HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

SECURITY
COMPETENCE

Hacking-Lab 2.0

Bachelorarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Frühjahrssemester 2018

Autor(en):	Janick Engeler, Yanick Gubler
Betreuer:	Ivan Bütler
Projektpartner:	Security Competence GmbH, 8645 Jona SG
Experte:	Benjamin Fehrensen
Gegenleser:	Daniel Keller

1 Aufgabenstellung

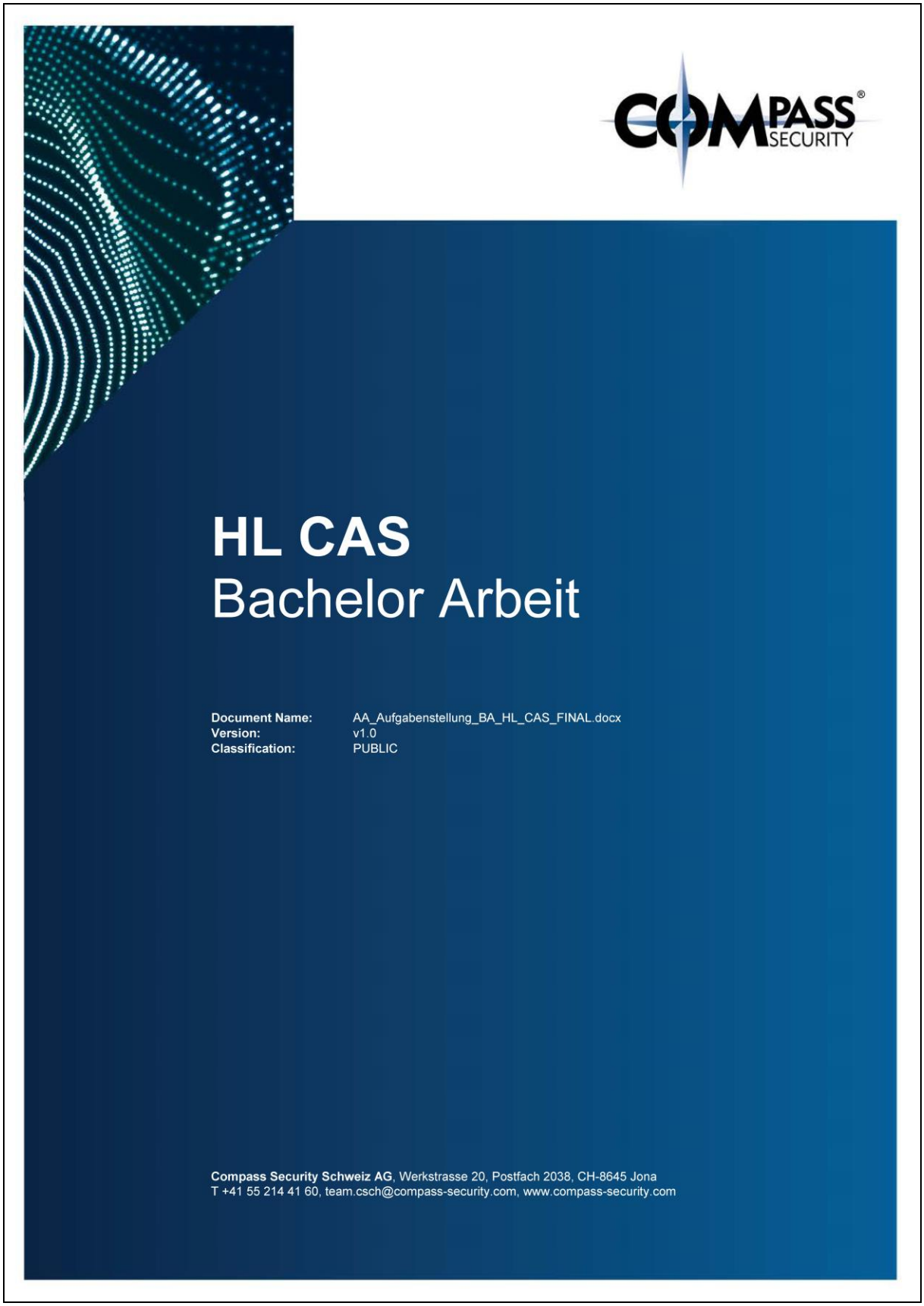


Abb. 1 Aufgabenstellung - Seite 1

Table of Contents

Abb. 2 Aufgabenstellung - Seite 2

1 Challenge Authoring System

1.1 Einleitung

Mit der zunehmenden Digitalisierung und der wachsenden Anzahl von Hackerangriffen wächst der Anspruch für eine adäquate Security Ausbildung von angehende Cyber Security Spezialisten. Neben der Vermittlung von Theorie wird dem praktischen Üben ein hoher Stellenwert beigemessen. Zu diesem Zweck stellt das Hacking-Lab seit dem Jahre 2007 eine Online Plattform mit sogenannten Security Challenges bereit. Das Hacking-Lab ist sehr erfolgreich mit der Bereitstellung von Übungen, aber vom Look & Feel auch ein wenig in die Jahre gekommen und das Erfassen von neuen Übungen und Challenges ist ausschliesslich den Betreibern von Hacking-Lab möglich.

1.2 Um was geht es

Mit dieser Bachelorarbeit soll ein Erfassungs- und Verwaltungssystem für Übungen (Challenges) entwickelt werden, welches im neuen Hacking-Lab 2.0 produktiv eingesetzt werden soll. Die Grundkonzeption von Hacking-Lab 2.0 wurde in der Studienarbeit von Janick Engeler und Yanick Gubler entwickelt und in ihrem Abschlussbericht dokumentiert. Nun geht es in der Bachelorarbeit darum das Konzept mit Mitarbeitern von Hacking-Lab zu validieren und umzusetzen.

1.3 Begriffsdefinitionen

Die folgenden Begriffe (nicht vollständig) werden in dieser Aufgabenstellung verwendet und eingesetzt. Detaillierte Angaben zu diesen Komponenten kann der Leser der Studienarbeit "Hacking-Lab 2.0" entnehmen.

Kürzel	Beschreibung	Details
CCS	Challenge Consumer System	Hauptanwendung aus der Sicht des Benutzers. Am CCS authentisiert kann, der Benutzer eine Liste von Aufgaben abrufen und die Details und Hints zu einer Aufgabe öffnen. Zudem bietet der Consumer eine einfache Möglichkeit für die Bereitstellung von Ressourcen, die bei einer Challenge verwendet wird (z.B. Docker, VM, File, …)
CAS	Challenge Authoring System	Web Anwendung die Verwaltung von Challenges. Ein Benutzer kann in dieser Anwendung eine Challenge erfassen, bestehende Challenges editieren oder eine Übersetzung von einer Challenge durchführen.
CRS	Challenge Ressource System	RESTful API für das Management von Ressourcen. Damit ein Benutzer bei einer Übung eine Ressource benutzen kann, muss diese bei Docker beispielsweise zuerst gestartet und instanziert werden. Das CRS System übernimmt die entsprechende START, STOP, RESTART, DELETE Funktion.
SSO	Single-Sign-On	Hacking-Lab 2.0 basiert auf einem SSO Service. Das bedeutet, dass die Benutzer-Registrierung und Authentisierung von den eigentlichen Anwendungen ausgelagert ist.
REALM	Bereich oder Territorium	Hacking-Lab 2.0 hat die Möglichkeit die Benutzer innerhalb des SSO in einem REALM zusammen zu fassen. Ein REALM ist ein organisatorisches Gefäss um alles was zu einem Kunden oder Mandaten gehört zu kapseln.
CDS	Challenge Directory System	Hacking-Lab 2.0 erlaubt es, dass ein Kunde mehrere eigene oder fremde CCS, CAS und CRS benutzt oder betreibt. Um die Übersicht über diese Systeme nicht zu verlieren, werden die Nutzungsbeziehungen der Systeme in einem zentralen Directory zusammengefasst. Damit kann man beispielsweise Auswertungen und Reports erzeugen die Auskunft darüber geben, wo und in welcher Sprache eine Challenge verfügbar ist.

Abb. 3 Aufgabenstellung - Seite 3

1.4 Grundlegender Ablauf aus Sicht eines Benutzers der eine Challenge löst

Der Benutzer besucht die Webseite seines CCS (www.ctf.com) und möchte dort einige Übungen zum Thema "Remote Attacks" durchführen. Nach der Registrierung erhält der Benutzer Zugriff auf eine Auswahl von bereitgestellten Challenges. Nun entscheidet sich der Benutzer für die "WordPress Challenge". Er klickt auf die entsprechende Challenge und erhält die Aufgabenstellung direkt im Browser angezeigt. Im Hintergrund hat der CCS via CRS einen On-Demand Docker mit einer verwundbaren WordPress Instanz gestartet, so dass der Benutzer diesen Docker auf Sicherheitslücken hin untersuchen kann.

Nachdem der Benutzer die Sicherheitslücke identifiziert und ausgenützt hat (Exploit), reicht dieser seine Lösung im CCS ein. Anschliessend wird die Lösung von einem Teacher bewertet und der Benutzer erhält für seine Leistung entsprechend Punkte. Nun hat der Benutzer die erste Challenge erst einmal abgeschlossen und kann mit der nächsten Challenge beginnen.

1.4.1 Steps, Instructions und Hints

Jede Challenge hat einen Titel, Abstract, gehört zu einer Kategorie, hat einen Schwierigkeitsgrad und besteht aus einer konkreten Aufgabenstellung, auch Mission genannt. Zudem kann man einer Challenge zusätzliche Steps mit Instruktionen oder Hints hinzufügen. Eine Instruktion ist dabei eine ganz konkrete Anweisung, was der Benutzer als nächstes für das Lösen der Challenge tun muss.

Example Instruction

```
discover the vulnerability with the command line tool "wpscan"
>> wpscan -u https://www.the-vulnerable-site.com/
```

Ein Hint ist weniger klar und eindeutig wie eine Instruction. Sie gibt dem Benutzer eher einen Hinweis in welche Richtung dieser beim Lösen der Aufgabe nachdenken soll. Beispielsweise folgender Hint:

Example Hint

```
find a tool that is specialized finding WordPress vulnerabilities
```

1.4.2 Write-Up und Gold-Nugget Lösungen

Das Challenge System unterstützt zwei Arten von Lösungen. Für die Lehre und bei der Vermittlung von Wissen mit einem Dozenten wird der Benutzer aufgefordert, ein Write-Up der Challenge einzureichen. Das Write-Up erklärt die identifizierte Schwachstelle, die Beschreibung oder das Tool für das Ausnützen der Sicherheitslücke und die empfohlenen Gegenmassnahmen für die Behebung des Problems. Die Korrektur des Write-Up ist ein manueller Prozess, indem ein Teacher das eingereichte Write-Up liest und bewertet.

Für Prüfungen und der Nutzung des Hacking-Lab in Wettbewerben und CTF (Capture-The-Flag) Events ist eine automatisierte Überprüfung der eingereichten Lösung möglich. Dabei findet der Benutzer nach dem Lösen der Challenge ein Lösungswort, auch Gold-Nugget genannt. Dieses Lösungswort soll der Benutzer anschliessend online einreichen und erhält ein unmittelbares Feedback ob die Lösung korrekt und akzeptiert wird. Sofern das System die Lösung akzeptiert, wird dem Benutzer die Punktezahl für die Challenge gutgeschrieben.

1.4.3 Punktevergabe

Wer eine Aufgabe ohne Instructions und Hints löst, erhält die volle Punktezahl. Falls der Benutzer auf Hints oder Instructions zurückgegriffen hat, so wird die maximale Punktezahl um einige Punkte reduziert. Falls der Benutzer alle Hints und alle Instructions eingesehen hat, dann erhält der Benutzer noch 50% der maximalen Punktezahl. Abhängig von der maximalen Anzahl von Hints und Instruktionen und den eingesehenen Hints und Instruktionen wird dem Benutzer pro-rata ein Abzug gemacht, jedoch maximal 50% der maximalen Punktezahl.

Abb. 4 Aufgabenstellung - Seite 4

1.5 Grundlegender Ablauf aus Sicht eines Challenge Autors der eine Challenge erfasst

Der Autor besucht die Webseite seines CCS (www.ctf.com) und meldet sich mit einem Account an, welcher über die Rolle "Autor" verfügt. Klickt der Autor im CCS auf "My Challenges", dann wird der Benutzer auf das Web GUI des CAS weitergeleitet. Das ist eine Web-Anwendung auf dem CAS System für die Verwaltung von Challenges. Die Web View des CCS ist abhängig von der Rolle des eingeloggten Benutzers.

1.5.1 CAS Rollenmodell

Der Benutzer mit der Rolle REALM CAS ADMIN kann auf sämtliche Challenges von seinem REALM zugreifen und diese editieren oder übersetzen. Der Benutzer mit der REALM CAS AUTHOR Rolle kann lediglich auf Challenges zugreifen, die er selbst erfasst hat. Der Benutzer mit der Rolle REALM CAS TRANSLATOR kann auf alle existierenden Challenges seines REALM zugreifen und diese mutieren oder übersetzen. Diese Rolle hat aber keine Rechte für das Erstellen von neuen Challenges. Ein Autor kann immer nur Challenges bearbeiten die seinem REALM zugeordnet sind.

1.5.2 Challenge Editor

Der Editor für die Erfassung von Challenges ist auf Einfachheit und Usability getrimmt, und an die Funktionalität wie auch Look&Feel von existierenden Web Editoren wie beispielsweise www.medium.com oder paper.dropbox.com angelehnt. Im Hintergrund sollen die Daten im Markdown Format gespeichert werden. s

1.5.3 Public versus Private Challenges

Der Challenge Autor kann eine Challenge als "public" markieren. Damit gibt er sich einverstanden, dass Nutzer von anderen REALMs die Challenge übernehmen dürfen. Wird eine Challenge als private markiert, so ist die Challenge ausschliesslich für den eigenen REALM gedacht und darf diesen REALM nicht verlassen. Hat ein Kunde jedoch mehrere CAS, so ist eine Verteilung der Challenge auf die eigenen CAS des gleichen REALM zugelassen. Das könnte heissen, dass eine neue Challenge die in Europa erstellt wird automatisch an das eigene Challenge System in den USA transferiert wird. Durch diesen Mechanismus ist gewährleistet, dass ein Kunde eigene Challenges auf den eigenen CAS weltweit nutzen kann, diese aber vor anderen Kunden des Hacking-Lab geheim bleiben.

1.5.4 Hacking-Lab Vault

Das Hacking-Lab steht nicht nur als Online Service zur Verfügung, sondern kann wahlweise auch als isolierte Umgebung im Intranet oder einem isolierten Netzwerk (Vault) betrieben werden. Das bedeutet, dass diese Installationen auch ohne Internetverbindung und ohne Zugriff auf das zentrale Internet CDN funktionieren muss. Bei der Vault Installation erhält der Kunde sein eigenes CAS System, damit er die installierten Challenges mutieren und auch neue Challenges erfassen kann. Der automatische Abgleich der installierten oder neuen Challenges mit dem zentralen Challenge Repository ist nicht möglich. Falls erwünscht kann der Abgleich manuell während dem Maintenance und Update Fenster gemacht werden. Dadurch kann das Vault System auch von Changes des Public Challenges Repositories profitieren.

1.5.5 Hacking-Lab Connect

Hacking-Lab empfiehlt ihren Kunden, dass ihre Hacking-Lab Installationen mit dem Internet verbunden sind. Dadurch ist gewährleistet, dass diese Installationen von Updates bei den Challenges, Verbesserungen wie auch Übersetzungen von Challenges profitieren. Nimmt der Kunde über sein eigenes CAS eine Veränderung an einer "public" Challenge vor, so wird diese Veränderung in das zentrale Internet Challenge Respository gespeichert. Andere Challenge Kunden können automatisch von dieser Veränderung profitieren und entscheiden, ob sie auf der "latest" Version oder dem aktuell installierten Release bleiben wollen. Eine "private" Challenge wird nicht mit dem Internet Challenge Repository abgeglichen, steht aber für alle CAS des Kunden analog Kapitel 1.5.3 bereit.

1.5.6 Review von Changes

Veränderung an einer Challenge ist innerhalb der Kundeninstallation sofort wirksam. Wird die Challenge jedoch auf dem zentralen Challenge Repository synchronisiert, so muss der Change durch einen Challenge Reviewer freigeschalten werden. Erst dann ist die überarbeitete Challenge für andere Kunden verfügbar.

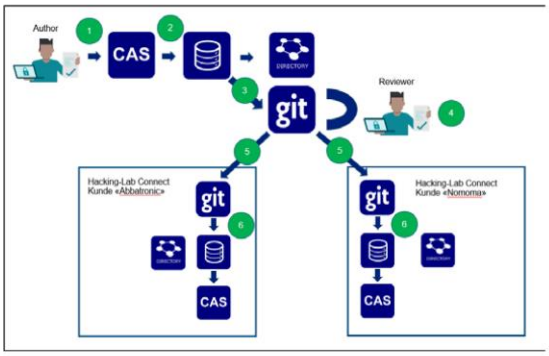
Abb. 5 Aufgabenstellung - Seite 5

1.6 Synchronisierung von Challenges

1.6.1 Einleitung

Hacking-Lab 2.0 ist ein weltweit verteiltes Challenge System. Das führende System von Challenges wird durch Hacking-Lab mittels einem zentralen Challenge Repository realisiert. Es ist der Dreh- und Angelpunkt für die Synchronisation von Challenges innerhalb von Hacking-Lab Installationen.

1.6.2 Update einer "public" Challenge durch Hacking-lab



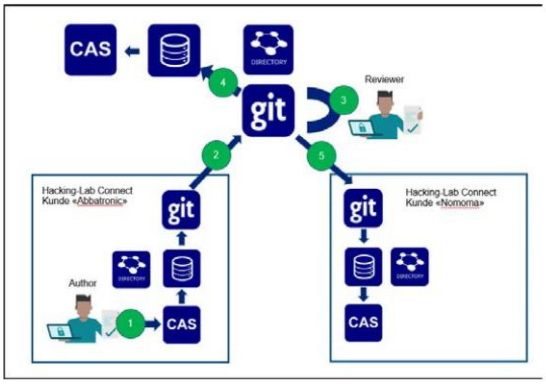
In der Darstellung links modifiziert ein Hacking-Lab Autor (1) im zentralen Hacking-Lab Challenge Repository eine Challenge. Die Metadaten der Challenge werden in der CAS Datenbank (2) persistiert.

Das CAS hat die Fähigkeit, alle zu einer Challenge gehörenden Daten (ohne Ressourcen) als Flat Files zu exportieren (3) und im zentralen Git Repository zu aktualisieren. Nach der Prüfung des Changes durch einen "Reviewer (4) im Git System" wird ein "git commit" gemacht und das Update auf das zentrale Challenge Repository geschrieben.

Kunden mit der "Hacking-Lab Connect" Installation können wählen, ob ihre Installation die (git) Changes automatisch übernimmt (5), oder ob der Kunde manuell die Changes prüfen und übernehmen will.

Falls der Kunde ein Update übernimmt, wird die aktualisierte Challenge vollständig über die CAS Import Funktion aus dem lokalen Git Repository übernommen und in die Datenbank geladen. Danach steht sie den Benutzern zur Verfügung.

1.6.3 Update einer "public" Challenge durch einen Hacking-Lab Connect Kunden



In der Darstellung links aktualisiert ein "Hacking-Lab Connect" Kunde eine Challenge (1). Dies könnte eine Anpassung an der Aufgabenstellung sein, oder er macht sich an eine Übersetzung der entsprechenden Challenge.

Der Autor erfasst die Changes über seine CAS Anwendung (1) und speichert das Update in der lokalen Datenbank. Anschliessend werden die Datenbank Informationen in das lokale GIT exportiert und dort via "git commit" (2) in eine spezielle "Inbound Git Queue" beim zentralen Git Repository übernommen.

Der Reviewer (3) überprüft den Change und gibt die finale Version frei, was die Übernahme der Changes auf dem zentralen Challenge Repository (4) und Übernahme bei anderen Hacking-Lab Connect Kunden (5) bewirkt.

Über diesen Mechanismus werden Changes von Hacking-Lab Kunden übernommen und verteilt. Dies gilt jedoch nur für Challenges die als "public" markiert sind.

Abb. 6 Aufgabenstellung - Seite 6

1.6.4 Erfassung einer "public" Challenge durch einen Hacking-Lab Connect Kunden

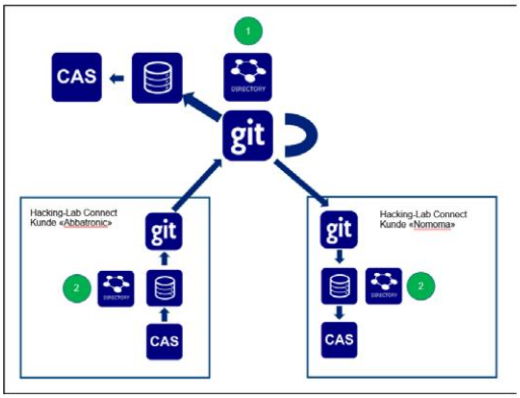
Wenn ein Hacking-Lab Connect Kunde eine neue Challenge erfasst, dann bedeutet dies implizit, dass ein neuer Eintrag im globalen Directory notwendig ist. Aus Sicherheitsüberlegungen möchten man jedoch Hacking-Lab Connect Kunden keine Zugriffsrechte auf das globale Directory geben. Daher werden die Daten für die Erzeugung des entsprechenden Directory Eintrag als Flat Command File beim Kunden erstellt und über den git commit an das zentrale Repository übergeben. Der Reviewer beim zentralen Challenge Repository

1.6.5 Git Konzept

Damit nicht jede Hacking-Lab Instanz auf alle Challenges im zentralen Git System zugreifen kann, wird das Git System wiefolgt aufgebaut.

- Im zentralen Git Repository erhält jede Challenge ein eigenes Repository
- Jeder Kunde erhält zudem ein eigenes Git Repository und wird darauf berechtigt (User Authentication)
  - Das Kunden Repository besteht aus den lizenzierten Challenges
  - Jede Challenge im Kunden Repo ist ein Git Submodule
  - Der Kunde ist auf dem Submodule berechtigt

1.7 Directory Service



Das Zentrale Directory System (1) ist ein Auskunftssystem über die gesamte Hacking-Lab Systemlandschaft. Sie gibt Auskunft über die weltweit verfügbaren "public" CCS, CAS, CRS und CRSS. Das Auskunftssystem hat kein Wissen über "private" Challenges oder Hacking-Lab Installationen, die als "geheim" eingestuft sind.

Aus diesem Grund verfügt jede Hacking-Lab Installation über ein eigenes Directory System. Dieses Auskunftssystem gibt ausschliesslich Auskunft über die CCS, CAS, CRS und CRSS innerhalb des gleichen REALM.

Diese Massnahme ist notwendig, damit überall die gleiche Architektur verwendet werden kann, unabhängig ob die Hacking-Lab Installation mit dem Internet verbunden (Hacking-Lab Connect), losgelöst vom Internet (Hacking-Lab Vault) oder anderweitig als "geheim" eingestuft wird.

1.7.1 Neue "public" Challenges eines Hacking-Lab Connect Kunden

Wenn ein Hacking-Lab Connect Kunde eine neue public Challenge erfasst, dann wird beim Erfassungsprozess das lokale Directory System benutzt. Bei der Synchronisation mit dem zentralen Git Repository gemäss der Beschreibung unter dem Kapitel 1.6.3 werden auch Informationen des lokalen Directory an das globale Directory übermittelt. Damit wird sichergestellt, dass das globale Directory über lokale Challenges informiert wird (sofern diese nicht private oder geheim sind).

1.8 Unterschrift

Jona, 15. Juli 2018, Ivan Bütler (ibuetler@hsr.ch)

Section 1 Challenge Authoring System 7/7

Abb. 7 Aufgabenstellung - Seite 7

2 Abstract

Die Security Competence GmbH betreibt seit dem Jahre 2007 eine Internet Security Hands-On Trainingsumgebung für angehende Security-Spezialisten. Das Portal mit dem Namen «Hacking-Lab» wird von über 80'000 Benutzern weltweit genutzt. Aufgrund der veralteten Technik und des hohen Verwaltungs- und Pflegeaufwands der bestehenden Lösung, soll das System auf den neusten Stand gebracht und sogleich der Grundstein für die Zukunft gelegt werden. In einer vorgelagerten Studienarbeit stellte man Überlegungen an, um die Ziele einer hohen Skalierbarkeit, der Internationalisierung des Systems, sowie des weltweiten Einsatzes über das Internet (Cloud) oder direkt vor Ort beim Kunden (On-Premise) zu erreichen. Aus dem daraus entstandenen Konzept (Hacking-Lab 2.0), soll in dieser Bachelorarbeit eine konkrete Implementierung entstehen, welche sich auf eine Kernaufgabe des neuen Hacking-Labs – die vereinfachte Erfassung von Übungseinheiten (Challenge-Editor) und deren Übersetzung – fokussiert. Als Herausforderung ist die fehlende Hacking-Lab 2.0 Architektur zu sehen, die für die Bereitstellung des Challenge-Editors vorausgesetzt wird (Benutzerverwaltung, Registrierung, Autorisierung, Load-Balancing) und parallel zur Entwicklung mit aufgebaut wurde.

Das Ergebnis dieser Bachelorarbeit ist eine funktionierende Hacking-Lab 2.0 Infrastruktur auf Basis von Docker, Load-Balancer, OAuth Identity Provider, Verwaltung und Orchestrierung von Übungseinheiten via Git und Submodules, als auch der User Federation via LDAP. Der darauf aufbauende Challenge-Editor ermöglicht, durch ein intuitives User Interface, die vereinfachte Erfassung, Mutation und Übersetzung von Challenges. Durch die gewählte Abstraktion der umgesetzten Services (Microservice Architektur) und der systematischen Verwendung eines RESTful APIs wird es für den Industriepartner möglich, die erstellte Software unmittelbar einzusetzen und zu integrieren. Zudem ist es bedenkenlos realisierbar, die Software bei einem beliebigen Cloud Provider (Amazon AWS, Azure, Digital Ocean, ...) zu deployen und zu nutzen.

3 Management Summary

3.1 Ausgangslage

Sicherheit bekam in den letzten Jahren in allen Bereichen der Gesellschaft einen immer höheren Stellenwert. Auch die IT ist immer stärker von dieser Thematik betroffen. Es werden laufend grössere Summen aus allen Bereichen der Wirtschaft in die IT-Sicherheit investiert. Dies kann einerseits darauf zurückgeführt werden, dass kriminelle Organisationen kontinuierlich neue Wege suchen, um ein System und dessen Benutzer anzugreifen und andererseits das Bewusstsein der Benutzer steigt, die mit solchen Systemen in Kontakt treten und deshalb verstärkt die Sicherheitsaspekte als entscheidende Anforderungen gesehen werden. Neben der Sicherheit eines Systems spielt der Wissensstand der Benutzer eine entscheidende Rolle. Dadurch wird es fortlaufend wichtiger, nicht nur der Schulung von Experten, sondern auch des einzelnen Benutzers Beachtung zu schenken. Um der Wichtigkeit des Faktors der Benutzerschulung bzw. Sensibilisierung Rechnung zu tragen, wurde vor einigen Jahren die Hacking-Lab Umgebung entwickelt.

Zusammengefasst dient das bestehende Hacking-Lab in erster Linie dazu, interessierten Benutzern eine Plattform zu bieten, auf welcher sie ihre erworbenen Kenntnisse zur IT-Sicherheit anhand verschiedenster Aufgaben (Challenges) prüfen, sich mit anderen Benutzern messen oder ihr Wissen erweitern können. Dabei ist zu beachten, dass die Benutzer diese Plattform vorwiegend in ihrer Freizeit verwenden und so ein Unternehmen nur geringfügig in den Genuss eines verbesserten Sicherheitsbewusstseins ihrer Angestellten kommt. Damit das neue Hacking-Lab System direkt von Firmen mit wenig Aufwand intern eingesetzt und betrieben werden kann, ist es Ziel dieser Arbeit, das Hacking-Lab als ein in sich geschlossenes System zu entwickeln, damit es direkt in bestehende Infrastrukturen integriert werden kann und das Unternehmen sehr schnell vom erweiterten Sicherheitsbewusstsein seiner Mitarbeiter profitiert.

Der Betrieb des bestehenden Hacking-Labs verursacht viel Aufwand. Einerseits um neue «Challenges» zu erfassen und andererseits um bestehende zu verbessern. Dies ist auf die umständliche Umsetzung dieses Prozesses zurückzuführen. Das Ziel dieser Arbeit beinhalten die Erfassung und Modifikation von «Challenges» für alle laufenden Instanzen zu vereinheitlichen und zu vereinfachen. Damit auch internationale Firmen oder andere Länder vom neuen Hacking-Lab profitieren können, war es eine weitere Anforderung, alle «Challenges» in eine beliebige Sprache übersetzen zu können.

Damit auch Kunden des neuen Hacking-Lab Systems von neu erstellten oder verbesserten «Challenges» profitieren können, ist es Ziel der Bachelorarbeit diese aus einem laufenden System exportieren und in einem anderen wieder importieren zu können. Um zu verhindern, dass falsche, oder Aufgaben minderer Qualität, in einem anderen System eingefügt werden, wird ein Review-Prozess integriert, durch welchen die Qualität der «Challenges» hochgehalten werden kann.

3.2 Vorgehen / Technologie

Die Arbeit baut auf der vorhergehenden Studienarbeit desselben Projektteams auf. Zum einen wurden dort bereits verschiedenste Konzepte für das neue Hacking-Lab entwickelt und zum anderen ein erster Entwurf eines Prototyps erstellt. Basierend auf diesen Erkenntnissen und den neu gestellten Anforderungen betreffend der Kapselung des Systems, wurde die bestehende Infrastruktur neu durchdacht und entsprechend angepasst. Der erstellte Prototyp erfuhr einzelne architektonische Änderungen und wurde anhand der neuen Ideen erweitert. Da der Überprüfungsprozess der Challenges eine zentrale Komponente des neuen Hacking-Labs darstellt, musste eine einfache Möglichkeit geschaffen werden, Aufgaben zu exportieren und anhand des Exports zu entscheiden, ob eine Änderung vorliegt. Falls dies der Fall ist, muss die Qualität der Neuerungen überprüft und bei Bedarf allen Benutzern mittels Import zur Verfügung gestellt werden.

Der Rahmen dieser Bachelorarbeit beinhaltet nicht den gesamten Umfang des neuen Hacking-Lab Systems, sondern nur einzelne Kernfunktionalitäten davon. Die anderen Teile wurden direkt vom Auftraggeber entwickelt, was eine genaue Abstimmung erforderte. Dies brachte Risiken mit sich, da somit eine exakte Trennung der verwendeten Dienste erstellt und eingehalten werden musste, um so eine gemeinsame Entwicklung zu ermöglichen. Weiter blieb dadurch die Wahl einer geeigneten Technologie aus, da diese bereits vorgegeben wurde.

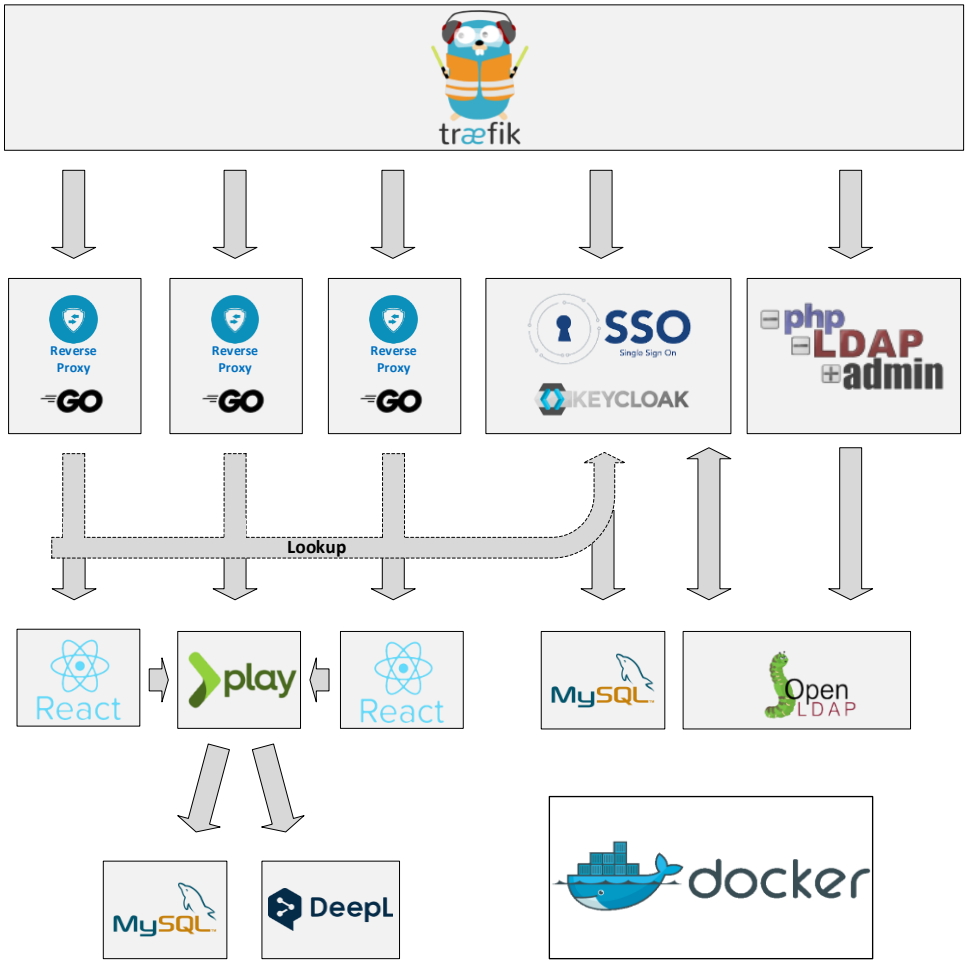


Abb. 8 Schematische Architektur und verwendete Technologie / Produkte

3.3 Ergebnisse

Als Resultat dieser Arbeit wurde neben der gesamten System-Infrastruktur ein Editor erstellt, mit welchem auf einfachste Weise neue «Challenges» erfasst und bestehende modifiziert werden können. Zusätzlich wurde eine Komponente entwickelt, mit welcher diese übersetzt und bestehende Übersetzungen bearbeitet werden können. Damit der Überprüfungsprozess für neu erstellte oder angepasste Challenges funktionieren kann, wurde eine weitere Applikation erstellt, die es ermöglicht, eine «Challenge» aus dem Dateisystem zu laden und in das Hacking-Lab System zu importieren, sowie den aktuellen Stand einer «Challenge» aus der Umgebung zu exportieren und wieder in ein Dateisystem zu persistieren. Dabei muss beachtet werden, dass während dem Import / Export Prozess kein Zeichen verändert werden darf, damit der Review korrekt durchgeführt werden kann.

Die System-Infrastruktur wurde so aufgebaut, dass eine Neuinstallation bzw. die Integration in eine bestehende Systemlandschaft einfach möglich wird. Dies wurde durch das Auslagern der Konfiguration der gesamten Architektur in eine zentrale Datei ermöglicht. Die Infrastruktur kann dadurch mit sehr geringem Aufwand auf nahezu jeder Umgebung in Betrieb genommen werden. Einer der Vorteile des neuen Systems ist, dass es fast beliebig erweitert und skaliert werden kann, indem die zentrale Konfigurationsdatei entsprechend angepasst wird.

3.4 Ausblick

Der Quellcode der entwickelten Lösung, sowie die gesamte Infrastruktur mit deren initialen Konfiguration ist innerhalb eines privaten Repositories abgelegt und wird nach Abschluss der Bachelorarbeit durch das Projektteam veröffentlicht. Dadurch können Interessierte die entwickelte Applikation bei sich in Betrieb nehmen und die implementierten Ergebnisse testen.

Ein von einem anderen Autor entwickelter Teil der «Challenge-Editor» Komponente, der während dieser Arbeit auf eine andere Softwarebibliothek adaptiert und grundlegend weiterentwickelt wurde, wird nach Projektende ebenfalls veröffentlicht, damit er von anderen Entwicklern verwendet oder weiterentwickelt werden kann.

Der nächste Schritt für die Ergebnisse dieser Bachelorarbeit, ist die teilweise Integration in den Kontext der vom Auftraggeber parallel dazu entwickelten Lösung. Da die Architektur modular aufgebaut und kontinuierliche Rücksprachen mit dem Auftraggeber erfolgten, sollte dies kein Problem darstellen.

4 Inhaltsverzeichnis

5 Technischer Bericht

5.1 Einführung

5.1.1 Ausgangslage und Rahmenbedingungen

Sicherheit bekam in den letzten Jahren in allen Bereichen der Gesellschaft einen immer höheren Stellenwert. Auch die IT ist immer stärker von dieser Thematik betroffen. Es werden laufend grössere Summen aus allen Bereichen der Wirtschaft in die IT-Sicherheit investiert. Dies kann einerseits darauf zurückgeführt werden, dass kriminelle Organisationen kontinuierlich neue Wege suchen, um ein System und dessen Benutzer anzugreifen und andererseits das Bewusstsein der Benutzer steigt, die mit solchen Systemen in Kontakt sind und deshalb verstärkt die Sicherheitsaspekte als entscheidende Anforderungen gesehen werden. Da neben der Sicherheit eines Systems, der Wissensstand der Benutzer eine entscheidende Rolle spielt, wird es immer wichtiger, nicht nur der Schulung von Experten, sondern auch des einzelnen Benutzers, Beachtung zu schenken. Um der Wichtigkeit des Faktors der Benutzerschulung bzw. Sensibilisierung Rechnung zu tragen, wurde im Jahr 2007 die Hacking-Lab Umgebung entwickelt.

Zusammengefasst dient das bestehende Hacking-Lab in erster Linie dazu interessierten Benutzern eine Plattform zu bieten, auf welcher sie ihre erworbenen Kenntnisse zur IT-Sicherheit anhand verschiedenster Aufgaben (Challenges) prüfen, mit anderen Benutzern messen oder erweitern können. Da das bestehende Hacking-Lab-System nicht mehr dem aktuellen Stand der Technik entspricht, wurde in der vorgelagerten Studienarbeit ein Konzept ausgearbeitet, mit dem Ziel das Hacking-Lab der Zukunft zu entwickeln. Dieses Konzept definiert die Rahmenbedingungen für die vorliegende Bachelorarbeit. Die Ergebnisse der Studienarbeit, wie der erstellte Prototyp und die damalige Docker-Infrastruktur, wurden als Grundlage für die zu erstellende Applikation verwendet. Zu Beginn der Bachelorarbeit wurden die bestehenden Komponenten analysiert, um sie anschliessend durch ein Re-Design optimal den nun genauer spezifizierten Anforderungen anzupassen.

5.1.2 Ziele

Ausgehend aus der Konzeptanalyse der Studienarbeit soll in dieser Bachelorarbeit eine in sich geschlossene Infrastruktur aufgebaut werden, die den neuen Anforderungen an das System, hohe Skalierbarkeit, Internationalisierung und dem weltweiten Einsatz über die Cloud oder direkt beim Kunden (On-Premise), gerecht werden soll. Dabei ist entscheidend, dass das System mit kleinem Aufwand in Betrieb genommen und sich in bestehende Umgebungen integrieren lässt, damit sie potenziellen Kunden mit wenigen Konfigurationsschritten innerhalb ihres bestehenden Netzwerks zur Verfügung steht. Als weitere zu implementierende Komponente wird eine Kernfunktionalität des neuen Hacking-Labs, die vereinfachte Erfassung und Übersetzung von Challenges, erwartet, die direkt in die Umgebung integriert sein soll.

Die in der Studienarbeit erstellten Prototypen werden analysiert, um diese im Anschluss zu optimieren und neue Funktionalitäten zu implementieren. Entscheidend ist die einfache Verwendung der einzelnen Komponenten sowie eine homogene Microservice-Architektur, damit diese in den extern entwickelten Kontext und damit in die aufgebaute Infrastruktur integriert werden können.

5.2 Problembeschreibung

5.2.1 Challenge-Editor Implementierung

Im folgenden Unterkapitel werden die Anforderungen an die zu implementierenden Kernfunktionalitäten, der Erfassung, Modifikation und Übersetzung von Challenges beschrieben. Die Hauptaufgabe bestand darin eine Webapplikation für die genannten Aufgaben umzusetzen, welche die nachfolgenden weiteren Anforderungen mit sich bringt.

5.2.1.1 Markdown im Browser

Da eine Anforderung der Gebrauch von Markdown für die eigentlichen Inhalte einer Challenge ist, muss es möglich sein, dass Markdown im Browser korrekt dargestellt wird.

5.2.1.2 Editor mit Hilfestellung

Damit weniger erfahrene Benutzer den Editor sinnvoll verwenden und ihre Inhalte für eine Challenge entsprechend formatieren und gestalten können, muss eine Möglichkeit implementiert werden, die es diesen Benutzern erlaubt, den Editor ohne die Kenntnis von Markdown zu bedienen.

5.2.1.3 Editor mit Markdown-Syntax Unterstützung

Damit erfahrene Benutzer das Styling ihrer Inhalte nicht zwingend anhand der beschriebenen Hilfestellung durchführen müssen und dadurch gegebenenfalls ausgebremst werden, muss die Möglichkeit der Markdown-Syntax Erkennung umgesetzt werden, die darauf korrekt im Browser dargestellt wird.

5.2.1.4 Persistierung von Markdown

Die im Editor erfassten Markdown-Inhalte einer Challenge müssen in der Markdown-Syntax auf der Datenbank persistiert werden.

5.2.1.5 Automatische Speicherung der Editorinhalte

Damit bei einem Absturz des Browsers oder anderen möglicherweise auftretenden Problemen keine Daten verloren gehen können, muss es möglich sein, dass die erstellten, bzw. editierten Inhalte kontinuierlich in der Datenbank gespeichert werden. Dabei dürfen keine Zeitverzögerungen entstehen, um so die User Experience hochzuhalten.

5.2.1.6 Automatische Übersetzung von Challenges

Damit eine bestehende Challenge in anderen Sprachen zur Verfügung gestellt werden kann, müssen diese automatisch übersetzt werden können.

5.2.1.7 Manuelle Übersetzung von Challenges

Da eine automatisch übersetzte Challenge tendenziell fehlerbehaftet ist, muss eine Möglichkeit der manuellen Übersetzung bzw. Modifikation von automatisch übersetzten Challenges implementiert werden.

5.2.2 Challenge Synchronisation

Im folgenden Unterkapitel werden die Anforderungen an die zu implementierende Challenge-Synchronisation beschrieben. Dabei wird nur der tatsächlich umgesetzte Aspekt beleuchtet und der extern durch die Security Competence GmbH realisierte Mechanismus weggelassen. Die Synchronisation wird später durch einen Reviewprozess ergänzt, damit die Qualität der Challenges trotz kundenspezifischen Anpassungen hochgehalten und Fehler minimiert werden können.

5.2.2.1 Import von extern erstellten Challenges

Damit die Security Competence GmbH bereits vor der Fertigstellung des Editors Challenges erfassen kann, muss eine Möglichkeit implementiert werden, um bestehende Challenges aus dem Filesystem in die Datenbank zu importieren. Diese Funktionalität muss später für den Synchronisationsmechanismus von Challenges für externe Kunden verwendete werden können.

5.2.2.2 Export von Challenges in das Dateisystem

Damit Änderungen von Challenges durch den Editor für andere Kunden zugänglich gemacht werden können, muss eine Möglichkeit implementiert werden, mit welcher bestehende Challenges aus der Datenbank in eine definierte Filestruktur persistiert werden können.

5.2.2.3 Erstellung eines automatisierbaren Jobs

Damit der Import- / Export-Prozess für die spätere Verwendung automatisiert werden kann, muss eine Möglichkeit umgesetzt werden, dass dieser durch eine extern umgesetzte Software-Komponente gestartet werden kann.

5.2.2.4 Gleichbleibende Hash-Werten der Dateien

Damit Änderungen an Challenges durch den Editor korrekt vom automatisierbaren Job detektiert und entsprechend verarbeitet werden können, darf während der Übertragung kein einziges Bit verändert werden. Es muss also eine Möglichkeit geschaffen werden, dass während der Übertragung einer Challenge vom Dateisystem zur Datenbank und zurück kein Zeichen verändert wird, falls sie nicht tatsächlich verändert wurde.

5.2.2.5 Unterstützung von nicht europäischen Zeichen

Damit auch russische, chinesische oder hebräische Challenges in die Datenbank importiert und von dort zurück in das Dateisystem exportiert werden können, muss die Möglichkeit der Unterstützung aller Zeichensätze implementiert werden.

5.2.3 Infrastruktur

Im Folgenden werden die Anforderungen an die Services beschrieben, welche in einer Docker-Infrastruktur betrieben werden sollen.

5.2.3.1 Verschlüsselte Kommunikation

Um eine sichere Kommunikation über alle Dienste gewährleisten zu können, muss der gesamte Netzwerkverkehr verschlüsselt übertragen werden. Dafür muss eine Lösung entworfen werden, welche dies möglichst einfach umsetzen kann.

5.2.3.2 Einmalige Authentifizierung über alle Services

Da die Architektur auf Mikroservices beruht, muss eine Lösung gefunden werden, bei der die Sitzung eines angemeldeten Hacking-Lab Benutzers, für alle Services verwendet werden kann. Dies muss ohne eine Benutzerinteraktion realisierbar sein. Dadurch wird es möglich, dass der Benutzer beim ersten Aufruf eines Service seine Anmeldeinformationen angeben muss und bei Aufrufen weiterer Services die bestehende Sitzung übernommen wird.

5.2.3.3 Wiederverwendung der Benutzerlogins für andere Dienste

Die Anmeldedaten der Benutzer müssen auch von anderen Diensten verwendet werden können. So sollen beispielsweise Docker-Container, welche von einem CRS stammen, mit dem gleichen Benutzerlogin verwendet werden können. Weitere Verwendungszwecke wäre ein VPN-Service, welcher die User anhand ihres Hacking-Lab Accounts authentifiziert.

5.2.3.4 Dezentrale/skalierbare Architektur der Services

Die Services, wie der CAS-Server, der CAS-Client und der Consumer müssen in einer dezentralen Architektur betrieben werden können. So soll es nicht darauf ankommen, in welcher örtlichen Zusammenstellung die jeweiligen Services betrieben werden. Weiter muss die Möglichkeit bestehen, gewisse Komponenten wie der CAS-Client oder das Consumer-System mehrfach zu betreiben. Dadurch werden die Dienste skalierbar und die jeweiligen Antwortzeiten können möglichst gering gehalten und Lastspitzen vermieden werden. Durch die Skalierbarkeit könnte auch der Problematik der geografischen Distanz entgegengewirkt werden.

5.2.3.5 Absicherung der Passwörter in den Docker-Containern

Typischerweise werden Passwörter, welche von den jeweiligen Services in einem Docker-Container verwendet werden, per Environment-Variablen oder Docker-Entrypoints übergeben. Da diese durch alle Benutzer, welche Zugang zu den jeweiligen Containern haben, ausgelesen werden können, muss eine sicherere Alternative entworfen werden.

5.3 Lösungskonzept

5.3.1 Challenge-Editor Implementierung

In den folgenden Unterkapiteln werden die während der Bachelorarbeit entwickelten konzeptionellen Lösungen für die im Zusammenhang mit dem Challenge-Editor beschriebenen Probleme dargelegt.

5.3.1.1 Markdown zu HTML Parser für Markdown im Browser

Damit Markdown korrekt im Browser dargestellt werden kann, wird ein Parser benötigt, der die verwendete Markdown-Syntax in HTML-Elemente umwandelt. Dieser muss mit der im Editor unterstützten Syntax, sowie mit der Markdown-Syntax der extern erstellten Challenges, die in die Datenbank importiert werden können, kompatibel sein. Es wird dazu die Standard Markdown-Syntax [1] vorausgesetzt.

5.3.1.2 Toolbar als Hilfestellung für den Editor

Damit der Editor für weniger erfahrene Benutzer nutzbar wird, gibt es verschiedene Möglichkeiten, von denen zwei eingehend analysiert wurden.

5.3.1.2.1 Variante 1

Die erste geprüfte Möglichkeit ist die Verwendung eines statischen Menüs am oberen Ende des Editors. Da eines der Hauptziele des Editors jedoch ein modernes Design ist, wirkt sich ein statisches Menü negativ darauf aus. Ein weiterer Nachteil dieser Variante ist, dass Platz auf der Seite verschwendet wird, die anderweitig genutzt werden könnte.

5.3.1.2.2 Variante 2

Die zweite analysierte Variante war eine dynamische Toolbar, die nach dem Markieren eines Textabschnitts angezeigt und danach wieder ausgeblendet wird. Hierdurch kann der Platzbedarf minimiert sowie ein ansprechendes Design des Editors erreicht werden.

5.3.1.2.3 Entscheidung

Nach Absprache mit dem Betreuer und dem Aufzeigen der Vor- und Nachteile der beiden Varianten, wurde entschieden die Variante 2 weiter zu verfolgen und anhand eines Prototyps zu implementieren.

5.3.1.3 Texterkennung für Markdown-Syntax Unterstützung des Editors

Damit direkt eingegebene Markdown-Syntax zeitnah erkannt und in das entsprechende HTML Element geparsed werden kann, wird eine Texterkennung benötigt, die kontinuierlich den Inhalt analysiert und bei korrekter Markdown Eingabe entsprechend darstellt.

5.3.1.4 Converter für die Persistierung von Markdown

Damit die HTML-Inhalte des Editors wieder als Markdown in der Datenbank gespeichert werden können, wird ein Converter benötigt, der HTML zu Markdown überführt. Dieser muss auf die im Editor verwendeten HTML-Elemente abgestimmt werden.

5.3.1.5 Websockets für automatische Speicherung der Editorinhalte

Damit Inhalte des Editors laufend auf der Datenbank persistiert werden können, ohne dass unzählige POST Requests verschickt werden müssen, was zu Zeiteinbussen führen würde, bietet sich die Verwendung eines Websockets an. Der Vorteil dabei ist, dass dieser bidirektional funktioniert und asynchron die Daten auf der Datenbank speichert. Damit wird es zugleich möglich, dass beim User Interface keine Verzögerung für die Datenübertragung bemerkbar sind.

5.3.1.6 API für automatische Übersetzung von Challenges

Damit eine bestehende Challenge automatisch in eine andere Sprache übersetzt werden kann, muss entweder ein eigener Sprachdienst implementiert oder ein bestehender verwendet werden. Da für diese Problematik bereits viele etablierte Dienste existieren, und der Fokus dieser Bachelorarbeit nicht auf einem Übersetzungsdienst liegt, wurde in diesem Zusammenhang entschieden einen konventionellen Dienst zu verwenden. Die Anforderungen an einen solchen sind neben der sinnvollen Übersetzung, die einfache Verwendung des APIs, sowie der Preis für eine Übersetzung der Inhalte.

5.3.1.7 Editor für manuelle Übersetzung von Challenges

Damit eine bereits bestehende Challenge manuell übersetzt werden kann, muss eine weitere Editorkomponente implementiert werden, die analog zum bereits beschriebenen Challenge-Editor funktioniert und somit Markdown Inhalte korrekt als HTML rendert und wieder zu Markdown zurück konvertiert. Dazu kann entweder eine eigene Komponente implementiert oder die bereits analysierte, verwendet werden. Da kein gravierender Unterschied zwischen den beiden Editoren vorhanden ist, wurde entschieden direkt den Challenge-Editor als Grundlage für den Übersetzungseditor zu verwenden.

5.3.2 Challenge Synchronisation

In den folgenden Unterkapiteln werden die während der Bachelorarbeit entwickelten konzeptionellen Lösungen für die im Zusammenhang mit der Challenge Synchronisation beschriebenen Probleme dargelegt.

5.3.2.1 REST-API für den Import von extern erstellten Challenges

Damit extern erstellte Challenges in die Datenbank importiert werden können, wird vom Server eine REST-URI zur Verfügung gestellt. Als Body kann die Challenge mit den verwendeten Medien als JSON-Objekt übergeben werden. Um zu gewährleisten, dass das JSON Objekt bei jeder der zu importierenden Challenges die gleiche Form hat, wurde ein Konzept entwickelt bzw. eine Spezifikation für das Objekt definiert. Für die Form des eigentlichen Imports wurden drei Varianten ausgearbeitet und mit dem Betreuer und dem Auftraggeber besprochen.

5.3.2.1.1 Variante 1

Pro Markdown-Datei einer Challenge wird ein POST Request erstellt, der als Header-Feld den Typ der Markdown-Datei enthält (ABSTRACT, SOLUTION, HINT, INSTRUCTION) und im Body der eigentliche Markdown-Inhalt encodiert beinhaltet. Dazu müssten die in den Markdown-Files referenzierten Medien zusätzlich in den einzelnen Requests enthalten sein.

Für die Metadaten einer Challenge wird ein separater POST Request abgesetzt, der innerhalb eines JSON-Objektes die relevanten Daten wie Level, Type usw. einer Challenge beinhaltet. Der Nachteil dieser Variante ist, dass mehrere Requests abgesetzt werden, wobei geprüft werden müsste, dass jeder Teil korrekt gesendet und auf dem Server erhalten und persistiert werden konnte.

5.3.2.1.2 Variante 2

Der gesamte Inhalt einer Challenge (Metadaten, Medien und Markdown-Dateien) wird als JSON-Objekt innerhalb eines Request zum Server gepostet. Der Nachteil dieser Variante wäre, dass ein einzelner POST Request sehr gross werden könnte, da neben dem eigentlichen Inhalt einer Challenge sogleich alle Medien als Base64 codierte Strings im gleichen JSON zum Server übermittelt werden. Jedoch würde der Nachteil der ersten Variante, dass geprüft werden müsste, ob ein Request fehlschlug, hinfällig. Falls ein Request fehlschlagen würde, könnte dieser einfach erneut übermittelt werden.

5.3.2.1.3 Variante 3

Als dritte Variante würde sich ein File-Upload anbieten, mit welchem direkt die einzelnen Dateien einer Challenge auf den Server gepostet werden können. Innerhalb des Headers müsste eine Referenz auf die eigentliche Challenge mitgeschickt werden, um die einzelnen Dateien auf dem Server korrekt zuordnen zu können. Die Unterscheidung der einzelnen Markdown-Typen könnte anhand des Dateinamens gewährleistet werden, wenn diese mit einer definierten Struktur erstellt werden.

Ein entscheidender Nachteil dieser Variante ist, dass jede Datei auf dem Server zuerst korrekt geparst werden müsste, um mit der eigentlichen Markdown-Syntax und nicht als ByteString persistiert zu werden. Weiter wäre hier wieder der Nachteil der ersten Variante gegeben, da einzelne Requests übermittelt werden müssten, die entsprechend auf ihre Vollständigkeit geprüft werden müssten.

5.3.2.1.4 Entscheidung

Nach der Analyse und dem Gespräch mit dem Betreuer und dem Auftraggeber wurde entschieden die Variante 2 weiter zu verfolgen. Diese ist die erfolgversprechendste und effizienteste, weil keine Prüfung der einzelnen Requests benötigt wird und beim erfolgreichen Import einer neuen Challenge angenommen werden kann, dass kein Bit während der Übertragung verändert wurde.

5.3.2.2 REST-API für den Export von Challenges in ein Dateisystem

Damit bestehende Challenges aus der Datenbank exportiert werden können, bietet sich die Implementierung eines REST-APIs an. Da bereits für den Import einer Challenge das Format als ein JSON-Objekt festgelegt wurde, wurde definiert, für den Export das gleiche Format zu verwenden. Auf diesem Weg müsste die Struktur jedoch vom Server und nicht vom Benutzer des APIs generiert werden.

5.3.2.3 Java Applikation für die Automatisierung des Import- / Export-Jobs

Damit eine Challenge aus einem Dateisystem gelesen, in die für den Import benötigte Struktur gebracht werden und dann über das REST-API zum Server übermittelt werden kann, wird eine Java Applikation implementiert, die diesen Teil des Imports übernimmt. Analog wird für den Export einer Challenge die gleiche Applikation verwendet und die durch das REST-API angeforderte Challenge in die definierte Filestruktur abgebildet. Damit die Applikation automatisiert werden kann, sind die jeweiligen Operationen (IMPORT / EXPORT) durch einen Parameter ausführbar und mittels einer Konfigurationsdatei auf das jeweilige Betriebssystem adaptierbar. So kann die Applikation sowohl unter Unix, als auch Windows verwendet werden.

5.3.2.4 Encoding für die Erhaltung des Hash-Wertes der Dateien

Damit der Hash-Wert von unveränderten Teilen einer Challenge vor und während der Übertragung gleichbleibt, muss ein Encoding verwendet werden, dass auf beiden Seiten der Übertragung gleich umgesetzt wird.

5.3.2.5 Encoding für die Unterstützung nicht europäischer Zeichensätze

Damit alle verschiedenen Zeichensätze für den Import- / Export-Prozess unterstützt werden können, muss auf beiden Seiten der Übertragung ein UTF-8 Encoding verwendet werden. Dieses muss auf die gleiche Weise umgesetzt werden, damit der Hash-Wert der unveränderten Datei erhalten bleibt.

5.3.3 Infrastruktur

Im Folgenden werden die in der Bachelorarbeit entwickelten konzeptionellen Lösungen für die beschriebenen Herausforderungen dargelegt.

5.3.3.1 Reverse Proxy für verschlüsselte Kommunikation

Ein Reverse Proxy kann die Sicherheit eines Systems erheblich erhöhen, da alle Services, die hinter einem solchen Proxy betrieben werden, gegenüber dem Internet verschleiert werden. Dadurch wird es möglich, die tatsächlichen Adressen der Services nicht nach Aussen bekannt zu gegeben. Somit sind von aussen nur generische Adressen eines Reverse Proxys zugänglich wobei dieser entscheidet, an welche tatsächliche Adresse er einen Request weiterleiten soll. Dieser Aspekt soll durch folgende Grafik verbildlicht werden:

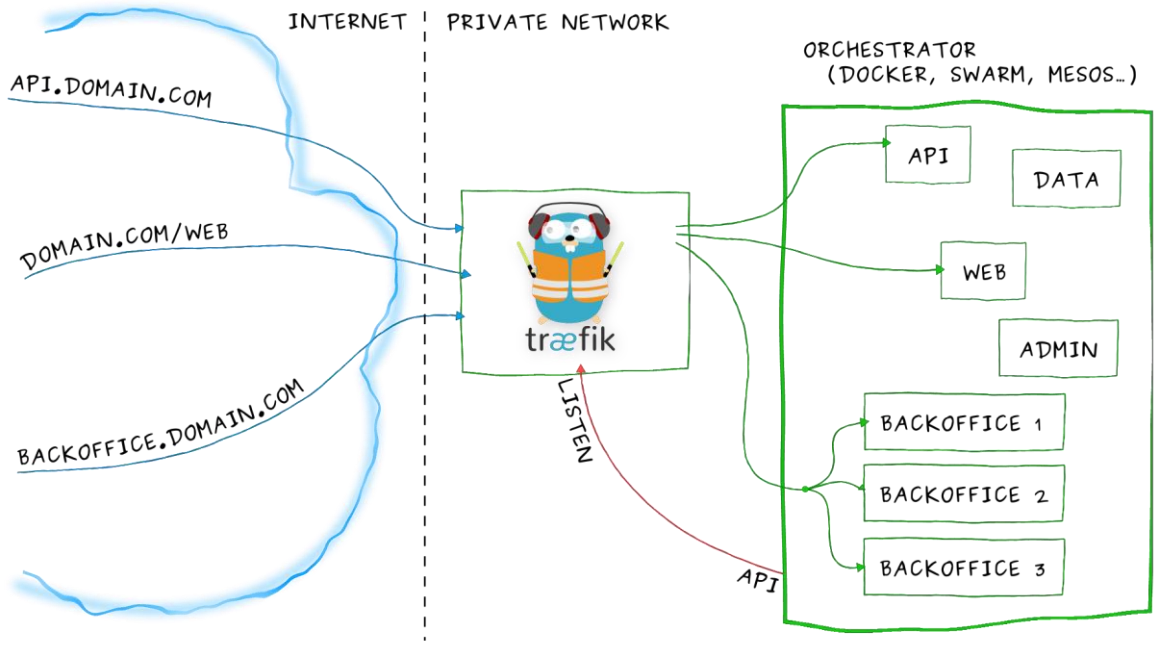


Abb. 9 Reverse Proxy - Traefik [2]

Da der Reverse Proxy als einziger Zugriffspunkt von ausserhalb des privaten Netzwerkes vorhanden ist, bietet es sich an, bereits an dieser Stelle den HTTP-Verkehr mittels TLS zu verschlüsseln. Dies bringt einige Vorteile mit sich. So muss nur ein einziges Serverzertifikat für die HTTPS-Verschlüsselung aller dahinterliegenden Dienste eingesetzt werden, was die Ersteinrichtung und den Unterhalt vereinfacht. Da der Reverse Proxy im selben Docker-Netzwerk wie die restlichen Container betrieben wird, ist es nicht notwendig, den Verkehr hinter dem Proxy zu verschlüsseln. Aus diesem Grund kann bei der Entwicklung der anderen Container und Services, welche hinter dem Proxy betrieben werden, die Verschlüsselung vernachlässigt werden, weil diese bereits vorgängig vom Proxy implementiert wird.

5.3.3.2 SSO-Service für Authentifizierung über alle Services

Um die Anmeldedaten über alle Services verwenden zu können und bereits etablierte Sitzungen für weitere Dienste ohne Benutzerinteraktion aufrechterhalten zu können, wurde das Konzept eines SSO-Services verwendet. Dadurch wird es möglich, dass sich ein Benutzer nur einmalig anmelden muss, um im Anschluss die etablierte Vertrauensstellung für andere Dienste zu verwenden.

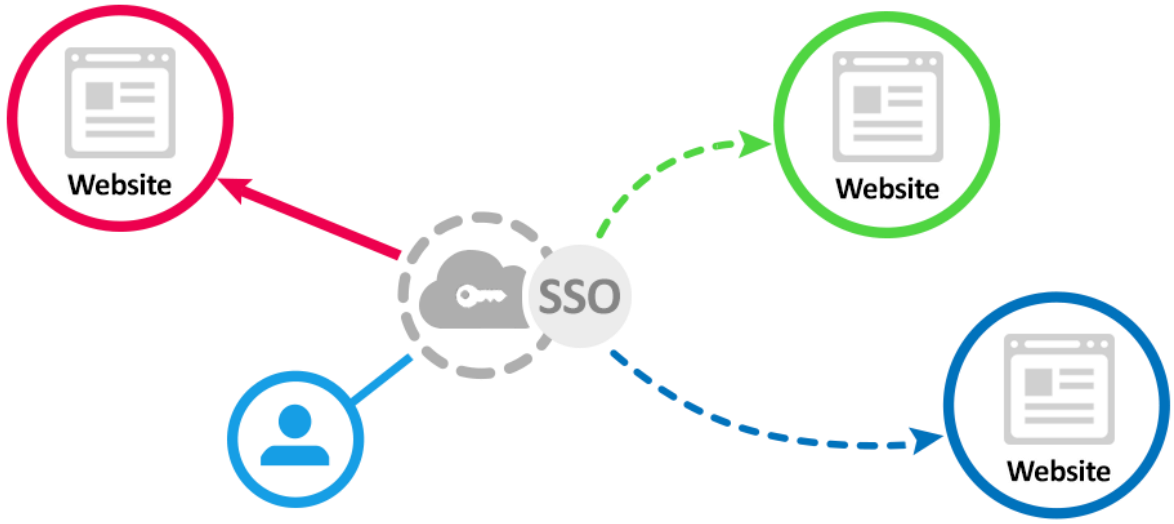


Abb. 10 Konzept eines SSO-Service [3]

5.3.3.3 Benutzerpersistierung in LDAP für dessen Wiederverwendung

Damit die Anmeldedaten der Hacking-Lab Benutzer auch für Dienste, welche das Protokoll eines SSO-Dienstes wie OIDC oder SAML nicht unterstützen, verwendet werden können, wurde auf die Vorteile der Persistierung der Benutzerdaten in einem LDAP-Server gesetzt. So wird sichergestellt, dass alle Anmeldedaten, welche von dem SSO-Service verwendet werden, in einem LDAP-Server persistiert sind. Weiter ergibt sich so der Vorteil, dass diese zentral gehalten und somit auch zentral verwaltet werden können. Da nun die Benutzerdaten in einem LDAP-Server gehalten werden, musste sichergestellt werden, dass auch der SSO-Service diese als Grundlage für die Authentifizierungs-Prozesse der Benutzer verwendet.

Da der LDAP-Server das zentrale System für die Speicherung der Anmeldedaten darstellt, können auch andere Komponenten, welche nicht direkt mit dem SSO-Service kommunizieren können, die Logins verwenden. So kann beispielsweise ein Linux Docker-Container eines CRS die Anmeldung eines Hacking-Lab Benutzers mit seinen persönlichen Anmeldedaten mittels Linux PAM unterstützen. Weitere Verwendungszwecke wären beispielsweise ein VPN. Insgesamt wird so die Bedienung des Hacking-Lab 2.0 vereinfacht.

5.3.3.4 Dezentrale/skalierbare Architektur durch Verwendung öffentlicher FQDNs

Durch die Verwendung von Docker-Containern und öffentlichen FQDNs für die Kommunikation unter den meisten Containern wird es einfach diese dezentral zu betreiben. Aufgrund der Verwendung einer Konfigurationsdatei über alle Container lassen sich die FQDNs und weitere Parameter für die Kommunikation zentral und übersichtlich konfigurieren. Wenn neue Nodes eines Systems eingesetzt werden sollen (z. B. zusätzlichen CAS-Client), kann dies auf einfache Weise bewerkstelligt werden. Dies indem das Git Repository, welches den Quellcode aller Services und die für den Betrieb notwendigen Docker-Container hält, geklont wird. Anschliessend müssen die gewünschten Parameter bezüglich Ports, Passwörter, Adressen etc. gesetzt und die jeweiligen Dienste gestartet werden.

5.3.3.5 Docker Secrets für die Passwortabsicherung

Damit Passwörter, welche in einem Container verwendet werden sollen, nicht über die Umgebungs-Variablen oder Docker Entrypoints übergeben werden müssen, was ein Sicherheitsrisiko darstellen würde, wurde auf die Vorteile von Docker Secrets gesetzt. Bei Docker Secrets werden Passwörter den Docker-Applikationen unter `/run/secrets/<secret_name>` als Datei zur Verfügung gestellt. Auf diese Dateien lassen sich optional Zugriffsberechtigungen setzten.

Docker Secrets können auf zwei verschiedene Arten verwendet werden [2], wobei sich diese innerhalb eines Containers nicht unterscheiden lassen. Auf dem Docker Host unterscheiden sich diese jedoch im Grundsatz.

- Zum einem lässt sich pro Passwort eine Datei mit dem Klartextpasswort als Inhalt auf dem Docker-Host anlegen, um dieses dann den jeweiligen Containern zur Verfügung zu stellen. Diese Variante ist aus Sicht der Sicherheit etwas fragwürdig, da die unverschlüsselten Passwörter auf dem Docker-Host persistiert sein müssen.
- Zum anderen wird mit dem Einsatz von Docker Secrets innerhalb eines Secret Stores eine wesentlich sicherere Variante angeboten. Dieser Store wird auf dem Docker-Host erstellt und hält die Passwörter verschlüsselt. Dadurch ist es nicht möglich, ein darin gespeichertes Passwort ohne Weiteres auf dem Host-System auszulesen.

5.3.3.5.1 Einsatz von Docker Swarm für Docker Secret Store

Mit dem Einsatz des Docker Secret Stores wird vorausgesetzt, dass Docker im Swarm Modus [3] betrieben wird. Da jedoch nicht sichergestellt werden kann, dass die später zu betreibenden Docker-Hosts im Swarm Mode operieren, soll es auf einfache Weise möglich sein, zwischen dem Betrieb im Swarm Mode oder dem Standalone Mode auszuwählen. Das Problem hierbei ist, dass sich die Compose-Dateien, in welchen die Servicekonfigurationen gehalten werden, in den beiden Modi stark voneinander unterscheiden. Aus diesem Grund wurde ein Konzept entwickelt, welches aus einer zwischen beiden Modi gültigen aber nicht vollständigen Compose-Datei und einer Modi spezifischen Datei für die Verlinkung des Docker Secrets per Datei oder Secret Store eine gültige Docker-Compose Datei zusammenstellen wird.

5.4 Umsetzung

Im folgenden Kapitel werden die konkreten Implementierungen und Umsetzungen der in der Problemstellung erklärten Anforderungen dargelegt. Zusätzlich wird die Umsetzung des Challenge Authoring Servers erläutert, für welchen keine konkrete Anforderung existierte.

5.4.1 Challenge Authoring System Server (REST-API)

Der Grundgedanke für das neue System war die «Lose Kopplung» der einzelnen Komponenten. Damit dies erreicht werden kann, wird ein zentrales API benötigt, auf das jede der Komponenten auf die gleiche Art zugreifen kann. Dafür bietet sich im Webbereich ein RESTful API an. Das RESTful API des Challenge Authoring System Servers wird durch das Play Framework [4] mit Java implementiert und nach aussen zur Verfügung gestellt. Genauere Informationen zum REST API können dem Kapitel 6.3 entnommen werden.

Der Challenge Authoring System Server wurde in seinen Grundzügen bereits im Prototyp der Studienarbeit verwendet und im Kontext dieser Bachelorarbeit weiterentwickelt und einem eingehenden Refactoring unterzogen. Zur Übersicht der implementierten Serverarchitektur, soll folgende schematische Darstellung dienen.

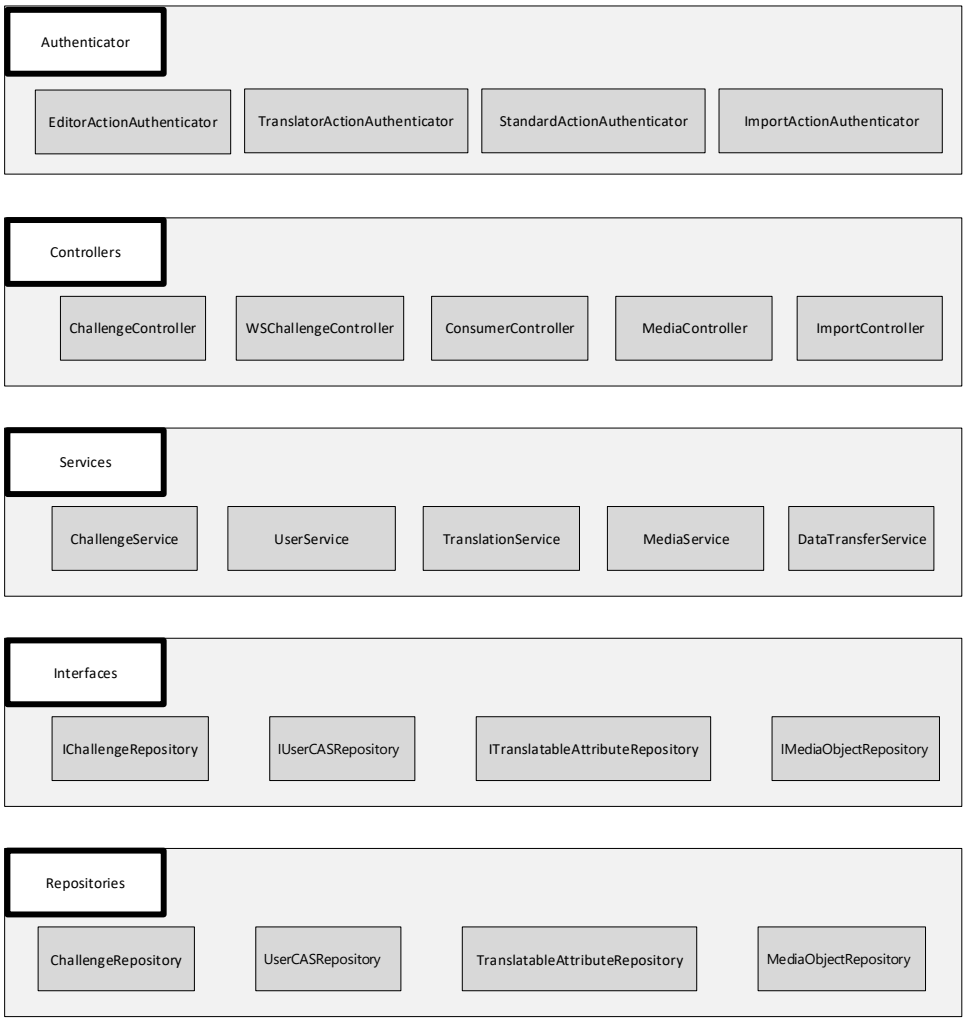


Abb. 11 Schematische Serverarchitektur

5.4.1.1 Authenticator

Zum ersten Layer und somit zur initialen Zugriffsschicht wurden die Authenticators implementiert. Sie definieren eine Autorisierungsschicht, welche anhand der Userrolle erkennt, ob die Verwendung der REST URI erlaubt ist oder nicht.

5.4.1.2 Controllers

Wenn ein Request vom Client die erste Zugriffschicht passiert hat, werden sie anhand der Routes-Definitionen des Servers an die einzelnen Controller weitergeleitet. Die implementierten Controller besitzen, bis auf einzelne Tests für das Vorhandensein von bestimmten Request-Headers, keine Logik. Sie delegieren die Aufrufe zu den Services, welche die eigentliche Serverlogik beinhalten.

5.4.1.3 Services

Ein durch den Controller weitergeleiteter Aufruf des Clients gelangt zu einem der fünf Services. Diese beinhalten die gesamte Business-Logik der Serverapplikation. Aus diesem Grund liegt das Hauptaugenmerk der implementierten Unit Tests auf diesem Layer des Servers.

Eine weitere Zuständigkeit der Services ist die Kapselung der Daten-Persistierung auf der Datenbank. Dazu verwenden sie verschiedene Repositories, die jeweils ein eigenes Interface implementieren, in welchem die CRUD Operationen pro verwendeter Model-Klasse definiert werden.

5.4.1.4 Interfaces

Die Repository-Interfaces wurden eingeführt, um die expliziten Zugriffe auf eine spezifische Datenbank zu abstrahieren. Dadurch wird die Möglichkeit gegeben, dass die Datenbank ausgetauscht werden kann und anstatt einer MySQL DB eine Oracle DB oder Ähnliches verwendet wird. Pro in der Datenbank persistierten Klasse wird ein eigenes Interface zur Verfügung gestellt, welches durch ein spezifisches Repository implementiert wird. Zusammen mit den Repositories bilden die Interfaces das DataAccess-Layer des Servers.

5.4.1.5 Repositories

Ein Repository ist für den konkreten Zugriff auf die Datenbank zuständig. Es implementiert die durch sein Interface definierten Methoden für die Erstellung, Modifikation und Löschung der Model-Klasse auf der Datenbank. Für ihre Operationen nutzen sie die Ebean-Server Klasse, welche die CRUD Methoden zur Verfügung stellt.

Damit bei zeitaufwendigen Datenbankoperationen der Ausführungskontext von Play nicht gestört wird, laufen diese Operationen in einem eigens dafür erstellten Datenbank-Ausführungskontext.

5.4.1.6 Abschluss

Das schematische Diagramm zur Serverarchitektur ist stark vereinfacht. Es werden nur die vier aussagekräftigsten Interfaces und Repositories dargestellt, damit die Übersichtlichkeit nicht darunter leidet. Weiter gibt es die eigentlichen Model-Klassen, die durch die Repositories in der Datenbank persistiert werden. Für genauere Informationen zur Architektur und Design des Servers sei hier auf das Kapitel 6.2 verwiesen, wo diese Punkte detaillierter aufgeschlüsselt werden.

5.4.2 Challenge-Editor Implementierung

In den folgenden Unterkapiteln werden die während der Bachelorarbeit konkret implementierten Komponenten, in Bezug auf den Challenge-Editor, beschrieben und die getroffenen Entscheidungen begründet. Es wird dafür von der Evaluation der einzelnen Komponenten zur konkreten Umsetzung vorgegangen.

5.4.2.1 Evaluation des Markdown Editors

Da bereits viel Arbeit in konkrete Implementierungen von Markdown-Editoren innerhalb einer Webapplikation eingeflossen sind, wurde mit dem Betreuer entschieden keine Eigenentwicklung voranzutreiben, sondern falls möglich einen bereits vorhandenen zu verwenden und diesen zu erweitern. Dies hat den Vorteil von allfälliger Weiterentwicklung, seitens des Entwicklers, profitieren zu können. Aus diesem Grund wurden neun bestehende JavaScript Editoren gesucht und anhand einer Nutzwertanalyse bewertet.

Für die Nutzwertanalyse wurden die folgenden Faktoren des Editors beurteilt:

- Leichtgewichtigkeit des Editors
- aktiv gewartet
- direktes Rendering von Markdown-Syntax Eingaben
- Live-Editierung
- Git-Integration
- Style

Nach der Beurteilung im Team wurde die Nutzwertanalyse mit dem Betreuer mit anderen Gewichtungen durchgeführt, aber dem gleichen Ergebnis. Im Folgenden werden die neun Editoren kurz vorgestellt und die für die Nutzwertanalyse entscheidenden Vor- und Nachteile beschrieben.

5.4.2.1.1 Medium Editor [5]

Dieser Editor wurde für den Prototyp der Studienarbeit verwendet. Er wurde dem Editor von Medium [6] nachempfunden.

Vorteil: dynamische Toolbar bei Markierung eines Textabschnitts ersichtlich, Drag & Drop von Bildern möglich, live-editierbar

Nachteil: Kein direktes Rendering von Markdown Syntax Eingaben unterstützt

5.4.2.1.2 Dropbox Paper [7]

Der Editor von Dropbox Paper wäre perfekt. Das Problem ist, dass kein Source Code oder Plug-In dafür vorhanden ist und deshalb für diese Bachelorarbeit nicht verwendet werden kann.

5.4.2.1.3 Demarcate [8]

Demarcate ist ein In-Place webbasierter Markdown-Editor.

Vorteil: -

Nachteil: Altmodisches Design, statisches Menü, kein direktes Rendering von Markdown Syntax Eingaben unterstützt, seit 4 Jahren keine Anpassung mehr, nicht live editierbar

5.4.2.1.4 Dillinger [9]

Dillinger ist ein mobilefähiger, offline-speicherbarer AngularJS HTML5 Markdown Editor.

Vorteil: Direktes Rendering von Markdown Syntax Eingaben unterstützt, Git Integration

Nachteil: Keine Toolbar (für wenig erfahrene Benutzer nicht bedienbar), nicht live editierbar

5.4.2.1.5 Stackedit.io [10]

Stackedit.io ist ein Echtzeit Markdown zu HTML Editor.

Vorteil: Tabellen und mathematische Ausdrücke unterstützt, offline Speicherung möglich, Google Drive / Dropbox / GitHub Integration, direktes Rendering von Markdown Syntax Eingaben unterstützt

Nachteil: Statisches Menü, zu überladen für den Gebrauch in diesem Bachelorarbeitskontext, nicht live editierbar

5.4.2.1.6 TOAST UI Editor [11]

TOAST UI Editor ist ein mächtiger Markdown Editor (GitHub Flavored Markdown).

Vorteil: UML / Charts / Tabellen Unterstützung

Nachteil: sehr langsam, schwierig erweiterbar, zu überladen für den Gebrauch in diesem Bachelorarbeitskontext, statisches Menü, nicht live editierbar

5.4.2.1.7 Woofmark [12]

Woofmark ist ein modular aufgebauter Markdown und HTML Editor.

Vorteil: -

Nachteil: Sehr komplizierter Code und dadurch schwierig erweiterbar, statisches Menü, nicht live editierbar

5.4.2.1.8 SimpleMDE [13]

Simple MDE ist ein simpel aufgebauter, leicht auf einer Webseite einbettbarer Markdown Editor.

Vorteil: Direktes Rendering von Markdown Syntax Eingaben unterstützt

Nachteil: Statisches Menü, nicht live editierbar

5.4.2.1.9 Pen [14]

Pen ist ein einfach aufgebauter, Editor mit Markdown-Erweiterung

Vorteil: Live editierbar, dynamische Toolbar bei Markierung eines Textabschnitts ersichtlich, leichtgewichtig, leicht verständlicher und damit leicht erweiterbarer Code, direktes Rendering von Markdown Syntax Eingaben unterstützt

Nachteil: seit 4 Jahren keine Anpassung mehr

Editor	Leicht-gewichtig	Aktiv gewartet	Markdown-Syntax-Unterstützung	Live editing	Git-Integration	Style	Wert
Gewicht	**20 %**	**15 %**	**20 %**	**30 %**	**5 %**	**10 %**	**100 %**
Medium Editor	5	5	1	6	1	5	4.3
Dropbox Paper	Wird nicht weiter angeschaut						
Demarcate	4	1	1	2	1	2	2
Dilinger	3	6	6	2	6	4	4
stackedit.io	3	6	6	2	6	4	4
TOAST UI Editor	2	6	6	2	1	3	3.45
woofmark	4	3	4	5	1	3	3.9
Simple MDE	4	3	6	3	1	4	3.8
Pen	6	2	6	6	1	6	5.15

Tab. 1 Nutzwertanalyse Markdown-Editoren

5.4.2.1.10 Auswertung

Aufgrund der durchgeführten Nutzwertanalyse wird der Pen-Editor für die Implementierung des Challenge-Editors verwendet.

5.4.2.2 Markdown-Editor Implementierung

Der Editor, der sich in der Nutzwertanalyse als bester herauskristallisiert hat, ist wie oben beschrieben der Pen-Editor. Er ist mit `npm install pen` sehr einfach in die bestehende Challenge-Editor Applikation integrierbar. Da eine der Anforderungen an den Challenge-Editor die Verwendung von React darstellt und der Pen-Editor lediglich als JavaScript-Modul vorhanden ist, wurde zu Beginn der Implementierung Pen in eine React Component adaptiert und eingehend getestet.

Als zusätzliche Anforderung des Betreuers an den Markdown-Editor wurde die Möglichkeit Bilder per Drag & Drop in den Inhalt einzufügen besprochen. Als Defizit des Pen-Editors, gegenüber dem bereits für den Prototyp der Studienarbeit verwendeten Medium Editors, fehlte diese Funktionalität. Durch den simplen Aufbau des Pen-Editors konnte diese in kurzer Zeit erweitert werden. Es wurden zwei zusätzliche Event-Listener hinzugefügt und Drag & Drop in separaten Methoden implementiert. Damit das eingefügte Bild korrekt in der Datenbank persistiert werden konnte, musste eine zusätzliche RESTful URI auf dem Server implementiert werden. Dieser kann das Bild als Base64 codierter String mit dem zu verwendenden Pfad übermitteln. Um das eingefügte Bild später wieder für die Bearbeitung einer Challenge korrekt im Editor anzeigen zu können, wurde die GET Funktionalität dazu zeitnah implementiert. Als Query String muss dazu der im Markdown referenzierte Pfad auf das Bild angegeben werden.

5.4.2.2.1 Editor-Toolbar

Einer der Vorteile des verwendeten Pen-Editors ist die Toolbar, die beim Markieren eines Textabschnitts eingeblendet wird. Dadurch kann der unerfahrene Benutzer seine Markdown-Inhalte entsprechend sauber stylen, ohne die Markdown-Syntax zu kennen und doch kann der gesamte Platz für den Text ausgenutzt werden.

Da die Toolbar des Pen-Editors neben den Standard-Markdown-Elementen, zusätzliche Einträge enthielt, die nicht in der Markdown-Syntax vorhanden sind, wurden diese entfernt und mit den noch fehlenden Elementen ergänzt.

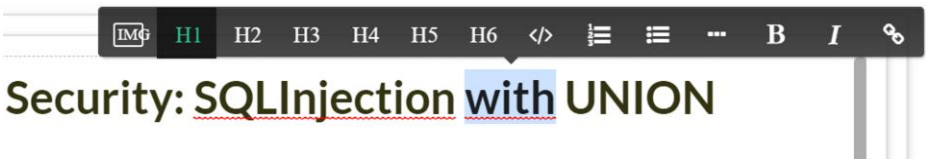


Abb. 12 Editor-Toolbar

5.4.2.2.2 Markdown-Syntax Unterstützung

Ein weiterer Vorteil des Pen-Editors ist das direkte Rendering von Markdown-Syntax Eingaben über die entsprechenden Text-Zeichen. Während der Analyse dieser Funktionalität sind einige Fehler in der Texterkennung gefunden und behoben worden. Weiter wurde sie um zusätzliche Elemente ergänzt. Die vom Pen Autoren zur Verfügung gestellte Markdown.js Datei, welche diese Texterkennung beinhaltet, wurde nach der Erweiterung des Codes in eine eigene React-Komponente ausgelagert und im Editor eingebunden.

Die Texterkennung funktioniert dadurch, dass einzelne Spezialzeichen, die für die Markdown-Syntax verwendet werden, registriert und in einem Stack abgelegt werden, wenn sie nacheinander auftreten. Wenn ein Leerschlag eingefügt wird, wird anhand verschiedener regulären Ausdrücken versucht die Sequenz einem Markdown-Element zuzuordnen. Falls dies gelingt, wird der dem Markdown-Element entsprechende HTML-Tag in den Editor eingefügt. Mit zwei Enter Klicks kann ein so eingefügter HTML-Tag wieder verlassen und der normale Textfluss wiederaufgenommen werden.

5.4.2.2.3 Evaluation Converter für die Persistierung von Challenges

Damit der HTML-Content aus dem Editor wieder in die eigentliche Markdown-Syntax konvertiert werden kann, muss ein intelligenter Converter gesucht oder selber geschrieben werden, der diese Aufgabe übernimmt. Nach ersten Versuchen der Eigenentwicklung wurden vier bestehende Converter aus dem Internet analysiert und anhand einer Nutzwertanalyse bewertet. Die fünf Bewertungskriterien sind die folgenden:

- Leichtgewichtigkeit
- Aktiv gewartet
- Funktioniert mit Pen-Editor
- Erweiterbar
- Unterstützung von «GitHub Flavored Markdown»

Die vier untersuchten Converter sind:

5.4.2.2.3.1 turndown [15]

Ein JavaScript Converter für HTML zu Markdown.

Vorteil: Vor 5 Monaten zuletzt angepasst, einfache Verwendung, erweiterbar, funktioniert gut mit Pen-Editor zusammen

Nachteil: -

5.4.2.2.3.2 upndown [16]

Ein JavaScript HTML zu Markdown Converter.

Vorteil: -

Nachteil: Vor einem Jahr zuletzt angepasst, nicht erweiterbar, funktioniert nicht mit Pen-Editor zusammen

5.4.2.2.3.3 to-markdown [17]

Ein HTML zu Markdown Converter, der mittels JavaScript implementiert wurde. Dieser ist deprecated und wird als turndown weitergeführt.

5.4.2.2.3.4 html-to-markdown [18]

Ein HTML zu Markdown Converter ohne externe Abhängigkeiten.

Vorteil: -

Nachteil: Vor drei Jahren zuletzt bearbeitet, fehlende Unterstützung für Images, Links, DIVs, was ihn nicht mit dem Pen-Editor kompatibel macht

HTML to Markdown	Leichtgewichtig	Aktiv gewartet	Funktioniert mit Pen Editor	Erweiterbar	GitHub Flavored Markdown	Wert
Gewicht	15 %	20 %	40 %	5 %	20 %	**100 %**
turndown	5	5	5	5	6	5.2
upndown	5	4	2	2	3	3.05
html-to-markdown	6	2	2	2	3	2.8

Tab. 2 Nutzwertanalyse HTML zu Markdown Converter

5.4.2.2.3.5 Auswertung

Turndown geht als klarer Sieger aus dieser Nutzwertanalyse hervor und wird aus diesem Grund als HTML zu Markdown Converter eingesetzt.

5.4.2.2.4 Evaluation Converter für die Bearbeitung von Challenges

Damit der in der Datenbank gespeicherte Markdown-Inhalt wieder korrekt in den Editor eingefügt werden kann, ist es notwendig, dass er in die für den Editor benötigte HTML-Struktur konvertiert wird. Dazu wurden verschiedene Converter untersucht und anhand einer Nutzwertanalyse bewertet. Die fünf Bewertungskriterien sind die folgenden:

- Leichtgewichtigkeit
- Aktiv gewartet
- Funktioniert mit Pen-Editor
- Erweiterbar
- Unterstützung von «GitHub Flavored Markdown»

Die folgenden Converter wurden analysiert:

5.4.2.2.4.1 markdown-it [19]

markdown-it ist ein Markdown Parser, der mit JavaScript implementiert wurde.

Vorteil: Vor drei Monaten zuletzt angepasst, erweiterbar

Nachteil: Ist nicht ohne weitere Anpassung mit Pen-Editor kompatibel

5.4.2.2.4.2 markdown-js [20]

Markdown-js ist ein in JavaScript geschriebener HTML zu Markdown Parser.

Vorteil: Unterstützt verschiedene Markdown-Dialekte

Nachteil: Vor vier Jahren zuletzt angepasst, nicht mit Pen-Editor kompatibel, kein «GitHub Flavored Markdown»

5.4.2.2.4.3 marked [21]

marked ist ein Markdown-Parser, der für einen optimalen Speed erstellt wurde.

Vorteil: Vor 30 Tagen zuletzt angepasst, sehr einfach gehalten, gut mit Pen kompatibel, «GitHub Flavored Markdown» Unterstützung

Nachteil: -

5.4.2.2.4.4 showdown [22]

Showdown ist ein in JavaScript geschriebener HTML zu Markdown Converter.

Vorteil: «GitHub Flavored Markdown» Unterstützung, vor 5 Monaten zuletzt bearbeitet

Nachteil: kompliziert aufgebaut, nicht mit Pen-Editor kompatibel

Markdown to HTML	Leichtgewichtig	Aktiv gewartet	Funktioniert mit Pen Editor	Erweiterbar	GitHub Flavored Markdown	Wert
Gewicht	15 %	20 %	40 %	5 %	20 %	**100 %**
Markdown-it	5	5	3	5	3	3.8
Markdown-js	5	1	2	5	3	2.6
marked	5	5	5	2	6	5.05
Showdown	2	4	2	3	6	3.25

Tab. 3 Nutzwertanalyse Markdown zu HTML Converter

5.4.2.2.4.5 Auswertung

marked geht als klarer Sieger aus dieser Nutzwertanalyse hervor und wird aus diesem Grund als Markdown zu HTML Converter eingesetzt.

5.4.2.2.5 Umsetzung der kontinuierlichen Speicherung von Challenges

Die kontinuierliche Speicherung der Challenge-Editor Inhalten wird wie bereits erwähnt einerseits mit Websocket-Verbindungen und andererseits, für die Metadaten, über POST Requests erledigt. Der Pen-Editor wird dazu bei der Initialisierung mit einer Websocket-Verbindung erweitert. Diese beinhaltet die für die Speicherung relevanten IDs der Markdown-Inhalte. Die IDs nehmen je nach Typ des Markdowns unterschiedliche Werte an. Für die Websocket-Verbindung eines Abstracts wird die ID der Challenge, sowie die neue ID des Abstracts übermittelt, wohingegen für die Hints und Instructions neben ihrer eigenen ID, die Referenz auf ihre Section benötigt wird. Auf dem Server wurde ein eigener Websocket-Controller eingerichtet, der für das Handling und die Initialisierung dieses Verbindungstyps verantwortlich ist.

Die Metadaten können über POST Requests behandelt werden, welche jeweils übermittelt werden, wenn die Eingabe über den Submit-Button bestätigt wird. Auf dem Server wird eine REST URI zur Verfügung gestellt, in der alle Metadaten POST Requests verarbeitet werden. Als Antwort des Servers wird ein JSON mit dem aktualisierten Wert zurückgegeben und auf dem Client für eine zukünftige Bearbeitung anhand des Reducers gespeichert.

5.4.2.3 Aufbau des Challenge-Editors

Der Challenge-Editor ist wie im folgenden Screenshot aufgebaut. Im linken Bereich wird eine Navigation angezeigt, durch welche sich der Benutzer im Editor zurechtfindet. Auf der rechten Seite befindet sich jeweils das neu zu erstellende oder zu bearbeitende Element. Im Bild unten wird die Mutation eines Abstracts anhand des Markdown-Editors verbildlicht. Die weiteren Elemente werden im Kapitel 6.2.5 aufgelistet.

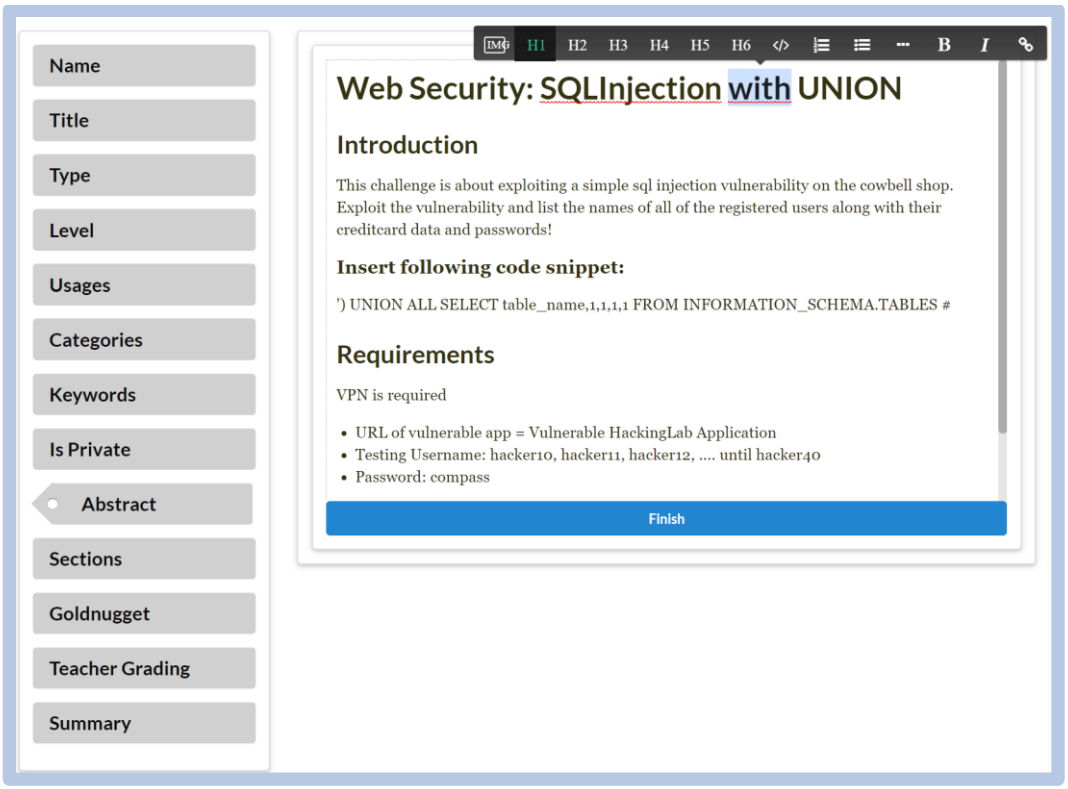


Abb. 13: Markdown-Editor

Nach der Erstellung und der Mutation einer Challenge kann das Resultat auf dem Summary-Navigationspunkt betrachtet werden. Zu Beginn werden die einzelnen verwendeten Challenge-Metadaten aufgelistet und darauffolgend die verschiedenen definierten Markdown-Inhalte.

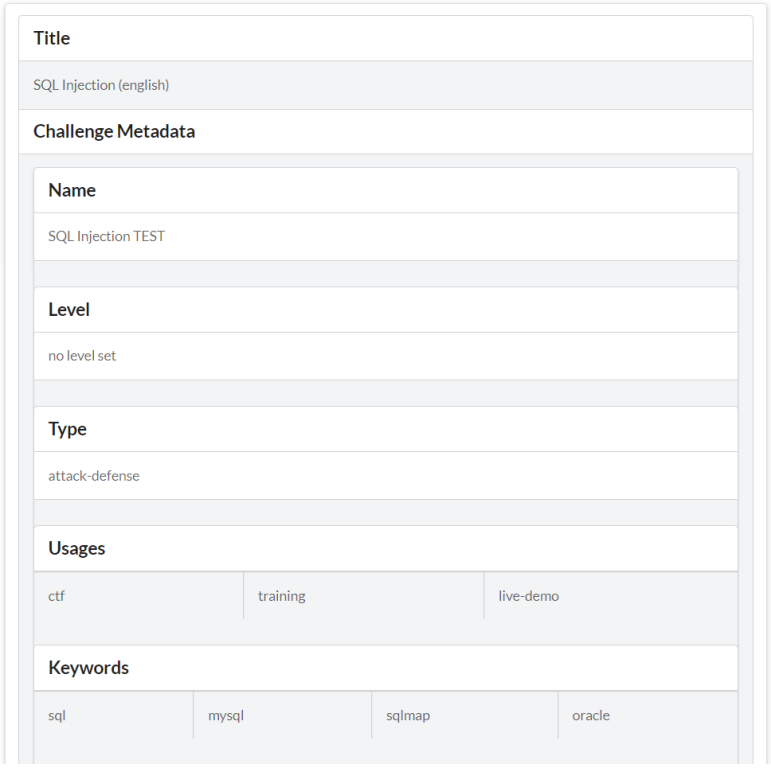


Abb. 14: Challenge-Summary

5.4.2.4 Umsetzung des automatischen Übersetzungs-API

Für die Implementierung des automatischen Übersetzungs-APIs wurde entschieden Deepl [23] zu verwenden, da dieses, nach unterschiedlichen Meinungen, die besten Übersetzungen zur Verfügung stellt und der Preis im normalen Rahmen liegt. Zusätzlich ist die Verwendung des APIs sehr intuitiv und kann ohne grossen Aufwand umgesetzt werden.

Für die Implementierung wurde auf dem Client ein zusätzlicher XMLHttpRequest aufgesetzt, dem der API-Key, ein zu übersetzender Text, sowie die Zielsprache im Body übergeben werden kann. Als Antwort wird von Deepl ein JSON-Objekt geliefert, das den übersetzten Text beinhaltet.

Als Schwierigkeit bei der Umsetzung sind Markdown-Abschnitte wie Code-Blöcke zu sehen, die nicht übersetzt werden sollen, da diese dadurch nicht mehr korrekt ausgeführt werden könnten. Aus diesem Grund wurde eine zusätzliche Methode implementiert, welche die nicht zu übersetzenden Blöcke durch einen Platzhalter ersetzen und nach der Übersetzung wieder mit dem initialen Text befüllt.

5.4.2.5 Umsetzung des Übersetzungseditors

Damit die durch das automatische API übersetzten Texte von Hand bearbeitet werden können, wurde eine zusätzliche Editor-Komponente implementiert. Im linken Bereich wird der initiale englische Text dargestellt und im rechten die Übersetzung desselben, falls sie vorhanden ist. Im unteren Ausschnitt wird diese Ansicht dargestellt.

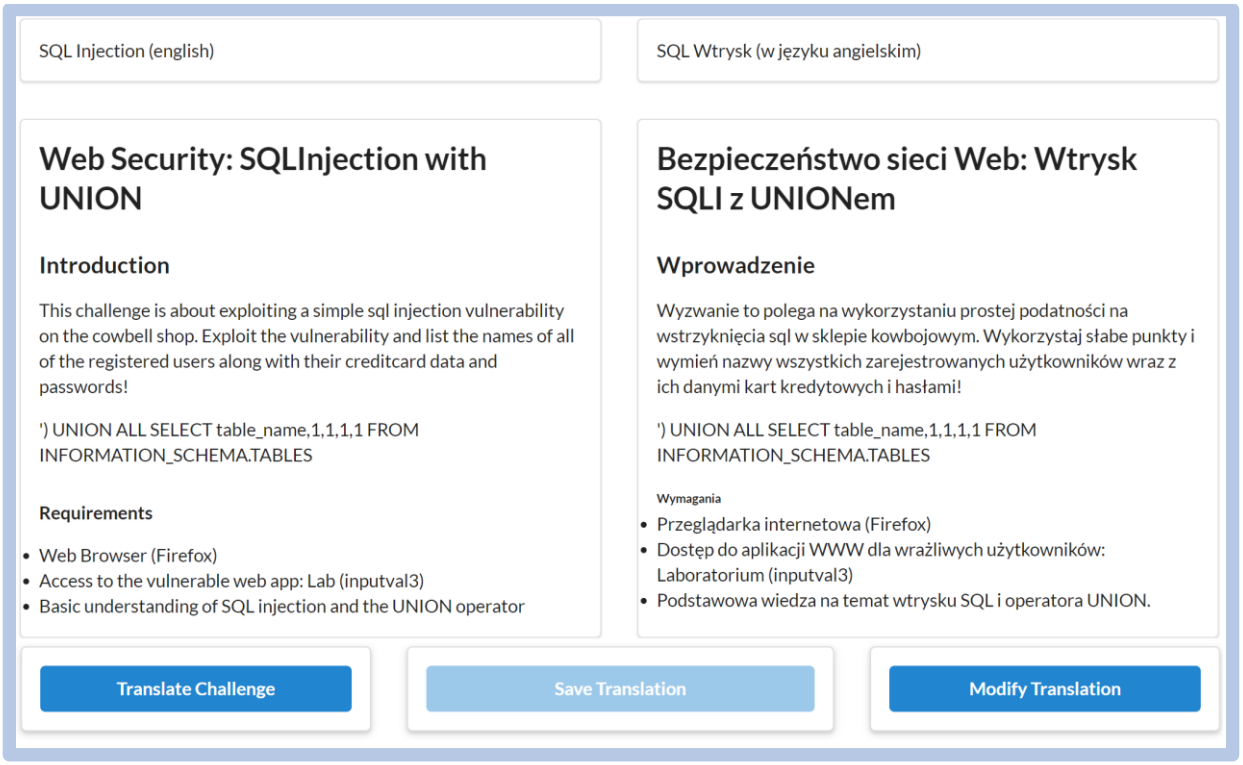


Abb. 15: Übersetzungseditor im Lese-Modus

Durch einen Klick auf «Modify Translation» wird die rechte Ansicht modifizierbar und der angepasste Pen-Editor kann für die Bearbeitung der Übersetzung verwendet werden. Dies soll folgende Illustration verbildlichen.

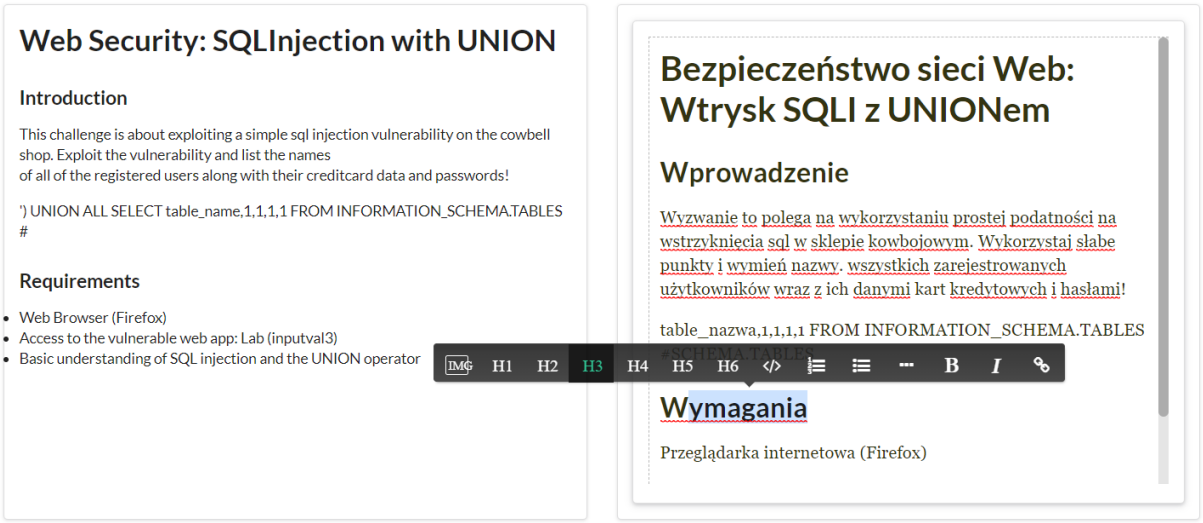


Abb. 16: Übersetzungseditor im Edit-Mode

5.4.3 Challenge Synchronisation

Für die Umsetzung der Challenge-Synchronisation ist die Definition der für den Import verwendeten Filestruktur sowie die Spezifikation des JSON-Objektes erforderlich. Diese werden im folgenden Abschnitt beschrieben werden. Als weiterer Punkt wird die Umsetzung der Encodings für das JSON-Objekt, sowie für die Unterstützung von UTF-8 beschrieben.

5.4.3.1 Spezifikation challenge.json Datei

Damit die Metadaten einer Challenge, welche jeweils nur in Englisch existieren, korrekt in die Datenbank importiert und die einzelnen Markdown-Dateien korrekt einer Challenge zugeordnet werden können, wird eine Datei challenge.json spezifiziert, welche den Container für diese Merkmale bildet. In Absprache mit dem Betreuer und dem Auftraggeber wurde diese Datei folgendermassen definiert.

```
{
  "id": "Challenge-UUID",
  "name": "Challenge-Name",
  "level": 1,
  "type": "Challenge-Typ",
  "usages": ["Array von Challenge-Usage-Strings"],
  "keywords": ["Array von Schlüsselwort-Strings"],
  "categories": ["Array von Challenge-Kategorie-Strings"],
  "created": "Erstellungsdatum im Format dd. MMMM yyyy und Local.US",
  "author": "Autor",
  "last update": "Letztes Modifizierungsdatum im Format dd. MMMM yyyy und Local.US",
  "last git commit": "Git Commit Referenz-ID",
  "resources": [{
    "id": "Ressourcen-UUID",
    "type": "Ressourcen-Typ"
  }],
  "goldnugget-type": "Goldnugget-Typ",
  "goldnugget-secret-static": "Statisches Goldnugget",
  "public": true,
  "sections": [{
    "id": "Section-UUID",
    "order": 0,
    "steps": [{
      "id": "Step-UUID",
      "order": 1,
      "type": "Step-Typ"
    }, {
      "id": "Step-UUID",
      "order": 1,
      "type": "Step-Typ"
    }]
  }]
}
```

Abb. 17 challenge.json Spezifikation

5.4.3.2 Spezifikation Dateistruktur

Durch eine Absprache mit dem Betreuer und dem Auftraggeber wurde die Dateistruktur für eine zu importierende Challenge festgelegt. Der Ordner, der alle Challenges beinhaltet, kann dabei in der Konfigurationsdatei der Java-Applikation festgelegt werden. Darin wird pro Challenge eine eigene Ordnerstruktur erstellt, dessen Name die UUID der Challenge verlangt. Pro Challenge muss ein challenge.json vorhanden sein, welches die Referenzen der Markdown-Dateien, sowie zusätzliche Metadaten beinhaltet. Weiter muss pro Challenge für jede verfügbare Sprache ein zusätzlicher Ordner angelegt werden. Diese beinhalten die tatsächlichen Markdown-Dateien (abstract.md, solution.md) und Merkmale, die pro Sprache unterschiedlich ausfallen können (title.txt, title.png). Innerhalb eines Sprach-Ordners, werden die zusätzlichen Strukturen «media» und «sections» ergänzt. Der media-Ordner wird in die verfügbaren Dateitypen (png, jpg usw.) unterteilt, in dem dann die expliziten Dateien abgelegt werden. Im sections-Ordner werden die Markdown-Dateien für die Section, sowie ein Ordner steps für die einzelnen Hint und Instruction Markdown-Dateien abgelegt. Die Dateinamen der Sections und Steps müssen so gewählt werden, dass das jeweilige Suffix (section-, step-) durch ihre tatsächliche UUID ergänzt wird, damit sie auf dem Server anhand der challenge.json-Datei korrekt zugeordnet werden können.

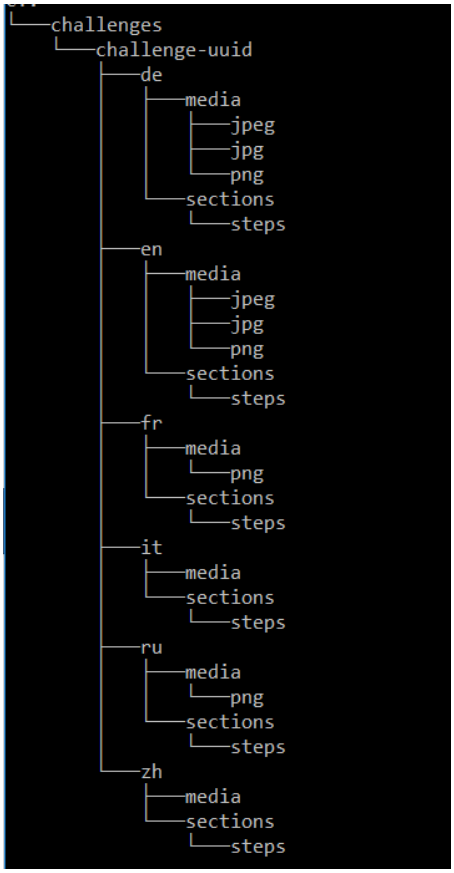


Abb. 18: Spezifikation Filestruktur

Die hier spezifizierte Dateistruktur wird anhand der Java-Applikation eingelesen und das JSON-Objekt für die Übermittlung generiert.

Die gleiche für den Import eingelesene Datei-Struktur wird bei einem Export der Challenge wiederaufgebaut und alle bereits vorhandenen Dateien überschrieben.

5.4.3.3 Spezifikation JSON-Objekt

Für die Übermittlung einer Challenge zum Server wurde entschieden der gesamte Prozess anhand eines JSON-Objekts abzuwickeln, damit nur ein POST Request erstellt werden muss. Die Spezifikation für dieses Objekt wurde mit dem Auftraggeber diskutiert und auf die folgende Struktur festgelegt.

```
{
  "challengeJson":{
    "Challenge-JSON Struktur"
  },
  "languageComponents":[ {
    "isoCode" : "Language ISO-Code",
    "titleMD" : "Übersetzter Titel",
    "titlePNG" : "Base64 codierte Titelbild-Datei",
    "solutionMD" : "Übersetzter Markdown-Solution Dateiinhalt",
    "abstractMD" : "Übersetzter Markdown-Abstract Dateiinhalt",
    "sections" : [ {
      "id" : "Section-UUID",
      "sectionMD" : "Übersetzter Markdown-Section Dateiinhalt",
      "steps" : [ {
        "id" : "Step-UUID",
        "stepMD" : "Übersetzter Markdown-Step Dateiinhalt"
      }, {
        "id" : "Step-UUID",
        "stepMD" : "Übersetzter Markdown-Step Dateiinhalt"
      } ]
    }],
    "medias" : {
      "png" : [ {
        "filePath" : "Datei-Pfad",
        "base64Encoded" : "Base64 codierte Bild-Datei"
      } ],
      "jpg" : null,
      "jpeg" : null,
      "gif" : null,
      "svg" : null,
      "mpeg" : null,
      "wav" : null,
      "mp3" : null,
      "mp4" : null
    }
  }]
}
```

Abb. 19: Spezifikation JSON-Objekt

Im ersten JSON-Attribut «challengeJson» wird die bereits spezifizierte challenge.json-Struktur verwendet. Im zweiten Attribut «languageComponents» wird ein JSON-Array der eigentlichen Sprachdefinitionen einer Challenge erwartet. Pro in der Dateistruktur definierten Sprache wird in diesem Array ein Eintrag erstellt und nach der Übermittlung auf der Datenbank persistiert.

5.4.3.4 Encoding für die Erhaltung des Hash-Wertes der Dateien

Damit der Hash-Wert jeder Datei vor, nach und während dem Import- /Export-Prozess immer derselbe bleibt, wurde für die Implementierung der JSON-Generierung auf beiden Seiten dieselbe Library verwendet (Jackson [24]). Mit Jackson ist es möglich anhand mit speziellen Annotationen versehenen Klassen ein JSON-Objekt zu generieren. Da auf beiden Seiten der Übermittlung eine Java-Applikation implementiert wurde, konnten diese Data-Transfer-Objekt Klassen für beide Applikationen verwendet werden. Da Jackson ein internes Encoding implementiert, musste dieses in diesem Zusammenhang nicht weiter beachtet werden.

5.4.3.4.1 Probleme bei der Implementierung

Das Problem bei der Übermittlung einer Challenge waren nicht die einzelnen Markdown-Dateien, sondern die als Base64 codierten Bilder, welche zusätzlich dem JSON-Objekt angehängt wurden. Diese konnten ohne Probleme auf beiden Seiten übertragen, jedoch nicht mit dem genau gleichen Hash-Wert wieder in der Dateistruktur erstellt werden. Der Grund für das Problem war die für das Decodieren des Base64-String in ein Byte-Array verwendete Library von Sun. Nach dem Wechsel auf die java.util.Base64 Klasse konnte die Bildgenerierung ohne weitere Probleme durchgeführt und die Datei mit dem gleichen Hash-Wert erstellt werden.

5.4.3.5 UTF-8 für die Unterstützung nicht europäischer Zeichensätze

Der Einsatz von UTF-8 Encodings, muss in der Java-Applikation für den Import / Export korrekt implementiert werden. Da die erstellte FileHelper Klasse die beiden einzigen Methoden für das Lesen und Schreiben von Dateien zur Verfügung stellt, musste nur an dieser Stelle das Encoding umgesetzt werden. Dazu kann in dem verwendeten FileOutputStream, sowie dem FileInputStream UTF-8 als Standard angegeben werden.

Auf dem Server wird das Encoding anhand des Connection-Strings zur Datenbank bereits korrekt umgesetzt.

5.4.4 Infrastruktur

5.4.4.1 Übersicht

Mit dem auf der nächsten Seite folgendem Deployment-Diagramm wird eine Übersicht über alle eingesetzten Docker-Container und deren Kommunikation untereinander vermittelt. Dieses soll dem Leser als Grundlage für dieses Kapitel dienen und ihn dabei unterstützen, die komplexe Infrastruktur schneller zu verstehen. Die verwendeten Farben haben dabei folgende Bedeutung:

- **Rot** Service welcher vom WWW erreichbar ist, sowie die Verbindungen vom WWW
- **Blau** es handelt sich um einen Docker-Container
- **Hellblau** Consumer Applikation
- **Grün** CAS-Client Applikation
- **Violett** CAS-Server Applikation
- **Orange** MySQL und Speicherort der Datenbank

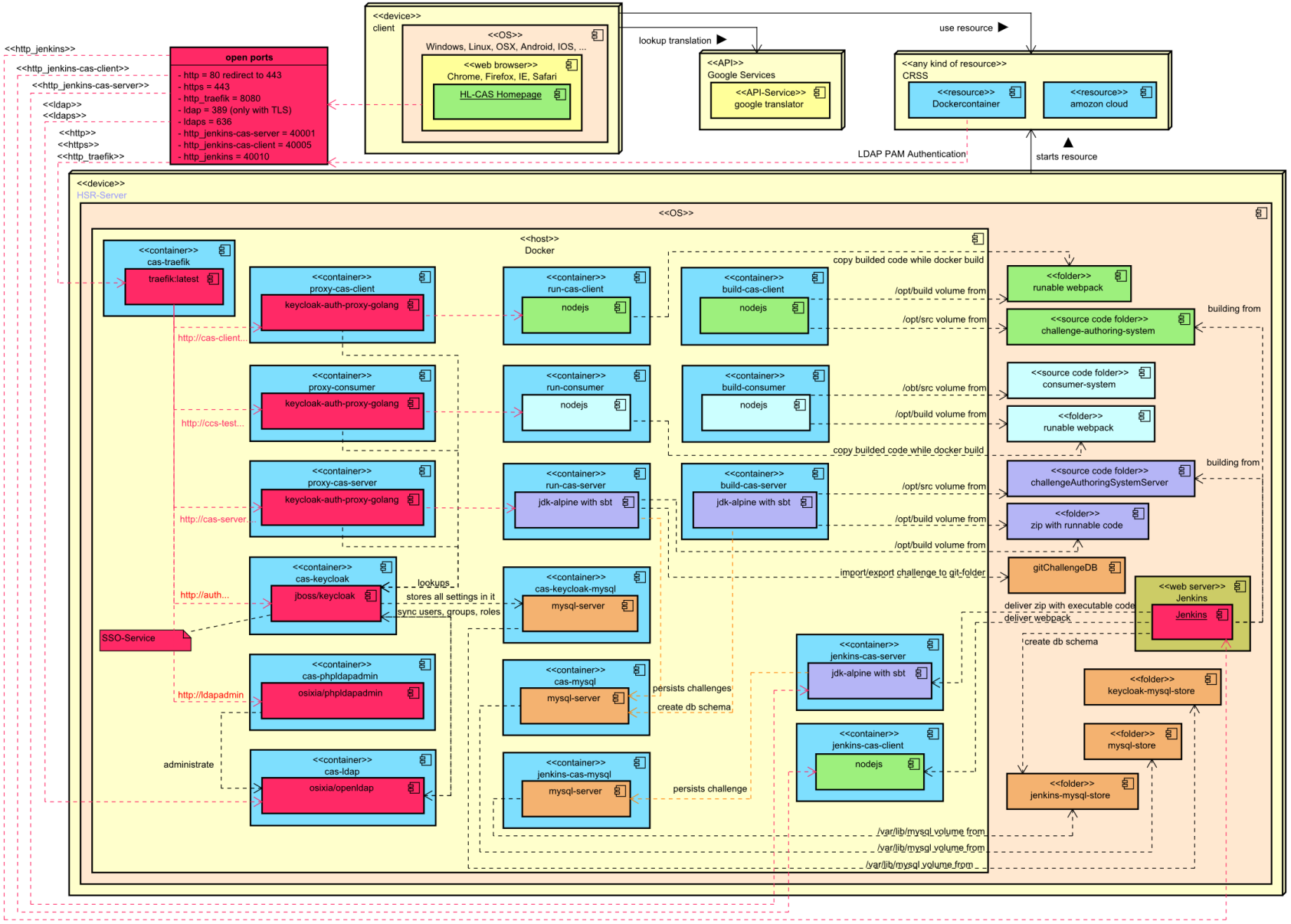


Abb. 20 Deployment Diagramm

5.4.4.2 Admin Tool

Um den Betreiber eines Challenge Authoring Systems bei den administrativen Aufgaben der Inbetriebnahme, Fehlersuche usw. zu unterstützen, wurde ein kleines Bash-Skript namens «admin_tool.sh» entwickelt. Darin sind alle notwendigen Operationen, wie auch Konfigurationsdateien, welche ein Betreiber für die Installation und den Betrieb benötigt enthalten. Dieses kleine Tool ist wie folgt aufgebaut:

```
root@sinv-56043:/var/lib/jenkins/workspace/ChallengeAuthoringSystem/docker# ./admin_tool.sh
 1) init_swarm
 2) create_docker_network
 3) create_docker_secrets
 4) createLDAPCerts.sh
 5) build_CAS
 6) build_CAS_hard
 7) run_CAS
 8) run_CAS_hard
 9) removeStacks
10) removeContainers
11) show_logs_build_CAS
12) show_logs_run_CAS
13) show_logs_cas-traefik
14) edit_docker-compose.common-services_template.yml
15) edit_docker-compose.build_template.yml
16) edit_docker-compose.run_template.yml
17) edit_.env
18) edit_traefik.toml
19) edit_cas-mysql_createCASUser.sql
20) edit_keycloak-mysql_createKeycloakUser.sql
21) edit_create_docker_secrets.sh
22) edit_ldap/image/environment/default.startup.yaml
23) edit_keycloak-proxy/configs/proxy-cas-server-cors-config.yml
24) edit_keycloak-proxy/configs/proxy-cas-client-cors-config.yml
25) Quit
Please enter your choice:
```

Abb. 21 admin_tool.sh Optionen

1. Setzt den Docker-Host in den Swarm Modus
2. Erstellt das CAS-Netzwerk, in welchem alle Container laufen werden
3. Wenn in der «.env»-Datei ENABLE_DOCKER_SWARM auf true steht, kann ein Docker Secret Store verwendet werden. Somit werden alle Passwörter, welche in den Docker Containern verwendet werden in einen solchen Store geschrieben. Ist die Einstellung in der «.env»-Datei auf false, werden die Passwörter in Secret Dateien geschrieben.
4. Führt ein Skript aus, um ein selbstsigniertes Zertifikat für LDAP zu erstellen
5. Buildet und startet die Container build-cas-server, build-cas-client, build-consumer und cas-mysql.
6. Gleich wie Option 5 jedoch werden dem Docker Build-Befehl die Argumente «`--force-rm –no-cache –pull`» angehängt, um sicherzustellen, dass keine zwischengespeicherten Daten verwendet werden.
7. Buildet und startet die Container run-cas-server, run-cas-client, run-consumer, cas-traefik, proxy-cas-server, proxy-cas-client, proxy-consumer, cas-keycloak, cas-phpldapadmin, cas-ldap, cas-keycloak-mysql und cas-mysql.
8. Gleich wie Option 7 jedoch werden dem Docker Build-Befehl die Argumente «`--force-rm –no-cache –pull`» angehängt, um sicherzustellen, dass keine zwischengespeicherten Daten verwendet werden.
9. Entfernt alle Stacks im Swarm Mode
10. Entfernt alle Container im Standalone Mode
11. Zeigt die Logs der Container aus Option 5 / 6 (nur im Standalone Mode)

12. Zeigt die Logs der Container aus Option 7 / 8 (nur im Standalone Mode)
13. Zeigt die Log-Datei von cas-traefik an
14. Öffnet die Docker-Compose-Datei vom cas-mysql in vi
15. Öffnet die Docker-Compose-Datei von den Containern der Option 5 zur Bearbeitung
16. Öffnet die Docker-Compose-Datei von den Containern der Option 7 zur Bearbeitung
17. Öffnet die zentrale Konfigurationsdatei zur Bearbeitung, in welcher alle Einstellungen wie öffentliche Adresse (FQDN), Ports, Pfade für Datenbanken, Zeitzone, Einstellung ob Docker Swarm Mode aktiviert ist etc. gemacht werden können.
18. Öffnet die Konfigurationsdatei für Traefik zur Bearbeitung
19. Öffnet die Datei zur Festlegung des cas-mysql Datenbankbenutzers und dessen Passwort zur Bearbeitung
20. Öffnet die Datei zur Festlegung des keycloak-mysql Datenbankbenutzers und dessen Passwort zur Bearbeitung
21. Öffnet die Passwortdatei, in der die Passwörter der Container stehen, zur Bearbeitung
22. Öffnet die Konfigurationsdatei für die Festlegung der LDAP Initialdaten zur Bearbeitung
23. Verlässt das Admin Tool

Wenn das Tool wie oben beschrieben verwendet wird, werden immer gleich alle Container der jeweiligen Gruppe behandelt. Wenn jedoch nur ein Container angesprochen werden soll, geht dies beispielsweise wie folgt:

```
./admin_tool.sh -rh cas-keycloak
```

Wenn ungültige Parameter übergeben wurden, erscheint folgende Meldung.

```
Usage: admin_tool.sh [option] [OPTIONAL_DOCKER_TO_BUILD (default: all) docker-service]
 option:  -b        to build containers and build CAS in docker cas_build_stack
          -bh       to build containers and build CAS in docker cas_build_stack (build --force-rm --no-cache --pull)
          -r        to build containers and run CAS in docker cas_run_stack
          -rh       to build containers and run CAS in docker cas_run_stack (build --force-rm --no-cache --pull)
          -lb       to docker service logs from cas_build_stack
          -lr       to docker service logs from cas_run_stack
```

Abb. 22 admin_tool.sh Bash Benutzung

Diese Variante eignet sich auch bestens, um die Services über ereignisgesteuerte Automatismen neu zu bauen. So wäre es denkbar, dass alle verteilten Systeme auf verschiedenen Hosts automatisch neu gebaut werden, sobald es ein Update im Git-Repository gibt.

5.4.4.3 Globale Konfigurationsdatei

Für eine einheitliche und übersichtliche Konfiguration der einzelnen Docker-Container wurde eine für die ganze Umgebung gültige Konfigurationsdatei namens «.env» erstellt, dessen Werte sowohl im Admin Tool, als auch in den Docker-Compose Dateien verwendet werden.

5.4.4.4 Docker Compose Dateien [25]

5.4.4.4.1 Template Dateien

Grundsätzlich wurden die Container auf drei Compose-Dateien verteilt. Zudem gibt es jeweils zwei weitere Compose-Dateien, welche die Docker Secrets auf unterschiedliche Weise verknüpfen. Daraus resultiert folgende Struktur:

- docker-compose.common-services_template_secretFile.yml
- docker-compose.common-services_template_secretStore.yml
- docker-compose.common-services_template.yml
  - cas-mysql
- docker-compose.build_template_secretFile.yml
- docker-compose.build_template_secretStore.yml
- docker-compose.build_template.yml
  - build-cas-server
  - build-cas-client
  - build-consumer
- docker-compose.run_template_secretFile.yml
- docker-compose.run_template_secretStore.yml
- docker-compose.run_template.yml
  - run-cas-server
  - run-cas-client
  - run-consumer
  - cas-traefik
  - proxy-cas-server
  - proxy-cas-client
  - proxy-consumer
  - cas-keycloak
  - cas-keycloak-mysql
  - cas-mysql
  - cas-ldap
  - cas-phpldapadmin

5.4.4.4.1.1 docker-compose.common-services_template.yml

Die Datei docker-compose.common-services_template.yml beinhaltet den MySQL Container, welcher vom CAS-Server verwendet wird. Da die MySQL-Datenbank beim Builden des CAS-Server erstellt und später im Betrieb vom CAS-Server verwendet wird, muss dieser Container sowohl in der Build-Umgebung, als auch in der Run-Umgebung verfügbar sein. Aus diesem Grund wurde der cas-mysql in einer separierten YML-Datei deklariert. Somit ist dessen Konfiguration nicht in zwei Dateien zu tätigen.

5.4.4.4.1.2 docker-compose.build_template.yml

In dieser Datei sind jene Container konfiguriert, welche dafür zuständig sind, die Quellcodes für den CAS-Server, den CAS-Client und den Consumer jeweils zu einer korrekten Konfigurationsdatei zusammenzufügen. Nach Fertigstellung eines Buildvorgangs wird der jeweilige Container beendet, um die Ressourcennutzung auf dem Docker-Host klein zu halten.

5.4.4.4.1.3 docker-compose.run_template.yml

In der «run_template» Datei werden alle Container, bis auf den cas-mysql, welche für den laufenden Betrieb benötigt werden deklariert. Somit kann die gesamte Laufzeitumgebung mit wenigen Kommandos in Betrieb genommen werden.

5.4.4.4.2 Stack Dateien

Die Template Dateien enthalten kaum statische Werte. Die meisten Werte wie beispielsweise der Name des zu verwendenden Docker Netzwerkes oder die Adressen der Container werden aus der «.env»-Datei im gleichen Ordner entnommen. Wenn mit dem Admin Tool Container gestartet werden, generiert das Admin Tool mit folgendem Befehl `docker-compose -f <filename> config` jeweils die tatsächlichen Compose-Files. Diese heissen wie folgt:

- docker-compose.stack_build.yml
- docker-compose.stack_common-services.yml
- docker-compose.stack_run.yml

Diese drei Dateien sind nun mit den statischen Werten bestückt. Anhand der Variable «ENABLE_DOCKER_SWARM» in der «.env»-Datei wird entschieden, welche Template-Datei für die Docker Secrets verwendet werden soll. Ist Docker Swarm im Einsatz, wird die Datei «*_template_secretStore.yml» verwendet, welches die Docker Secrets aus dem Passwort-Store liest. Andernfalls wird die Datei «*_template_secretFile.yml» benutzt, welches die Docker Secrets anhand einer Datei pro Passwort realisiert. Diese Docker-Secret Konfigurationen werden mit dem Admin Tool automatisch in die «docker-compose.stack_*.yml» geschrieben.

5.4.4.5 Docker Modus

Die Container können in zwei verschiedenen Modi ausgeführt werden. Zum einen im Standalone Mode, welcher aber keinen Docker-Secret-Store unterstützt, und zum andern im Docker-Swarm-Mode [3], welcher auch Clustering unterstützt.

Damit der Betrieb wahlweise in einem der beiden Modi operieren kann, wurden die Compose-Dateien so angelegt, dass diese mit beiden kompatibel sind. So sind gewisse Konfigurationen in den YML-Dateien nur in einem oder dem anderen Modus verfügbar. Nicht verfügbare Optionen werden automatisch nicht beachtet. Es wurde sichergestellt, dass die Funktionalität in beiden Modi dieselbe ist.

Die Auswahl des Modus kann anhand der Variable «ENABLE_DOCKER_SWARM» in der «.env»-Datei festgelegt werden. Ansonsten muss nichts weiter beachtet werden. Das Admin Tool liest diese Variable und verhält sich dementsprechend. So werden die Container im Swarm-Mode automatisch als Service gestartet. Die dafür notwendigen Befehle unterscheiden sich in den Modi grundsätzlich. Daher wurde das Admin Tool so aufgebaut, dass immer mit den gleichen Befehlen, unabhängig vom Docker Mode, gearbeitet werden. Somit wird es dem Benutzer möglichst einfach gemacht, zwischen den Modi zu wechseln. Eine Ausnahme bildet die Ausgabe der Log-Dateien, welche im Swarm Mode manuell ohne Admin Tool ausgegeben werden müssen.

5.4.4.6 Docker Secrets [2]

Um Docker Secrets im Standalone wie auch im Swarm Mode verwenden zu können, wurde ein Bash-Skript namens «create_docker_secrets.sh» entwickelt. In dieser Datei sind die Passwörter im Klartext vorhanden. Wenn das Skript ausgeführt wird, wird in der «.env»-Datei überprüft, in welchem Docker Mode sich der Docker-Host befindet. Ist man im Swarm Mode, werden die Passwörter in einen Secret Store geschrieben. Ist man hingegen im Standalone Mode werden die Passwörter in einzelne Dateien geschrieben.

Damit Docker Secrets eingesetzt werden können, muss die im Container betriebene Applikation das Lesen des zu setzenden Passwortes aus einer Datei unterstützen, da die Secrets auf diese Weise unter `/run/secrets/<secret_name>` innerhalb eines Containers zur Verfügung gestellt werden. Die meisten vorgefertigten Docker-Images, welche in dieser Arbeit zur Verwendung kamen, unterstützen dies jedoch nicht. Um dies zu umgehen, kam ein im Internet aufzufindendes Skript namens «env_secrets_expand.sh» [26] zum Einsatz. Dieses wurde nur geringfügig abgeändert.

Um dieses Skript zu verwenden, wird anstelle des Klartextpassworts folgender Platzhalter in die Docker Umgebungsvariable geschrieben `DOCKERSECRET:<secret_name>`. Das Skript «env_secrets_expand.sh» wird entweder als Bash Source im Entrypoint-Skript an oberster Stelle hinzugefügt oder man ersetzt den Docker Entrypoint komplett. Wenn der Entrypoint ersetzt wird, muss beachtet werden, dass im neuen Entrypoint der Alte am Schluss wieder aufgerufen wird.

Das Skript «env_secrets_expand.sh» durchsucht beim Starten der Umgebung die Umgebungsvariablen nach einem Wert, der mit `DOCKERSECRET` beginnt. Somit weiss es, dass es diese Umgebungsvariable dem Prozess, welches das Skript ausführt, mit dem tatsächlichen Passwortinhalt zur Verfügung stellen muss. Darauf liest das Skript das Passwort aus der Secrets-Datei unter `/run/secrets/<secret_name>` und exportiert dieses in die prozesslokalen Umgebungsvariablen. Dies stellt sicher, dass nur der Prozess, welcher das Secret auch benötigt, in seinen Umgebungsvariablen darauf Zugriff hat.

5.4.4.7 Container

Im Folgenden werden alle verwendeten Container genauer beschrieben.

5.4.4.7.1 cas-traefik

Dieser Container implementiert den Reverse Proxy, welcher unter anderem für die Umleitung des HTTP-Verkehrs auf einen verschlüsselten HTTPS-Verkehr zuständig ist. Er übernimmt auch die TLS-Verschlüsselung des HTTPS-Verkehrs. Als Reverse Proxy wurde das Produkt Traefik [27] eingesetzt. Für die Generierung der zu verwendenden Zertifikate wurde ein Skript entwickelt, welches OpenSSL Zertifikate auf einfache Weise erstellen lässt.

Für Traefik werden Frontend-Regeln definiert, welche beschreiben, auf welche Ports und URLs gehört werden muss. Jede dieser Regeln verweist anschliessend auf eine Backend-Regel. Diese stellt den effektiven Host im privaten Netzwerk dar, welcher nur via Frontend-Regel vom Internet erreichbar ist. Die Regeln lassen sich entweder in einer Traefik-Konfigurationsdatei definieren oder man gibt in der Konfigurationsdatei an, dass Traefik diese Regeln anhand von Labels [28] der Container automatisch generieren soll. Da in der Traefik-Konfigurationsdatei nicht mit Variablen aus der «.env»-Datei gearbeitet werden kann, ist die zweite Variante besser für die Hacking-Lab 2.0 Umgebung geeignet. Somit

wurden bei allen Containern, welche vom Internet erreichbar sein müssen, ein Traefik-Label konfiguriert. Da die Services CAS-Server, CAS-Client und Consumer nicht ohne Authentifizierung erreichbar sein sollen, wurde das Traefik-Label nicht auf diesen Containern gesetzt, sondern auf den vor jedem dieser Services vorangestellten Keycloak Reverse Auth Proxy.

Der folgende Bildschirmausschnitt zeigt das Traefik Dashboard der Hacking-Lab 2.0 Umgebung, welches einen guten Überblick über die verbundenen Hosts vermittelt.

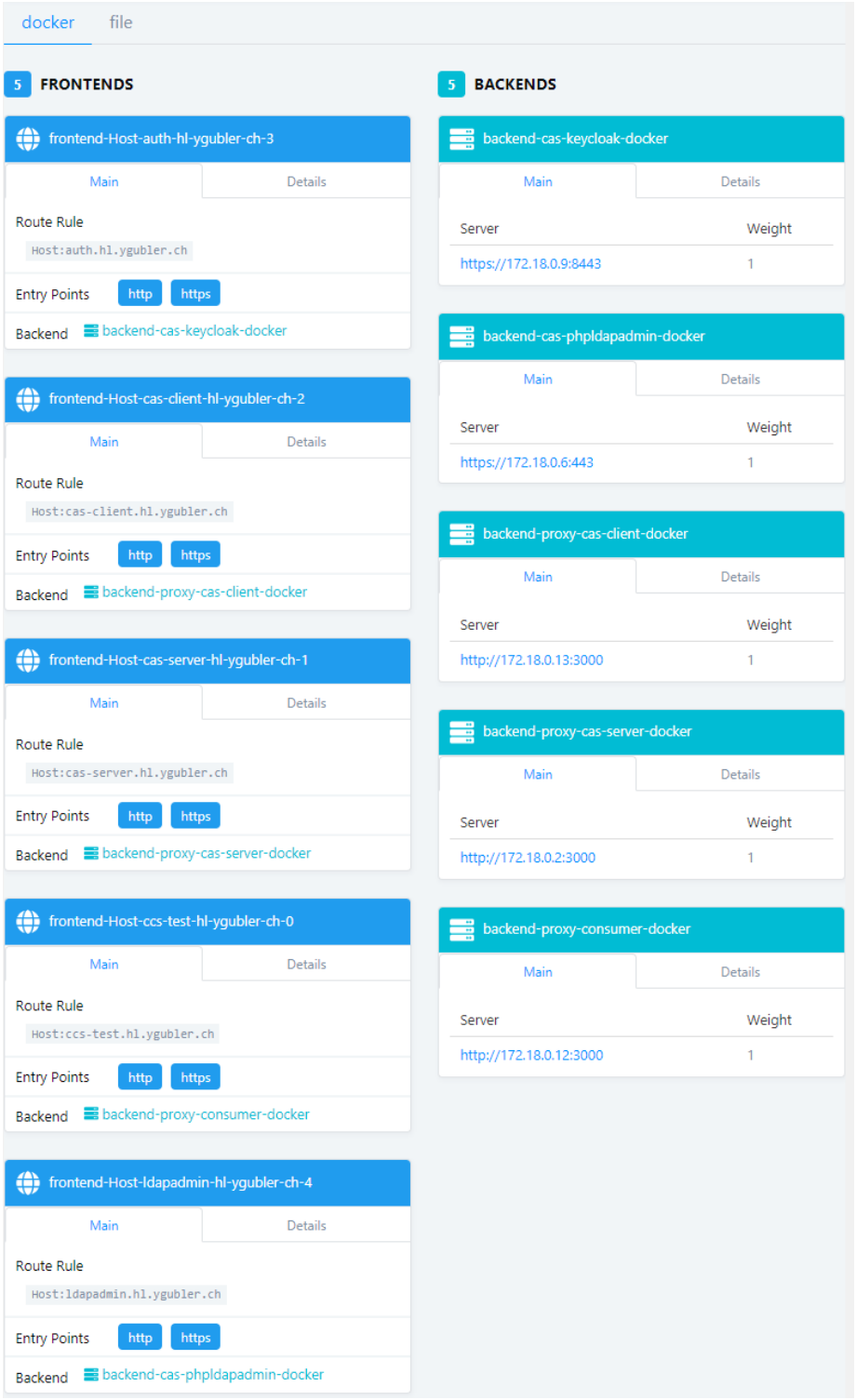


Abb. 23 Traefik Dashboard

5.4.4.7.1.1 Zertifikat-Generator

Der cas-traefik verwendet anders als der cas-ldap ein OpenSSL-Zertifikat für die Absicherung seiner Verbindungen. Um auch die Generierung solcher Zertifikate zu vereinfachen, wurden zwei Scripts implementiert, welche anhand einer Konfigurationsdatei die Erstellung übernehmen. Das erste Skript dient nur dazu ein Serverzertifikat zu erstellen, wobei das zweite zusätzlich eine CA erstellt.

5.4.4.7.2 cas-keycloak

Keycloak [29] ist eine Open Source Identitäts- und Zugriffsverwaltungslösung der Firma Red Hat. Diese Software bietet eine einfache Lösung an, einen SSO-Dienst zu betreiben.

Durch die Verwendung dieses SSO-Service wird erreicht, dass die Hacking-Lab Benutzer alle Microservices mit demselben Login verwenden können. Sobald sie sich an einer Stelle authentifiziert haben, ist es ihnen möglich alle Services mit der bereits aufgebauten Vertrauensstellung zu benutzen, ohne sich ein weiteres Mal anzumelden. Andere Vorteile sind vorgefertigte Abläufe für Login, Logout, Benutzerregistrierung, Zwei-Faktor-Authentifizierung und vieles mehr.

5.4.4.7.2.1 Übersicht

Im Folgenden wird ein Einblick auf die Hauptkomponenten von Keycloak vermittelt, um ein Verständnis für dessen Aufbau und Funktionsweise zu schaffen.

Folgende Grafik soll dabei helfen, den Aufbau zu verbildlichen.

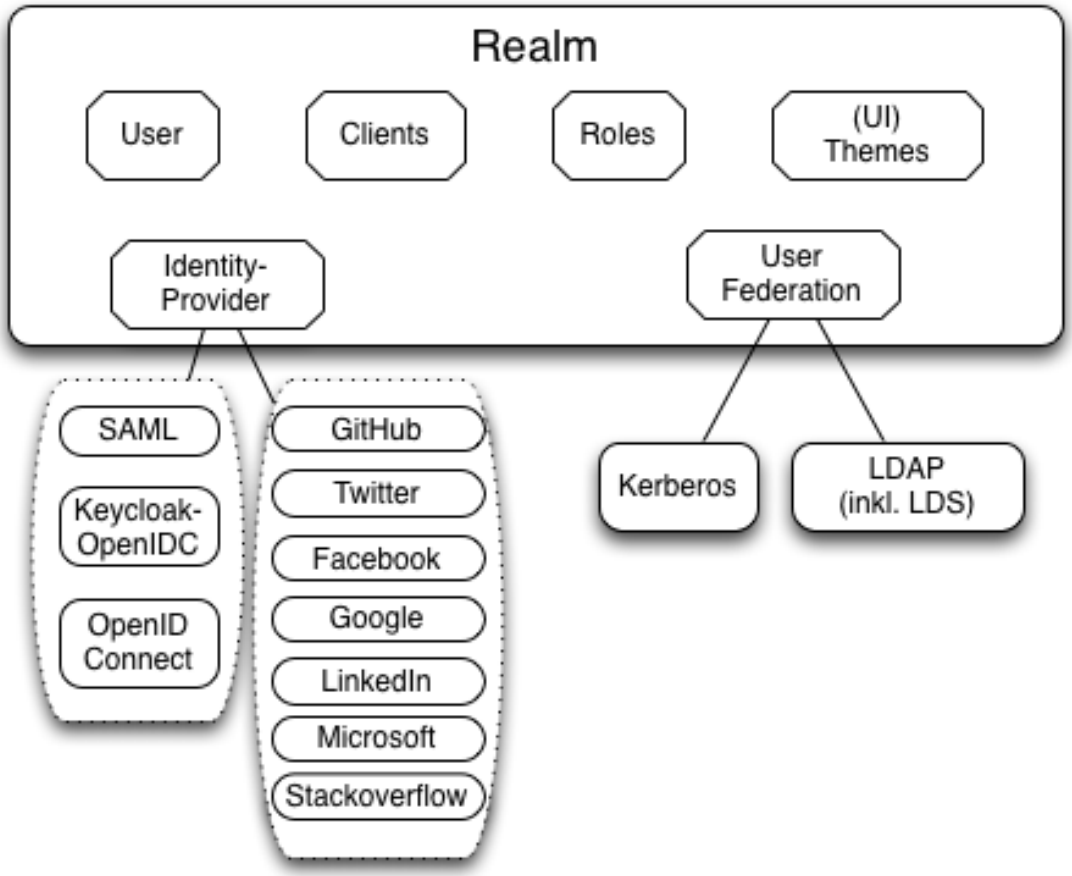


Abb. 24 Keycloak Übersicht [30]

5.4.4.7.2.2 Realm

Keycloak arbeitet mit sogenannten Realms, welche jeweils ein Bereich oder Territorium abbilden. Pro Realm gibt es eine eigene Benutzerverwaltung, Berechtigungen, Workflows, Identity Provider, User Federations und UI-Themes.

Im Kontext des Hacking-Lab 2.0 wird es eine globale Realm Hacking-Lab geben. Diese beschreibt das Territorium, welches das bestehende Hacking-Lab ablösen wird. Da das neue Hacking-Lab aber einen Schritt weitergeht, ist es in der Version 2.0 auch möglich, eine Realm für einen Kunden zu erstellen, der das Hacking-Lab 2.0 als Umgebung einkauft. So wird es dem Käufer möglich, seine Privatsphäre zu schützen, indem er mit seiner eigenen Realm einen komplett unabhängigen Bereich schafft.

Auf der Ebene Realm lässt sich konfigurieren, welche Interaktionen dem Benutzer auf der mitgelieferten Anmeldeseite zur Verfügung stehen. Im Folgenden werden die möglichen Optionen aufgelistet, welche auf einfache Weise de-/aktiviert werden können.

- User registration
- Email as username
- Edit username
- Forgot password
- Remember Me
- Verify email
- Login with email
- Require SSL

Die Authentifizierungsprotokolle, welche Keycloak verwendet, erfordern kryptografische Signaturen oder teilweise Verschlüsselung. Keycloak verwendet dafür asymmetrische Schlüsselpaare (einen privaten und einen öffentlichen Schlüssel). Der Service hält immer ein aktives und ein oder mehrere passive Schlüsselpaare. Ein solches Paar wird bei der Erstellung einer neuen Realm automatisch generiert. Um bessere Sicherheit gewährleisten zu können, empfiehlt es sich alle drei bis sechs Monate dieses zu erneuern und nach ein bis zwei Monaten nach der Erstellung der neuen Schlüssel die alten manuell zu löschen.

Weiter lässt sich festlegen, wie lange die SSO-Tokens und die Client-Logins maximal gültig sind oder nach welcher Zeitspanne von Inaktivität diese ungültig werden. Diese Werte wurden bei den Standardeinstellungen belassen.

Um zu verhindern, dass ein Brute-Force-Angriff erfolgreich verlaufen könnte, wurde die «Brute-Force-Detection» aktiviert, welche Benutzer nach überschreiten der maximal ungültigen Anmeldeversuche für eine gewisse Zeit aussperrt.

5.4.4.7.2.3 Client

Für jede Applikation, die mit Keycloak abgesichert werden soll, muss ein Client Objekt auf dem Keycloak-Server erfasst werden. Solche Clients sind dazu in der Lage JWTs auszustellen, welche einen angemeldeten Benutzer identifizieren und unter anderem, dessen Berechtigungen (Roles) beinhaltet. Ein solches Token gilt jedoch nur immer für den Client, welcher dieses auch ausgestellt hat. Wird nun auf einen anderen Client zugegriffen, kann durch die noch vorhandene Vertrauensstellung zwischen Benutzer und Keycloak-Server ein neues JWT (ohne Interaktion des Benutzers) für den anderen Client ausgestellt werden.

Um zu verhindern, dass ein potenzieller Angreifer sich als ein Client ausgibt, braucht ein Client immer eine Client-ID und ein Client-Secret. Diese werden vom Keycloak automatisch generiert und müssen beim Client hinterlegt werden. Im Rahmen dieser Bachelorarbeit wurden drei Keycloak Authentication Reverse Proxys eingesetzt, welche als Clients fungieren.

5.4.4.7.2.3.1 Mapper

Pro Client kann man Mappers konfigurieren. Mappers dienen dazu, Attribute wie Benutzernamen, E-Mail-Adressen oder sonstige Werte, welche in einem Keycloak-Userobjekt vorhanden sind, in das vom Client ausgestellte JWT einzufügen. Somit kann pro Auth Proxy entschieden werden, welche Informationen in das Token gelangen, das beim Backendsystem in Form von X-Auth-Headers verfügbar wird.

5.4.4.7.2.3.2 Client Roles

Roles können nicht nur auf der Ebene der Realm erstellt werden, sondern auch auf der Ebene Client. Somit wird es beispielsweise möglich, dass ein Benutzer auf der Ebene Realm nur die Rolle «hl_user» hat. Da dieser auf einem bestimmten Client Administrationsrechte benötigt, kann diesem eine Client-Role hinzugefügt werden. Diese ist dann nur spezifisch für diesen Client gültig. Auf diese Abstraktionsebene wurde bewusst verzichtet, da es bei der Hacking-Lab Umgebung vollends ausreicht, Roles auf der Ebene Realm zu erfassen.

5.4.4.7.2.3.3 Scope

Der Scope bezieht sich auf die Client Roles. Standardmässig ist «Full Scope Allowed» aktiv. Dies bedeutet, dass in jedem Client JWT alle Client Roles enthalten sind. Dies ist nicht zu empfehlen und sollte somit deaktiviert werden. Anschliessend kann manuell festgelegt werden, welche Client Roles, bei welchem Client im JWT enthalten sein sollen.

5.4.4.7.2.4 Roles

Roles bezeichnen Benutzerrollen, die einem Client / Auth Proxy weitere Möglichkeiten zum Blockieren oder Durchlassen eines Requests geben. So wird es möglich, dass Client Anfragen mit gewissen URL-Pfaden und HTTP-Methoden nur akzeptiert werden, wenn diese auch die entsprechende Role enthalten.

Das Hacking-Lab 2.0 besitz die folgenden drei Roles:

- hl_user
- hl_editor
- hl_translator

Jeder Benutzer ist standardmässig der Rolle «hl_user» zugewiesen, denn die Proxys blockieren alle Anfragen, welche nicht mit einem JWT, welches diese Rolle beinhaltet, versehen sind. Die übrigen zwei Rollen werden, von den Proxys nicht verwendet und auch nicht an das Backendsystem weitergereicht. Eine Ausnahme bildet jedoch der Proxy des CAS-Servers, bei welchem nicht nur die Rolle «hl_user», sondern auch die die restlichen («hl_editor» und «hl_translator») dem JWT und den X-Auth-Headers hinzugefügt werden. Diese werden vom CAS-Server analysiert, um gewisse Operationen nur zuzulassen, wenn der Benutzer / die Anfrage die benötigte Rolle besitzt. So können Challenges beispielsweise nur erfasst werden, wenn der Eintrag «hl_editor» im Request enthalten ist.

5.4.4.7.2.5 User Federation

Um zu erreichen, dass auch andere Dienste wie Linux PAM oder ein VPN-Dienst die Benutzeranmeldedaten vom Keycloak nutzen können, wurde hier eine LDAP-Anbindung vollzogen.

Da der verbundene LDAP-Server ein selbstsigniertes Zertifikat für das LDAPS-Protokoll verwendet, muss das Vertrauen in dieses erst etabliert werden. Geschieht dies nicht, lässt das System keine Verbindung mit der User Federation zu. Damit eine Verbindung aufgebaut werden kann, wird beim Buildvorgang des Docker-Containers «cas-keycloak» das Skript «add_cert_to_java_truststore.sh» in das Docker Entrypoint-Skript eingefügt. Dieses fügt dem Java Keystore das öffentliche Zertifikat des LDAP-Servers hinzu. Somit wurde die Vertrauensstellung erfolgreich etabliert und die Verbindung kann aufgebaut werden.

Wenn der «Edit Mode Writable» konfiguriert wird, ist die Synchronisation zwischen Keycloak und LDAP bidirektional. Aufgrund dieser Einstellung sind Änderungen an Benutzerobjekten immer in beiden Systemen vorhanden, unabhängig davon, auf welchem System diese ausgeführt wurden. Dabei ist aber zu beachten, dass neue oder überarbeitete Objekte im LDAP keine Synchronisation mit Keycloak auslösen, anders als wenn diese direkt im SSO-Dienst getätigt werden. Dies löste automatisch eine Synchronisation aus. Aus diesem Grund wurde «Periodic Full Sync» aktiviert, was bewirkt, dass periodisch ein kompletter bidirektionaler Abgleich aller Benutzerobjekte stattfindet.

Damit der «Edit Mode Writable» verwendet werden kann, sind an der Standardkonfiguration der LDAP-Mappers Änderung vorzunehmen. Da durch die Synchronisation in beide Richtungen auch Attribute aus dem LDAP im Keycloak überschrieben werden müssen, ist es notwendig die vorkonfigurierten Mappings anzupassen. Denn ohne Anpassungen sind diese als Read Only konfiguriert, wodurch die Updates vom LDAP nicht im Keycloak übernommen werden. Aber auch die Synchronisation in die andere Richtung ist im Auslieferungszustand nicht möglich. Um diese zu ermöglichen, müssen die Mapper «creation date» und «modify date» entfernt werden. Ansonsten versucht der SSO-Server die Attribute in die LDAP-Struktur zu schreiben. Dies lässt der verwendete LDAP-Dienst jedoch nicht zu, da diese dort schon bestehen und selbst vom System verwaltet werden.

Standardmässig werden nur Benutzerobjekte jedoch nicht Groups und Roles synchronisiert. Um auch diese mit dem Directory-Dienst abzugleichen, müssen Mappers auf der LDAP-Anbindung konfiguriert werden. Im Folgenden werden diese Mapper genauer erklärt. Bei der Verwendung dieser wird man bei der Nutzung der Default Groups und Roles eingeschränkt. In Keycloak kann definiert werden, welchen Gruppen und Rollen einem neu erstellten Benutzer zugewiesen wird. Sobald jedoch ein LDAP-Mapper auf Groups eingerichtet wird, funktioniert das Zuweisen des neuen Benutzers in eine oder mehrere Standardgruppen nicht mehr. Dies ist jedoch ein den Entwickler bekanntes Problem. Dieses wurde im Verlauf dieser Arbeit von einem Entwickler gelöst, jedoch noch nicht veröffentlicht. Weiter Informationen ist dem Open Issue [31] auf der JBoss Webseite zu entnehmen.

5.4.4.7.2.5.1 Realm Role Mapper

Dieser Mapper musste manuell erstellt werden.

Er wird im Modus «LDAP_ONLY» betrieben. Somit werden die Rollen immer vom LDAP gelesen. Diese Einstellung ist notwendig, da die Rollen sonst nicht im LDAP und im Keycloak gleichzeitig und konsistent angezeigt werden können. Dies ist die einzige der drei Einstellungsmöglichkeiten, bei der die Role Mitgliedschaften der Benutzer direkt auf Keycloak gemacht werden können und diese trotzdem im LDAP vorhanden sind. Ein weiterer Vorteil liegt darin, dass man die Roles auf diese Weise sowohl auf

dem LDAP, als auch auf dem Keycloak administrieren kann. Die beiden anderen Einstellungsmöglichkeiten sind die folgenden:

- IMPORT: Diese ist ähnlich wie READ_ONLY. Jedoch würden die Rollen Mitgliedschaften der Benutzer nur ein einziges Mal vom LDAP gelesen werden. Dies wäre der Fall, wenn ein User vom LDAP importiert wird und ist daher nicht wirklich praktikabel.
- READ_ONLY: Hier würde keine Synchronisation mit LDAP stattfinden. Die Rollen Zugehörigkeiten der Benutzer würden immer von beiden Datenbanken (Keycloak und vom LDAP) gelesen und zusammengeführt werden. Somit müsste man alle Role Zugehörigkeiten auf dem LDAP machen, damit Konsistenz darüber herrscht.

Weiter wurde festgelegt, wie die Rollen vom LDAP empfangen werden. Dazu wurde die User Roles Retrieve Strategy «LOAD_ROLES_BY_MEMBER_ATTRIBUTE» gewählt. Damit die Rollenzugehörigkeiten der User auf Keycloak angezeigt werden können, muss diese Option gewählt werden. Somit sieht man im Keycloak unter den Rollen zwar nicht die dazugehörigen Benutzer, man kann dafür auf dem Keycloak Userobjekt sehen, in welchen Rollen derjenige zugeteilt ist. Wenn für eine Rolle alle dazugehörigen Benutzer angezeigt werden sollen, kann dies im LDAP nachgeschaut werden, da dort für eine Rolle die dazugehörigen Benutzer angezeigt werden. Jedoch ist es dort wiederum nicht möglich für ein Userobjekt alle zugewiesen Rollen anzuzeigen. Also genau umgekehrt wie unter Keycloak. Die beiden anderen Einstellungsmöglichkeiten für die User Roles Retrieve Strategy wären:

- GET_ROLES_FROM_USER_MEMBEROF_ATTRIBUTE: Würde die Role Zugehörigkeiten anhand des User Memberof Attribute mappen. Dies funktionierte jedoch nicht, weil es genau anders herum im LDAP persistiert wird.
- LOAD_ROLES_BY_MEMBER_ATTRIBUTE_RECURSIVELY: für Active Directory vorgesehen.

Wenn im Keycloak eine Rolle erstellt wird, ist diese erst im LDAP sichtbar, wenn im role-ldap-mapper auf «Sync Keycloak Roles To LDAP» geklickt wird, oder ein User mit der neuen Role verknüpft wird, da beim Verändern des Benutzerobjekts automatisch eine Synchronisation gestartet wird.

Auch wenn die Roles in Keycloak gelöscht werden können, werden diese bei der nächsten Synchronisation der Roles mit LDAP wieder in Keycloak erstellt. Bei den ldap-group-mappern gibt es die Option «Drop non existing…», was dieses Problem beheben würde. Da es diese Option aber für die Roles nicht gibt, müssen die entsprechenden Roles auf dem LDAP und im Keycloak gelöscht werden, damit die Roles wirklich endgültig entfernt werden.

5.4.4.7.2.5.2 Client Role Mapper

Es wurde in dieser Umgebung auf die Verwendung von Rollen auf der Ebene von Clients verzichtet. Wenn diese eingesetzt werden würden, müsste für jeden Client ein eigener Mapper erstellt werden, welcher die clientspezifischen Rollen mit LDAP synchronisiert. Dies weil im Mapper angegeben werden muss, von welchem Client die Rollen mit LDAP synchronisiert werden sollen.

5.4.4.7.2.5.3 Group Mapper

Dieser Mapper wurde analog dem Role Mapper manuell erstellt. Dabei gibt es einen kleinen Unterschied bei den Einstellungsmöglichkeiten. Denn diese Mapper bieten die Möglichkeit zum Aktivieren der Option «Drop non-existing groups during sync». Diese ermöglicht anders als beim Role Mapper, dass Gruppen nur im LDAP oder im Keycloak gelöscht werden müssen und diese beim nächsten Abgleich auf beiden Systemen verschwunden sind.

5.4.4.7.2.6 Identity Provider

Keycloak macht es möglich, Identity Provider für die Anmeldung zu nutzen. So wäre es möglich, dass Benutzerkonten von einem SAML-, OpenID Connect-Dienst oder einem sozialen Dienst wie Facebook, GitHub, Google etc. für die Authentifizierung verwendet werden. Auf diese erweiterte Funktionalität wurde aber komplett verzichtet.

5.4.4.7.2.7 Export / Import

Keycloak bietet verschiedenste Varianten an, wie ein Export oder Import einer Konfiguration durchgeführt werden kann. Die einfachste ist jene, welche über die Keycloak Administrationswebseite verfügbar ist. Diese ist jedoch nur eingeschränkt verwendbar, da beim Export nur Groups, Roles und Clients exportiert werden. Client Secrets und Benutzer sind vom Export ausgeschlossen.

Eine weitere von Keycloak zur Verfügung gestellte Variante dient dazu, die Datenbank, in welcher die gesamten Daten inklusive Keycloak-Konfiguration persistiert werden, austauschen zu können. Dieser Export / Import wird beim Starten der Applikation / des Containers ausgeführt. Um dem Administrator dies zu erleichtern, wurde in der «.env»-Datei die Variable «KEYCLOAK_MIGRATION_ACTION» hinzugefügt. Ist diese leer, geschieht nichts. Ist sie jedoch mit dem Wert «export» initialisiert, speichert Keycloak den gesamten Inhalt seiner Datenbank in eine JSON-Datei pro Realm und je 50 Benutzer pro Realm. Steht sie hingegen auf «import» werden alle JSON-Dateien aus dem deklarierten Ordner in die Datenbank importiert.

5.4.4.7.2.8 Themes

Keycloak bietet die Möglichkeit, die vorgefertigten Themes der folgenden Bereiche zu individualisieren. Diese können auch mehrsprachig definiert werden.

- Anmeldeseite
- Benutzerkontenübersicht
- Administrationskonsole
- E-Mail

Im Rahmen dieser Arbeit wurde die Anmeldeseite geringfügig angepasst. Wie der folgende Ausschnitt zeigt, ist nun das Hacking-Lab Logo auf der Seite eingebunden.

Abb. 25 Keycloak Anmeldeseite

Falls ein neues Theme entworfen wird, kann dies in einem vordefinierten Ordner, aus dem der Keycloak Docker-Container gebuildet wird, abgelegt werden. Somit erscheint der neue Style auf der Keycloak-Administrationsseite automatisch zur Auswahl.

5.4.4.7.2.9 Benutzerregistration

Sobald auf ein vom Auth Proxy geschütztes System ohne gültiges JWT zugegriffen wird, leitet dieser den Benutzer auf die Anmeldeseite vom SSO-Service um. Wenn dem Zugreifenden keine Anmeldedaten zur Verfügung stehen, kann sich dieser selbstständig registrieren.

Um zu verhindern, dass Benutzer verschiedene Konten erstellen und um sicherzustellen, dass die vom Benutzer angegebene E-Mail-Adresse korrekt ist, wurde die Option der E-Mailadressbestätigung konfiguriert. Dazu wurde auf der Ebene Realm der SMTP-Server der HSR angegeben. Sobald sich nun ein neuer Hacking-Lab Benutzer registriert, wird dieser dazu aufgefordert, den Link einer vom SSO-Dienst automatisch ausgelösten Mail zu öffnen, um so seine Adresse zu bestätigen.

5.4.4.7.3 keycloak-mysql

Keycloak speichert alle Einstellungen und Konfigurationen in einer MySQL-Datenbank. Daher basiert dieser Container auf dem Standard MySQL Docker-Image, welches von Docker Hub bezogen werden kann. Der Pfad zur Datenbank kann in der Datei «.env»-Datei festgelegt werden. Wenn der Container beendet wurde, kann anhand dieses Pfades die Datenbank ohne Probleme auf ein anderes System migriert werden.

5.4.4.7.4 Keycloak Reverse Authentication Proxy

Als Grundlage für den Reverse Auth Proxy dient das Docker-Image von Ivan Bütler, welches er auf Docker-Hub veröffentlicht hat (ibuetler/keycloak-auth-proxy-golang:latest). Dieses Image basiert wiederum auf dem Image «8gears/keycloak-auth-proxy:goproxy», welches den offiziellen Keycloak Proxy [32] Server von Java auf Go adaptiert hat.

Dieses Docker-Image bietet die Möglichkeit nur Anfragen von authentifizierten und autorisierten Benutzer an den jeweiligen Backend-Service weiterzureichen. Ob eine Anfrage dazu berechtigt ist oder nicht hängt vom JWT, welches mit dem Request mitgeschickt wird, ab. Der Inhalt eines JWT sieht beispielsweise wie folgt aus:


```
{
    "jti": "d47bf3bf-02f4-48bd-a008-66cdc51c1d86",
    "exp": 1527783338,
    "nbf": 0,
    "iat": 1527783038,
    "iss": "https://auth.hl.ygubler.ch/auth/realms/Hacking-Lab",
    "aud": "cas_client_eu",
    "sub": "becab80e-960f-46c0-ad5e-1b95d4b65256",
    "typ": "Bearer",
    "azp": "cas_client_eu",
    "auth_time": 1527783038,
    "session_state": "f0a95e4a-2064-4294-b660-876e52239a4e",
    "acr": "1",
    "allowed-origins": [
        "https://cas-client.hl.ygubler.ch"
    ],
    "realm_access": {
        "roles": [
            "hl_user"
        ]
    },
    "resource_access": {},
    "preferred_username": "test_user",
    "given_name": "",
    "family_name": ""
}
```


Der Proxy entscheidet dann anhand der realm_access→roles, ob er die Anfrage weiterreichen soll. Wenn sie nicht die im Proxy definierte Role im JWT enthält, kann nicht zugegriffen werden. Wenn jedoch gar kein JWT vorhanden ist, wird der Benutzer auf die Anmeldeseite vom Keycloak-Server weitergeleitet. Dort kann er seinen Benutzernamen und sein Passwort angeben. Anschliessend wird er per Callback wieder an die ursprünglich aufgerufene Webseite zurückgeleitet. Da nun ein JWT im Request enthalten ist, wird überprüft, ob die entsprechende Role vorhanden ist und allenfalls weitergeleitet oder geblockt. Die Manipulation eines solchen JWTs ist nicht möglich, da sie vom Proxy signiert und so Manipulationen erkannt werden. Des Weiteren wäre es auch möglich die Tokens zu verschlüsseln, somit kann ausschliesslich der zuständige Proxy den Inhalt des Tokens einsehen. Eine entsprechende Option um die Verschlüsselung zu aktivieren gibt es in der «.env»-Datei, die dazu notwendige Variable lautet KEYCLOAK_PROXY_ENABLE_ENCRYPTED_TOKEN. Auch wenn das Token verschlüsselt übertragen wird, ergänzt der Proxy den Request mit dem entschlüsselten JWT in einem X-Auth-Token Header. Zudem entnimmt er auch gewisse Informationen aus dem Token und packt diese in die X-Auth-Headers, wie man auf folgender Illustration erkennen kann.

```
GET / HTTP/1.0
Host: cas.requestcatcher.com
Connection: close
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip
Accept-Language: de,en-US;q=0.7,en;q=0.3
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJzLTBzZ0Y1LUJyTmM4M0RzaGNKXy
Cache-Control: max-age=0
Connection: close
Cookie: kc-access=+CbVfxpN97w7pEosq6+vDencQnU71Q1mdo83AHBYAFO/DN3Vvkkvg2FxbSQuUFF6krumVaMcXkg6V4hiD0
Referer: https://auth.hl.ygubler.ch/auth/realms/Hacking-Lab/protocol/openid-connect/auth?client_id=c
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:59.0) Gecko/20100101 Firefox/59.0
X-Auth-Audience: cas_server_eu
X-Auth-Email: ygubler@hsr.ch
X-Auth-Expiresin: 2018-04-18 11:32:17 +0000 UTC
X-Auth-Groups:
X-Auth-Roles: hl_user,cas_server_eu:user
X-Auth-Subject: c7dda3e6-06b8-4777-8052-025e51b0d231
X-Auth-Token: eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJzLTBzZ0Y1LUJyTmM4M0RzaGNKXy1yVkdkMU
X-Auth-Userid: test
X-Auth-Username: test
X-Forwarded-For: 172.16.60.18
X-Forwarded-For: 172.16.60.18
X-Forwarded-Host:
X-Forwarded-Port: 443
X-Forwarded-Proto: https
X-Forwarded-Server: 8e1b145030d5
X-Real-Ip: 172.16.60.18
```

Abb. 26 X-Auth-Header Proxy --> Backend-Service

So ist es für die Backend-Services ein Einfaches Informationen, welche aus dem JWT stammen, aus den X-Auth Headers zu entnehmen.

5.4.4.7.4.1 Header Manipulationsschutz

Die X-Auth-Header lassen sich nicht manipulieren, aus diesem Grund muss im Backend auch nicht überprüft werden, ob diese Informationen nicht gefälscht wurden. Der Test für diese Behauptung ist der folgende:
Es wurde geprüft was passiert, wenn ein Angreifer versucht die X-Auth Headers bereits beim Absenden des Request mitzugeben. Sie werden, wenn sie bereits im JWT vorhanden sind, restlos ersetzt. Vom JWT nicht verwendete X-Auth-Headers werden aber vom Proxy belassen.

Für den Test wurde versucht einen X-Auth-Role Header, beim Senden desselben Requests wie oben, bereits vorher zu setzen. Dies wurde hier mit der Role «admin» und der Gruppe «testgroup» versucht. Wenn diese Attacke funktionieren würde, käme der Angreifer so zu Administrationsrechten im Backend. Weiter wurden noch nicht vom JWT verwendete Headers wie «tst» und «X-Auth-Test» eingeschleust. Die manipulierten Header sind in der folgenden Grafik ersichtlich.

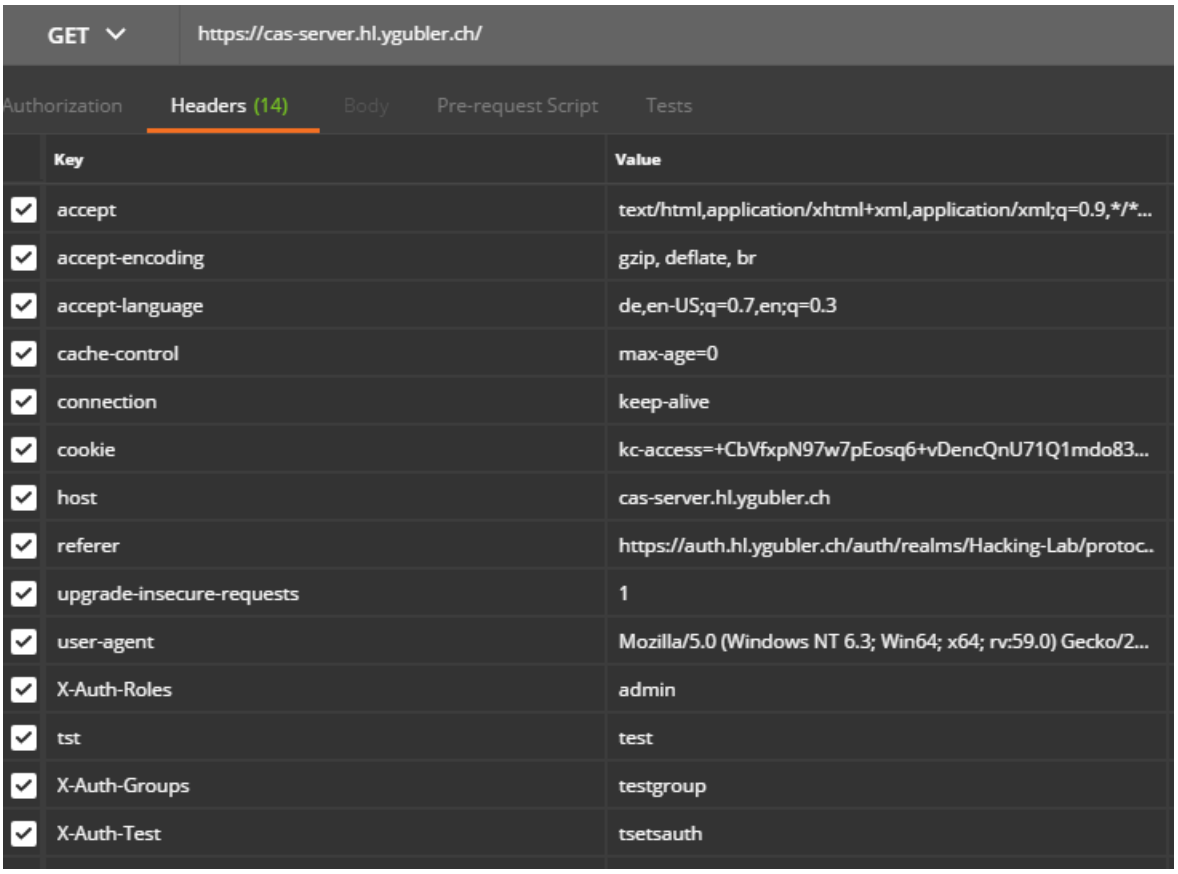


Abb. 27 Versuch eines X-Auth-Header-Angriffs

Die untenstehende Grafik zeigt nun wieder die Header, welche bei der Verbindung zwischen Proxy und Backend-Service vorhanden sind.

```
GET / HTTP/1.0
Host: cas.requestcatcher.com
Connection: close
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip
Accept-Language: de,en-US;q=0.7,en;q=0.3
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJzLTBzZ0Y1LUJyTmM4M0RzaGNKXy1y
Cache-Control: max-age=0
Cache-Control: no-cache
Connection: close
Cookie: kc-access=+CbVfxpN97w7pEosq6+vDencQnU71Q1mdo83AHBYAFO/DN3Vvkkvg2FxbSQuUFF6krumVaMcXkg6V4hiD0+b
Postman-Token: 3b4440de-8a5b-41fc-b4b0-0fd6e5b00d8c
Referer: https://auth.hl.ygubler.ch/auth/realms/Hacking-Lab/protocol/openid-connect/auth?client_id=cas
Tst: test
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:59.0) Gecko/20100101 Firefox/59.0
X-Auth-Audience: cas_server_eu
X-Auth-Email: ygubler@hsr.ch
X-Auth-Expiresin: 2018-04-18 11:32:17 +0000 UTC
X-Auth-Groups:
X-Auth-Roles: hl_user,cas_server_eu:user
X-Auth-Subject: c7dda3e6-06b8-4777-8052-025e51b0d231
X-Auth-Test: tsetsauth
X-Auth-Token: eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJzLTBzZ0Y1LUJyTmM4M0RzaGNKXy1yVkdkMUc3
X-Auth-Userid: test
X-Auth-Username: test
X-Forwarded-For: 172.16.60.18
X-Forwarded-For: 172.16.60.18
X-Forwarded-Host:
X-Forwarded-Port: 443
X-Forwarded-Proto: https
X-Forwarded-Server: 8e1b145030d5
X-Real-Ip: 172.16.60.18
```

Abb. 28 X-Auth-Header Proxy --> Backend-Server nach Angriff

Wie die obige Grafik zeigt, wurden die Headers, welche an der Anfrage vorhanden waren, mit den Werten aus dem Token ersetzt. So steht nun bei X-Auth-Groups nichts mehr, anstelle der «testgroup» welche einzuschleusen versucht wurde. Der X-Auth-Roles-Header ist wieder mit dem Wert «hl_user,cas_server_eu:user» aus dem JWT befüllt.

5.4.4.7.4.2 Vertrauensstellung zwischen Keycloak und Keycloak Auth Proxy

Um die Vertrauensstellung zwischen Keycloak und den Keycloak Auth Proxys herzustellen, muss jedem Proxy eine in Keycloak konfigurierte Client-ID und ein dazugehöriges Client-Secret via Docker Secret übergeben werden. Beim Starten der Proxy Container machen diese eine Konfigurationsanfrage mittels Client-ID und Client-Secret an den definierten Keycloak-Server. Falls dieser während dieses Vorgangs nicht erreichbar ist, beendet sich der Container wieder. Dies stellt insofern ein Problem dar, dass wenn alle Container gleichzeitig hochgefahren werden, die Proxys sicherlich vor dem Keycloak-Server bereitstehen würden und durch fehlenden Konfigurationsbezug gleich wieder beendet werden. Aus diesem Grund wurden die Reverse Proxys so konfiguriert, dass diese sich bei nicht Erhalt der Konfiguration automatisch neu starten.

5.4.4.7.4.3 Verwendete Proxy Instanzen

Im Rahmen dieser Bachelorarbeit wurde der beschriebene Proxy dreimal verwendet, sodass jegliche Anfragen die Rolle «hl_user» enthalten müssen, um zum jeweiligen Backendserver weitergeleitet werden zu können:

- proxy-cas-server
- proxy-cas-client
- proxy-consumer

5.4.4.7.5 cas-mysql

Diese Instanz eines MySQL-Servers in der Version 5.7.21 wurde mit dem offiziellen Docker-Image von Docker Hub realisiert. Innerhalb dieses RDBMS wird eine Datenbank betrieben, welche es dem CAS-Server ermöglicht, seine Daten (z. B. Challenges) zu persistieren. Der Name dieser Datenbank wie auch deren Anmeldedaten können durch Anpassungen der entsprechenden Variablen in der «.env.»-Datei verändert werden, dadurch wird auch der CAS-Server beim nächsten Start die neuen Angaben verwenden. Dieser Container speichert seine Daten (Datenbanken) auf dem Docker-Host an dem Pfad, welcher in der genannten Konfigurationsdatei hinterlegt wird. So können diese bei angehaltenem Container auch auf andere Systeme übertragen oder gesichert werden.

Dieser Container wird als einziger für die Build-Umgebung wie auch für die Run-Umgebung gebraucht.

5.4.4.7.6 CAS-Server

Um den CAS-Server in Betrieb zu nehmen, muss dessen Quellcode erst in ein ausführbares Programm übersetzt werden, welches anschliessend in Form einer ZIP-Datei zur Verfügung gestellt wird. Bei diesem Vorgang wird auch das Datenbankschema auf dem cas-mysql erstellt. Falls es bereits vorhanden ist, wird es via einer Datenbank-Evolution überschrieben. Im Anschluss daran kann das in der komprimierten Datei enthaltene Programm zur Ausführung gebracht werden. Diese beiden Schritte wurden jeweils in einen Docker ausgelagert, welche kurz vorgestellt werden. Beide fundieren auf einem offiziellen Alpine basierten OpenJDK Docker-Image, bei welchen nachträglich noch SBT installiert wurde.

5.4.4.7.6.1 build-cas-server

Beim Start des Containers werden zwei Laufwerke vom Host eingebunden. Das eine enthält den Quellcode des CAS-Servers und der andere dient zum Speichern der ZIP-Datei, welche das anschliessend ausführbare Programm enthält. Der Buildvorgang wird automatisch gestartet und bei dessen Beendigung stoppt auch der Container. Wenn die Datenbank auf dem cas-mysql noch nicht vorhanden ist, wird diese bei diesem Vorgang kreiert, andernfalls wird der Schritt zur Erstellung der Datenbank übersprungen.

5.4.4.7.6.2 run-cas-server

Dieser Dienst verbindet sich beim Starten mit dem Ordner auf dem Docker-Host, in welchem die vom build-cas-server generierte ZIP-Datei gespeichert wurde. Diese Datei wird im ersten Schritt im Container entpackt. Im Anschluss daran wird ein Skript aus der ZIP-Datei ausgeführt, welches die CAS-Serverapplikation startet.

5.4.4.7.7 CAS-Client

Wie der CAS-Server wird auch der Client in zwei Schritten mithilfe von zwei Containern zur Ausführung gebracht. Beide basieren auf dem jeweils aktuellsten Docker-Image von Node.js, welches von Docker Hub bezogen wird.

5.4.4.7.7.1 build-cas-client

Beim Starten des Containers werden zwei Ordner vom Docker-Host verbunden und die aktuellste Version des Node-Modules Semantic-UI installiert, welches für den Buildprozess benötigt wird. Das eine, der verbundenen Volumen, hält den Quellcode für den CAS-Client und das andere dient zum Speichern des durch Webpack fertig gebuildeten Codes.

5.4.4.7.7.2 run-cas-client

Dieser Docker installiert beim Starten die folgenden Node-Module, welche für die Lauffähigkeit, der durch Webpack generierten Dateien benötigt werden:

- express
- cors
- store
- body-parser
- xhr2

Bevor jedoch diese Module installiert werden, wird die vom build-cas-client fertig gebuildete Applikation in den Docker-Container kopiert, um diese anschliessend mit einem Node.js-Server den Benutzern zur Verfügung zu stellen. Der Server liest alle Parameter, welche für die Verbindung zum CAS-Server notwendig sind, sowie jene die der Node-Server für die Festlegung des zu verwendenden Protokolls und Port braucht, aus der «.env» Konfigurationsdatei.

Zum Abschluss wird der Node.js-Server gestartet und ist somit bereit die Client-Applikation den Benutzern zur Verfügung zu stellen.

5.4.4.7.8 CAS-Consumer

Der Ablauf verhält sich analog zu jenem des CAS-Client. So werden auch hier zwei Container mit dem aktuellsten Node.js Docker-Image verwendet.

5.4.4.7.8.1 build-consumer

Zwischen diesem Service und dem build-cas-client gibt es keine Unterschiede, bis auf die vom Host verbundenen Ordner. Der eine verweist auf den Quellcode des Consumers und der andere auf einen eigenen Pfad für die Speicherung des Consumer-Applikationscodes.

5.4.4.7.8.2 run-consumer

Dieser Docker installiert beim Starten die folgenden Node-Module, welche für die durch Webpack generierten Dateien benötigt werden:

- express
- xhr2

Bis auf die installierten Module verhält sich dieser Container analog dem run-cas-client.

5.4.4.7.9 LDAP

Um den vom Keycloak verwendeten LDAP-Server zu realisieren, wurden vorgefertigte Docker-Images des Docker Hub Repository von Osixia [33] verwendet, welche auf OpenLDAP basieren.

5.4.4.7.9.1 cas-ldap

Dieser OpenLDAP-Dienst ist vom Internet her erreichbar und steht nicht hinter dem Traefik Reverse Proxy. Aus diesem Grund muss die Verbindung gesondert mithilfe eines eigenen Zertifikats verschlüsselt werden. Hierfür ist es notwendig, dass das verwendete Zertifikat mit GnuTLS generiert wird. Diesbezüglich gibt es bei den Entwicklern von OpenLDAP einen Open Issue [34].

Insgesamt wurden drei Docker-Volumen verbunden, damit dieser Dienst neu gebuildet werden kann, ohne dass Daten/Konfigurationen verloren gehen. Die genannten Verzeichnisse sind dafür zuständig, dass die Konfiguration, die Datenbank und die Zertifikate zur Verfügung gestellt werden. Diese Ordner wurden auch ins Git-Repository aufgenommen, damit auf einem neuen Docker-Host die DNs für die Keycloak Roles und Groups bereits vorhanden sind. Denn diese müssten ansonsten manuell erfasst werden, damit der LDAP-Sync von Keycloak nicht fehlschlägt.

5.4.4.7.9.1.1 default.startup.yaml

Diese Datei wird gebraucht, um die benötigten Umgebungsvariablen für die LDAP-Konfiguration zu setzen. Nach dem Start des Containers wird diese Datei automatisch gelöscht, wodurch die Werte nicht in den Container-Umgebungsvariablen ersichtlich sind. Somit bietet diese Datei eine sichere Variante, um Passwörter und andere sensible Werte zu übergeben. Jedoch muss sie gesondert von der «.env»-Datei konfiguriert werden.

5.4.4.7.9.1.2 createLDAPCerts.sh

Um die Erstellung eines neuen Zertifikats für den LDAP zu vereinfachen, wurde dieses Skript entwickelt. Nach dem Start werden die für die Generierung benötigten Parameter direkt auf der Konsole verlangt. Sobald dieser Vorgang beendet ist, wird das CA Zertifikat und das Serverzertifikat dem Docker-Image zur Verfügung gestellt. Beim nächsten Start des Containers wird somit automatisch das neue Zertifikat verwendet. Es sollte jedoch vermieden werden, dieses Skript laufen zu lassen, wenn kein neues Zertifikat erstellt werden soll. Da ansonsten das bestehende überschrieben wird.

5.4.4.7.9.1.3 Verschlüsselte LDAP Verbindung

Um sicherzustellen, dass nur verschlüsselte Daten vom LDAP übertragen werden, wurden die übermittelten Pakete mit Wireshark analysiert. Denn anders als ausserhalb des Docker-Netzwerks ist innerhalb nicht nur der verschlüsselte LDAPS-Port 636 geöffnet, sondern auch der Port 389, welcher normalerweise für eine unverschlüsselte LDAP-Verbindung verwendet wird. Jedoch konnte anhand der Aufzeichnung der Netzwerkpakete die Korrektheit der Konfiguration, dass nur verschlüsselte Verbindungen mit LDAP zugelassen sind, bestätigt werden. Denn wenn versucht wird einen unverschlüsselten Kanal zu öffnen, wird dieser sofort abgewiesen. Erst wenn LDAP auf Port 389 mit TLS verschlüsselt oder direkt LDAPS auf Port 636 verwendet wird, wird der Verkehr zugelassen.

5.4.4.7.9.1.4 POSIX-Account für Linux PAM

Damit die LDAP Objekte auch für Linux-PAM verwendet werden können, müssen diese zusätzlich mit Attributen eines POSIX-Accounts angereichert werden. Die noch fehlenden und nicht durch Keycloak befüllten Attribute sind die folgenden:

- gidNumber
- homeDirectory
- uidNumber

Da das manuelle Eintragen dieser Werte keine praktikable Möglichkeit darstellt, müsste ein automatischer sich wiederholender Job entwickelt werden, welcher diese Felder ausfüllt. Weiter dürfen diese nicht aus statischen Werte bestehen und müssen daher über alle Benutzer eindeutig sein. Eine solche Logik konnte aufgrund mangelnder Zeit nicht mehr umgesetzt werden.

5.4.4.7.9.2 cas-phpldapadmin

Damit das LDAP-Schema auf einfachste Weise grafisch betrachtet werden kann, sowie neue Einträge mit einer Weboberfläche getätigt werden können, wurde das phpLDAPadmin-Image eingesetzt. Dieses bietet eine über Traefik erreichbare Webseite, welche nach erfolgreicher Anmeldung das Schema wie folgt anzeigt und diverse Operationen zur Verwaltung und Erstellung neuer Objekte bietet.

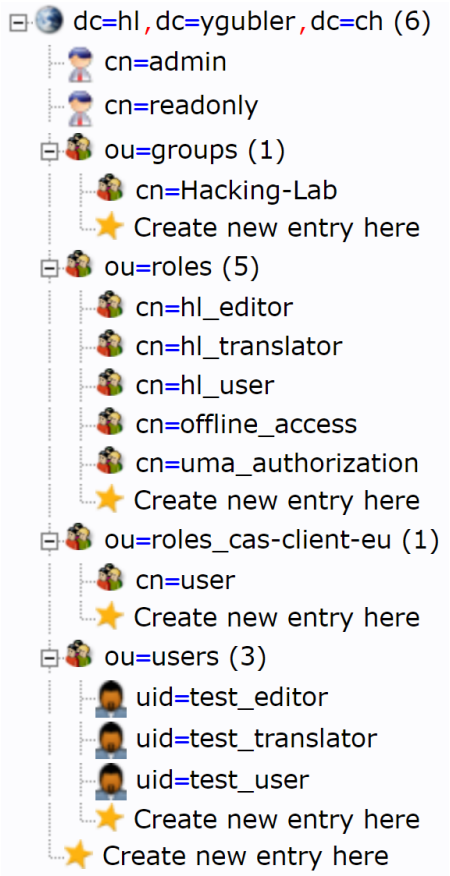


Abb. 29 phpLDAPadmin Schema Übersicht

Da der LDAP-Server ein selbstsigniertes Zertifikat verwendet, ist es wie beim cas-keycloak notwendig, dass das öffentliche Zertifikat beim cas-phpldapadmin bekannt gemacht wird. Dies wurde erreicht, indem es als Docker-Volume verbunden wird und innerhalb des Containers an den Standardspeicherort der Zertifikate verlinkt wird. So wird es standardmässig für den Verbindungsaufbau verwendet.

5.4.4.7.9.3 Backup

Um die LDAP-Daten sichern zu können, gibt es folgende drei Varianten, wobei auf die containerbasierte Variante verzichtet wurde.

- Sichern der Ordner, welcher der Service cas-ldap per Docker-Volume verbunden hat
- Mittels cas-phpldapadmin einen Export als CSV, DSML, LDIF oder VCARD des LDAP-Schemas durchführen
- Sichern mit Hilfe eines eigens von Osixia dafür entwickelten Docker-Container [35]

5.4.4.7.10 Jenkins

Dieses webbasierte CI-System wurde im Rahmen dieses Projekts für die wichtigsten Kernsysteme verwendet. Somit kann sichergestellt werden, dass die zu entwickelnden Systeme immer in der neusten Version auf einem Server zur Ausführung gebracht werden. Jenkins wurde direkt auf dem Docker-Host verwendet.

Sobald ein neuer Commit auf dem Master-Branch getätigt wird, löst dies einen neuen Build-Job im CI-System aus. Dieser führt alle implementierten Tests der Systeme CAS-Server und CAS-Client aus. Auf das Consumer-System wurde bewusst verzichtet, da dieses im Umfang der Bachelorarbeit keine entscheidende Komponente ist. Wenn die Tests und der Build der Systeme erfolgreich durchgeführt werden konnten, werden die generierten lauffähigen Applikationen auf den jeweiligen Docker-Container bereitgestellt. Somit wird erreicht, dass stets ein Zugriff auf die Komponente in der neusten Version zur Verfügung steht. Bei diesen Systemen wurde jedoch auf die Verwendung von Traefik und Keycloak verzichtet.

Die Services werden mithilfe der folgenden Container bereitgestellt, wobei sich diese nicht von den Containern in der Run-Umgebung unterscheiden. Damit diese Systeme keinen Einfluss auf das schlussendlich abzuliefernde Produkt haben, wurden dessen Konfigurationen in abgesonderten Dateien vorgenommen.

- jenkins-cas-server
- jenkins-cas-client
- jenkins-cas-mysql

5.4.4.8 Zugriff aus dem WWW

Dieses Diagramm soll die textuell beschriebenen Umsetzungen bezüglich Reverse Proxys für die verschlüsselte Kommunikation, wie auch für die Authentifizierung im Zusammenhang mit Keycloak und LDAP verbildlicht darstellen.

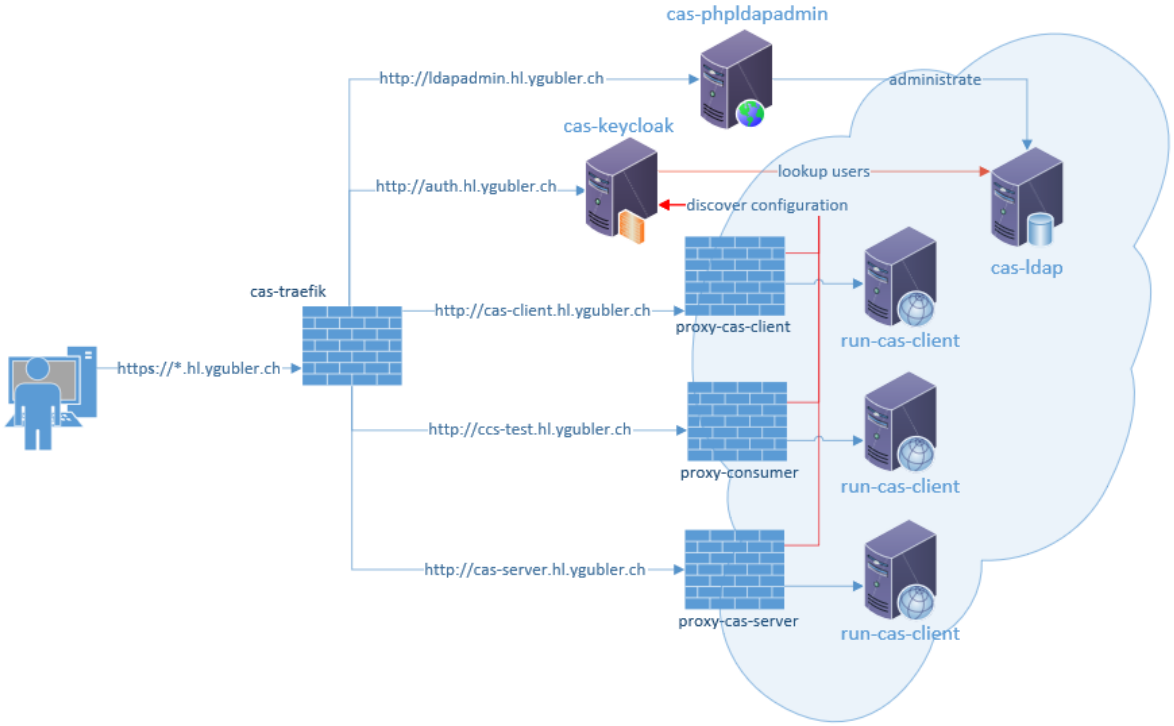


Abb. 30 Netzwerkdiagramm

Es ist klar ersichtlich, dass ein Request nur bis zum Traefik Reverse Proxy mittels HTTPS übertragen wird und anschliessend an den entsprechenden Dienst, mittels unverschlüsseltem HTTP-Protokoll, weitergeleitet wird. Weiter ist zu erkennen, dass alle Anfragen an die Dienste CAS-Server, CAS-Client und Consumer den Keycloak Auth Reverse Proxy passieren müssen. Dafür muss die Realm Rolle «hl_users» im JWT des Requests enthalten sein. Weiter Aspekte sind, die Erreichbarkeit der Services cas-keycloak und cas-phpldapadmin aus dem Internet über den Traefik.

5.4.4.9 Ablauf eines Verbindungsaufbaus mittels SSO

Da der Aufbau einer Verbindung bis zu den in der Bachelorarbeit implementierten Systemen durch die Verwendung von zwei unterschiedlichen Reverse Proxy's (Traefik und Keycloak Auth Proxy) und einem SSO-Dienst sehr komplex ist, wird dieser im folgenden Sequenzdiagramm auf zwei Seiten veranschaulicht. Dabei gehört alles im grün hinterlegten Bereich Ersichtliche, zum Ablauf bei einem Erstaufruf des run-cas-client ohne die Übergabe eines JWTs und das violett hinterlegte zum Zugriff auf den run-cas-server, wobei hier bereits das Token als Cookie vorhanden ist. Es ist zu erkennen, dass der gesamte Verkehr zwischen Benutzer und Traefik mit TLS gesichert ist. Der Verkehr hinter Traefik wird anschliessend unverschlüsselt mit HTTP übertragen.

5.4.4.9.1 Ablauf Erstaufruf

Beim Aufruf der CAS-Client Webseite ist noch kein Token vorhanden, welches den Zugriff, durch den proxy-cas-client geschützten Dienst, erlauben würde. Aus diesem Grund leitet der Proxy den Browser mit einem Callback zum Keycloak-Server weiter. Dort hat sich der Benutzer zu authentifizieren, wobei der SSO-Dienst die Anmeldedaten beim cas-ldap verifiziert. Bei einer erfolgreichen Verifikation wird das JWT gegenüber dem cas-keycloak dem Browser übermittelt. Dieses bemächtigt den proxy-cas-client zum Ausstellen eines für ihn gültiges JWT. Mit diesem kann abschliessend die anfänglich angefragte Seite ausgeliefert werden.

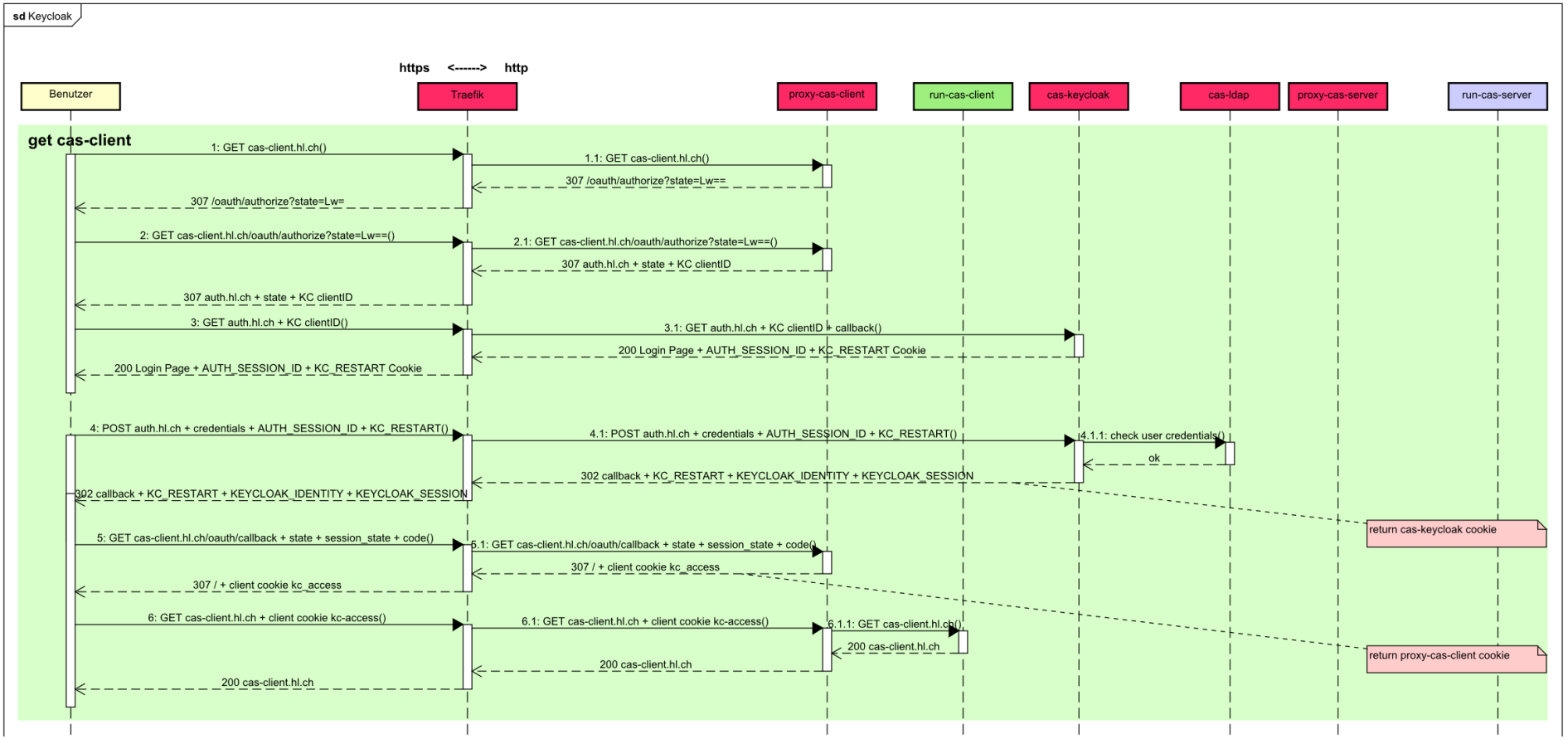


Abb. 31 Sequenzdiagramm Verbindungsaufbau Teil 1

5.4.4.9.2 Ablauf bei vorhandener Vertrauensstellung

Im zweiten Teil des Sequenzdiagramms wird der Aufruf des run-cas-servers visualisiert. Es ist zu beachten, dass der Benutzer hier bereits ein JWT, welches vom proxy-cas-client ausgestellt wurde, besitzt und dieser auch gegenüber dem Keycloak-Server eine Vertrauensstellung aufgebaut hat. Somit wird der Benutzer nicht ein erneutes Mal aufgefordert seine Anmeldedaten einzugeben. Für ihn sieht es beim Aufruf des Service so aus, als ob dieser keine Authentifizierung benötigt, da diese komplett automatisch im Hintergrund mittels den bereits vorhandenen Tokens vom Browser durchgeführt wird.

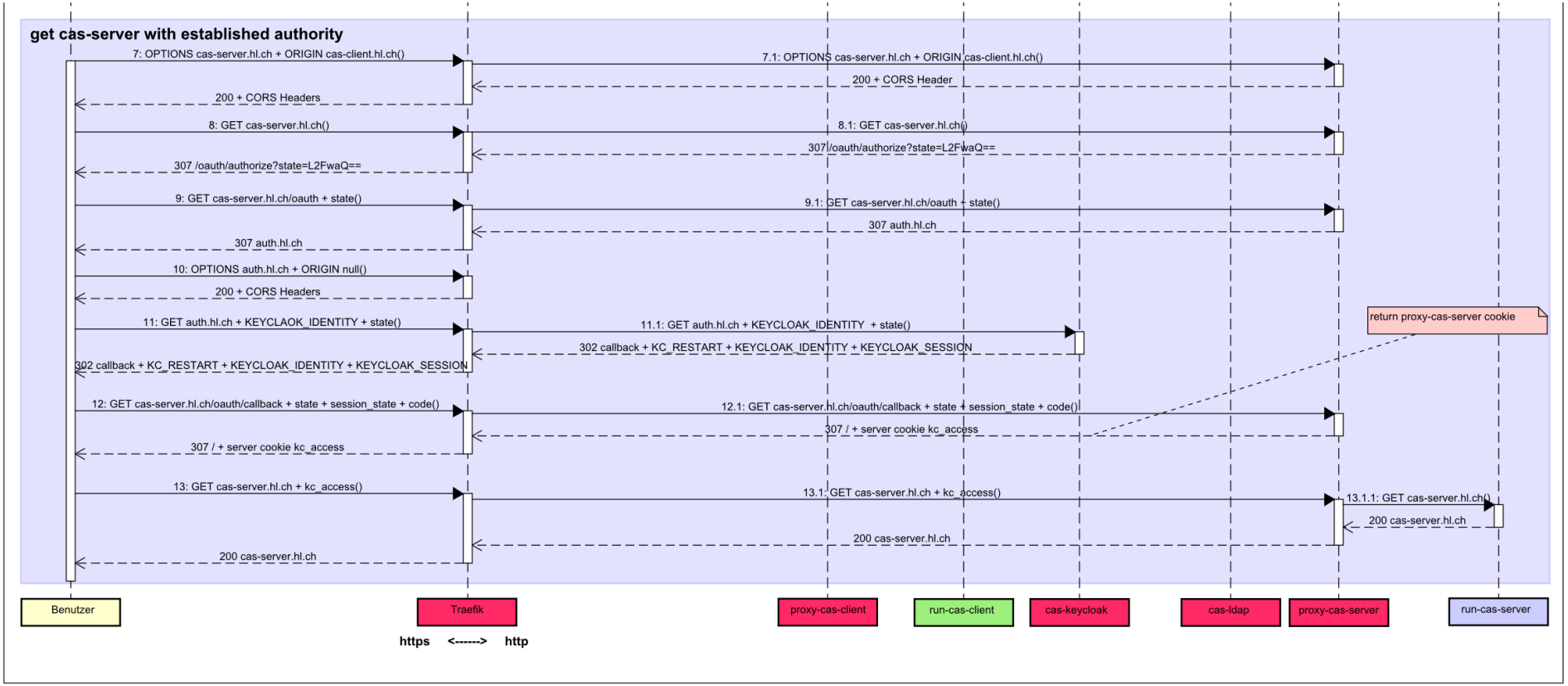


Abb. 32 Sequenzdiagramm Verbindungsaufbau Teil 2

5.4.4.10 CORS

Um gegen XXS-Attacken (Cross-Site-Scripting) geschützt zu sein, verwenden moderne Browser die SOP (Same-Origin-Policy), um Cross-Origin-Anfragen zu unterbinden. Dadurch ist es nicht möglich, dass eine Anfrage der Seite A an eine andere mit abweichender Domain ohne Vorkehrungen durchgeführt werden kann. Um dies zu ermöglichen, muss der angefragte Server mit einem CORS-Header antworten, welcher die im Request enthaltene Domain von A als Origin, mit einem Access-Control-Allow-Origin bestätigt. Erst mit dieser zusätzlichen Information lassen es die Webbrowser zu, dass für diese bestimmte Seite Cross-Origin-Anfragen abgesetzt werden können. Bei der Umsetzung wurde diese Thematik im Zusammenhang mit Keycloak und dessen Reverse Proxys zu einer grösseren Herausforderung.

Wie im Kapitel 5.4.4.9 ersichtlich wird, ist man nach dem Zugriff auf den CAS-Client und einer erfolgreichen SSO-Anmeldung erst gegenüber dem «proxy-cas-client» und «cas-keycloak» authentifiziert. Sobald die Client Seite (https://cas-client...) geladen wurde, wird automatisch eine Anfrage an den CAS-Server (https://cas-server...) gemacht, um eine Liste aller verfügbaren Consumer zu erhalten. Da diese aber die Same-Origin verlässt, veranlasst der Browser eine OPTIONS-Request an den Server, welche aber nicht durch diesen beantwortet werden kann, weil diese durch fehlende Authentifizierung gegenüber dem «proxy-cas-server» abgefangen wird. Aus diesem Grund wurden die «Access-Control-Allow-…» Header-Antworten auf diesem Proxy konfiguriert. Somit könnten die OPTIONS- Requests auch bei fehlendem Token gegenüber dem anzufragenden Proxy beantwortet werden.

Nachdem nun vom Proxy mit korrekten Headers geantwortet wurde, man aber noch das korrekte JWT benötigt, leitet dieser den Browser, für den Bezug eines neuen «proxy-cas-server»-Token, an den «cas-keycloak» weiter. Dieser muss somit auch wieder mit CORS-Headers auf OPTIONS- Requests antworten können, da der Browser bei der Token-Anfrage ein weiteres Mal die SOP beim Zugriff (https://auth...) verletzt. Somit muss auch der «cas-keycloak» auf CORS-Anfragen antworten können, da diese Software diesbezüglich keine Unterstützung bietet, musste eine Lösung gefunden werden, um auf OPTIONS-Requests antworten zu können. Bei der Analyse dieser Requests hat sich weiter herausgestellt, dass diese als Origin den Wert null enthalten, anstelle der URL des CAS-Clients. Da dieser Wert, unabhängig von welchem Client der Request ausgelöst wird, konstant null ist, konnte auf dem «cas-traefik» ein statischer Eintrag gemacht werden. Dieser antwortet auf alle Anfragen an den Keycloak-Server mit den, auf folgendem Bildausschnitt ersichtlichen, Access-Control-Allow-Headers.

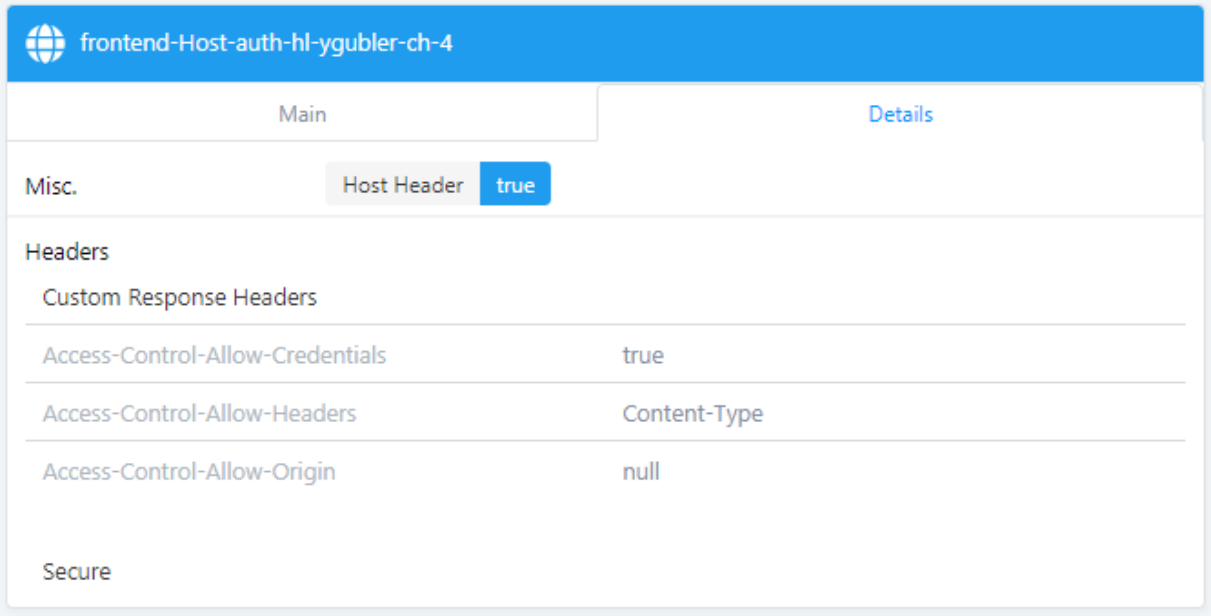


Abb. 33 Traefik Frontend Rule - Keycloak CORS Header

Um zu gewährleisten, dass diese Umsetzung funktionstüchtig ist, musste beim Absetzen der Anfrage einiges beachtet werden. Denn damit allen Redirects gefolgt werden kann und dabei dennoch die Anmeldedaten für Keycloak beinhaltet werden, bedingt es dem Request folgende Argumente zu übergeben.

- credentials: 'include'
- redirect: 'follow'

Damit der Antwort mittels JavaScript Daten entnommen werden können, muss der Request wie folgt als CORS-Anfrage deklariert werden.

- mode: 'cors'

Die Gesamtheit dieser Konfigurationen schafft es, eine voll funktionsfähige Umgebung mit CORS-Unterstützung zu etablieren.

5.4.5 KeycloakUserImporter

Um die bestehenden Benutzer aus dem aktuellen Hacking-Lab ins neue überführen zu können, wurde im Rahmen der Bachelorarbeit der KeycloakUserImporter implementiert, um einen Benutzerimport mittels einer CSV-Datei zu bewerkstelligen.
Diese kleine Java Applikation verwendet das Keycloak Java API [36], um eine Verbindung mit Keycloak aufzubauen. Dieses API funktioniert nur in der Java Version 8 und unterstützt Java 10 noch nicht.

Das Programm lässt es zu folgende Attribute zu importieren:

- Username
- Lastname
- Firstname
- Email
- Password
- Realm Roles
- Groups
- Default Passwort (wenn keines im CSV vorhanden ist)
- isMailVerified

5.4.5.1 config.properties

Es folgt der Inhalt der Datei «config.properties». Mittels dieser Konfigurationsdatei lassen sich alle Einstellungen für den Import der Benutzer zentral verwalten.

```
// keycloak connection settings
keycloakAuthURL=https://auth.hl.ygubler.ch/auth
keycloakAuthRealm=master
// leave empty to read username from console
keycloakAuthUsername=admin
// leave empty to read password from console
keycloakAuthPassword=
keycloakAuthClientId=admin-cli
keycloakRealm=Hacking-Lab

// cert settings
httpsPubCertKeycloakPath=C:\\Git\\ChallengeAuthoringSystem\\KeycloakUserImporter\\cert.pem
javaKeystorePassword=changeit
javaKeystoreCaAlias=keycloak

// select steps to execute
writeToLogfile=false
ConsolePrintUsersFromCSV=true
createUsers=true
assignUsersToRealmRoles=true
assignUsersToGroups=true
deleteUsersAfterCreation=true

// csv settings
csvPath=C:\\Git\\ChallengeAuthoringSystem\\KeycloakUserImporter\\Users.csv
csvSeperator=;
csvUseQuotes=false
csvQuote="
csvDefaultPassword=test1234
// set to true if users mailaddress should be verified after import
isEmailVerified=true
// csv mappings
csvUsername=username
csvLastname=lastname
csvFirstname=firstname
csvEmail=email
csvPassword=password
csvRealmRoles=realmRoles
csvGroups=groups
```

5.4.5.2 Java Keystore Generierung

Da der Keycloak-Server (cas-keycloak) hinter dem Traefik Reverse Proxy betrieben wird, welcher den HTTP-Verkehr mittels TLS sichert (HTTPS), ist es notwendig, das öffentliche TLS-Zertifikat vom Traefik in einen neuen applikationseigenen Java Keystore zu importieren. Diese Notwendigkeit ist darauf zurückzuführen, dass das für die Bachelorarbeit verwendete Zertifikat nicht öffentlich signiert ist. Somit ist dieser Import notwendig, um der Java Applikation das Vertrauen gegenüber dem HTTPS-Endpoint (Traefik) zu gewährleisten. Das zu importierende Zertifikat wird in der Datei «config.properties», welche im KeycloakUserImporter Ordner gespeichert ist, mittels Pfadangabe bekannt gegeben.

5.4.5.3 Konsolenausgabe der gelesenen Benutzer aus der CSV-Datei

Nachdem der neu angelegte Java Keystore geladen wurde, werden alle Benutzer aus der CSV-Datei geladen und auf der Konsole ausgegeben.

5.4.5.4 Benutzererstellung

In diesem Schritt werden die Benutzer aus der CSV-Datei in Keycloak UserRepresentation-Objekte geladen und anschliessend zum Keycloak-Server übermittelt, welcher diese ins LDAP persistiert. Damit diese und die folgenden Schritte performant sind, werden die Operationen nebenläufig ausgeführt.

5.4.5.5 Benutzerzuweisung an Rollen und Gruppen

Bei der Zuweisung der neuen Benutzer an Rollen/Gruppen, werden die Rollen/Gruppen-Zuweisungen nur erstellt, wenn diese bereits in der Keycloak-Datenbank existieren. Ist dies nicht der Fall, wird eine entsprechende Fehlermeldung auf der Konsole ausgegeben. Wenn in der Konfigurationsdatei die Option entsprechend gesetzt ist werden die Fehler zusätzlich in eine Logdatei geschrieben. Die nichtexistierenden Rollen/Gruppen müssen händisch in Keycloak erfasst werden.

5.4.5.6 Entfernen der Benutzer

Dieser Schritt erleichtert das Löschen der Benutzer erheblich, da dies sonst für jeden Benutzer im Keycloak Web-GUI einzeln gemacht werden muss.

5.5 Ergebnisdiskussion mit Ausblick

5.5.1 Ergebnis

Auf der Grundlage der vorgelagerten Studienarbeit wurde in der Bachelorarbeit die Docker-Laufzeitumgebung verbessert und mit weiteren Komponenten ergänzt. Somit stehen nach dieser Arbeit Dienste wie Challenge-Import/-Export, Challenge-Editor, Challenge-Translator, OAuth Identity Provider, User Federation via LDAP, SSO, Benutzerverwaltung/-registrierung/-import, Autorisierung durch Reverse Auth Proxys und Load-Balancing zur Verfügung.

Durch die Entwicklung des Challenge-Editors können nun auf einfachste Weise neue Challenges erfasst und bearbeitet werden, welche anschliessend durch den neuen Challenge-Translator in eine andere Sprache übersetzt werden kann.

Bei der Implementierung wurde ein RESTful API verwendet, welches es erlaubt, die Software einfach in den von der Security Competence GmbH entwickelten Kontext zu integrieren. Weiter wurden die Services als Microservices aufgebaut, so ist eine einfache Skalierung der verschiedenen Applikationen möglich.

Die folgende Tabelle soll eine Übersicht über die vom Auftraggeber geforderten Komponenten und deren Status der Zielerreichung vermitteln.

Komponente	Priorität	Gelöst	Kommentar
Challenge-Editor	Hoch	Ja	Wurde vollumfänglich implementiert
Challenge-Translator	Hoch	Ja	Wurde vollumfänglich implementiert
ChallengeHandler (für Import/Export)	Hoch	Ja	Wurde vollumfänglich implementiert
KeycloakUserImporter	Hoch	Ja	Wurde vollumfänglich implementiert
CAS-Server	Hoch	Ja	Wurde vollumfänglich implementiert
CAS-Client	Hoch	Ja	Wurde vollumfänglich implementiert
CAS-Consumer	Niedrig	Teilweise	Wurde nur als Mock gebraucht
Keycloak-SSO	Hoch	Ja	Wurde vollumfänglich implementiert
LDAP User Federation	Hoch	Ja	Wurde vollumfänglich implementiert
Traefik Load-Balancer	Hoch	Ja	Wurde vollumfänglich implementiert
Reverse Auth Proxys	Hoch	Ja	Wurde vollumfänglich implementiert
Linux-PAM	Niedrig	Nein	Da Keycloak das Hinzufügen von POSIX-Attributen bei der LDAP User Federation nicht unterstützt, müsste eine Logik implementiert werden, die dies direkt auf dem LDAP übernimmt. Aus Zeitknappheit wurde komplett darauf verzichtet.
Mandantenfähigkeit	Hoch	Teilweise	Wurde nicht umgesetzt, jedoch wurde dies Domain Model berücksichtigt und kann in einem weiteren Schritt einfach umgesetzt werden.
Admin Tool	Mittel	Ja	Damit die Infrastruktur einfach eingerichtet und betrieben werden kann, wurde dieses Tool entwickelt. Denn nur so ist es möglich, dass die gestellten Anforderungen an die Docker-Umgebung erfüllt werden können (wechseln zwischen Swarm und Standalone Mode, Secret Store Verwendung im Swarm Mode).

Tab. 4 Zielerreichungstabelle der zu entwickelnden Komponenten

5.5.2 Ausblick für die Weiterentwicklung

5.5.2.1 Komponentenintegration in das extern entwickelte Hacking-Lab

Während dieser Bachelorarbeit hat die Security Competence GmbH das Hacking-Lab 2.0 parallel entwickelt. Nun gilt es Komponenten, wie beispielsweise den Challenge-Editor oder den Challenge-Translator, in dieses zu integrieren.

5.5.2.2 Mandantenfähigkeit

Vom Kunden kam der Wunsch, dass die Applikation mandantenfähig ist. Dies wurde so im Datenmodel umgesetzt. Somit kann dieses ohne grundlegende Anpassungen einfach umgesetzt werden.

5.5.2.3 Belohnungssystem

Damit die Motivation der Benutzer zur Challenge-Erstellung und Übersetzung gesteigert werden kann, muss das, in der Studienarbeit konzipierte, Belohnungskonzept implementiert werden. So sollen Benutzer für ihre Arbeit belohnt werden, indem sie beispielsweise weitere Challenges zur Verfügung gestellt bekommen.

5.5.2.4 Webapplikation zum Lösen von Challenges

Um das Hacking-Lab vollumfänglich nutzen zu können, muss eine Webapplikation entworfen werden, welche das Lösen von Challenges ermöglicht. Diese Software gehört jedoch nicht zum Lieferumfang der Bachelorarbeit, da diese den zeitlichen Rahmen erheblich gesprengt hätte.

5.5.2.5 Ressourcenserver

Gewisse Challenges benötigen zum Lösen zusätzliche Ressourcen. Diese können beispielsweise in Form von Docker-Containern oder virtuellen Maschinen zur Verfügung gestellt werden. Der Server, welche diese Ressourcen instanziiert und allfällige benutzerabhängige Challenge-Lösungen (Goldnugget) generiert, müsste noch entwickelt werden.

5.5.2.6 LDAP-Benutzer mit POSIX-Attributen erweitern

Der LDAP-Server sollte erweitert werden, damit die Benutzerobjekte mit POSIX-Attributen ausgestattet sind. So wird es möglich, dass diese Objekte auch für die Linux-PAM Authentifizierung verwendet werden können, welche in den Docker-Container des Ressourcenservers verwendet werden. So wird es möglich, dass die Hacking-Lab Benutzer ihr persönliches Login verwenden, um bei einem Ressourcen Container anzumelden. Diese Implementation war, aufgrund von Zeitknappheit und Ressourcenmangel, nicht Ziel dieser Bachelorarbeit. Da Keycloak eine solche Umsetzung der POSIX-Attribute nicht unterstützt, müsste eine Logik entwickelt werden, die die Benutzerobjekte nach der Erstellung durch Keycloak erweitert.

6 SW-Engineering Anhänge

6.1 Anforderungsspezifikation & Domänenanalyse

6.1.1 Einführung

Das vorliegende Dokument umfasst die analytischen Aspekte der Anforderungen an das zu entwickelnde Produkt, die anhand der Aufgabenstellung und kontinuierlichen Meetings definiert wurden. Mittels einer allgemeinen Beschreibung des HL-CAS wird ein Überblick über die Applikation verschafft und dazu die einzelnen Use Cases, (nicht) funktionale Anforderungen, sowie die Domäne analysiert und beschrieben. Die beschreibenden Texte werden nach Möglichkeit zur einfacheren Verständlichkeit mit Diagrammen ergänzt.

6.1.2 Allgemeine Beschreibung

6.1.2.1 Auftrag

Mit der zunehmenden Digitalisierung und der wachsenden Anzahl von Hackerangriffen wächst der Anspruch für eine adäquate Security Ausbildung von angehenden Cyber Security Spezialisten. Neben der Vermittlung von Theorie wird dem praktischen Üben ein hoher Stellenwert beigemessen. Zu diesem Zweck stellt das Hacking-Lab seit dem Jahre 2007 eine Onlineplattform mit sogenannten Security Challenges bereit. Das Hacking-Lab ist sehr erfolgreich mit der Bereitstellung von Übungen, ist aber vom Look & Feel ein wenig in die Jahre gekommen. Ein weiterer Nachteil der bestehenden Lösung ist, dass das Erfassen von neuen Übungen und Challenges ausschliesslich den Betreibern von Hacking-Lab möglich ist.

Mit dieser Bachelorarbeit soll ein Erfassungs- und Verwaltungssystem für Übungen entwickelt werden, welches im neuen Hacking-Lab 2.0 produktiv eingesetzt werden soll. Das Grundkonzept wurde bereits in der vorhergehenden Studienarbeit entwickelt und im dort erstellten Bericht dokumentiert.

6.1.3 Aktoren

Das Erfassungs- und Verwaltungssystem für Challenges (HL-CAS) besitzt die folgenden Aktoren, welche die nachstehenden Interessen vertreten.

- *Challenge-Editor*: Dieser hat das Interesse das HL-CAS mit einer geeigneten User Experience und einer sauberen Usability zu benutzen. Das System sollte ihm die Erfassung oder Modifikation einer Challenge so einfach und schnell wie möglich machen, damit keine Zeiteinbussen hingenommen werden müssen. Durch einen kontinuierlichen Datentransfer zur zugrundeliegenden Datenbank sollen keine Daten verloren gehen, wenn ein Verbindungsfehler oder ähnliche Probleme auftreten.
- *Challenge-Translator*: Dieser hat das Interesse das HL-CAS mit einer geeigneten User Experience und einer sauberen Usability zu benutzen. Das System sollte ihm die Neuerstellung einer Übersetzung oder die Modifikation einer Bestehenden so einfach und schnell wie möglich machen, damit keine Zeiteinbussen hingenommen werden müssen. Damit die Übersetzungen

nicht zwingend von Hand erstellt werden müssen, wird durch ein API die Möglichkeit gegeben, ganze Challenges automatisch zu übersetzen.

- *CCS*: Das Consumer System hat das Interesse, dass die Challenges durch ein einfach zu benutzendes RESTful API angefragt werden können und in einem geeigneten Format zur Verfügung gestellt werden. Das Standard Format für diesen Anwendungsfall wurde als JSON definiert.
- *Export/Import*: Der Job für den Export / Import von Challenges hat das Interesse, dass die Challenges durch ein einfach zu handhabendes RESTful API bezogen und wieder geschickt werden können. Das Format für den Datentransfer wurde als JSON definiert, welches alle pro Challenge relevanten Daten in einem Request beinhaltet. Die Struktur wurde in Zusammenarbeit mit dem Auftraggeber festgelegt.

6.1.4 Use Cases

Die folgende Illustration beinhaltet die Use Cases des CAS-Client Systems. Die Use Cases sind mit ihren jeweiligen Aktoren verknüpft. Die beiden Aktoren CAS_Admin und CCS_Admin besitzen alle Use Cases des jeweiligen Systems, da sie Rechte für alle dieser Operationen besitzen.

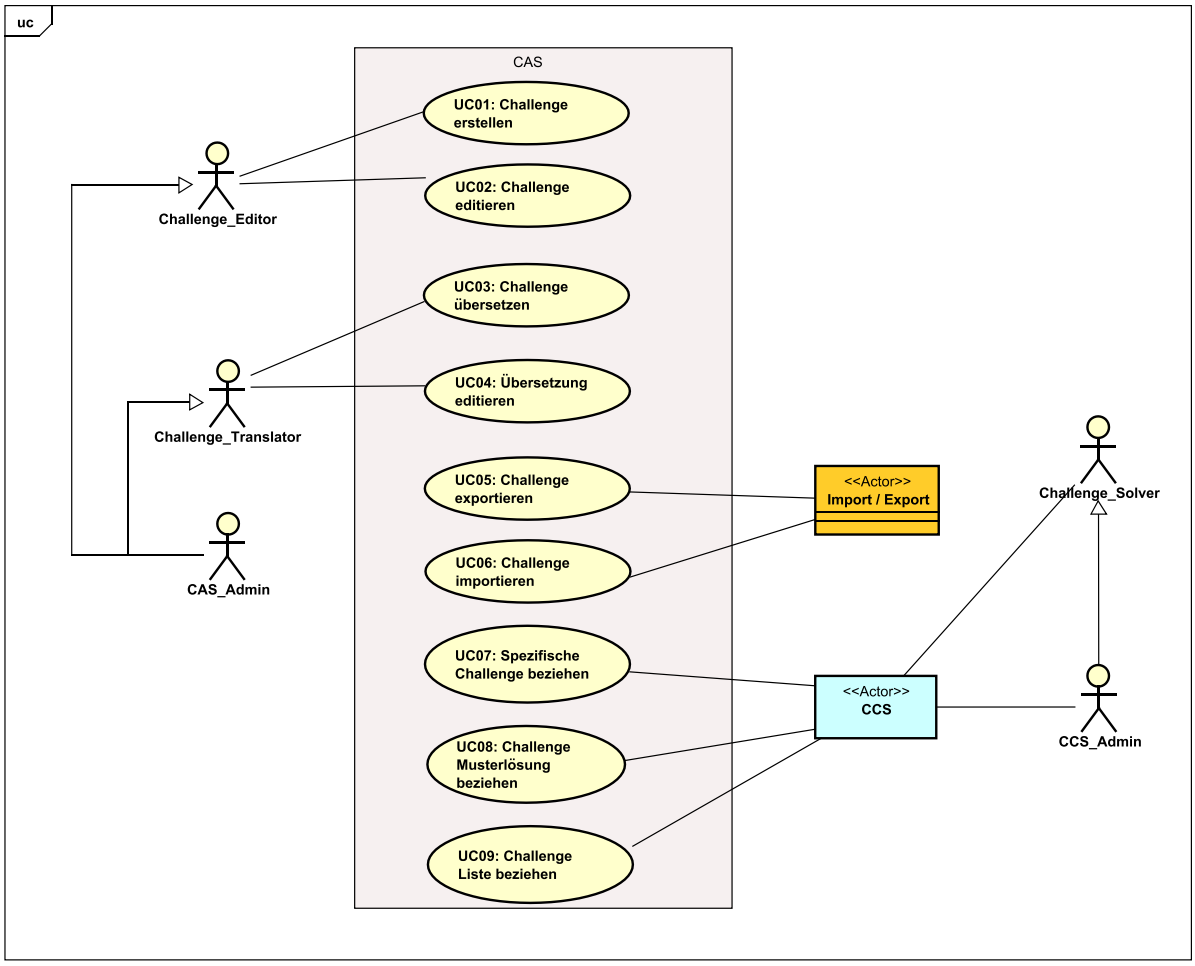


Abb. 34 Use Case Diagramm

6.1.4.1 UC00: Autorisation

In den nachfolgenden Unterkapiteln werden die einzelnen Use Cases kurz beschrieben und anhand eines Sequenzdiagrammes visualisiert. Um Redundanzen zu vermeiden wird der bereits im Kapitel 5.4.4.9 beschriebene Authentifizierungsprozess hier vereinfacht dargestellt und bei den danach folgenden Use Cases als bereits durchgeführt vorausgesetzt.

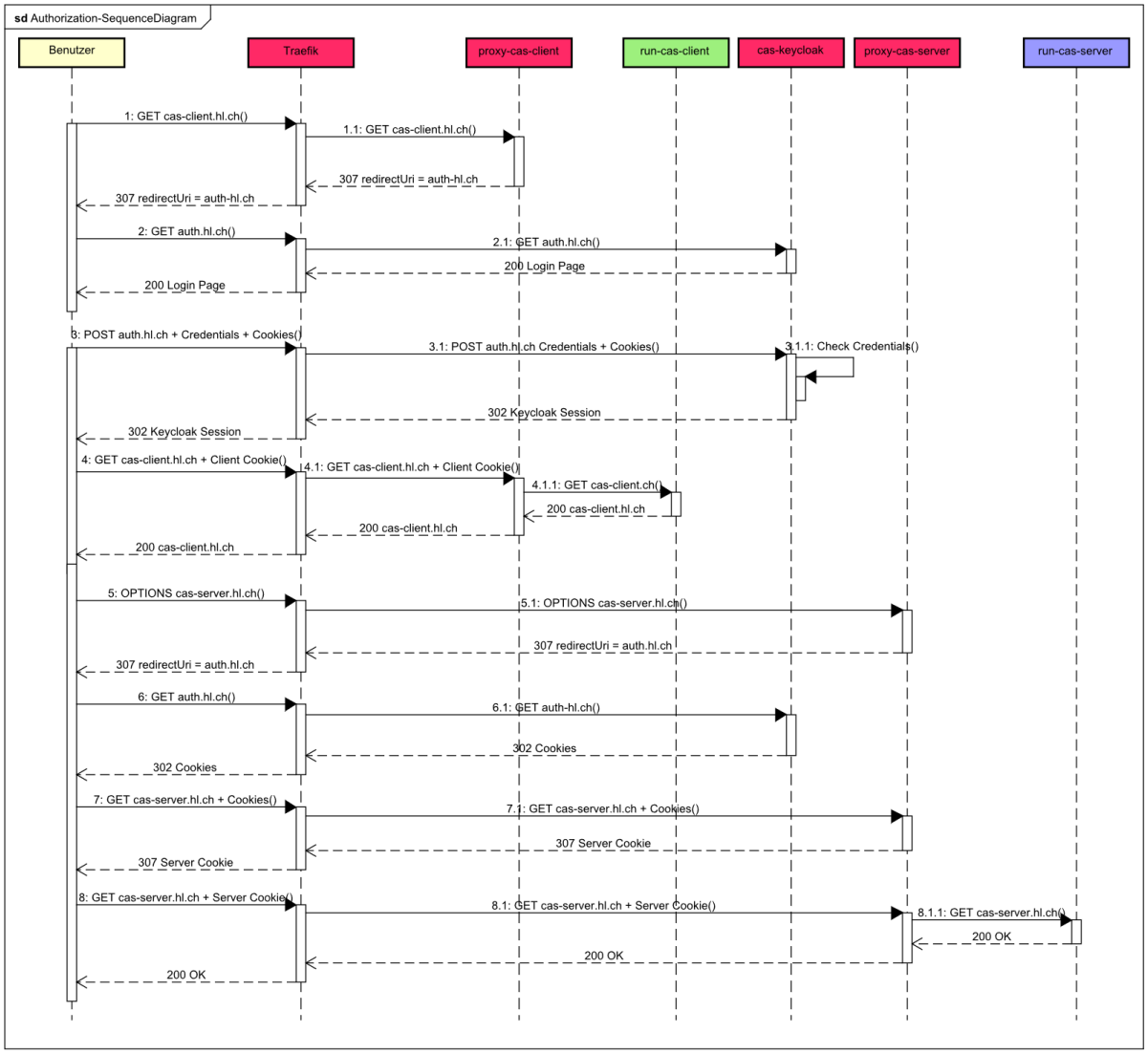


Abb. 35 Autorisierungsprozess Sequenzdiagramm

1. Der Benutzer navigiert im Browser zum CAS-Client System
2. Der Reverse Proxy für den CAS-Client leitet den Browser aufgrund des fehlenden Client-Cookies zum CAS-Keycloak weiter
3. Der Browser fragt CAS-Keycloak zum Anzeigen der Login Page an
4. CAS-Keycloak sendet die Login Page zurück
5. Der Benutzer gibt seinen Namen und das zugehörige Passwort ein und bestätigt die Eingabe
6. Die User Credentials werden vom Browser zum Keycloak geschickt und dort geprüft
7. CAS-Keycloak stellt eine Session mit dem Benutzer aus und sendet diese und das Client Cookie zurück.
8. Der Browser sendet die Anfrage an das CAS-Client System erneut.
9. Das CAS-Client System sendet die HTML-Page des CAS-Clients zurück

10. Der Browser führt das JavaScript der HTML-Page aus und sendet dem CAS-Server einen OPTIONS Request für die Aushandlung der CORS Header (da andere Domain)
11. Der Reverse Proxy des Servers leitet den Browser aufgrund des fehlenden Server-Cookies zum CAS-Keycloak weiter
12. CAS-Keycloak sieht die bestehende Session, stellt das Server-Cookie aus und sendet dieses zurück.
13. Der Browser sendet die Anfrage an den CAS-Server erneut und bekommt Antwort vom Server

6.1.4.2 UC01: Challenge erstellen

Ein Benutzer mit der Challenge Editor Rolle kann eine neue Challenge mit dem CAS-Client erfassen. Die Challenge wird kontinuierlich über das RESTful API auf der Datenbank gespeichert.

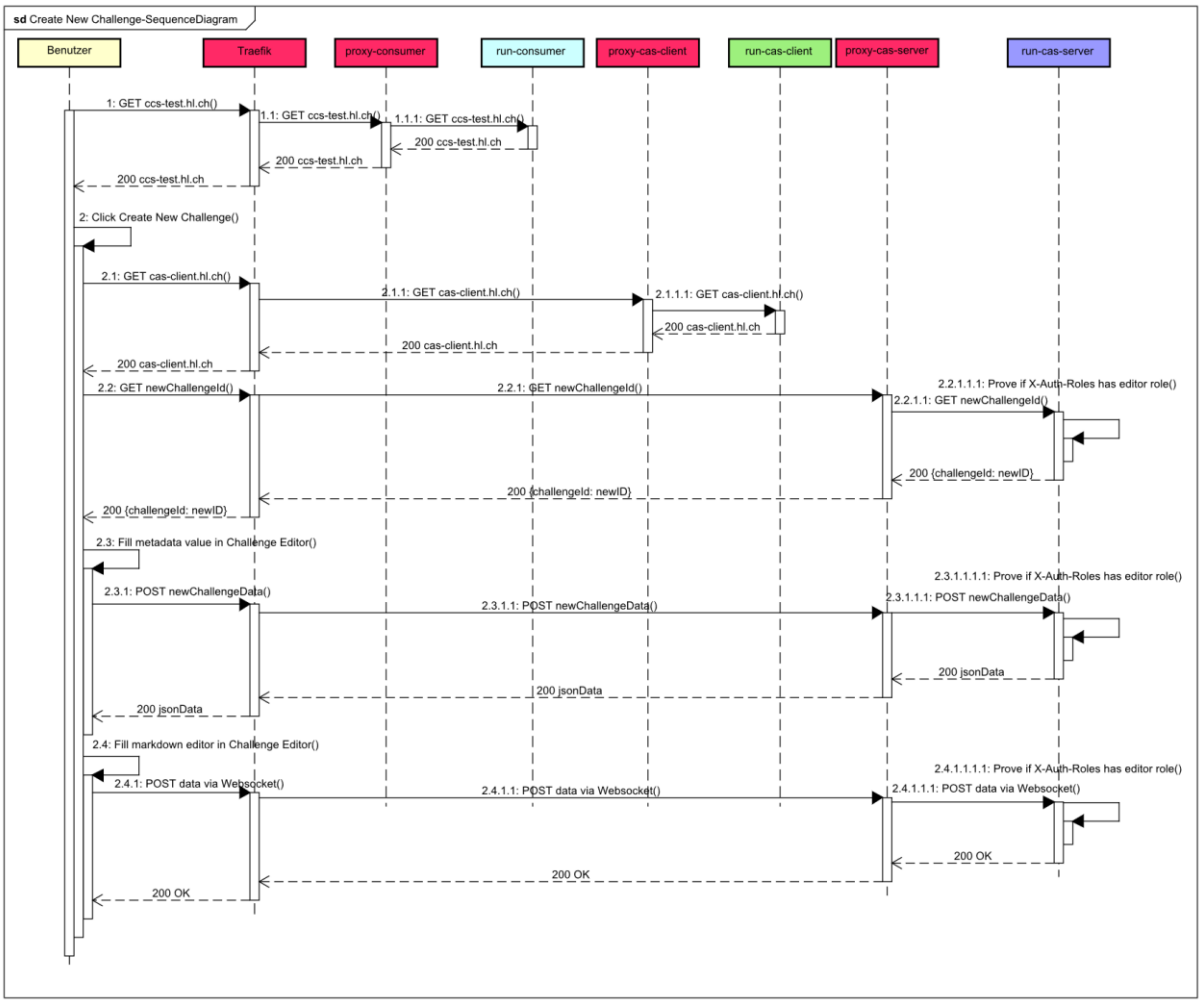


Abb. 36 Challenge Erstellung Sequenzdiagramm

1. Der Benutzer navigiert im Browser zum Consumer System
2. Das Consumer System gibt dem Benutzer die HTML-Page mit der Challenge Übersicht zurück
3. Der Benutzer klickt auf «Create New Challenge»
4. Das Consumer System leitet den Benutzer weiter zum CAS-Client System
5. CAS-Client System gibt dem Benutzer die HTML-Page mit dem Challenge-Editor zurück
6. Der Benutzer befüllt ein Metadatenfeld innerhalb des Challenge-Editors und bestätigt die Eingabe (wird einige Male durchgeführt)

7. Der CAS-Server prüft die Rolle des Benutzers und die Eingabe und speichert den Wert in die Datenbank
8. Der Benutzer befüllt ein Markdown-Feld innerhalb des Challenge-Editors. Es werden anhand eines Websockets kontinuierlich Daten an den Server geschickt
9. Der CAS-Server überprüft die Rolle des Benutzers, sowie die Eingabe und speichert den Wert in die Datenbank

6.1.4.3 UC02: Challenge editieren

Ein Benutzer mit der Challenge Editor Rolle kann eine bestehende Challenge aus einer Liste wählen und mit dem Challenge-Editor des CAS-Clients bearbeiten. Die getätigten Änderungen werden kontinuierlich über das RESTful API auf der Datenbank gespeichert.

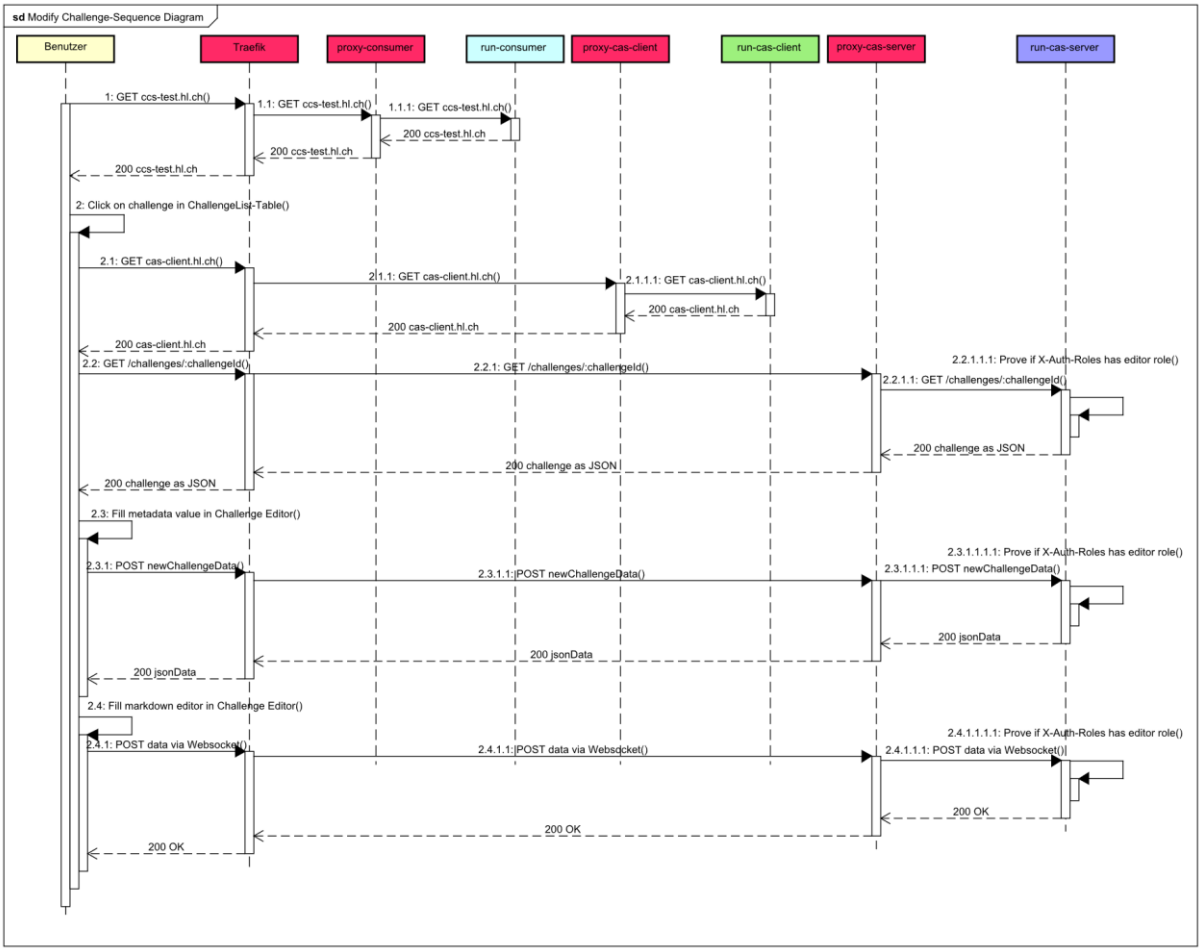


Abb. 37 Challenge editieren Sequenzdiagramm

1. Der Benutzer navigiert im Browser zum Consumer System
2. Das Consumer System gibt dem Benutzer die HTML-Page mit der Challenge Übersicht zurück
3. Der Benutzer wählt eine bestehende Challenge aus der Übersicht
4. Das Consumer System leitet den Benutzer weiter zum CAS-Client System
5. CAS-Client System gibt dem Benutzer die HTML-Page mit dem befüllten Challenge-Editor zurück
6. Der Benutzer verändert ein Metadatenfeld innerhalb des Challenge-Editors und bestätigt die Eingabe (wird einige Male durchgeführt)
7. Der CAS-Server prüft die Rolle des Benutzers und die Eingabe und speichert den Wert in die Datenbank

8. Der Benutzer verändert ein Markdown-Feld innerhalb des Challenge-Editors. Es werden anhand eines Websockets kontinuierlich Daten an den Server geschickt
9. Der CAS-Server überprüft die Rolle des Benutzers, sowie Eingabe und speichert den Wert in die Datenbank

6.1.4.4 UC03: Challenge übersetzen

Ein Benutzer mit der Challenge Translator-Rolle kann eine bestehende Challenge von Englisch in eine andere Sprache übersetzen. Er kann über das Deepl-API eine ganze Challenge automatisch oder im Modifikationsmodus von Hand übersetzen. Nach der Speicherung durch den Benutzer wird die neue Übersetzung in die Datenbank geschrieben.

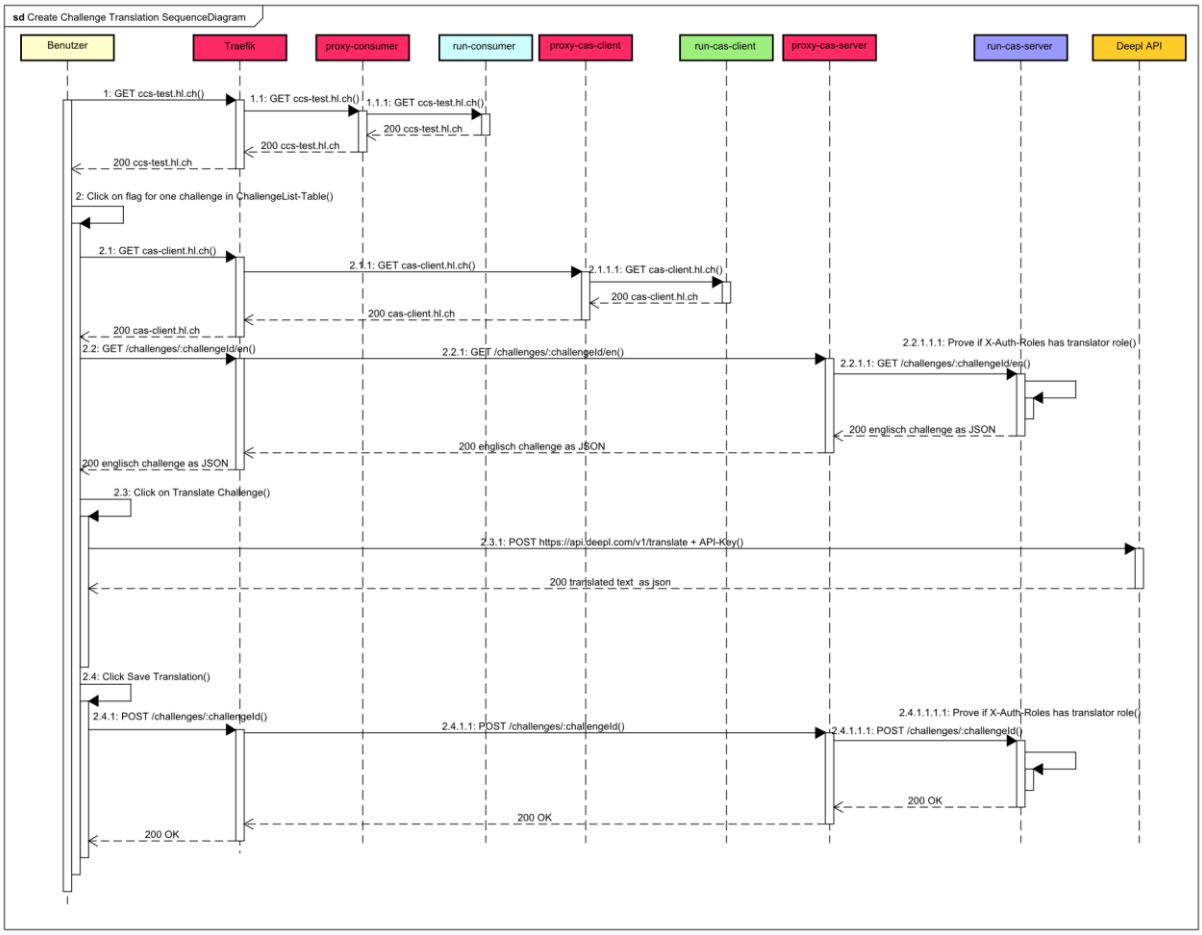


Abb. 38 Challenge übersetzen Sequenzdiagramm

1. Der Benutzer navigiert im Browser zum Consumer System
2. Das Consumer System gibt dem Benutzer die HTML-Page mit der Challenge Übersicht zurück
3. Der Benutzer wählt eine bestehende Challenge aus der Übersicht für die Übersetzung und klickt auf eine Flagge für die gewünschte Übersetzung
4. Das Consumer System leitet den Benutzer weiter zum CAS-Client System
5. Das CAS-Client System startet anhand der ausgewählten Challenge einen Request zum CAS-Server
6. Der CAS-Server prüft, ob der Benutzer die Rolle besitzt eine englische Challenge zu holen.
7. CAS-Client System gibt dem Benutzer die HTML-Page für die Erfassung einer Übersetzung für die englische Challenge
8. Der Benutzer klickt auf «Translate Challenge»

9. Der CAS-Client erstellt einen Request zum Deepl API, um die Challenge dort zu übersetzen
10. Das Deepl API sendet ein JSON mit der Übersetzung zurück
11. Der Benutzer klickt auf «Save Translation»
12. Der CAS-Client erstellt einen POST Request zum CAS-Server
13. Der CAS-Server überprüft die Rolle des Benutzers, und speichert die Übersetzung in der Datenbank

6.1.4.5 UC04: Übersetzung editieren

Ein Benutzer mit der Challenge Translator Rolle kann eine bestehende Übersetzung einer Challenge bearbeiten. Falls ein Teil einer Challenge bereits von Hand übersetzt wurde und das Deepl API verwendet wird, gibt der CAS-Client eine Meldung aus, dass die bestehende Änderung überschrieben wird. Der Benutzer kann die Meldung bestätigen oder zur Bearbeitung zurückkehren.

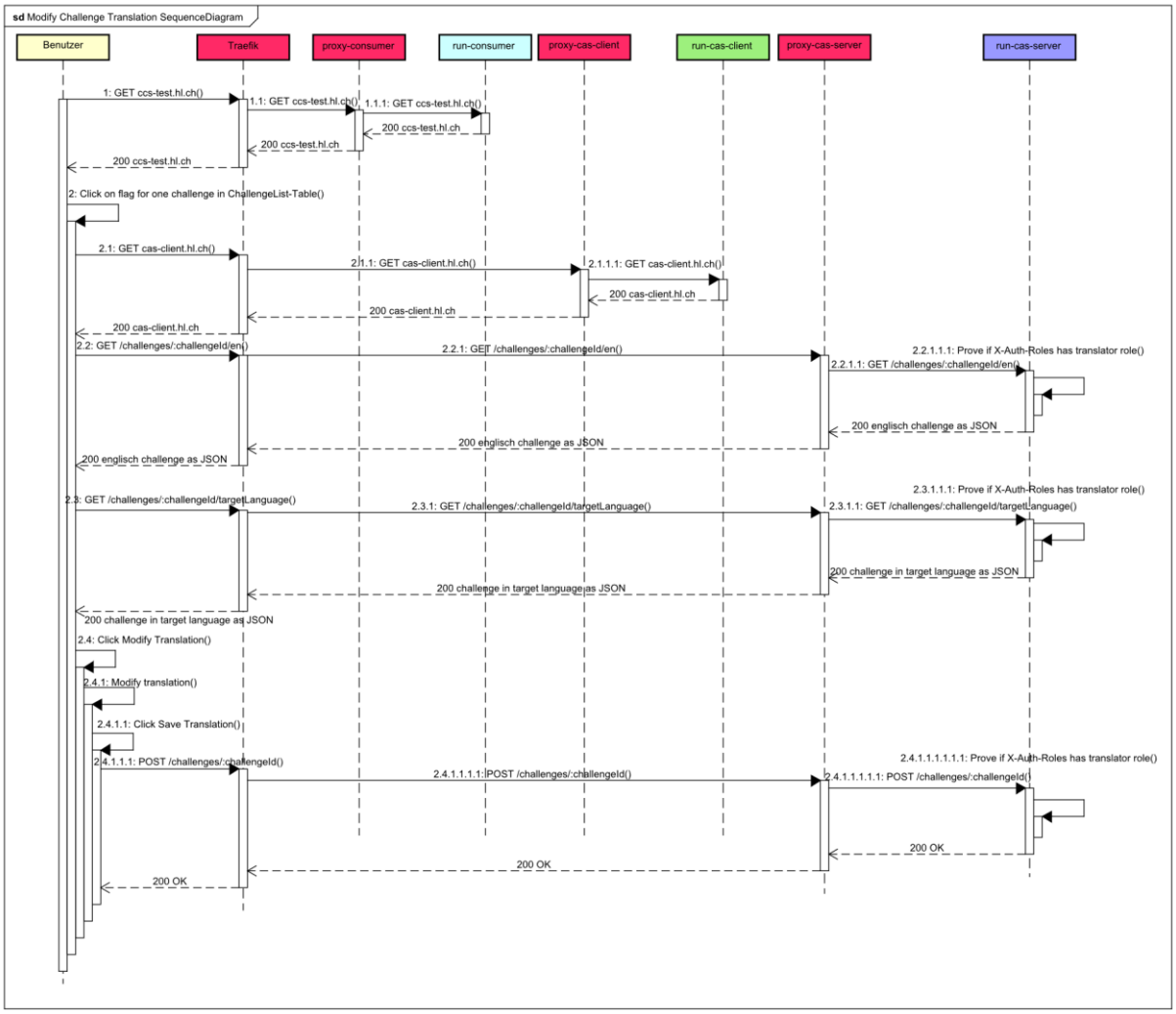


Abb. 39 Challenge-Übersetzung editieren Sequenzdiagramm

1. Der Benutzer navigiert im Browser zum Consumer System
2. Das Consumer System gibt dem Benutzer die HTML-Page mit der Challenge Übersicht zurück
3. Der Benutzer wählt eine bestehende Challenge aus der Übersicht für die Übersetzung und klickt auf eine Flagge für die gewünschte Übersetzung
4. Das Consumer System leitet den Benutzer weiter zum CAS-Client System

5. Das CAS-Client System startet anhand der ausgewählten Challenge einen Request zum CAS-Server
6. Der CAS-Server prüft, ob der Benutzer die Rechte besitzt eine englische Challenge zu holen
7. Das CAS-Client System startet anhand der ausgewählten Challenge und der Zielsprache einen Request zum CAS-Server
8. Der CAS-Server prüft, ob der Benutzer die Rolle besitzt eine Challenge mit der gewünschten Sprache zu holen.
9. CAS-Client System gibt dem Benutzer die HTML-Page für die Modifikation einer Übersetzung für die englische Challenge und die bereits übersetzte Challenge in der Zielsprache zurück
10. Der Benutzer klickt auf «Modify Translation»
11. Der Benutzer ändert eine Übersetzung
12. Der Benutzer klickt auf «Save Translation»
13. Der CAS-Client erstellt einen POST Request zum CAS-Server
14. Der CAS-Server überprüft die Rolle des Benutzers, und speichert die Übersetzung in der Datenbank

6.1.4.6 UC05: Alle Challenges exportieren

Der Export-Job kann über ein RESTful API alle Challenges exportieren. Das CAS bereitet die Challenges mit allen benötigten Daten in ein JSON-Array auf und sendet dem Job als Antwort das JSON zurück.

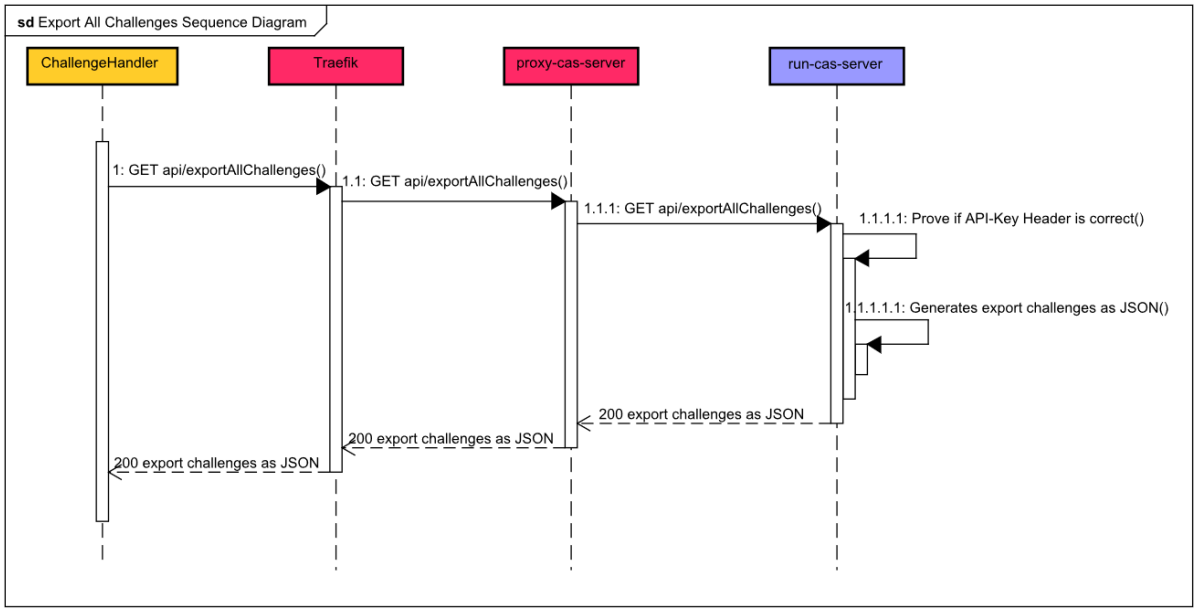


Abb. 40 alle Challenges exportieren Sequenzdiagramm

6.1.4.7 UC05: Eine spezifische Challenge exportieren

Der Export-Job kann über ein RESTful API eine spezifische Challenge exportieren. Das CAS bereitet die Challenge mit allen benötigten Daten in ein JSON auf und sendet dem Job als Antwort das JSON.

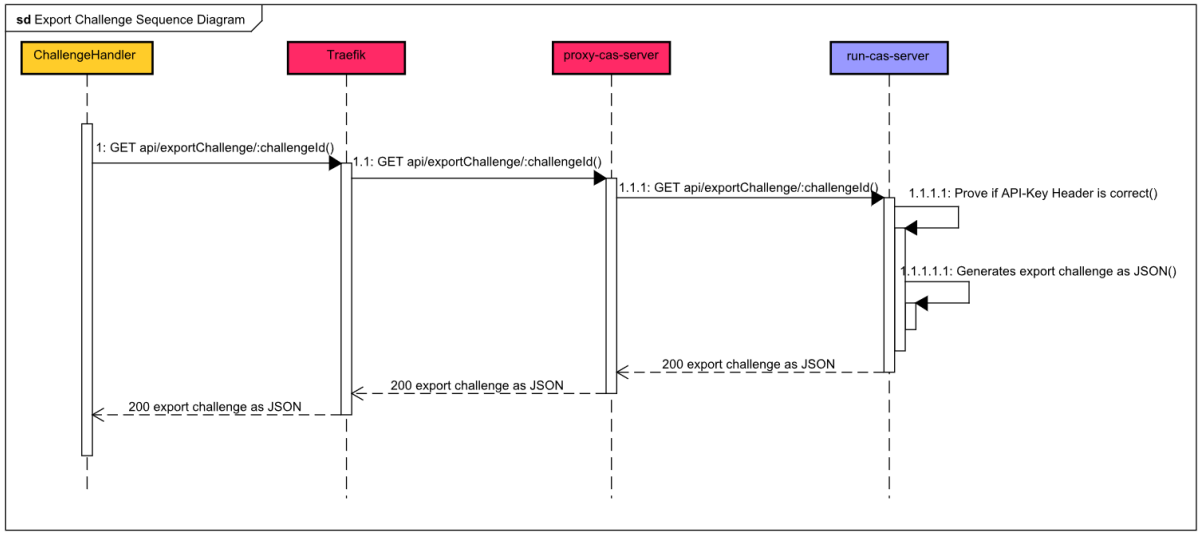


Abb. 41 spezifische Challenge exportieren Sequenzdiagramm

6.1.4.8 UC06: Challenge importieren

Der Import-Job kann über ein RESTful API eine spezifische Challenge in das CAS importieren. Das CAS wandelt die JSON Datei in eine Challenge um und überschreibt oder erstellt die Challenge auf der Datenbank.

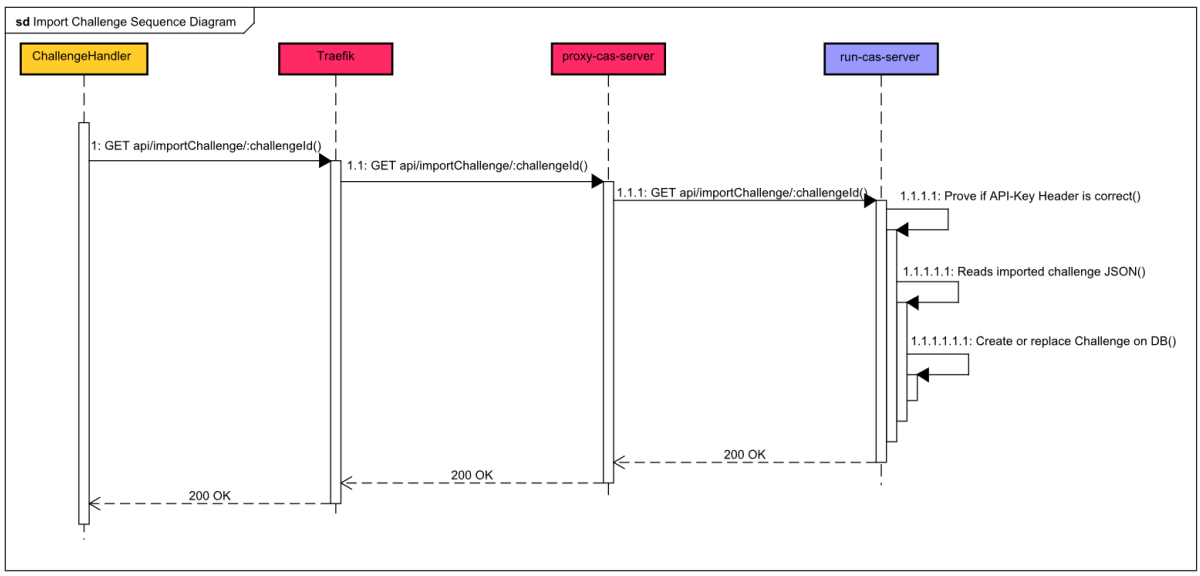


Abb. 42 Spezifische Challenge importieren Sequenzdiagramm

6.1.4.9 UC07: Spezifische Challenge beziehen

Das CCS kann über ein RESTful API eine spezifische Challenge anfragen. Das CAS generiert aus den vorhandenen Daten ein JSON-Objekt und sendet dem CCS als Antwort die Challenge als JSON.

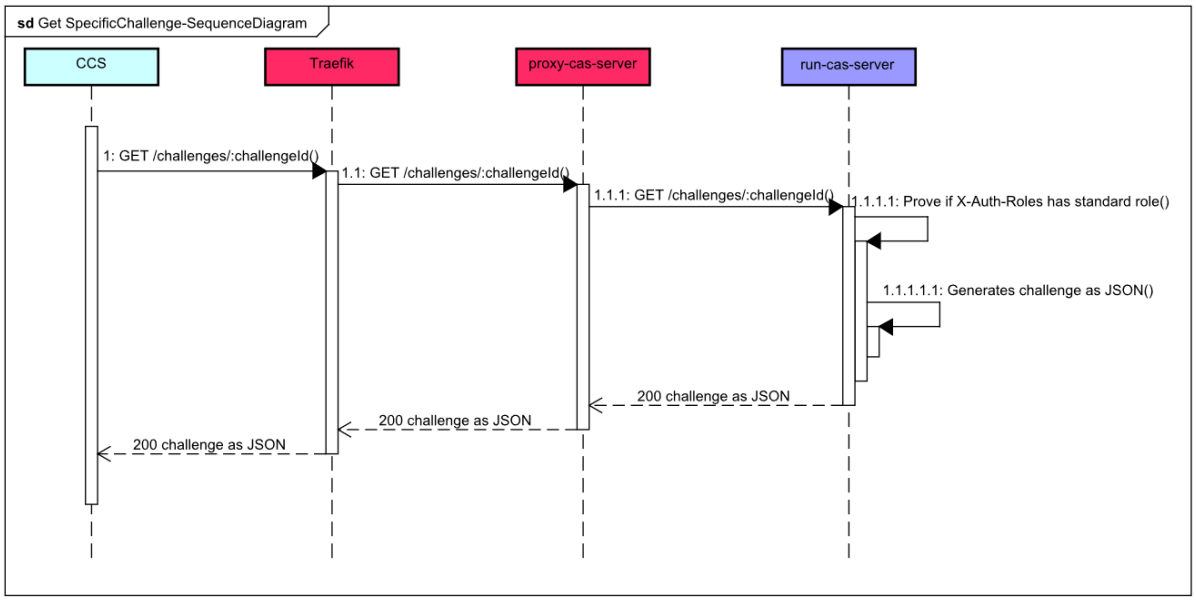


Abb. 43 spezifische Challenge beziehen Sequenzdiagramm

6.1.4.10 UC08: Challenge Musterlösung beziehen

Das CCS kann über ein RESTful API die Musterlösung einer spezifischen Challenge anfragen. Das CAS schickt die vorhandene Musterlösung als JSON Objekt dem CCS als Antwort zurück.

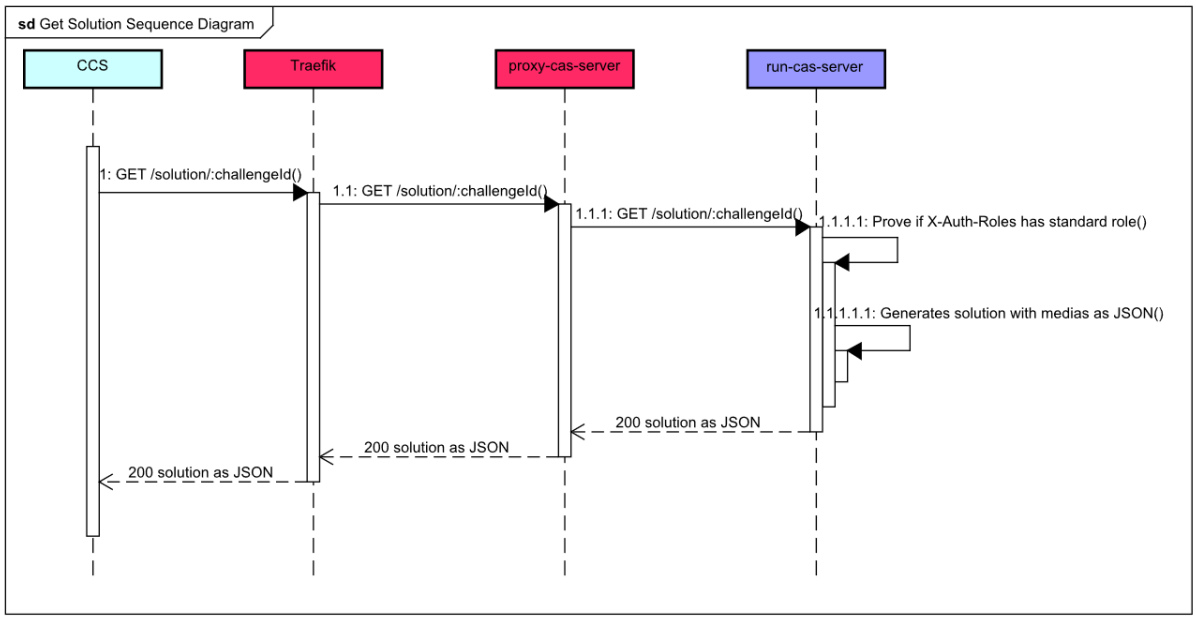


Abb. 44 Challenge Musterlösung beziehen Sequenzdiagramm

6.1.4.11 UC09: Challenge Liste beziehen

Das CCS kann über ein RESTful API alle auf dem CAS vorhandenen Challenges abfragen. Das CAS generiert eine Liste aus den benötigten Metadaten und schickt dem CCS diese als JSON-Array in der Antwort zurück.

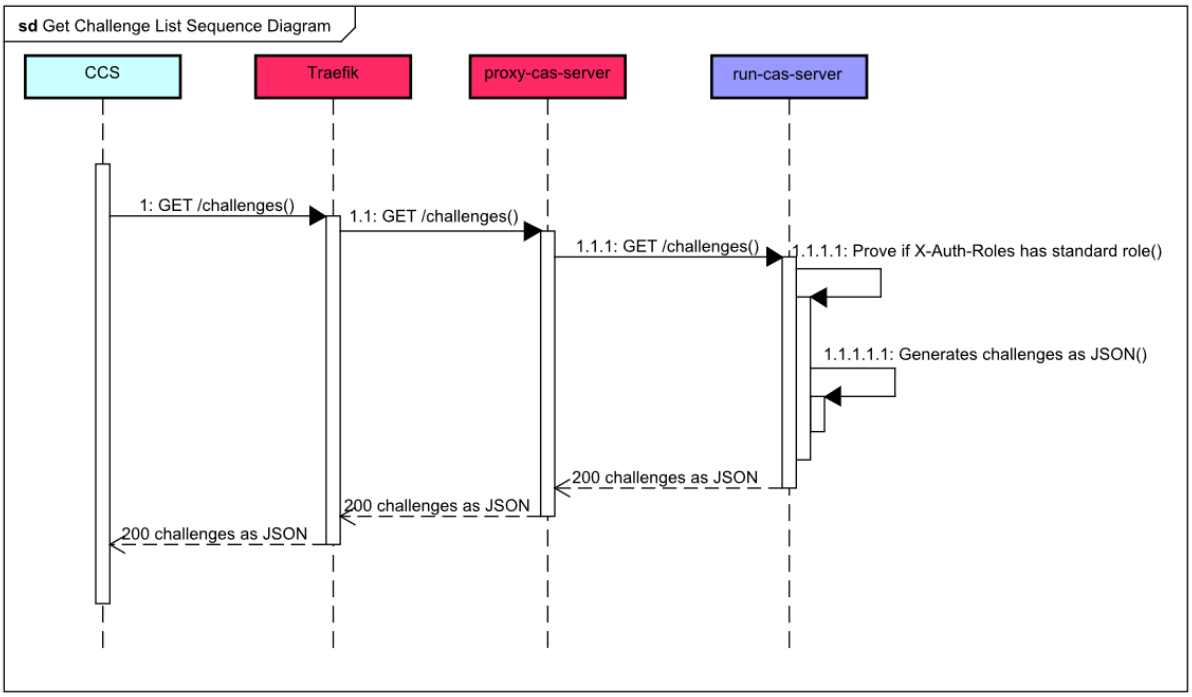


Abb. 45 Challenge Liste beziehen Sequenzdiagramm

6.1.5 Nicht funktionale Anforderungen

6.1.5.1 Sicherheit

Da den Hacking-Lab Benutzern Sicherheit im IT-Bereich vermittelt werden soll, spielt Sicherheit in der Implementierung und der gesamten Infrastruktur eine wichtige Rolle. Es muss darauf geachtet werden, dass keine Kommunikation von ausserhalb mitgelesen werden kann. Dazu muss die gesamte Kommunikation des Systems mit der Aussenwelt verschlüsselt werden.

Damit keine entscheidenden Daten von nicht berechtigten Nutzern gestohlen werden können, muss sichergestellt werden, dass es ausschliesslich authentifizierten Benutzern möglich ist, auf das CAS System zu zugreifen. Weiter muss das gesamte RESTful API durch eine Autorisierungsschicht abgesichert werden, damit keine Daten von nicht berechtigten Benutzern angeschaut oder verändert werden können.

6.1.5.2 Wiederherstellbarkeit

Da das korrekte Erfassen einer Challenge viel Zeit benötigen kann, muss sichergestellt werden, dass nach einem Absturz der Anwendung oder beim Beenden der Browsersession keine Daten verloren gehen und der Erfassungsprozess weitergeführt werden kann. In einer ersten Version wird die aktuelle Browsersession jedoch nicht wiederhergestellt werden können.

6.1.5.3 Fehlertoleranz

Da das neue Hacking-Lab System von mehreren Benutzern gleichzeitig benutzt werden können muss, muss sichergestellt werden, dass bei einem Fehler in der Anwendung nur die aktuelle Operation des verursachenden Benutzers abgebrochen wird und somit andere allfällige Teilnehmer nicht davon betroffen sind. Es ist also massgeblich für den Erfolg des Systems entscheidend, dass Fehlersituationen sauber behandelt werden und nicht das gesamte System zum Absturz gebracht werden kann.

6.1.5.4 Verfügbarkeit

Da es sich beim neuen CAS-System um ein Kernsystem handelt und bei Nichtverfügbarkeit finanzielle und Imageschäden entstehen können, muss die Applikation hochverfügbar sein. Konkret bedeutet das eine Verfügbarkeit von mindestens 99 % im Jahr (3.65 Tage Downtime sind noch im akzeptablen Rahmen).

6.1.5.5 Skalierbarkeit

Die im neuen Hacking-Lab System je nach Installation oder für den verteilten Betrieb die Lasten sehr unterschiedlich ausfallen können, muss die gesamte Infrastruktur so ausgelegt werden, dass sie ohne grossen Aufwand automatisch skaliert und so die unterschiedlichen Lasten kein Problem für den Betrieb darstellen.

6.1.5.6 Deployment

Das neue Hacking-Lab System muss so einfach wie möglich bei einem neuen Kunden in Betrieb genommen werden können. Es muss möglich sein, dass mit einigen wenigen Konfigurationsanpassungen eine neue Umgebung aufgesetzt und in Betrieb genommen werden kann.

6.1.5.7 Wartbarkeit

Da das neue Hacking-Lab System an verschiedenste Kunden verkauft wird, muss die Infrastruktur in einfachster Weise wartbar sein. Das bedeutet, dass in höchstens zwei Stunden eine Komponente ausgetauscht und der Betrieb wieder gewährleistet werden kann.

6.1.5.8 Integrierbarkeit

Da das neue Hacking-Lab in unterschiedlichen Umgebungen integriert werden können muss, muss in der Implementierung der Infrastruktur darauf geachtet werden, dass keine Abhängigkeiten nach Aussen bestehen, sodass das neue System auf unterschiedlichen Betriebssystemen und in unterschiedlichen Netzwerken ohne Probleme zum Laufen gebracht werden kann.

6.1.5.9 Fazit

Um der «nicht funktionalen Anforderung» für das Deployment gerecht werden zu können, bietet sich die Docker Container Technologie an. Damit wird es einfach möglich eine komplette Umgebung mit wenigen Befehlen hochzufahren und anhand einer zentralen .env-Konfigurationsdatei für das jeweilige System zu konfigurieren.

Auch für die Wartbarkeit und Integrierbarkeit ist Docker geeignet, da eine fehlerhafte Komponente einfach ausgetauscht und aufgrund der Architektur einer Containerlösung auf alle Betriebssysteme zum Laufen gebracht werden kann.

Auch eine erhöhte Skalierbarkeit kann mit Docker erreicht werden, da der Docker Swarm Mode genau für diese Problematik entwickelt wurde.

Aus den genannten Punkten wird ersichtlich, dass die Infrastruktur des neuen Hacking-Labs auf Docker basieren muss.

6.1.6 Domain Model

Im folgenden Diagramm wird eine Übersicht über das in der Applikation verwendete Datenmodell vermittelt. Es wurde ausgehend aus dem in der vorgelagerten Studienarbeit entwickelten Datenmodell erweitert. Damit die Erkenntnisse der Studienarbeit nicht erneut erklärt werden, sei hier auf diese verwiesen. [37]

Als entscheidende neue Anforderung an das Modell, wurde gewünscht die Medien-Dateien wie Bilder usw. pro Sprache persistieren zu können. Aus diesem Grund war die wichtigste Anpassung, neben einfachen Datentyp Änderungen und einigen erweiternden Tabellenspalten, die Ergänzung einer Struktur für die effiziente Persistierung von Mediendateien in der Datenbank.

Damit Bilder und Medien pro Sprache gespeichert werden können, bedarf es einer Referenz-Tabelle (MediaReference), in der die URL auf die entsprechende Datei hinterlegt werden kann. Pro MediaReference werden mehrere Medien-Objekte mit ihrem Base64 codierten Inhalt pro Sprache persistiert. Damit eine effiziente Suche möglich wird, wurde die Tabelle MarkdownMediaReference erdacht, in der pro Markdown-Datei, wo die Medien-Dateien im Text referenziert werden, die Verbindung auf die verwendeten Medien-Objekte hinterlegt werden. Dadurch wird es möglich anhand der ID der Markdown-Datei die darin verwendeten Medien zu suchen und innerhalb eines einzigen Requests zurückzugeben.

Eine weitere neue Anforderung war die zusätzliche Speicherung eines Wertes, um eine Übersetzung zu kennzeichnen, ob sie von einem Benutzer oder nur über das automatische API erstellt wurde. Dadurch kann auf dem Client während der Bearbeitung einer Übersetzung angezeigt werden, wenn eine benutzerdefinierte Übersetzung durch eine Übersetzung des Deepl APIs ersetzt werden soll und eine entsprechende Warnung angezeigt werden.

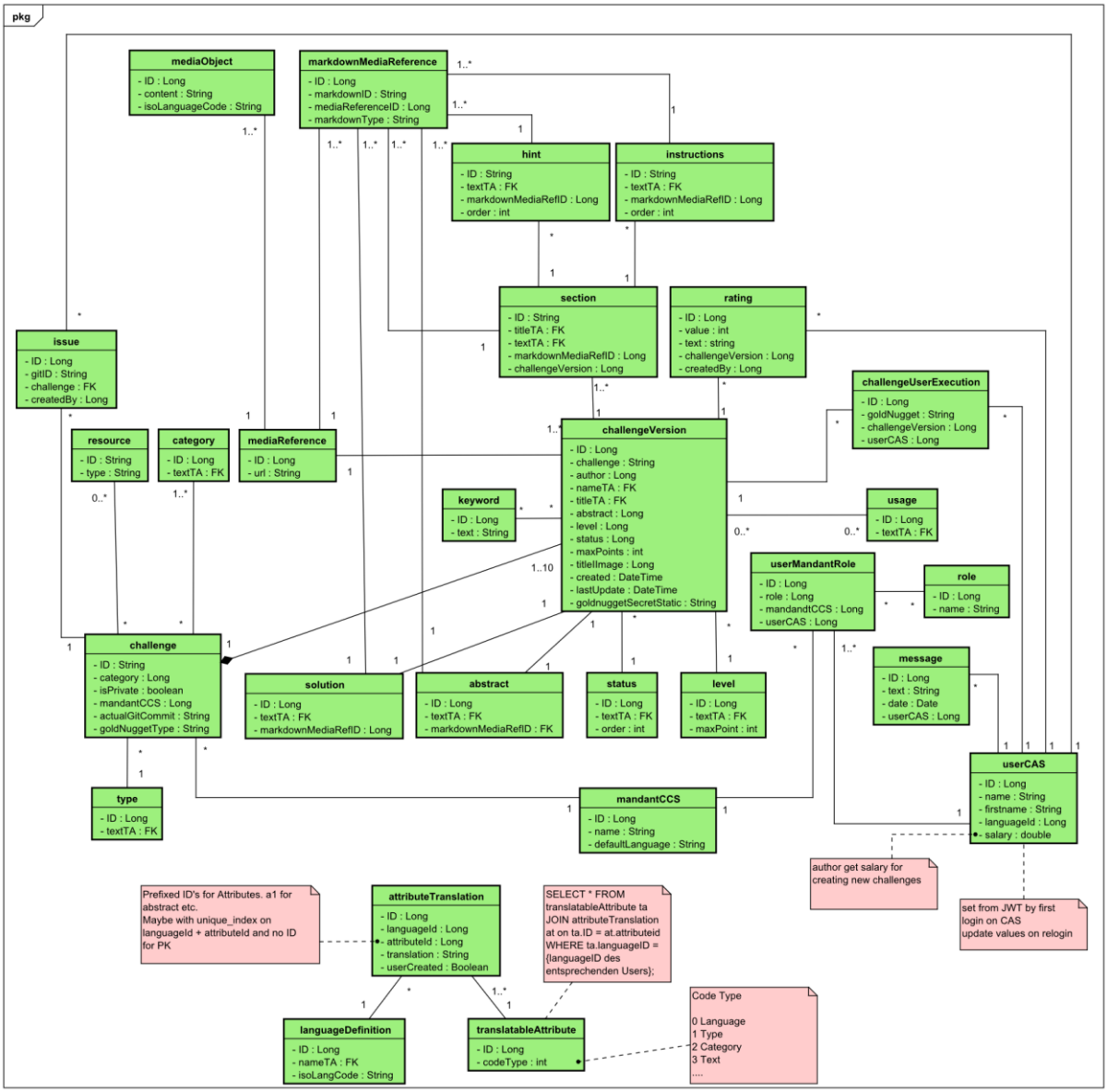


Abb. 46 Domain Model

6.2 Software Architektur Dokument

6.2.1 Einführung

Das folgende Kapitel gibt Aufschluss über die implementierte Architektur der Softwarekomponenten. Da bereits in der Studienarbeit [37] auf die einzelnen Komponenten eingegangen und diese beschrieben wurden, werden in diesem Dokument nur die wichtigsten Neuerungen und Anpassungen beschrieben. Somit werden die verwendeten Repositories und deren Interfaces nicht mehr beschrieben, da die Grundidee deren Implementierung, bereits während der Studienarbeit dargelegt wurden.

Zu Beginn wird die verwendete Architektur beschrieben. Darauffolgend wird die logische Architektur erklärt, sowie das umgesetzte User Interface visualisiert.

6.2.2 Architektur Beschreibung

Die eigentliche Server-Applikation stützt sich auf eine MVC Architektur. Damit ein sauberes Unit Testing möglich wird, sowie die Codeduplizität minimiert werden kann, wird diese Grundstruktur noch weiter unterteilt.

6.2.2.1 Server

Im Folgenden wird die Unterteilung des Servers beschrieben. Es soll nicht das Ziel sein jede verwendete Klasse zu beschreiben, sondern ein grober Überblick über die verwendete Struktur des Backends zu geben. Weiter muss beachtet werden, dass viele Teile bereits in der Studienarbeit beschrieben wurden und hier nur die neu Implementierten Komponenten Platz finden.

6.2.2.1.1 Authenticator

Die Authenticators stellen die initiale Zugriffsschicht dar. Sie definieren eine Autorisierungsschicht, welche anhand der Userrolle erkennt, ob die Verwendung der REST URI erlaubt ist oder nicht. Die Userrolle wird über den Reverse Proxy in die Request Headers (X-Auth-Roles) eingefügt und zum Server übertragen. Falls ein User nicht für eine Aktion berechtigt ist, wird vom Server der Status 401 zurückgemeldet und dem User die entsprechende Meldung angezeigt.

Es können beliebige Authenticators implementiert werden, indem die vom Play-Framework zur Verfügung gestellte Security.Authenticator Klasse als Superklasse verwendet wird. Aus der grundsätzlich Funktionalitäten des Clients, Übersetzen und Erstellen von Challenges, können zwei solche Authenticators abgeleitet werden. Da jedoch der normale Benutzer der Challenges lösen bzw. aus einer Liste auswählen möchte weitere API-Funktionen des Servers nutzen möchte, muss ein weiterer Authenticator implementiert werden, der die Standardfunktionalität, ohne dass etwas auf dem Server geändert werden kann, gegen unerlaubten Zugriff absichert. Zusätzlich sind API-Funktionen vorhanden, die nicht direkt von einem User, sondern von einer anderen Applikation aufgerufen werden. Da hier kein expliziter User vorhanden ist, muss ein weiterer Authenticator implementiert werden, der nicht anhand einer Rolle, sondern anhand eines API-Keys den Zugriff autorisiert. Aus diesen Überlegungen werden die vier ActionAuthenticators **EditorAuthenticator**, **StandardActionAuthenticator**, sowie **TranslatorActionAuthenticator** und **ImportActionAuthenticator** abgeleitet und implementiert.

StandardActionAuthenticator: Durch den StandardActionAuthenticator werden die API-Methoden gesichert, die für die Anzeige der Challenges verwendet werden. Mit diesen kann kein Wert auf dem Server geändert werden und benötigen deshalb auch keine spezielle Rolle. Standardmässig wird die Rolle «hl_user» (wird für jeden Benutzer gesetzt) oder «hl_admin» zugelassen. Alternativ kann der Standardwert durch eine Umgebungsvariable innerhalb der Docker-Infrastruktur geändert werden.

EditorActionAuthenticator: Durch den EditorActionAuthenticator werden die API-Methoden gesichert, die für die Erstellung und Mutation der Challenges verwendet werden. Standardmässig wird die Rolle «hl_editor» oder «hl_admin» zugelassen. Alternativ kann der Standardwert durch eine Umgebungsvariable innerhalb der Docker-Infrastruktur geändert werden.

TranslatorActionAuthenticator: Durch den TranslatorActionAuthenticator werden die API-Methoden gesichert, die für die Übersetzung der Challenges verwendet werden. Standardmässig wird die Rolle «hl_translator» oder «hl_admin» zugelassen. Alternativ kann der Standardwert durch eine Umgebungsvariable innerhalb der Docker-Infrastruktur geändert werden.

ImportActionAuthenticator: Durch den ImportActionAuthenticator werden die API-Methoden gesichert, die für den Import und Export der Challenges verwendet werden. Da diese API-Methoden ausschliesslich von einer externen Applikation aufgerufen werden, wird hier der übermittelte API-Key geprüft. Dieser kann in der application.conf Datei innerhalb des Servers angepasst werden.

6.2.2.1.1.1 Verwendung

Authenticators werden als Annotation einer Controller-Methode angefügt und werden vor dem Ausführen der Methode geprüft. Im Fehlerfall wird die gesicherte Methode niemals ausgeführt.

6.2.2.1.2 Controllers

Die Implementierung des REST-APIs wird in den Controllern behandelt. Sie dienen nach den Authenticators als erste Anlaufstelle vom Client zum Server. Die Controller delegieren ihre Aufrufe weiter an die Services, welche die eigentliche Serverlogik beinhalten. Da der Server für fünf unterschiedliche Verwendungszwecke Methoden zur Verfügung stellt, wurden fünf verschiedene Controller implementiert.

Im Gegensatz zur Studienarbeit beinhalten die Controller keine eigene Business-Logik mehr, da diese, nach einem Refactoring, vollständig in die Services ausgelagert wurde. Die einzige verwendete Logik besteht aus Checks für die Query-Strings oder allfälligen Header-Werten.

6.2.2.1.2.1 Challenge Controller

Der Challenge Controller stellt Methoden für den Challenge-Editor zur Verfügung. Neben den einfachen GET-Methoden für die initialen Metadaten Werte (Typen, Kategorien usw.), stellt er die Implementierung für die POST-Methode zur Erstellung und Bearbeitung einer Challenge zur Verfügung.

6.2.2.1.2.2 Consumer Controller

Der Consumer Controller stellt die GET-Methoden für die Consumer Liste, sowie für die Übermittlung der URL zu einem spezifischen Consumer System zur Verfügung.

6.2.2.1.2.3 Import Controller

Der Import Controller stellt die Methoden für den Synchronisationsprozess der Challenges zur Verfügung. Er implementiert die URIs für den Import und Export der Challenges.

6.2.2.1.2.4 Media Controller

Der Media Controller stellt die GET-, POST- und DELETE-Methoden für Bilder zur Verfügung.

6.2.2.1.2.5 Websocket Controller

Der Websocket Controller implementiert die GET-Methode zum Aufbau einer Websocket Verbindung. Dieser Controller ist der einzige, der nicht durch Authenticators gesichert wird. Aus diesem Grund ist er auch der einzige, der eigene Logik besitzt, die er dazu verwendet, die Rolle des Benutzers, sowie ein Origin-Check anhand der gelieferten Header zu erledigen.

6.2.2.1.3 Actors

Es wird ein ParentActor implementiert, der eine Factory-Methode für die Erstellung einer Websocket Verbindung anhand eines eingehenden Requests zur Verfügung stellt. Durch die Factory Methode wird die Websocket-Verbindung zwischen Client und Server hergestellt. Im durch die Factory-Methode erstellten Child-Actor wird das Handling des, durch die Websocket-Verbindung, asynchron verschickten Inhalts durchgeführt. Die unterschiedlichen Markdown-Typen (Abstract, Hint, Instruction, Solution) werden voneinander getrennt behandelt und anhand eines Service in die Datenbank persistiert.

6.2.2.1.4 Services

Damit die Controller nur die Implementierung des REST-APIs und wenig eigene Logik beinhalten, wird pro Controller eine eigene Service-Klasse implementiert. Diese kapseln die Repository Zugriffe und zentralisieren Methoden, die von unterschiedlichen Controllers oder anderen Services verwendet werden können. Durch diese zusätzliche Abstraktion wird es möglich, dass die Persistenz-Schicht ohne grosse Anpassungen ausgewechselt werden kann.

6.2.2.1.5 Model

Die Klassen des Model Packages sind ähnlich aufgebaut wie bereits während der Studienarbeit. Eine wichtige Neuerung in diesem Bereich sind weitere Data-Transfer-Objekt Klassen, welche für den Import- / Export-Prozess verwendet werden.

6.2.2.1.6 Utils

Die Klassen des Utils Package wurden grundlegend angepasst. Einerseits wurden die bestehenden Klassen entfernt, da deren Funktionalität auf andere Art zur Verfügung gestellt werden konnte (JWTUtil) und andererseits neue Komponenten ergänzt.

6.2.2.1.6.1 ControllerUtil

Die ControllerUtil Klasse stellt Methoden bereit, welche von verschiedenen Controller und Services verwendet werden. Wie z. B. einer tryParse Methode für Long-Werte.

6.2.2.1.6.2 TestUtil

Das TestUtil implementiert zwei Methoden, welche für die Tests verwendet werden. Die erste generisch implementierte Methode wird im Testing benötigt, um anhand von Reflection auch private Methoden aufrufen zu können. Die zweite wird zum vereinfachten Lesen von Testdateien benötigt.

6.2.2.2 Client

Die Struktur des Clients aus der vorangehenden Studienarbeit ist weitestgehend erhalten geblieben. Eine wichtige Anpassung ist, dass die Middleware entfernt, sowie die Components Klassen einem eingehenden Refactoring unterzogen und durch neue Komponenten erweitert wurden. Zusätzlich wurden neue Stylesheets erstellt, die für das Styling einzelner Client Komponenten verwendet werden.

Damit die «actions» Methoden, sowie der Reducer keine Logik mehr enthalten, wurden die implementierten Funktionen in Helpers ausgelagert und von dort in die gewünschten Dateien importiert.

Weitere kleinere Anpassungen gab es im Bereich der Konfiguration mittels Webpack, wo zusätzliche Komponenten eingefügt und einzelne fehlerhafte Teile ersetzt wurden.

6.2.3 Logische Architektur

Im folgenden Ausschnitt wird die logische Architektur aus Sicht der Serverapplikation vereinfacht dargestellt. Die Abbildung sollte selbsterklärend sein und wird aus diesem Grund nicht mehr weiter beschrieben.

Abb. 47 Logische Architektur

6.2.4 Datenspeicherung

Die vom Server zur Verfügung gestellten Daten werden in einer MySQL-Datenbank persistiert. Diese wurde als eigener Docker-Container umgesetzt. In der aktuellen Implementierung wird, wie bereits erwähnt, der Zugriff auf die Datenbank durch Repository-Klassen, die ein dafür vorgesehenes Interface implementieren, gekapselt. Dadurch kann die Datenbank und die Repositories ohne weitere Auswirkungen durch eine alternative Implementierung ersetzt werden.

6.2.4.1 Evolution

Damit das verwendete Datenbankschema auf allen Systemen aktuell bleibt wird die Play-interne Database Evolution eingesetzt. Bei Änderungen an Model-Klassen wird die Evolutionsdatei angepasst und die neuen Tabellen oder Spalten beim Starten des Servers in der Datenbank eingefügt. Die Evolutionsdatei wird jeweils in das Git-Repository commited.

6.2.5 Benutzeroberfläche

Da die gesamte Benutzeroberfläche für diese Bachelorarbeit neu aufgebaut wurde, wird im folgenden Kapitel versucht anhand von Screenshots eine Übersicht über das UI zu geben.

6.2.5.1 Challenge-Editor

6.2.5.1.1 Navigation

Durch die Navigationskomponente wird es dem Bearbeiter einer Challenge möglich, auf die verschiedenen Schritte zur Erstellung einer Challenge direkt zu greifen zu können. Die Navigation wird nur für die Bearbeitung einer Challenge im gesamten Ausmass angezeigt. Bei der initialen Erstellung wird sie nur bis zum aktuellen Editor-Schritt angezeigt.

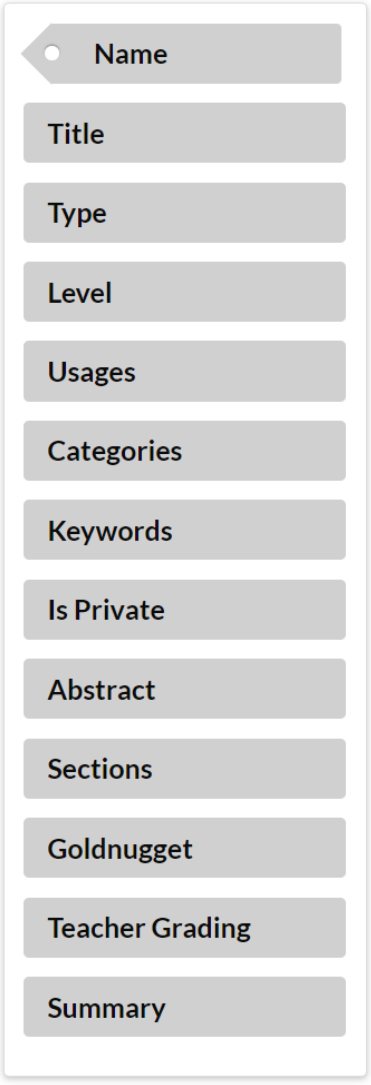


Abb. 48 Challenge-Editor Navigation

6.2.5.1.2 Metadaten Components

Im Folgenden werden die einzelnen Editorschritte für die Metadaten einer Challenge visualisiert.

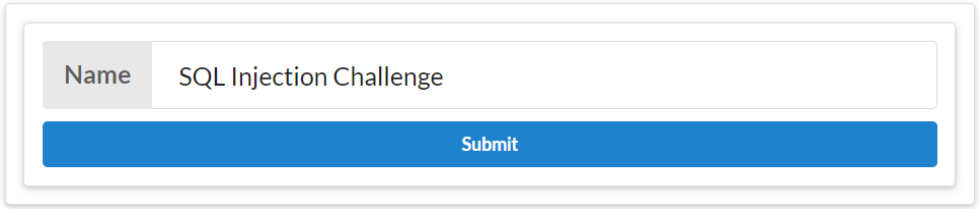


Abb. 49 Challenge-Editor Name

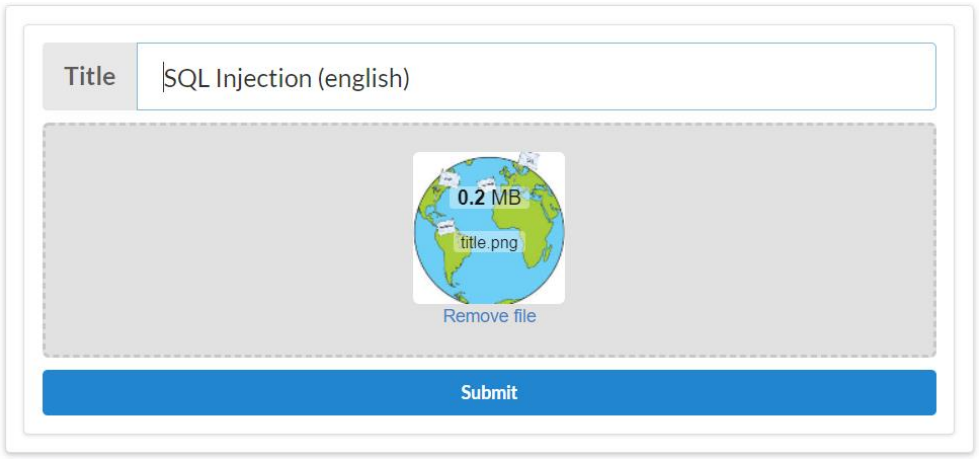


Abb. 50 Challenge-Editor Titel mit hochgeladenem Bild

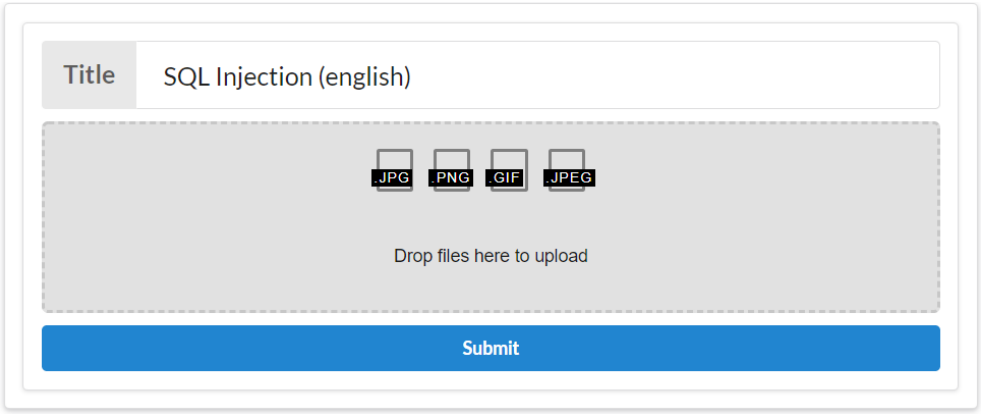


Abb. 51 Challenge-Editor Titel ohne Bild

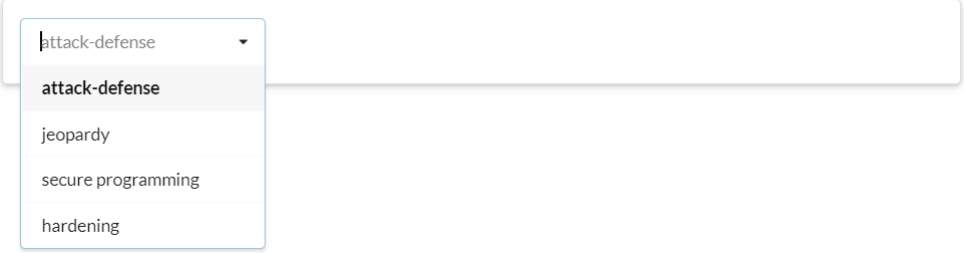


Abb. 52 Challenge-Editor Type Dropdown

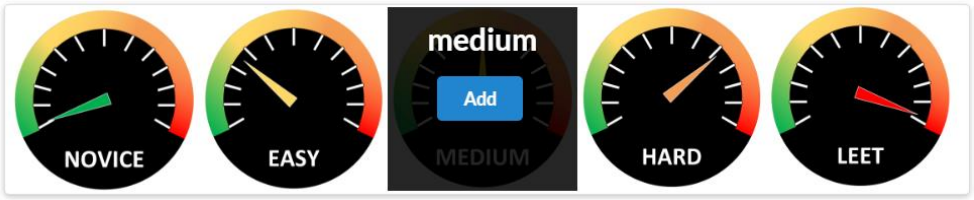


Abb. 53 Challenge- Editor Level

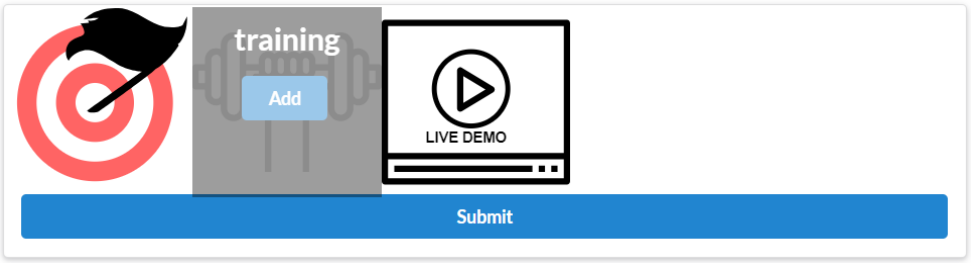


Abb. 54 Challenge-Editor Usages

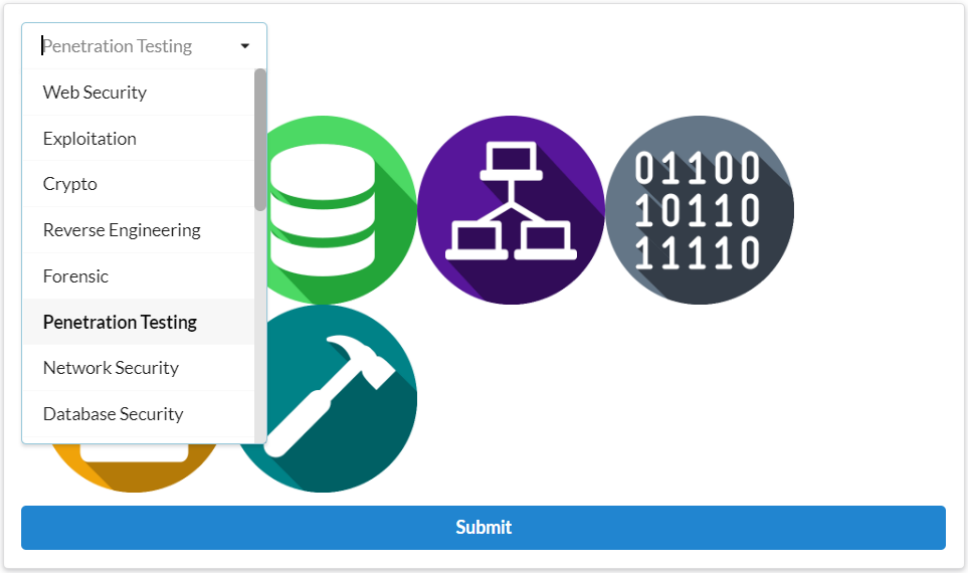


Abb. 55 Challenge-Editor Categories

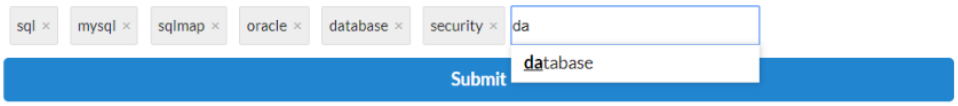


Abb. 56 Challenge-Editor Keywords

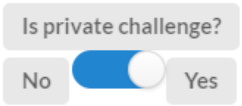


Abb. 57 Challenge-Editor Private Checkbox

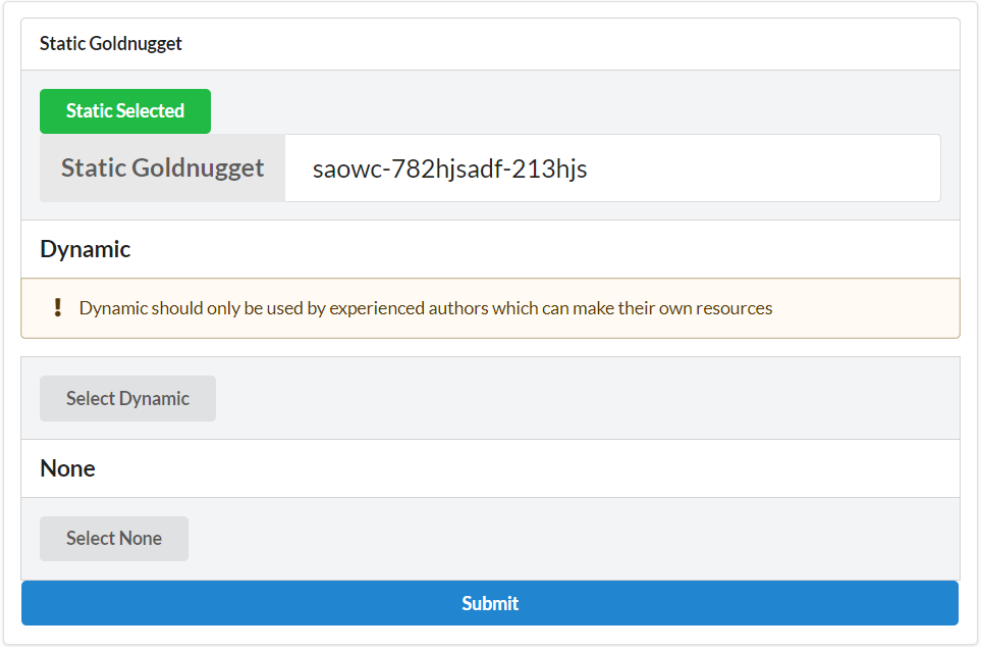


Abb. 58 Challenge-Editor Goldnugget

6.2.5.1.3 Markdown Editoren

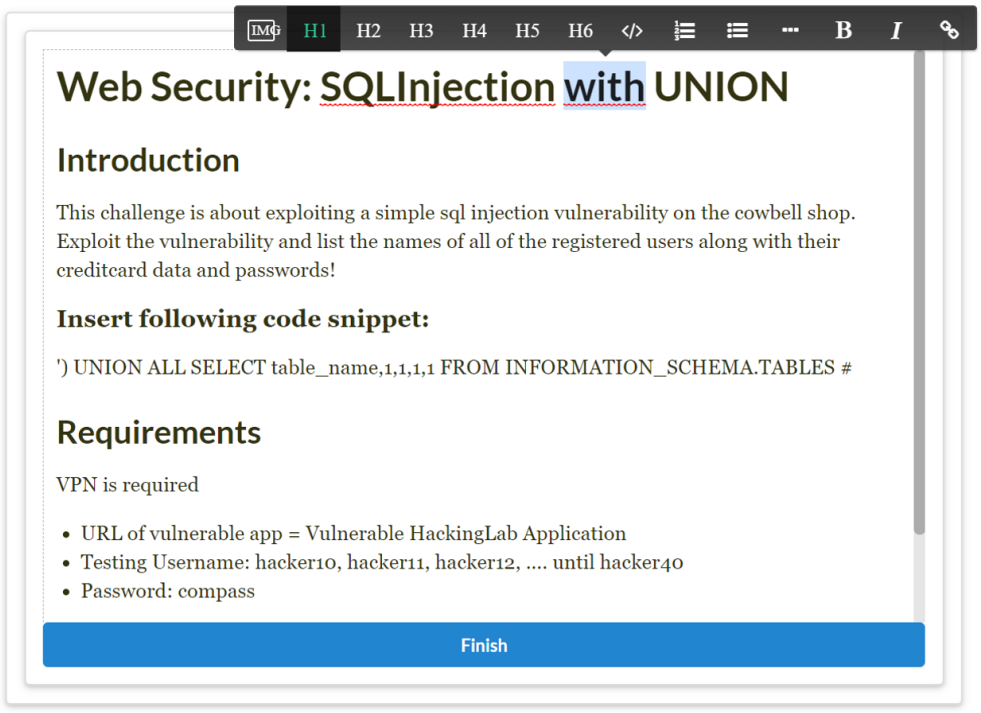


Abb. 59 Challenge-Editor Abstract

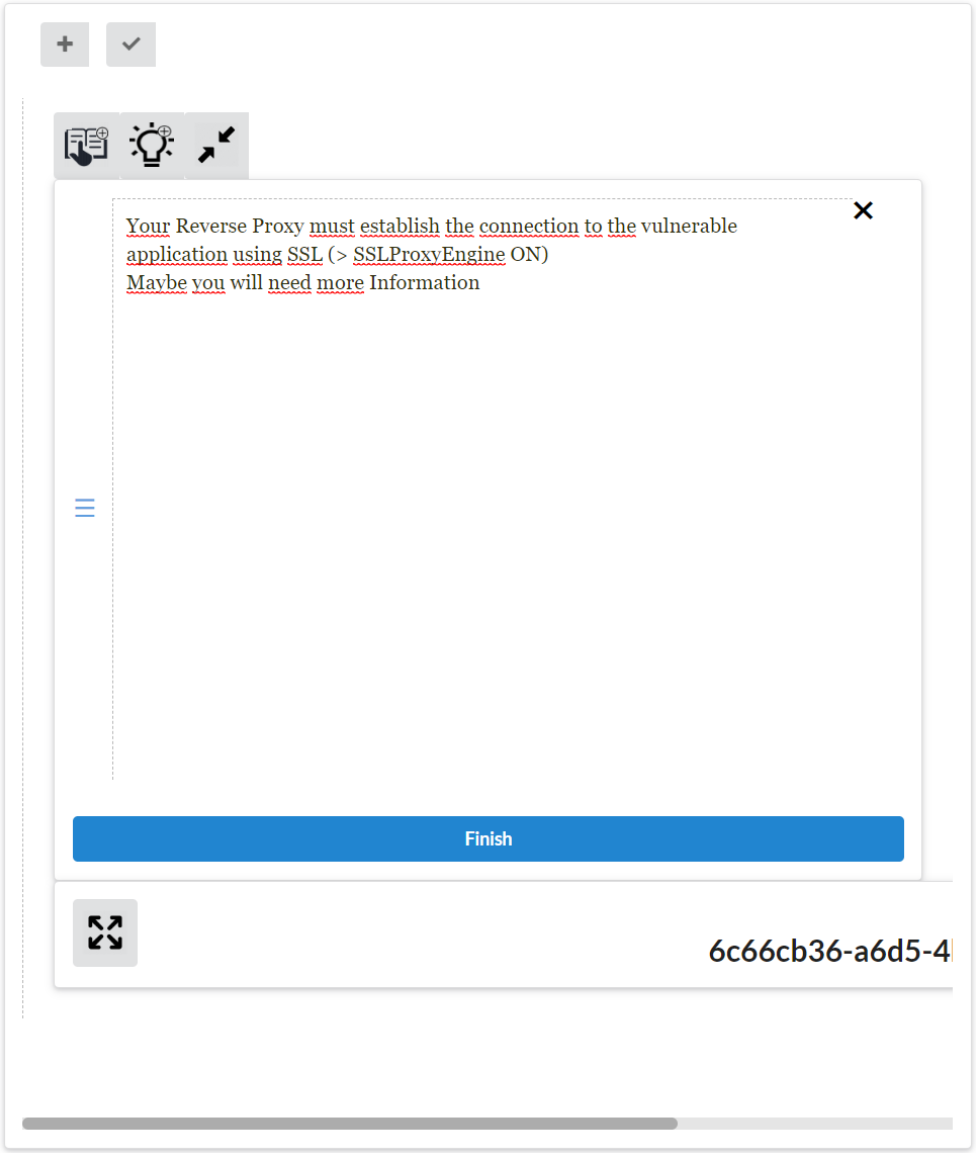


Abb. 60 Challenge-Editor Section

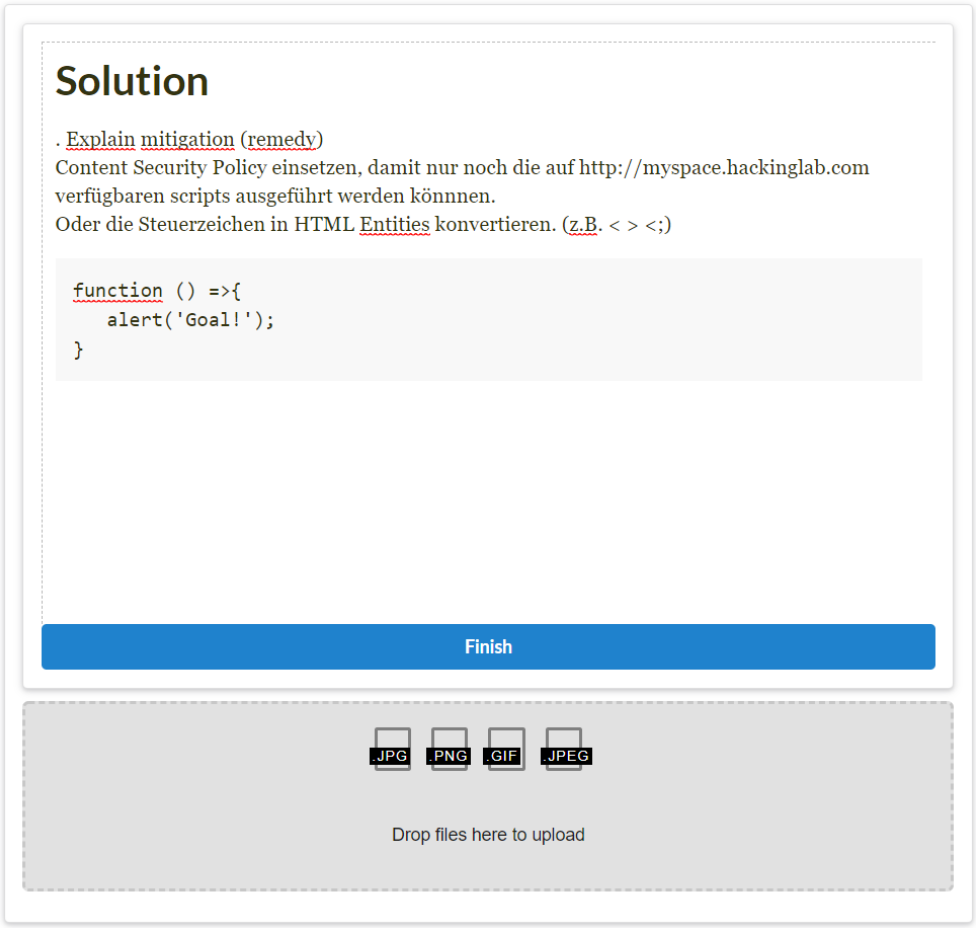


Abb. 61 Challenge-Editor Solution

6.2.5.1.4 Challenge Summary

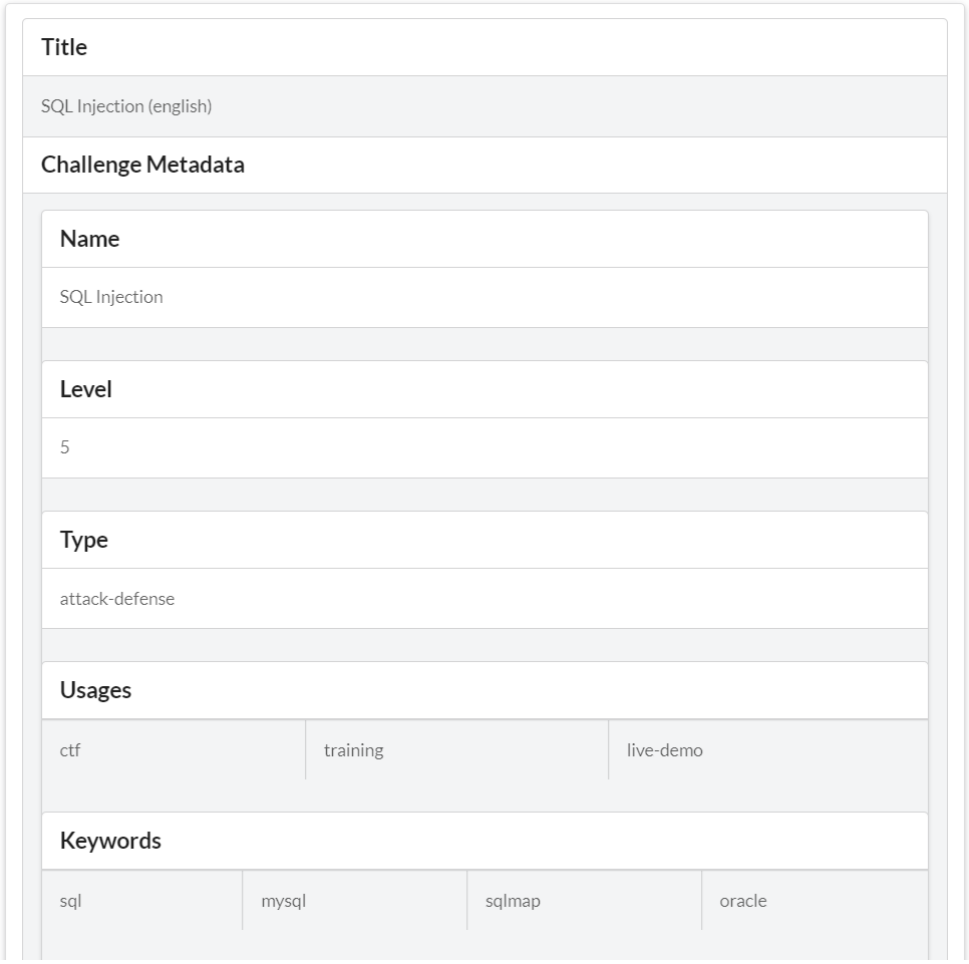


Abb. 62 Challenge-Editor Summary 1

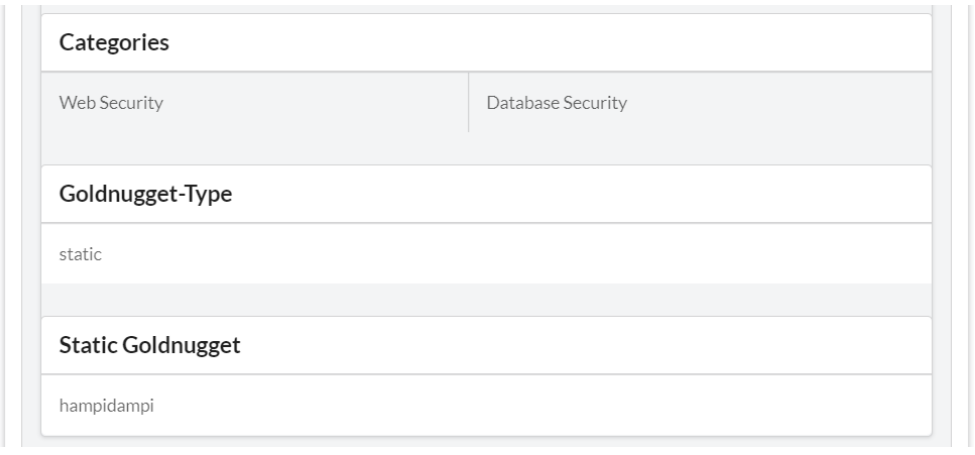


Abb. 63 Challenge-Editor Summary 2

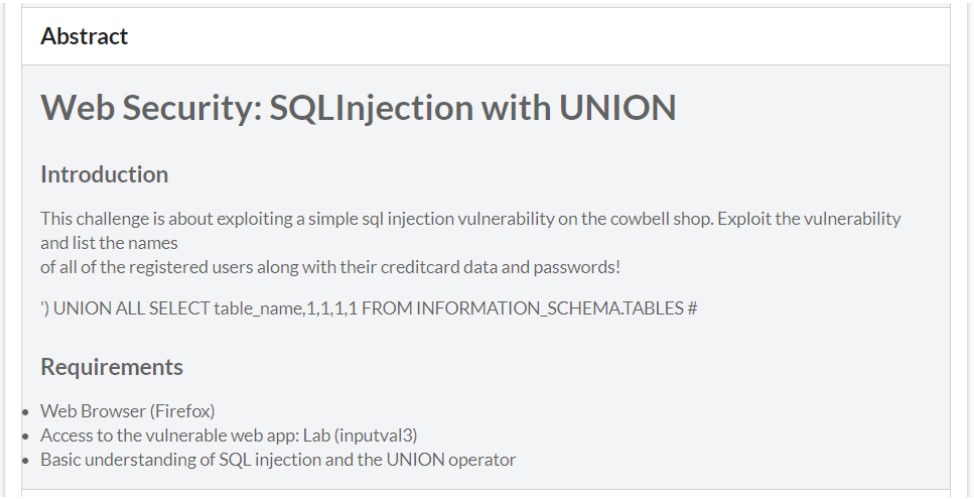


Abb. 64 Challenge-Editor Summary 3

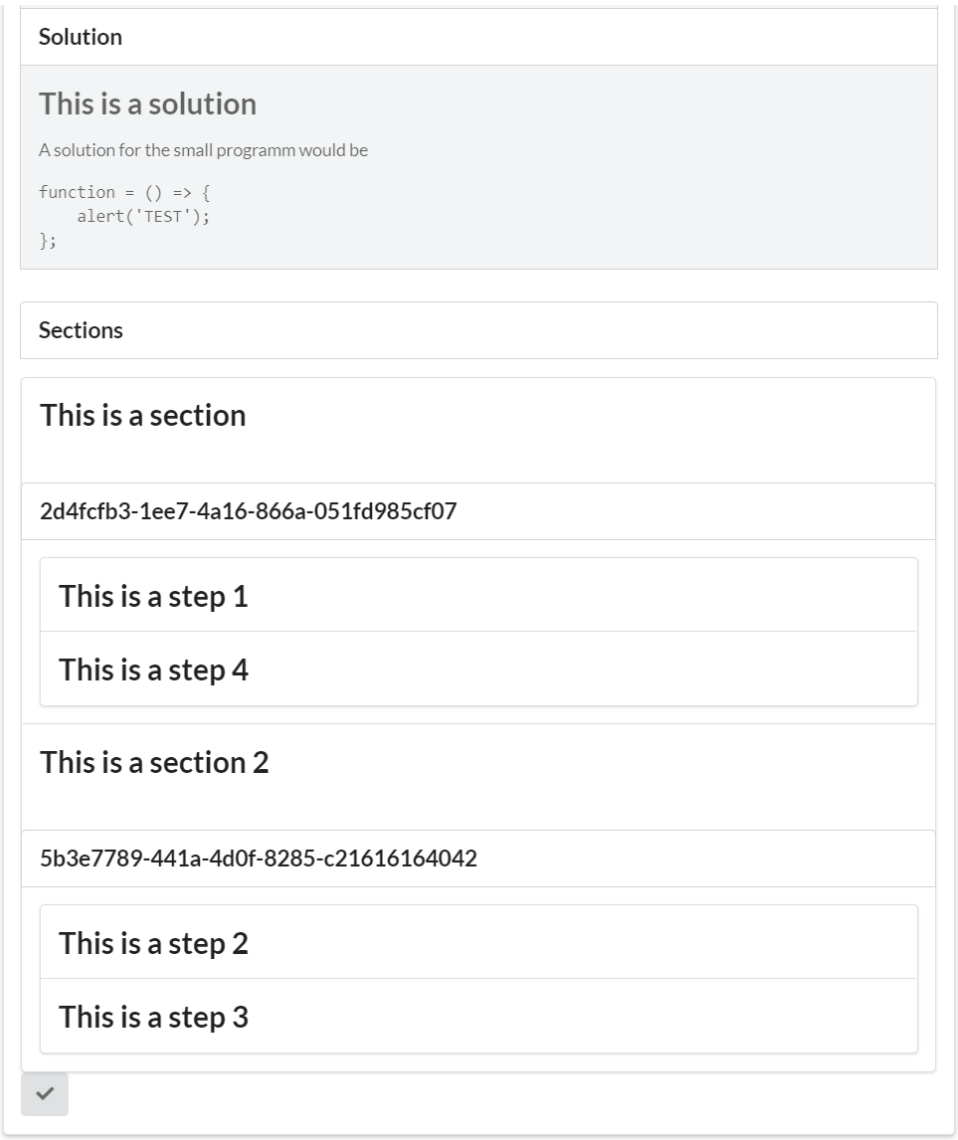


Abb. 65 Challenge-Editor Summary 4

6.2.5.2 Challenge-Translator

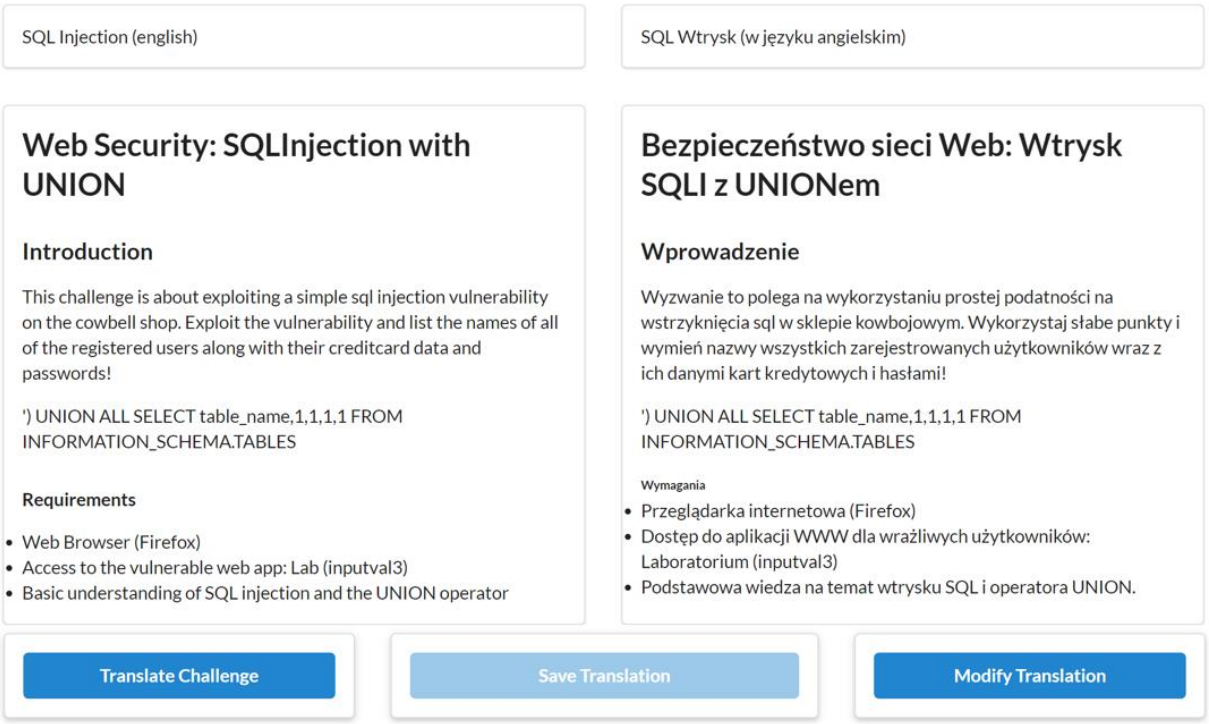


Abb. 66 Challenge-Translator Read-Mode

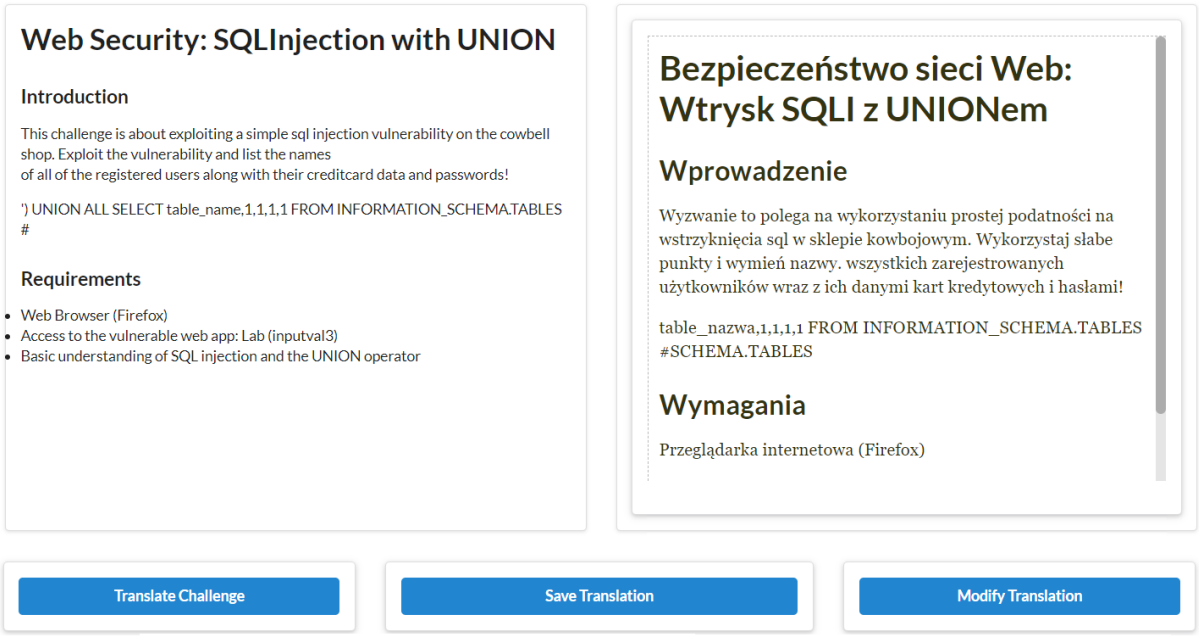


Abb. 67 Challenge-Translator Edit Mode

6.2.6 Test

Im Folgenden werden die verwendeten Testmethoden beschrieben. Es wird dabei eine Unterteilung zwischen Server- und Client-Test gemacht, da der Aufbau dieser komplett verschieden ist.

6.2.6.1 Server Unit Tests

Das verwendete Testframework für den Server ist JUnit, welches Annotationen zur Verfügung stellt, um simple Testklassen aufzubauen.

Aufgrund der modular aufgebauten Server-Architektur wird es möglich, jede Komponente einzeln zu prüfen. Die Priorität haben dabei die Services, da diese die gesamte Businesslogik der Applikation beinhalten. Damit nicht nur die als «public» definierten Methoden geprüft werden können, wurde ein TestUtil aufgebaut, welches anhand von Reflection die privaten Methoden sichtbar und damit zusätzlich testbar macht.

Einer der Vorteile der Verwendung des Play Frameworks ist, dass Mechanismen zur Verfügung gestellt werden, die es erlauben eine gesamte Serverapplikation, an welche Anfragen versendet werden können, zu mocken. Dadurch können auch Controller bequem geprüft werden. Da die ActionAuthenticators für jede Methode des Controllers verwendet werden, wird es zusätzlich möglich diese in die Tests einzubeziehen und dadurch die Autorisierungsschicht des Servers zu testen.

Falls genügend Zeit bleibt, wird neben den entscheidenden Service-Methoden auch der gesamte Controller Code getestet.

6.2.6.2 Client Unit Tests

Für die Client Tests wird Jest (JavaScript Testing), das «except» Testing Framework für die Assertions, sowie «enzyme», das Hilfsmethoden für das Testing von React zur Verfügung stellt, verwendet.

Da die einzelnen React Components nicht sinnvoll zu prüfen sind, werden die Tests auf dem Client auf die konkreten Methoden-Implementierungen der eingesetzten Helper-Klassen beschränkt.

6.3 API-Beschreibungen und Verwendungsbeispiele

Methode	URL	Verwendung / Beschreibung
GET	/api	
GET	/api/challenges	**StandardActionAuthenticator**: Die hl_user Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. Gibt die Challenge-Metadaten für die Anzeige der verfügbaren Challenges zurück.
GET	/api/challenges/:challengeId	**StandardActionAuthenticator**: Die hl_user Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. **Headers**: Challenge-Language für die Auswahl einer Sprache. Falls der Wert leer gelassen wird, werden alle verfügbaren Sprachen exportiert. Gibt eine spezifische Challenge als JSON-Objekt zurück.
POST	/api/challenges/:challengeId	**TranslatorActionAuthenticator**: Die hl_translator Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. **Body**: Challenge als JSON-Objekt. Persistiert die im Body enthaltene Challenge in der Datenbank.
GET	/api/solution/:challengeId	**StandardActionAuthenticator**: Die hl_user Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. **Headers**: Challenge-Language für die Auswahl einer Sprache. Falls der Wert leer gelassen wird, wird die englische Solution exportiert. Gibt die Solution für eine spezifische Challenge zurück.
GET	/api/translatedChallenges/:challengeId	**StandardActionAuthenticator**: Die hl_user Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. **Headers**: Challenge-Language für die Auswahl einer Sprache. Muss angegeben werden. Gibt die Übersetzung einer spezifischen Challenge als JSON zurück.
GET	/api/challenge/getNewChallengeId	**StandardActionAuthenticator**: Die hl_user Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. Gibt eine neue, nicht bereits eingesetzte UUID zurück.
GET	/api/section/getNewSectionId	**EditorActionAuthenticator**: Die hl_editor Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. **SectionQueryString**:challengeId als Query-String Parameter erforderlich. Gibt ein Section JSON-Objekt zurück (sectionId und order).
GET	/api/step/getNewInstructionId	**EditorActionAuthenticator**: Die hl_editor Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. **SectionQueryString**: challengeId und sectionId als Query-String Parameter erforderlich. Gibt ein Instruction JSON-Objekt zurück (sectionId, instructionId, type: 'instruction', order).
GET	/api/step/getNewHintId	**EditorActionAuthenticator**: Die hl_editor Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. **SectionQueryString**: challengeId und sectionId als Query-String Parameter erforderlich. Gibt ein Hint JSON-Objekt zurück (sectionId, hintId, type: 'hint', order).
GET	/api/challenge/getAbstractId	**EditorActionAuthenticator**: Die hl_editor Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. **MarkdownItemQueryString:** challengeId als Query-String Parameter erforderlich. Gibt die aktuelle AbstractID der spezifischen Challenge zurück.
GET	/api/challenge/getSolutionId	**EditorActionAuthenticator**: Die hl_editor Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. **MarkdownItemQueryString:** challengeId als Query-String Parameter erforderlich. Gibt die aktuelle SolutionID der spezifischen Challenge zurück.
GET	/api/challenge/getNewAbstractId	**EditorActionAuthenticator**: Die hl_editor Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. **MarkdownItemQueryString:** challengeId als Query-String Parameter erforderlich. Initialisiert ein neues Abstract für die spezifische Challenge auf der Datenbank und gibt die neue AbstractID zurück.
GET	/api/challenge/getNewSolutionId	**EditorActionAuthenticator**: Die hl_editor Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. **MarkdownItemQueryString:** challengeId als Query-String Parameter erforderlich. Initialisiert eine neue Solution für die spezifische Challenge auf der Datenbank und gibt die neue SolutionID zurück.
POST	/api/section/createNewChallenge	**EditorActionAuthenticator**: Die hl_editor Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt **Body**: JSON-Objekt für eine Challenge (challengeName, challengeTitle, goldnuggetType, staticGoldnuggetSecret, challengeId, challengeLevel, challengeUsages, challengeKeywords, challengeCategories, isPrivate, challengeType). Persistiert anhand des Bodys eine neue Challenge auf der Datenbank bzw. modifiziert eine bestehende.
GET	/api/challenge/wsCreateNew/*queryString	**EditorActionAuthenticator**: Die hl_editor Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt.

		Origin-Check: Request Origin wird geprüft. Wert kann anhand Umgebungsvariable dynamisch eingefügt werden. Initialisiert einen neuen Websocket für die kontinuierliche Speicherung einer Markdown-Datei.
GET	/api/consumers	**StandardActionAuthenticator**: Die hl_user Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. Gibt ein JSON-Array der in der restdb.io enthaltenen Consumer zurück.
GET	/api/consumer	**StandardActionAuthenticator**: Die hl_user Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. Gibt Werte eines spezifischen Consumers als JSON zurück.
GET	/api/levels	**StandardActionAuthenticator**: Die hl_user Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. Gibt die definierten Levels zurück.
GET	/api/types	**StandardActionAuthenticator**: Die hl_user Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. Gibt die definierten Typen zurück.
GET	/api/usages	**StandardActionAuthenticator**: Die hl_user Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. Gibt die definierten Usages zurück.
GET	/api/categories	**StandardActionAuthenticator**: Die hl_user Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. Gibt die definierten Kategorien zurück.
GET	/api/keywords	**StandardActionAuthenticator**: Die hl_user Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. Gibt die definierten Keywords zurück.
POST	/api/upload_image	**EditorActionAuthenticator**: Die hl_editor Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. **MediaQueryString**: challengeId, languageIsoCode, markdownType als Query-String Parameter erforderlich. Zusätzlich markdownId möglich. Upload einer Image-Datei mit Persistierung in der Datenbank. Gibt die ID des erstellten Datenbank Entries zurück.
POST	/api/upload_image_base64	**EditorActionAuthenticator**: Die hl_editor Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. **MediaQueryString**: challengeId, languageIsoCode, markdownType, markdownId als Query-String Parameter erforderlich. **Body**: JSON-Objekt mit base64Image und filePath. Upload eines base64 codierten Images mit Persistierung in der Datenbank. Gibt die ID des erstellten Datenbank Entries zurück.
DELETE	/api/removeImage	**EditorActionAuthenticator**: Die hl_editor Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. **MediaQueryString**: challengeId, mediaId als Query-String Parameter erforderlich. Löschen eines Images aus der Datenbank.
DELETE	/api/ step/removeStep/:stepId	**EditorActionAuthenticator**: Die hl_editor Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. Löscht ein Hint, Instruction oder Section aus der Datenbank.
GET	/api/media/*mediaPath	**StandardActionAuthenticator**: Die hl_user Rolle bzw. ein per Umgebungsvariable eingefügter String wird für den Benutzer vorausgesetzt. Gibt ein Image-File für den übergebenen Pfad zurück.
POST	/api/importChallenge	**ImportActionAuthenticator**: Der spezifizierte API-Key muss als Header-Value (auth_key) übergeben werden. Das übermittelte JSON-Objekt wird geparst und mit allen Medien-Dateien in der Datenbank persistiert.
GET	/api/exportChallenge/:challengeId	**ImportActionAuthenticator**: Der spezifizierte API-Key muss als Header-Value (auth_key) übergeben werden. Eine spezifische Challenge wird als JSON-Objekt mit allen Medien-Dateien zurückgegeben.
GET	/api/exportAllChallenges	**ImportActionAuthenticator**: Der spezifizierte API-Key muss als Header-Value (auth_key) übergeben werden. Alle vorhandenen Challenge werden als JSON-Array mit allen Medien-Dateien zurückgegeben.
GET	/api/assets/*file	Gibt statische Ressourcen aus dem /public Path zurück.

Tab. 5 REST-API Beschreibung

Da in der implementierten Infrastruktur vor dem Server ein Reverse Proxy installiert wurde, wird der OPTIONS-Request direkt von diesem beantwortet. Ansonsten müsste pro zur Verfügung gestellten REST-URI eine weitere Methode OPTIONS implementiert werden, welche je nach Anforderung, die benötigten Access-Control-Allow-* – Headers zurückliefert, damit «Cross Origin Resource Sharing» korrekt funktioniert.

6.4 Test-Logs

6.4.1 Systemtests-Infrastruktur

Die nachfolgenden Systemtests prüfen die Korrektheit der Docker-Infrastruktur und deren durch das Admin Tool zur Verfügung gestellten Funktionalitäten. Dafür wurde jeweils ein Testprotokoll erstellt. Dabei sind alle Tests so deklariert, dass diese zu einem späteren Zeitpunkt reproduzierbar sind.

6.4.1.1 Umgebung

Die Tests wurden auf dem von der HSR zur Verfügung gestelltem virtuellen Server getätigt, welcher folgende Umgebung für die Ausführung bietet.

Beschreibung	Wert
Betriebssystem	Ubuntu 16.04.4 LTS
Festplattengrösse	35 GB
RAM-Grösse	4 GB
Prozessor	Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20 GHz
Anzahl Prozessorkerne	2
Docker-Version	18.03.1-ce
Docker-Compose-Version	1.20.1
Domäne	hl.ygubler.ch

Tab. 6 Systemtests-Infrastruktur – Umgebung

Weiter wurde das Git-Repository «ChallengeAuthoringSystem» heruntergeladen, welches die Grundlage für die Tests bildet. Aus diesem wird auch die Konfiguration (.env-Datei und andere Konfigurationsdateien) übernommen, welche auf die Umgebung/Domäne der Bachelorarbeit zugeschnitten ist.

6.4.1.2 Docker-Netzwerk erstellen

Beschreibung	Wert
Test-Nummer	1
Beschreibung	Docker-Netzwerkerstellung mithilfe des Admin Tools im Docker Standalone Mode (ENABLE_DOCKER_SWARM=false in .env)
Ausgeführte Schritte	./admin_tool.sh Please enter your choice: 2 Please enter your choice: 25 docker network ls
Erwartetes Ergebnis	create cas network <Docker-Network ID> NETWORK ID NAME DRIVER SCOPE <network_id> cas-network bridge local
Tatsächliches Ergebnis	create cas network f0cad9547cf7e411efb15a661c0ff60f14f615b6262ef97886e36df766611ba0 NETWORK ID NAME DRIVER SCOPE f0cad9547cf7 cas-network bridge local
Testergebnis	OK

Tab. 7 Systemtest - Docker-Netzwerk erstellen - Standalone Mode

Beschreibung	Wert
Test-Nummer	2
Beschreibung	Docker-Netzwerkerstellung mithilfe des Admin Tools im Docker Swarm Mode (ENABLE_DOCKER_SWARM=true in .env)
Ausgeführte Schritte	./admin_tool.sh Please enter your choice: 2 Please enter your choice: 25 docker network ls
Erwartetes Ergebnis	create cas network <Docker-Network ID> NETWORK ID NAME DRIVER SCOPE <network_id> cas-network overlay swarm
Tatsächliches Ergebnis	create cas network cas-network vfam6opq5hye4hsqigzv3381l6 NETWORK ID NAME DRIVER SCOPE vfam6opq5hye cas-network overlay swarm
Testergebnis	OK

Tab. 8 Systemtest - Docker-Netzwerk erstellen - Swarm Mode

6.4.1.3 Docker Secrets erstellen

Beschreibung	Wert
Test-Nummer	3
Beschreibung	Docker Secrets-Erstellung mithilfe des Admin Tools im Docker Standalone Mode (ENABLE_DOCKER_SWARM=false in .env)
Ausgeführte Schritte	./admin_tool.sh Please enter your choice: 3 Please enter your choice: 25 cat secrets secrets/database_passwort.txt
Erwartetes Ergebnis	create_docker_secrets total 64 -rw-r--r-- 1 root root 13 Jun 11 21:30 database_password.txt -rw-r--r-- 1 root root 0 Apr 16 20:47 .gitkeep -rw-r--r-- 1 root root 14 Jun 11 21:30 keycloak_database_password.txt -rw-r--r-- 1 root root 12 Jun 11 21:30 keycloak_mysql_root_password.txt -rw-r--r-- 1 root root 14 Jun 11 21:30 keycloak_password.txt -rw-r--r-- 1 root root 37 Jun 11 21:30 keycloak_proxy_cas_client_client_secret.txt -rw-r--r-- 1 root root 33 Jun 11 21:30 keycloak_proxy_cas_client_encryption_key.txt -rw-r--r-- 1 root root 37 Jun 11 21:30 keycloak_proxy_cas_server_client_secret.txt -rw-r--r-- 1 root root 33 Jun 11 21:30 keycloak_proxy_cas_server_encryption_key.txt -rw-r--r-- 1 root root 37 Jun 11 21:30 keycloak_proxy_consumer_client_secret.txt -rw-r--r-- 1 root root 33 Jun 11 21:30 keycloak_proxy_consumer_encryption_key.txt -rw-r--r-- 1 root root 10 Apr 25 17:38 ldap_admin_password.txt -rw-r--r-- 1 root root 9 Apr 25 17:38 ldap_config_password.txt -rw-r--r-- 1 root root 9 Apr 25 17:38 ldap_readonly_user_password.txt -rw-r--r-- 1 root root 12 Jun 11 21:30 mysql_root_password.txt cat secrets/database_password.txt <database password from create_docker_secrets.sh>
Tatsächliches Ergebnis	create_docker_secrets total 64 -rw-r--r-- 1 root root 13 Jun 11 21:30 database_password.txt -rw-r--r-- 1 root root 0 Apr 16 20:47 .gitkeep -rw-r--r-- 1 root root 14 Jun 11 21:30 keycloak_database_password.txt -rw-r--r-- 1 root root 12 Jun 11 21:30 keycloak_mysql_root_password.txt -rw-r--r-- 1 root root 14 Jun 11 21:30 keycloak_password.txt -rw-r--r-- 1 root root 37 Jun 11 21:30 keycloak_proxy_cas_client_client_secret.txt -rw-r--r-- 1 root root 33 Jun 11 21:30 keycloak_proxy_cas_client_encryption_key.txt -rw-r--r-- 1 root root 37 Jun 11 21:30 keycloak_proxy_cas_server_client_secret.txt -rw-r--r-- 1 root root 33 Jun 11 21:30 keycloak_proxy_cas_server_encryption_key.txt -rw-r--r-- 1 root root 37 Jun 11 21:30 keycloak_proxy_consumer_client_secret.txt -rw-r--r-- 1 root root 33 Jun 11 21:30 keycloak_proxy_consumer_encryption_key.txt -rw-r--r-- 1 root root 10 Apr 25 17:38 ldap_admin_password.txt -rw-r--r-- 1 root root 9 Apr 25 17:38 ldap_config_password.txt -rw-r--r-- 1 root root 9 Apr 25 17:38 ldap_readonly_user_password.txt -rw-r--r-- 1 root root 12 Jun 11 21:30 mysql_root_password.txt cat secrets/database_password.txt 6rNhNAPY7yXf
Testergebnis	OK

Tab. 9 Systemtest – Docker Secrets erstellen - Standalone Mode

Beschreibung	Wert
Test-Nummer	4
Beschreibung	Docker-Secrets-Erstellung mithilfe des Admin Tools im Docker Swarm Mode (ENABLE_DOCKER_SWARM=true in .env)

Ausgeführte Schritte	`./admin_tool.sh`
	`Please enter your choice: 2`
	`Please enter your choice: 25`
	`docker network ls`

Erwartetes Ergebnis	`create_docker_secrets`
	`<Docker-Secret ID>`
	`<Docker-Secret ID>`
	`…`
	`<print secret store>`

Tatsächliches Ergebnis	`create_docker_secrets`
	`bc8uwucyujuwsg6sbsvcwtdsb`
	`pjlup6s1ob5faglfgv3hkhawb`
	`ki0fh982fi7hmt9v57hv6dmq8`
	`kaom8t9fd9bg8q9vlo84qfzn7`
	`t7s5bxchmq44c619dhgzij8r8`
	`u2z8l2rhdrubgw9he5l0kch41`
	`uvytpecpzxu4yea71yf9iavc5`
	`5nkivjlm3dxemg5i5oOwlwdrt`
	`94lkful0k7q4m8crs78rd2to3`
	`ko02hehi06fm8d9w1xdufxk29`
	`42co38xhcbnwcruqyk4n60wca`
	`ID NAME ...`
	`uvytpecpzxu4yea71yf9iavc5 database_password ...`
	`u2z8l2rhdrubgw9he5l0kch41 keycloak_database_password ...`
	`42co38xhcbnwcruqyk4n60wca keycloak_mysql_root_password ...`
	`bc8uwucyujuwsg6sbsvcwtdsb keycloak_password ...`
	`pjlup6s1ob5faglfgv3hkhawb keycloak_proxy_cas_client_client_secret ...`
	`ki0fh982fi7hmt9v57hv6dmq8 keycloak_proxy_cas_client_encryption_key ...`
	`5nkivjlm3dxemg5i5oOwlwdrt keycloak_proxy_cas_server_client_secret ...`
	`t7s5bxchmq44c619dhgzij8r8 keycloak_proxy_cas_server_encryption_key ...`
	`ko02hehi06fm8d9w1xdufxk29 keycloak_proxy_consumer_client_secret ...`
	`94lkful0k7q4m8crs78rd2to3 keycloak_proxy_consumer_encryption_key ...`
	`kaom8t9fd9bg8q9vlo84qfzn7 mysql_root_password ...`

Testergebnis	OK

Tab. 10 Systemtest – Docker Secrets erstellen - Swarm Mode

6.4.1.4 Build Container ausführen

Beschreibung	Wert
Test-Nummer	5
Beschreibung	build_CAS Container ausführen und Logs folgen mithilfe des Admin Tools im Docker Standalone Mode (ENABLE_DOCKER_SWARM=false in .env)
Voraussetzung	ENABLE_DOCKER_SWARM=false in .env Docker Netzwerk im «bridge-mode» mit Option 2 im Admin Tool Docker Secrets Dateien anstelle Secret Store mit Option 3 im Admin Tool

Ausgeführte Schritte	`./admin_tool.sh`
	`Please enter your choice: 5`
	`Please enter your choice: 11`
	`Please enter your choice: 25`
	`docker ps`

Erwartetes Ergebnis	`Creating build-cas-server ... done`
	`Creating build-consumer ... done`
	`Creating build-cas-client ... done`
	`Creating cas-mysql ... done`
	Und erfolgreiches Resultat der Buildvorgänge der drei Services ohne Fehler im Log
	• build-consumer erfolgreiche Webpackerstellung
	• build-cas-client erfolgreiche Webpackerstellung
	• build-cas-server erfolgreiche Erstellung des ZIPs
	Nach dem Build-Vorgang sollten alle Container bis auf den cas-mysql beendet sein

Tatsächliches Ergebnis	`Creating build-cas-server ... done`
	`Creating build-consumer ... done`
	`Creating build-cas-client ... done`
	`Creating cas-mysql ... done`
	Logs:

```
build-consumer  | Hash: 821c8a48ec5b847b19e9
build-consumer  | Version: webpack 1.15.0
build-consumer  | Time: 65592ms
build-consumer  |                                      Asset     Size       Chunks
build-consumer  |  a046592bac8f2fd96e994733faf3858c.woff   63.7 kB      [emitted]
build-consumer  |   13db00b7a34fee4d819ab7f9838cc428.eot   98.6 kB      [emitted]
build-consumer  |   701ae6abd4719e9c2ada3535a497b341.eot   31.2 kB      [emitted]
build-consumer  |   46661d6d65debc63884004fed6e37e5c.svg  41 bytes      [emitted]
build-consumer  |   aa4adf7b80b62efdd292096501bcccd5.svg  82 bytes      [emitted]
build-consumer  |   c5ebe0b32dc1b5cc449a76c4204d13bb.ttf   98.4 kB      [emitted]
build-consumer  |   b87b9ba532ace76ae9f6edfe9f72ded2.ttf    106 kB      [emitted]
build-consumer  |   ad97afd3337e8cda302d10ff5a4026b8.ttf   30.9 kB      [emitted]
build-consumer  |   9c74e172f87984c48ddf5c8108cabe67.png   28.1 kB      [emitted]
build-consumer  |   8e3c7f5520f5ae906c6cf6d7f3ddcd19.eot    106 kB      [emitted]
build-consumer  | e8c322de9658cbeb8a774b6624167c2c.woff2   54.5 kB      [emitted]
build-consumer  |  faff92145777a3cbaf8e7367b4807987.woff   50.5 kB      [emitted]
build-consumer  | 0ab54153eeeca0ce03978cc463b257f7.woff2   40.1 kB      [emitted]
build-consumer  |  ef60a4f6c25ef7f39f2d25a748dbecfe.woff   14.7 kB      [emitted]
build-consumer  | cd6c777f1945164224dee082abaea03a.woff2   12.2 kB      [emitted]
build-consumer  |                                 main.js   1.42 MB   0  [emitted]
build-consumer  |                             main.js.map   8.09 MB   0  [emitted]
build-consumer  |                              index.html 547 bytes      [emitted]
build-consumer  |                        webpack.stats.json   3.62 kB      [emitted]
build-consumer  |    [0] multi main 40 bytes {0} [built]
build-consumer  |        factory:0ms building:9ms = 9ms
build-consumer  |     + 1427 hidden modules
build-consumer  |
build-consumer  | WARNING in ./app/actions/actions.js
build-consumer  |
build-consumer  | /opt/src/app/actions/actions.js
```

```
build-consumer  |   36:19  warning  Missing function expression name  func-names
build-consumer  |   37:17  warning  Unexpected console statement      no-console
build-consumer  |   38:29  warning  Unexpected console statement      no-console
build-consumer  |
build-consumer  | ✖ 3 problems (0 errors, 3 warnings)
build-cas-client    | Hash: b3939bcc28e3965f2733
build-cas-client    | Version: webpack 1.15.0
build-cas-client    | Time: 101334ms
build-cas-client    |                                    Asset      Size          Chunks
build-cas-client    |   49453d12ba4db732d2d3da2c3031d543.png   6.04 kB          [emitted]
build-cas-client    |   ca69f7d972f8bdf7094d480d279029ad.eot   12.6 kB          [emitted]
build-cas-client    |   8e3c7f5520f5ae906c6cf6d7f3ddcd19.eot    106 kB          [emitted]
build-cas-client    |   701ae6abd4719e9c2ada3535a497b341.eot   31.2 kB          [emitted]
build-cas-client    |   722a52f42bd09423dfc10ba6bbfacb74.png   2.93 kB          [emitted]
build-cas-client    |   f063a0d61bda917ba1b28e00735b457a.png   5.51 kB          [emitted]
build-cas-client    |   3733b59ced423e7d818c1702ed144cf1.png   4.76 kB          [emitted]
build-cas-client    |   c108eebeff851ade0afb5205fb511cfc.png   4.67 kB          [emitted]
build-cas-client    |   b09f4aa42d7beba3c36761013953fe0c.ttf   12.4 kB          [emitted]
build-cas-client    |   c5ebe0b32dc1b5cc449a76c4204d13bb.ttf   98.4 kB          [emitted]
build-cas-client    |   b87b9ba532ace76ae9f6edfe9f72ded2.ttf    106 kB          [emitted]
build-cas-client    |   ad97afd3337e8cda302d10ff5a4026b8.ttf   30.9 kB          [emitted]
build-cas-client    |   c8f4557fd0be72bbec603b41b10cece7.svg   13.2 kB          [emitted]
build-cas-client    |   1de4669ddea4bb68781282b164ea8d54.svg  695 bytes          [emitted]
build-cas-client    |   a217eb3bd1f34b5180289b1b9a636237.svg  942 bytes          [emitted]
build-cas-client    |   8f592b8d511038b2fe44452cb303a2eb.svg  943 bytes          [emitted]
build-cas-client    |   773ce55d1983d8aec072d76ed3b64ec7.svg   1.75 kB          [emitted]
build-cas-client    |   c556d9148a4c99dfe1c529d24f11053d.svg   1.24 kB          [emitted]
build-cas-client    |   4792d60bca0471862ee5fe6cd1a3588c.svg   3.77 kB          [emitted]
build-cas-client    |   e2c040514156961f18dfeea3ee3d90e1.svg    499 kB          [emitted]
build-cas-client    |   800ac9cf6c344bd5da9ed0fe87c44d7b.svg    378 kB          [emitted]
build-cas-client    |   7037c771cd5b9e7aa4ba46cac98e2655.svg    104 kB          [emitted]
build-cas-client    |   c9ab2b5cdf01bca330a13f150ab5c7e8.png   6.53 kB          [emitted]
build-cas-client    |   2b97bc467d6897114e724d34f0d52673.png   6.57 kB          [emitted]
build-cas-client    |   5841e6948aab9ba90dac0c2efaa90237.png   4.08 kB          [emitted]
build-cas-client    |   5886b993d478267d1552979d19b6853e.png   4.29 kB          [emitted]
build-cas-client    |   b38c89596afdd4e5f62645692433bf6a.png   4.77 kB          [emitted]
build-cas-client    |   3459c6435b984fd41b61b0f87db6379f.png   3.62 kB          [emitted]
build-cas-client    |   696115950dee1bc2088c3eafc80374fd.png   3.92 kB          [emitted]
build-cas-client    |   17682d718991c2864c972b41bf05618a.png   4.94 kB          [emitted]
build-cas-client    |   13db00b7a34fee4d819ab7f9838cc428.eot   98.6 kB          [emitted]
build-cas-client    |   d759f9531c683aff9d5103382b650a50.png   4.32 kB          [emitted]
build-cas-client    |   a76a31eea3899d66115be902082a5e33.png   6.98 kB          [emitted]
build-cas-client    |   4ab94eb7e9a7e33ff1e60d5ef9615aaa.png   3.41 kB          [emitted]
build-cas-client    |   035903890e847884e9db6b2149fc65e9.png   4.02 kB          [emitted]
build-cas-client    |   493fa05a4015ddff23c4ddb4f5a62e3d.png   4.33 kB          [emitted]
build-cas-client    |   d7bc717b8d6f64955a454d82064fb4a6.png   3.29 kB          [emitted]
build-cas-client    |   936336e0921fac79377950a0da0c44ea.png    5.8 kB          [emitted]
build-cas-client    |   47dacc1cee352356787c69397a5c4765.png   5.44 kB          [emitted]
build-cas-client    |   fa295bceb3c581c2601720671816c88.png   6.58 kB          [emitted]
build-cas-client    |   81c53858fd962673a3efe01627a22900.png   4.54 kB          [emitted]
build-cas-client    |   ba8bd474f455cf7c0efa6fdeb77d5dda.png   7.19 kB          [emitted]
build-cas-client    |   bf4596d349b2a9758c9502e32728a800.png   6.26 kB          [emitted]
build-cas-client    |   8ae3c63d28fbfb29352b4a0114423faf.PNG   28.4 kB          [emitted]
build-cas-client    |   9028dc1633670ae8ea6fc7c1c7582fb5.PNG   29.3 kB          [emitted]
build-cas-client    |   75c03f10dee2892a635f1b4c3413c4e5.png   26.1 kB          [emitted]
build-cas-client    |   f80abf36f15111fed60978ff05613faa.PNG   28.6 kB          [emitted]
build-cas-client    |   4b9278c12948367a81154bc80edc4d68.png   1.55 kB          [emitted]
build-cas-client    |   cb43ed86efdb33ac98bcd52d5f9f2f3c.PNG   29.3 kB          [emitted]
build-cas-client    |   9c74e172f87984c48ddf5c8108cabe67.png   28.1 kB          [emitted]
build-cas-client    |  a046592bac8f2fd96e994733faf3858c.woff   63.7 kB          [emitted]
build-cas-client    | e8c322de9658cbeb8a774b6624167c2c.woff2   54.5 kB          [emitted]
build-cas-client    |  faff92145777a3cbaf8e7367b4807987.woff   50.5 kB          [emitted]
build-cas-client    | 0ab54153eeeca0ce03978cc463b257f7.woff2   40.1 kB          [emitted]
build-cas-client    |  ef60a4f6c25ef7f39f2d25a748dbecfe.woff   14.7 kB          [emitted]
build-cas-client    | cd6c777f1945164224dee082abaea03a.woff2   12.2 kB          [emitted]
build-cas-client    |                                 main.js   1.85 MB       0  [emitted]
```

```
build-cas-client    |                          main.js.map    10.5 MB       0  [emitted]
build-cas-client    |                           index.html  916 bytes          [emitted]
build-cas-client    |                   webpack.stats.json    9.01 kB          [emitted]
build-cas-client    |    [0] multi main 40 bytes {0} [built]
build-cas-client    |        factory:1ms building:2ms = 3ms
build-cas-client    |     + 1536 hidden modules
build-cas-client    |
build-cas-client    | WARNING in ./app/actions/actions.js
build-cas-client    |
build-cas-client    | /opt/src/app/actions/actions.js
build-cas-client    |   796:34  warning  Unexpected console statement  no-console
build-cas-client    |
build-cas-client    | ✖ 1 problem (0 errors, 1 warning)
build-cas-client    |
build-cas-client    |
build-cas-client    | WARNING in ./app/components/TranslationComponent.js
build-cas-client    |
build-cas-client    | /opt/src/app/components/TranslationComponent.js
build-cas-client    |   214:16   warning  Unexpected confirm                    no-alert
build-cas-client    |   353:432  warning  Value must be omitted for boolean
build-cas-client    |   attributes react/jsx-boolean-value
build-cas-client    |   392:467  warning  Value must be omitted for boolean
build-cas-client    |   attributes react/jsx-boolean-value
build-cas-client    |   488:224  warning  Value must be omitted for boolean
build-cas-client    |   attributes react/jsx-boolean-value
build-cas-client    |   509:224  warning  Value must be omitted for boolean
build-cas-client    |   attributes react/jsx-boolean-value
build-cas-client    |
build-cas-client    | ✖ 5 problems (0 errors, 5 warnings)
build-cas-client    |
build-cas-client    |
build-cas-client    | WARNING in ./app/components/SectionComponent.js
build-cas-client    |
build-cas-client    | /opt/src/app/components/SectionComponent.js
build-cas-client    |   310:81  warning  Value must be omitted for boolean
build-cas-client    |   attributes /jsx-boolean-value
build-cas-client    |   317:81  warning  Value must be omitted for boolean
build-cas-client    |   attributes react/jsx-boolean-value
build-cas-client    |
build-cas-client    | ✖ 2 problems (0 errors, 2 warnings)
build-cas-server  | [info] Test run finished: 0 failed, 0 ignored, 44 total, 535.489s
build-cas-server  | [info] Passed: Total 84, Failed 0, Errors 0, Passed 84
build-cas-server  | [success] Total time: 1209 s, completed Jun 11, 2018 10:44:28 PM
build-cas-server  | [info] Done packaging.
build-cas-server  | [info] Your package is ready in
build-cas-server  | /opt/src/target/universal/challengeauthoringsystemserver-0.0.1.zip
build-cas-server  | [success] Total time: 21 s, completed Jun 11, 2018 10:44:50 PM
build-cas-server  | [INFO] [06/11/2018 22:44:50.604]
build-cas-server  | [Thread-2] [CoordinatedShutdown(akka://sbt-web)]
build-cas-server  | Starting coordinated shutdown from JVM shutdown hook
```

docker ps:

```
CONTAINER ID        IMAGE                  …      NAMES
58f850cf76b6        hl-cas/cas-mysql       …      cas-mysql
```

Testergebnis OK

Tab. 11 Systemtest - Build Container ausführen – Standalone Mode

Beschreibung	Wert
Test-Nummer	6
Beschreibung	build_CAS Container ausführen und Logs folgen im Docker Swarm Mode (ENABLE_DOCKER_SWARM=true in .env)
Voraussetzung	ENABLE_DOCKER_SWARM=false in .env Docker ist im Swarm Mode und mit Option 1 im Admin Tool wurde Swarm initiiert Docker Netzwerk im «overlay» Modus mit Option 2 im Admin Tool Docker Secret Store mit Option 3 im Admin Tool erstellt

Ausgeführte Schritte	`./admin_tool.sh`
	`Please enter your choice: 5`
	`Please enter your choice: 25`
	`docker service logs cas-build-stack_build-consumer`
	`docker service logs cas-build-stack_build-cas-client`
	`docker service logs cas-build-stack_build-cas-server`
	`docker ps`

Erwartetes Ergebnis	`Creating service cas-common-services-stack_cas-mysql`
	`Creating service cas-build-stack_build-consumer`
	`Creating service cas-build-stack_build-cas-client`
	`Creating service cas-build-stack_build-cas-server`
	Und erfolgreiches Resultat der Buildvorgänge der drei Services ohne Fehler im Log analog der Ausgabe des Tests 5
	• build-consumer erfolgreiche Webpackerstellung
	• build-cas-client erfolgreiche Webpackerstellung
	• build-cas-server erfolgreiche Erstellung des ZIPs
	Nach dem Build-Vorgang sollten alle Services bis auf den cas-mysql beendet sein

Tatsächliches Ergebnis	`Creating service cas-common-services-stack_cas-mysql`
	`Creating service cas-build-stack_build-consumer`
	`Creating service cas-build-stack_build-cas-client`
	`Creating service cas-build-stack_build-cas-server`
	Logs:
	`cas-build-stack_build-consumer | Hash: 821c8a48ec5b847b19e9`
	`cas-build-stack_build-consumer | Version: webpack 1.15.0`
	`cas-build-stack_build-consumer | : 68434ms`
	`cas-build-stack_build-consumer | Asset Size`
	`cas-build-stack_build-consumer | a046592bac8f2fd96e994733faf3858c.woff 63.7 kB [emitted]`
	`cas-build-stack_build-consumer | 13db00b7a34fee4d819ab7f9838cc428.eot 98.6 kB [emitted]`
	`cas-build-stack_build-consumer | 701ae6abd4719e9c2ada3535a497b341.eot 31.2 kB [emitted]`
	`cas-build-stack_build-consumer | 46661d6d65debc63884004fed6e37e5c.svg 41 bytes [emitted]`
	`cas-build-stack_build-consumer | aa4adf7b80b62efdd292096501bcccd5.svg 82 bytes [emitted]`
	`cas-build-stack_build-consumer | c5ebe0b32dc1b5cc449a76c4204d13bb.ttf 98.4 kB [emitted]`
	`cas-build-stack_build-consumer | b87b9ba532ace76ae9f6edfe9f72ded2.ttf 106 kB [emitted]`
	`cas-build-stack_build-consumer | ad97afd3337e8cda302d10ff5a4026b8.ttf 30.9 kB [emitted]`
	`cas-build-stack_build-consumer | 9c74e172f87984c48ddf5c8108cabe67.png 28.1 kB [emitted]`
	`cas-build-stack_build-consumer | 8e3c7f5520f5ae906c6cf6d7f3ddcd19.eot 106 kB [emitted]`
	`cas-build-stack_build-consumer | e8c322de9658cbeb8a774b6624167c2c.woff2 54.5 kB [emitted]`
	`cas-build-stack_build-consumer | faff92145777a3cbaf8e7367b4807987.woff 50.5 kB [emitted]`
	`cas-build-stack_build-consumer | 0ab54153eeeca0ce03978cc463b257f7.woff2 40.1 kB [emitted]`
	`cas-build-stack_build-consumer | ef60a4f6c25ef7f39f2d25a748dbecfe.woff 14.7 kB [emitted]`
	`cas-build-stack_build-consumer | cd6c777f1945164224dee082abaea03a.woff2 12.2 kB [emitted]`
	`cas-build-stack_build-consumer | main.js 1.42 MB [emitted]`
	`cas-build-stack_build-consumer | main.js.map 8.09 MB [emitted]`
	`cas-build-stack_build-consumer | index.html 547 bytes [emitted]`
	`cas-build-stack_build-consumer | webpack.stats.json 3.62 kB [emitted]`
	`cas-build-stack_build-consumer | [0] multi main 40 bytes {0} [built]`
	`cas-build-stack_build-consumer | factory:0ms building:2ms = 2ms`
	`cas-build-stack_build-consumer | + 1427 hidden modules`
	`cas-build-stack_build-consumer |`
	`cas-build-stack_build-consumer | WARNING in ./app/actions/actions.js`

```
cas-build-stack_build-consumer |
cas-build-stack_build-consumer | /opt/src/app/actions/actions.js
cas-build-stack_build-consumer | 36:19  warning  Missing function expression name func-names
cas-build-stack_build-consumer | 37:17  warning  Unexpected console statement    no-console
cas-build-stack_build-consumer | 38:29  warning  Unexpected console statement    no-console
cas-build-stack_build-consumer |
cas-build-stack_build-consumer | ✖ 3 problems (0 errors, 3 warnings
cas-build-stack_build-cas-client | Hash: b3939bcc28e3965f2733
cas-build-stack_build-cas-client | Version: webpack 1.15.0
cas-build-stack_build-cas-client | Time: 104030ms
cas-build-stack_build-cas-client |                                    Asset       Size  Chunks
cas-build-stack_build-cas-client | 49453d12ba4db732d2d3da2c3031d543.png   6.04 kB  [emitted]
cas-build-stack_build-cas-client | ca69f7d972f8bdf7094d480d279029ad.eot   12.6 kB  [emitted]
cas-build-stack_build-cas-client | 8e3c7f5520f5ae906c6cf6d7f3ddcd19.eot    106 kB  [emitted]
cas-build-stack_build-cas-client | 701ae6abd4719e9c2ada3535a497b341.eot   31.2 kB  [emitted]
cas-build-stack_build-cas-client | 722a52f42bd09423dfc10ba6bbfacb74.png   2.93 kB  [emitted]
cas-build-stack_build-cas-client | f063a0d61bda917ba1b28e00735b457a.png   5.51 kB  [emitted]
cas-build-stack_build-cas-client | 3733b59ced423e7d818c1702ed144cf1.png   4.76 kB  [emitted]
cas-build-stack_build-cas-client | c108eebeff851ade0afb5205fb511cfc.png   4.67 kB  [emitted]
cas-build-stack_build-cas-client | b09f4aa42d7beba3c36761013953fe0c.ttf   12.4 kB  [emitted]
cas-build-stack_build-cas-client | c5ebe0b32dc1b5cc449a76c4204d13bb.ttf   98.4 kB  [emitted]
cas-build-stack_build-cas-client | b87b9ba532ace76ae9f6edfe9f72ded2.ttf    106 kB  [emitted]
cas-build-stack_build-cas-client | ad97afd3337e8cda302d10ff5a4026b8.ttf   30.9 kB  [emitted]
cas-build-stack_build-cas-client | c8f4557fd0be72bbec603b41b10cece7.svg   13.2 kB  [emitted]
cas-build-stack_build-cas-client | 1de4669ddea4bb68781282b164ea8d54.svg 695 bytes  [emitted]
cas-build-stack_build-cas-client | a217eb3bd1f34b5180289b1b9a636237.svg 942 bytes  [emitted]
cas-build-stack_build-cas-client | 8f592b8d511038b2fe44452cb303a2eb.svg 943 bytes  [emitted]
cas-build-stack_build-cas-client | 773ce55d1983d8aec072d76ed3b64ec7.svg   1.75 kB  [emitted]
cas-build-stack_build-cas-client | c556d9148a4c99dfe1c529d24f11053d.svg   1.24 kB  [emitted]
cas-build-stack_build-cas-client | 4792d60bca0471862ee5fe6cd1a3588c.svg   3.77 kB  [emitted]
cas-build-stack_build-cas-client | e2c040514156961f18dfeea3ee3d90e1.svg    499 kB  [emitted]
cas-build-stack_build-cas-client | 800ac9cf6c344bd5da9ed0fe87c44d7b.svg    378 kB  [emitted]
cas-build-stack_build-cas-client | 7037c771cd5b9e7aa4ba46cac98e2655.svg    104 kB  [emitted]
cas-build-stack_build-cas-client | c9ab2b5cdf01bca330a13f150ab5c7e8.png   6.53 kB  [emitted]
cas-build-stack_build-cas-client | 2b97bc467d6897114e724d34f0d52673.png   6.57 kB  [emitted]
cas-build-stack_build-cas-client | 5841e6948aab9ba90dac0c2efaa90237.png   4.08 kB  [emitted]
cas-build-stack_build-cas-client | 5886b993d478267d1552979d19b6853e.png   4.29 kB  [emitted]
cas-build-stack_build-cas-client | b38c89596afdd4e5f62645692433bf6a.png   4.77 kB  [emitted]
cas-build-stack_build-cas-client | 3459c6435b984fd41b61b0f87db6379f.png   3.62 kB  [emitted]
cas-build-stack_build-cas-client | 696115950dee1bc2088c3eafc80374fd.png   3.92 kB  [emitted]
cas-build-stack_build-cas-client | 17682d718991c2864c972b41bf05618a.png   4.94 kB  [emitted]
cas-build-stack_build-cas-client | 13db00b7a34fee4d819ab7f9838cc428.eot   98.6 kB  [emitted]
cas-build-stack_build-cas-client | d759f9531c683aff9d5103382b650a50.png   4.32 kB  [emitted]
cas-build-stack_build-cas-client | a76a31eea3899d66115be902082a5e33.png   6.98 kB  [emitted]
cas-build-stack_build-cas-client | 4ab94eb7e9a7e33ff1e60d5ef9615aaa.png   3.41 kB  [emitted]
cas-build-stack_build-cas-client | 035903890e847884e9db6b2149fc65e9.png   4.02 kB  [emitted]
cas-build-stack_build-cas-client | 493fa05a4015ddff23c4ddb4f5a62e3d.png   4.33 kB  [emitted]
cas-build-stack_build-cas-client | d7bc717b8d6f64955a454d82064fb4a6.png   3.29 kB  [emitted]
cas-build-stack_build-cas-client | 936336e0921fac79377950a0da0c44ea.png    5.8 kB  [emitted]
cas-build-stack_build-cas-client | 47dacc1cee352356787c69397a5c4765.png   5.44 kB  [emitted]
cas-build-stack_build-cas-client | fa295bceb3c581c260172067181f6c88.png   6.58 kB  [emitted]
cas-build-stack_build-cas-client | 81c53858fd962673a3efe01627a22900.png   4.54 kB  [emitted]
cas-build-stack_build-cas-client | ba8bd474f455cf7c0efa6fdeb77d5dda.png   7.19 kB  [emitted]
cas-build-stack_build-cas-client | bf4596d349b2a9758c9502e32728a800.png   6.26 kB  [emitted]
cas-build-stack_build-cas-client | 8ae3c63d28fbfb29352b4a0114423faf.PNG   28.4 kB  [emitted]
cas-build-stack_build-cas-client | 9028dc1633670ae8ea6fc7c1c7582fb5.PNG   29.3 kB  [emitted]
cas-build-stack_build-cas-client | 75c03f10dee2892a635f1b4c3413c4e5.png   26.1 kB  [emitted]
cas-build-stack_build-cas-client | f80abf36f15111fed60978ff05613faa.PNG   28.6 kB  [emitted]
cas-build-stack_build-cas-client | 4b9278c12948367a81154bc80edc4d68.png   1.55 kB  [emitted]
cas-build-stack_build-cas-client | cb43ed86efdb33ac98bcd52d5f9f2f3c.PNG   29.3 kB  [emitted]
cas-build-stack_build-cas-client | 9c74e172f87984c48ddf5c8108cabe67.png   28.1 kB  [emitted]
cas-build-stack_build-cas-client | a046592bac8f2fd96e994733faf3858c.woff  63.7 kB  [emitted]
cas-build-stack_build-cas-client | e8c322de9658cbeb8a774b6624167c2c.woff2 54.5 kB  [emitted]
cas-build-stack_build-cas-client | faff92145777a3cbaf8e7367b4807987.woff  50.5 kB  [emitted]
cas-build-stack_build-cas-client | 0ab54153eeeca0ce03978cc463b257f7.woff2 40.1 kB  [emitted]
cas-build-stack_build-cas-client | ef60a4f6c25ef7f39f2d25a748dbecfe.woff  14.7 kB  [emitted]
```

```
cas-build-stack_build-cas-client | cd6c777f1945164224dee082abaea03a.woff2 12.2 kB  [emitted]
cas-build-stack_build-cas-client |                              main.js   1.85 MB  [emitted]
cas-build-stack_build-cas-client |                          main.js.map   10.5 MB  [emitted]
cas-build-stack_build-cas-client |                           index.html 916 bytes  [emitted]
cas-build-stack_build-cas-client |                     webpack.stats.json   9.01 kB  [emitted]
cas-build-stack_build-cas-client |    [0] multi main 40 bytes {0} [built]
cas-build-stack_build-cas-client |        factory:1ms building:2ms = 3ms
cas-build-stack_build-cas-client |     + 1536 hidden modules
cas-build-stack_build-cas-client |
cas-build-stack_build-cas-client | WARNING in ./app/actions/actions.js
cas-build-stack_build-cas-client |
cas-build-stack_build-cas-client | /opt/src/app/actions/actions.js
cas-build-stack_build-cas-client | 796:34  warning  Unexpected console statement  no-console
cas-build-stack_build-cas-client |
cas-build-stack_build-cas-client | ✖ 1 problem (0 errors, 1 warning)
cas-build-stack_build-cas-client |
cas-build-stack_build-cas-client |
cas-build-stack_build-cas-client | WARNING in ./app/components/TranslationComponent.js
cas-build-stack_build-cas-client |
cas-build-stack_build-cas-client | /opt/src/app/components/TranslationComponent.js
cas-build-stack_build-cas-client | 214:16 warning Unexpected confirm               no-alert
cas-build-stack_build-cas-client | 353:432 warning Value must be omitted for boolean
cas-build-stack_build-cas-client | attributes  react/jsx-boolean-value
cas-build-stack_build-cas-client | 392:467 warning Value must be omitted for boolean
cas-build-stack_build-cas-client | attributes  react/jsx-boolean-value
cas-build-stack_build-cas-client | 488:224 warning Value must be omitted for boolean
cas-build-stack_build-cas-client | attributes  react/jsx-boolean-value
cas-build-stack_build-cas-client | 509:224 warning Value must be omitted for boolean
cas-build-stack_build-cas-client | attributes  react/jsx-boolean-value
cas-build-stack_build-cas-client |
cas-build-stack_build-cas-client | ✖ 5 problems (0 errors, 5 warnings)
cas-build-stack_build-cas-client |
cas-build-stack_build-cas-client | WARNING in ./app/components/SectionComponent.js
cas-build-stack_build-cas-client |
cas-build-stack_build-cas-client | /opt/src/app/components/SectionComponent.js
cas-build-stack_build-cas-client |   310:81  warning  Value must be omitted for boolean
cas-build-stack_build-cas-client | attributes  react/jsx-boolean-value
cas-build-stack_build-cas-client |   317:81  warning  Value must be omitted for boolean
cas-build-stack_build-cas-client | attributes  react/jsx-boolean-value
cas-build-stack_build-cas-client |
cas-build-stack_build-cas-client | ✖ 2 problems (0 errors, 2 warnings)
cas-build-stack_build-cas-server | [info] Test run finished: 0 failed, 0 ignored, 44 total,
                                   542.286s
cas-build-stack_build-cas-server | [info] Passed: Total 84, Failed 0, Errors 0, Passed 84
cas-build-stack_build-cas-server | [success] Total time: 1224 s, completed Jun 11, 2018
                                   11:31:23 PM
cas-build-stack_build-cas-server | [info] Done packaging.
cas-build-stack_build-cas-server | [info] Your package is ready in
                                   /opt/src/target/universal/challengeauthoringsystemserver-0.0.1.zip
cas-build-stack_build-cas-server | [success] Total time: 21 s, completed Jun 11, 2018 23:31:44
cas-build-stack_build-cas-server    |    [INFO]    [06/11/2018    23:31:44.685]    [Thread-2]
[CoordinatedShutdown(akka://sbt-web)] Starting coordinated shutdown from JVM shutdown hoo
```

docker ps:

```
CONTAINER ID  IMAGE                     NAMES
adebadf9f234  hl-cas/cas-mysql:latest   cas-common-services-stack_cas-mysql.1.yb0j0i…
```

Testergebnis	OK

Tab. 12 Systemtest - Build Container ausführen – Swarm Mode

6.4.1.5 Run Container ausführen

Beschreibung	Wert
Test-Nummer	7
Beschreibung	run_CAS Container ausführen und Logs folgen mithilfe des Admin Tools im Docker Standalone Mode (ENABLE_DOCKER_SWARM=false in .env)
Voraussetzung	Die Voraussetzungen sind: ENABLE_DOCKER_SWARM=false Docker Netzwerk im «bridge» Modus mit Option 2 im Admin Tool Docker Secrets Dateien anstelle Secret Store mit Option 3 im Admin Tool Build wurde erfolgreich durchgeführt

Ausgeführte Schritte	`./admin_tool.sh`
	`Please enter your choice: 7`
	`Please enter your choice: 12`
	`Please enter your choice: 25`
	`docker ps`

Erwartetes Ergebnis	`Creating run-cas-server ... done`
	`Creating run-consumer ... done`
	`Creating cas-traefik ... done`
	`Creating cas-keycloak-mysql ... done`
	`Creating run-cas-client ... done`
	`Creating cas-ldap ... done`
	`Creating cas-keycloak ... done`
	`Creating proxy-cas-server ... done`
	`Creating proxy-consumer ... done`
	`Creating proxy-cas-client ... done`
	`Creating cas-phpldapadmin ... done`
	Und erfolgreiche Starts der Services ohne Fehler im Log
	Anschliessend sollten alle run Container gestartet sein

Tatsächliches Ergebnis	`Creating run-cas-server ... done`
	`Creating run-consumer ... done`
	`Creating cas-traefik ... done`
	`Creating cas-keycloak-mysql ... done`
	`Creating run-cas-client ... done`
	`Creating cas-ldap ... done`
	`Creating cas-keycloak ... done`
	`Creating proxy-cas-server ... done`
	`Creating proxy-consumer ... done`
	`Creating proxy-cas-client ... done`
	`Creating cas-phpldapadmin ... done`
	Logs:
	`run-cas-server | [info] p.c.s.AkkaHttpServer - Listening for HTTP on /0.0.0.0:9000`
	`run-consumer | Running at Port 3000`
	`cas-traefik | time="2018-06-11T21:39:21Z" level=warning msg="top-level traefikLogsFile has been deprecated -- please use traefiklog.filepath"`
	`cas-keycloak-mysql | [Entrypoint] Starting MySQL 5.7.21-1.1.3`
	`run-cas-client | Running at Port 3001`
	`cas-ldap | *** Running /container/run/process/slapd/run...`
	`cas-ldap | 5b1eec0d @(#) $OpenLDAP: slapd (Aug 10 2017 19:12:46) $`
	`cas-ldap | Debian OpenLDAP Maintainers <pkg-openldap-devel@lists.alioth.d>`
	`cas-ldap | TLS: warning: ignoring dhfile`
	`cas-ldap | 5b1eec0d slapd starting`
	`cas-keycloak | 23:40:26,051 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0060: Http management interface listening on http://0.0.0.0:9990/management`
	`cas-keycloak | 23:40:26,051 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0051: Admin console listening on http://0.0.0.0:9990`
	`cas-keycloak | 23:40:26,052 INFO [org.jboss.as] (Controller Boot Thread)`

```
                        WFLYSRV0025: Keycloak 4.0.0.Beta3 (WildFly Core 3.0.8.Final)
                        started in 27711ms - Started 546 of 882 services
                        (604 services are lazy, passive or on-demand)
proxy-cas-server    | info successfully retrieved openid configuration from the discovery
proxy-cas-server    | info enabled reverse proxy mode, upstream url
                       {"url": "http://run-cas-server:9000"}
proxy-cas-server    | info    protecting resource
                       {"resource": "uri: /*, methods: DELETE,GET,HEAD,OPTIONS,
                        PATCH,POST,PUT,TRACE, required: hl_user"}
proxy-cas-server    | info    keycloak proxy service starting {"interface": ":3000"}
proxy-consumer      | info successfully retrieved openid configuration from the discovery
proxy-consumer      | info    enabled reverse proxy mode, upstream url
                      {"url": "http://run-consumer:3000"}
proxy-consumer      | info    protecting resource
                      {"resource": "uri: /*, methods: DELETE,GET,HEAD,OPTIONS,
                       PATCH,POST,PUT,TRACE, required: hl_user"}
proxy-consumer      | info    keycloak proxy service starting {"interface": ":3000"}
proxy-cas-client    | info successfully retrieved openid configuration from the discovery
proxy-cas-client    | info    enabled reverse proxy mode, upstream url
                      {"url": "http://run-cas-client:3001"}
proxy-cas-client    | info    protecting resource
                      {"resource": "uri: /*, methods: DELETE,GET,HEAD,OPTIONS,
                       PATCH,POST,PUT,TRACE, required: hl_user"}
proxy-cas-client    | info    keycloak proxy service starting {"interface": ":3000"}
cas-phpldapadmin    | *** Running runit daemon...
cas-phpldapadmin    | *** runit daemon started as PID 1022
```

docker ps

```
CONTAINER ID        IMAGE                                 NAMES
423f9515f95d        osixia/phpldapadmin:latest            cas-phpldapadmin
2dd1820ffd7d        hl-cas/keycloak-auth-proxy-golang     proxy-cas-client
9228ed8aadec        hl-cas/keycloak-auth-proxy-golang     proxy-consumer
cac4f4f17c49        hl-cas/keycloak-auth-proxy-golang     proxy-cas-server
7e728eefe50f        hl-cas/cas-keycloak                   cas-keycloak
87a5fbe6af5c        hl-cas/cas-ldap                       cas-ldap
47e575cb50aa        hl-cas/run-cas-client                 run-cas-client
508b8543c871        hl-cas/cas-keycloak-mysql             cas-keycloak-mysql
8a2859372102        hl-cas/cas-traefik                    cas-traefik
c253826ed463        hl-cas/run-consumer                   run-consumer
b8b5c657762c        hl-cas/run-cas-server                 run-cas-server
bb4186a47d47        hl-cas/cas-mysql                      cas-mysql
```

Testergebnis	OK

Tab. 13 Systemtest - Run Container ausführen – Standalone Mode

Beschreibung	Wert
Test-Nummer	8
Beschreibung	run_CAS Container ausführen und Logs folgen mithilfe des Admin Tools im Docker Swarm Mode (ENABLE_DOCKER_SWARM=true in .env)
Voraussetzung	Die Voraussetzungen sind: ENABLE_DOCKER_SWARM=true Docker Netzwerk im «overlay» Modus mit Option 2 im Admin Tool Docker Secrets Store anstelle Secret Dateien mit Option 3 im Admin Tool Build wurde erfolgreich durchgeführt

Ausgeführte Schritte	`./admin_tool.sh`
	`Please enter your choice: 7`
	`Please enter your choice: 25`
	`docker service logs cas-run-stack-run-cas-server`
	`docker service logs cas-run-stack-run-consumer`
	`docker service logs cas-run-stack-cas-traefik`
	`docker service logs cas-run-stack-cas-keycloak-mysql`
	`docker service logs cas-run-stack-run-cas-client`
	`docker service logs cas-run-stack-cas-ldap`
	`docker service logs cas-run-stack-cas-keycloak`
	`docker service logs cas-run-stack-proxy-cas-server`
	`docker service logs cas-run-stack-proxy-consumer`
	`docker service logs cas-run-stack-proxy-cas-client`
	`docker service logs cas-run-stack-cas-phpldapadmin`
	`docker ps`

Erwartetes Ergebnis	`Creating service cas-common-services-stack_cas-mysql`
	`Creating service cas-run-stack_proxy-cas-server`
	`Creating service cas-run-stack_cas-phpldapadmin`
	`Creating service cas-run-stack_cas-keycloak-mysql`
	`Creating service cas-run-stack_cas-traefik`
	`Creating service cas-run-stack_run-consumer`
	`Creating service cas-run-stack_cas-ldap`
	`Creating service cas-run-stack_proxy-cas-client`
	`Creating service cas-run-stack_run-cas-client`
	`Creating service cas-run-stack_run-cas-server`
	`Creating service cas-run-stack_cas-keycloak`
	`Creating service cas-run-stack_proxy-consumer`
	Und erfolgreiche Starts der Services ohne Fehler im Log
	Anschliessend sollten alle run Container gestartet sein

Tatsächliches Ergebnis	`Creating service cas-common-services-stack_cas-mysql`
	`Creating service cas-run-stack_proxy-cas-server`
	`Creating service cas-run-stack_cas-phpldapadmin`
	`Creating service cas-run-stack_cas-keycloak-mysql`
	`Creating service cas-run-stack_cas-traefik`
	`Creating service cas-run-stack_run-consumer`
	`Creating service cas-run-stack_cas-ldap`
	`Creating service cas-run-stack_proxy-cas-client`
	`Creating service cas-run-stack_run-cas-client`
	`Creating service cas-run-stack_run-cas-server`
	`Creating service cas-run-stack_cas-keycloak`
	`Creating service cas-run-stack_proxy-consumer`
	Logs:
	`cas-common-services-stack_cas-mysql | [Entrypoint] MySQL Docker Image 5.7.21-1.1.3`
	`cas-common-services-stack_cas-mysql | [Entrypoint] Starting MySQL 5.7.21-1.1.3`
	`cas-run-stack_proxy-cas-server | info successfully retrieved openid configuration from the discovery`
	`cas-run-stack_proxy-cas-server | info enabled reverse proxy mode, upstream url {"url": "http://run-cas-server:9000"}`

```
cas-run-stack_proxy-cas-server | info protecting resource
                                {"resource": "uri: /*, methods: DELETE,GET,HEAD,
                                 OPTIONS,PATCH,POST,PUT,TRACE, required: hl_user"}
cas-run-stack_proxy-cas-server | info keycloak proxy service starting
                                {"interface": ":3000"}
cas-run-stack_cas-phpldapadmin | *** Running runit daemon...
cas-run-stack_cas-phpldapadmin | *** runit daemon started as PID 1022
cas-run-stack_cas-keycloak-mysql | [Entrypoint] MySQL Docker Image 5.7.21-1.1.3
cas-run-stack_cas-keycloak-mysql | [Entrypoint] Starting MySQL 5.7.21-1.1.3
cas-run-stack_cas-traefik      | time="2018-06-11T22:11:08Z" level=warning
                                 msg="top-level traefikLogsFile has been deprecated -
                                 please use traefiklog.filepath"
cas-run-stack_run-consumer     | Running at Port 3000
cas-run-stack_cas-ldap | *** Running /container/run/process/slapd/run...
cas-run-stack_cas-ldap | 5b1ef381 @(#) $OpenLDAP: slapd  (Aug 10 2017 19:12:46) $
cas-run-stack_cas-ldap | Debian OpenLDAP Maintainers
                        <pkg-openldap-devel@lists.alioth.debian.org>
cas-run-stack_cas-ldap | TLS: warning: ignoring dhfile
cas-run-stack_cas-ldap | 5b1ef382 slapd starting
cas-run-stack_proxy-cas-client | info  successfully retrieved openid configuration
                                 from the discovery
cas-run-stack_proxy-cas-client | info  enabled reverse proxy mode, upstream url
                                {"url": "http://run-cas-client:3001"}
cas-run-stack_proxy-cas-client | info  protecting resource
                                {"resource": "uri: /*, methods: DELETE,GET,HEAD,
                                 OPTIONS,PATCH,POST,PUT,TRACE, required: hl_user"}
cas-run-stack_proxy-cas-client | info  keycloak proxy service starting
                                 {"interface": ":3000"}
cas-run-stack_run-cas-client   | Running at Port 3001
cas-run-stack_run-cas-server   | [info] p.c.s.AkkaHttpServer
                                 - Listening for HTTP on /0.0.0.0:9000
cas-run-stack_cas-keycloak     | INFO WFLYSRV0060: Http management interface
                                 listening on http://0.0.0.0:9990/management
cas-run-stack_cas-keycloak     | INFO WFLYSRV0051: Admin console
                                 listening on http://0.0.0.0:9990
cas-run-stack_cas-keycloak    | INFO WFLYSRV0025: Keycloak 4.0.0.Beta3
                                (WildFly Core 3.0.8.Final) started in 37290ms - Started
                                546 of 882 services (604 services are lazy,
                                passive or on-demand)
cas-run-stack_proxy-consumer  | info  successfully retrieved openid configuration
                                from the discovery
cas-run-stack_proxy-consumer  | info  enabled reverse proxy mode, upstream url
                                {"url": "http://run-consumer:3000"}
cas-run-stack_proxy-consumer  | info  protecting resource
                                {"resource": "uri: /*, methods: DELETE,GET,HEAD,
                                 OPTIONS,PATCH,POST,PUT,TRACE, required: hl_user"}
cas-run-stack_proxy-consumer  | info  keycloak proxy service starting
                                {"interface": ":3000"}
```

docker ps

```
IMAGE                                        NAMES
hl-cas/keycloak-auth-proxy-golang:latest     cas-run-stack_proxy-cas-client.1.s5
hl-cas/keycloak-auth-proxy-golang:latest     cas-run-stack_proxy-cas-server.1.si
hl-cas/keycloak-auth-proxy-golang:latest     cas-run-stack_proxy-consumer.1.q4l5
hl-cas/cas-keycloak:latest                   cas-run-stack_cas-keycloak.1.hdg8pq
hl-cas/run-cas-server:latest                 cas-run-stack_run-cas-server.1.1t4t
hl-cas/run-cas-client:latest                 cas-run-stack_run-cas-client.1.zfax
hl-cas/cas-ldap:latest                       cas-run-stack_cas-ldap.1.rznb8dxe7t
hl-cas/run-consumer:latest                   cas-run-stack_run-consumer.1.2s73ij
hl-cas/cas-traefik:latest                    cas-run-stack_cas-traefik.1.gnmfqcy
hl-cas/cas-keycloak-mysql:latest             cas-run-stack_cas-keycloak-mysql.1.
osixia/phpldapadmin:latest                   cas-run-stack_cas-phpldapadmin.1.se
hl-cas/cas-mysql:latest                      cas-common-services-stack_cas-mysql
```

Testergebnis	OK

Tab. 14 Systemtest - Run Container ausführen – Swarm Mode

6.4.1.6 Traefik Frontend-Rule setzen

Beschreibung	Wert
Test-Nummer	9
Beschreibung	Traefik Frontend Rules anhand Docker-Labels via .env-Datei setzen im Standalone Mode (ENABLE_DOCKER_SWARM=false in .env)
Ausgeführte Schritte	Folgende Werte wurden in der .env-Datei wie folgt konfiguriert: • ADDRESS_PHPLDAPADMIN = ldapadmin.hl.ygubler.ch • ADDRESS_KEYCLOAK = auth.hl.ygubler.ch • CUSTOM_RESPONSE_HEADERS_CAS_KEYCLOAK = Access-Control-Allow-Origin:null||Access-Control-Allow-Credentials:true||Access-Control-Allow-Headers:Content-Type,Challenge-Language • KEYCLOAK_PROXY_PORT = 3000 • ADDRESS_CAS_SERVER = cas-server.hl.ygubler.ch • ADDRESS_CAS_CLIENT = cas-client.hl.ygubler.ch • ADDRESS_CONSUMER = ccs-test.hl.ygubler.ch

Erwartetes Ergebnis	Für die fünf Container mit Traefik-Labels soll auf dem Traefik-Dashboard (http://152.96.56.43:8080/dashboard/) eine Frontend Regel mit den Werten aus den ausgeführten Schritten angezeigt werden. Die Zuweisung der Werten an die Container sieht wie folgt aus:
	• cas-phpldapadmin
	  ○ ADDRESS_PHPLDAPADMIN
	• cas-keycloak
	  ○ ADDRESS_KEYCLOAK
	  ○ CUSTOM_RESPONSE_HEADERS_CAS_KEYCLOAK
	• proxy-cas-server
	  ○ KEYCLOAK_PROXY_PORT
	  ○ ADDRESS_CAS_SERVER
	• proxy-cas-client
	  ○ KEYCLOAK_PROXY_PORT
	  ○ ADDRESS_CAS_CLIENT
	• proxy-consumer
	  ○ KEYCLOAK_PROXY_PORT
	  ○ ADDRESS_CONSUMER

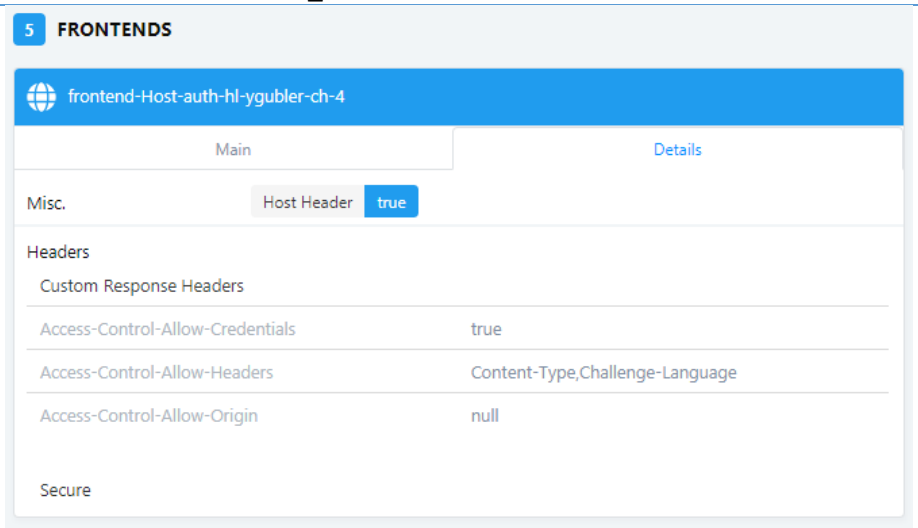
Tatsächliches Ergebnis	

Abb. 68 Systemtest - Traefik Dashboard

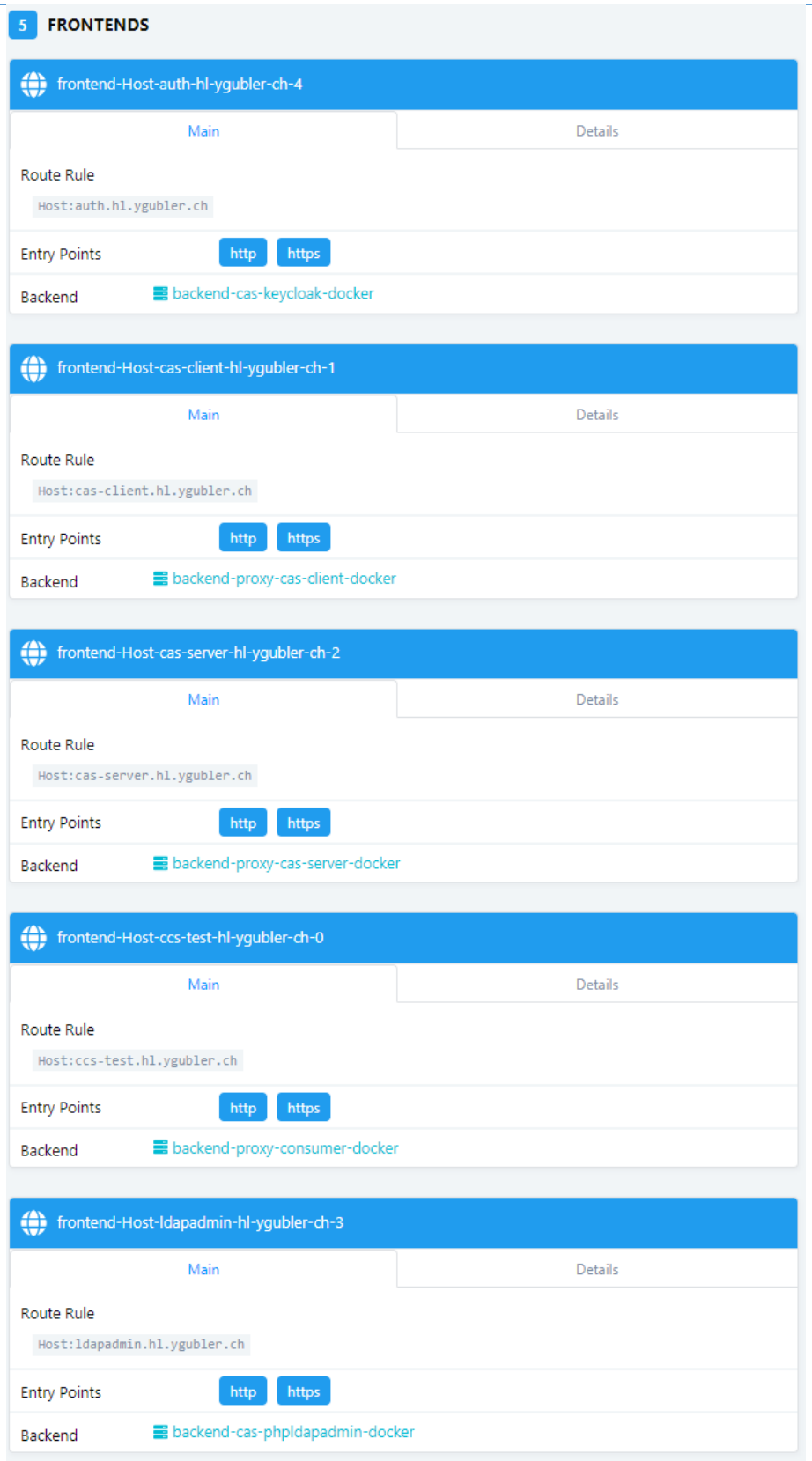


Abb. 69 Systemtest - Traefik Dashboard

Testergebnis	OK

Tab. 15 Systemtest - Traefik Frontend-Rule setzen -Standalone Mode

Beschreibung	Wert
Test-Nummer	10
Beschreibung	Traefik Frontend Rules anhand Docker-Labels via .env-Datei setzen im Swarm Mode (ENABLE_DOCKER_SWARM=true in .env)
Ausgeführte Schritte	Folgende Werte wurden in der .env-Datei wie folgt konfiguriert: • ADDRESS_PHPLDAPADMIN = ldapadmin.hl.ygubler.ch • ADDRESS_KEYCLOAK = auth.hl.ygubler.ch • CUSTOM_RESPONSE_HEADERS_CAS_KEYCLOAK = Access-Control-Allow-Origin:null||Access-Control-Allow-Credentials:true||Access-Control-Allow-Headers:Content-Type,Challenge-Language • KEYCLOAK_PROXY_PORT = 3000 • ADDRESS_CAS_SERVER = cas-server.hl.ygubler.ch • ADDRESS_CAS_CLIENT = cas-client.hl.ygubler.ch • ADDRESS_CONSUMER = ccs-test.hl.ygubler.ch

Erwartetes Ergebnis	Für die fünf Container mit Traefik-Labels soll auf dem Traefik-Dashboard (http://152.96.56.43:8080/dashboard/) eine Frontend Regel mit den Werten aus den ausgeführten Schritten angezeigt werden. Die Zuweisung der Werten an die Container sieht wie folgt aus:
	• cas-phpldapadmin
	  ○ ADDRESS_PHPLDAPADMIN
	• cas-keycloak
	  ○ ADDRESS_KEYCLOAK
	  ○ CUSTOM_RESPONSE_HEADERS_CAS_KEYCLOAK
	• proxy-cas-server
	  ○ KEYCLOAK_PROXY_PORT
	  ○ ADDRESS_CAS_SERVER
	• proxy-cas-client
	  ○ KEYCLOAK_PROXY_PORT
	  ○ ADDRESS_CAS_CLIENT
	• proxy-consumer
	  ○ KEYCLOAK_PROXY_PORT
	  ○ ADDRESS_CONSUMER

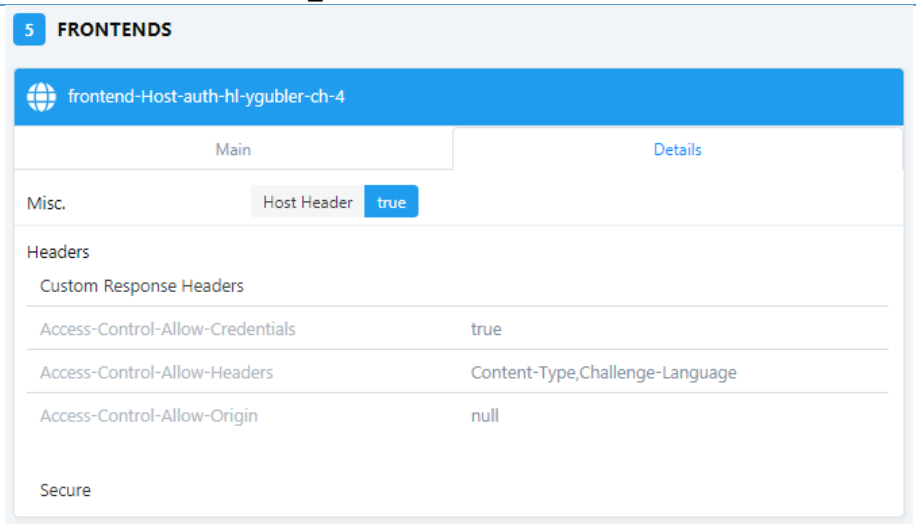
Tatsächliches Ergebnis	

Abb. 70 Systemtest - Traefik Dashboard

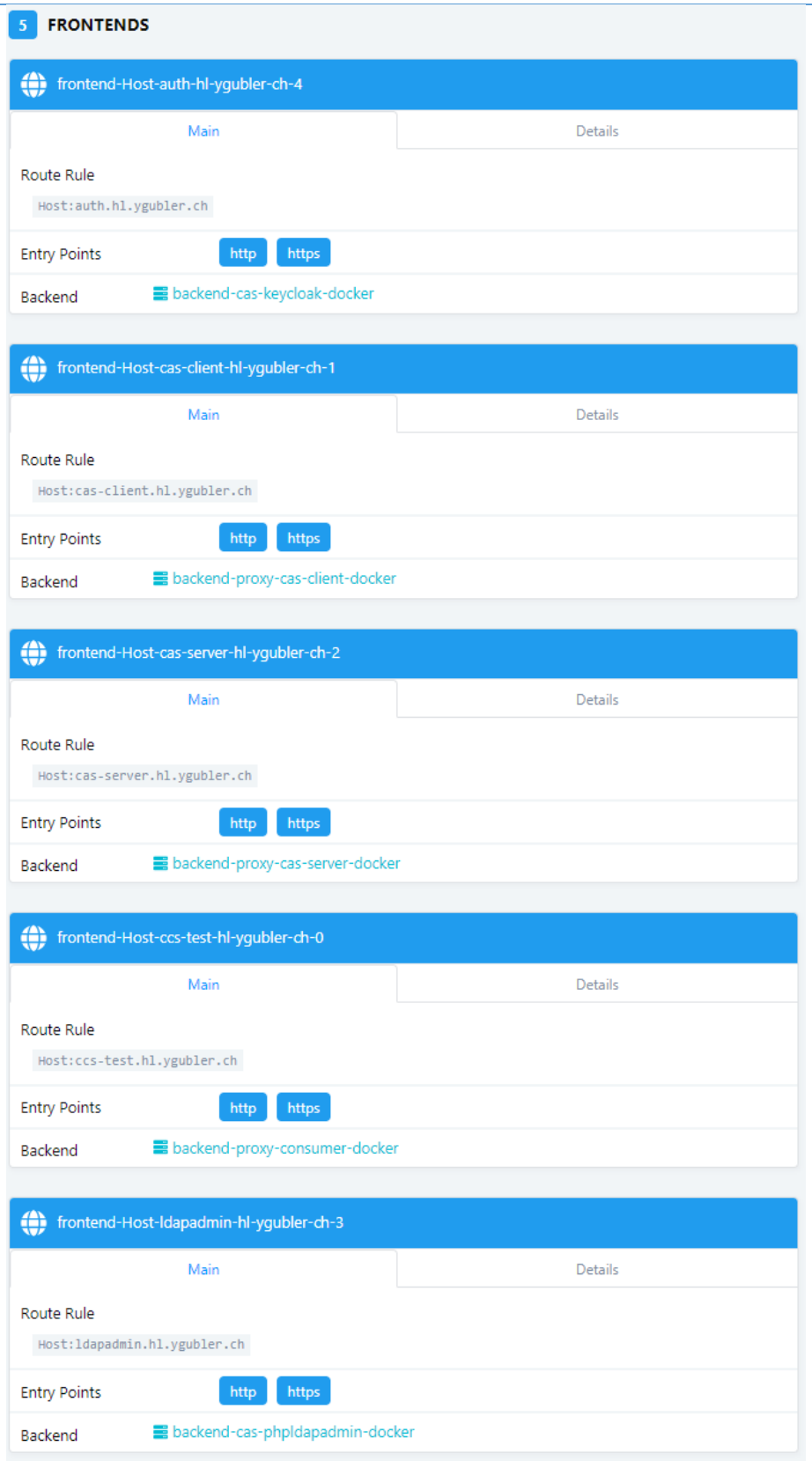


Abb. 71 Systemtest - Traefik Dashboard

Testergebnis	OK

Tab. 16 Systemtest - Traefik Frontend-Rule setzen - Swarm Mode

6.4.1.7 Step by Step Anleitung – Einstellungen der Bachelorarbeit

Beschreibung	Wert
Test-Nummer	11
Beschreibung	Dieser Test bezieht sich auf die Schritt für Schritt Anleitung für das Aufsetzen der Umgebung mit den in der Bachelorarbeit verwendeten Einstellungen und Domänennamen. Dabei wird die Umgebung im Standalone Mode betrieben (ENABLE_DOCKER_SWARM=false in .env)
Ausgeführte Schritte	Für diesen Test wurde das Git-Repository in `/opt/hl2/` geklont und alle Schritte gemäss der Anleitung im Kapitel 6.5.6 ausgeführt.
Erwartetes Ergebnis	Die Umgebung ist vollumfänglich verfügbar. Die zu überprüfenden Schritte sind der Anleitung zu entnehmen.
Tatsächliches Ergebnis	Die Umgebung konnte mit der neuen Umgebung hochgefahren werden. Dabei stimmt diese mit der Anleitung überein.
Testergebnis	OK

Tab. 17 Systemtest - Step by Step Anleitung - Einstellungen der Bachelorarbeit - Standalone Mode

Beschreibung	Wert
Test-Nummer	12
Beschreibung	Dieser Test bezieht sich auf die Schritt für Schritt Anleitung für das Aufsetzen der Umgebung mit den in der Bachelorarbeit verwendeten Einstellungen und Domänennamen. Dabei wird die Umgebung im Swarm Mode betrieben (ENABLE_DOCKER_SWARM=true in .env)
Ausgeführte Schritte	Für diesen Test wurde das Git-Repository in `/opt/hl2/` geklont und alle Schritte gemäss der Anleitung im Kapitel 6.5.6 ausgeführt.
Erwartetes Ergebnis	Die Umgebung ist vollumfänglich verfügbar. Die zu überprüfenden Schritte sind der Anleitung zu entnehmen.
Tatsächliches Ergebnis	Die Umgebung konnte mit der neuen Umgebung hochgefahren werden. Dabei stimmt diese mit der Anleitung überein.
Testergebnis	OK

Tab. 18 Systemtest - Step by Step Anleitung - Einstellungen der Bachelorarbeit - Swarm Mode

6.4.1.8 Step by Step Anleitung – für neue Domänennamen

Beschreibung	Wert
Test-Nummer	13
Beschreibung	Dieser Test bezieht sich auf die Schritt für Schritt Anleitung für das Aufsetzen der Umgebung mit einem anderen Domänennamen als jener, welcher standardmässig in der .env-Datei enthalten ist. Dabei wird die Umgebung im Standalone Mode betrieben (ENABLE_DOCKER_SWARM=false in .env)
Ausgeführte Schritte	Für diesen Test wurde das Git-Repository in `/opt/hl2/` geklont und alle Schritte gemäss der Anleitung im Kapitel 6.5.5 ausgeführt. Dabei wurde der Domänenname hl2.ygubler.ch verwendet. Der Host-Teil der FQDNs der Dienste wurde so belassen wie er standardmässig eingestellt ist. (z.B. cas-client.hl.ygubler.ch → cas-client.hl2.ygubler.ch). Um dies zu realisieren wurde auf dem DNS-Server zwei weiterer DNS-A-Record (*.hl2.ygubler.ch und hl2.ygubler.ch) erfasst, welcher auf die gleiche Adresse wie hl.ygubler.ch zeigen.
Erwartetes Ergebnis	Die Umgebung ist mit dem neuen Domänennamen vollumfänglich verfügbar. Die zu überprüfenden Schritte sind der Anleitung zu entnehmen.
Tatsächliches Ergebnis	Die Umgebung konnte mit der neuen Umgebung hochgefahren werden. Dabei stimmt diese mit der Anleitung überein.
Testergebnis	OK

Tab. 19 Systemtest - Step by Step Anleitung – für neue Domänennamen - Standalone Mode

Beschreibung	Wert
Test-Nummer	14
Beschreibung	Dieser Test bezieht sich auf die Schritt für Schritt Anleitung für das Aufsetzen der Umgebung mit einem anderen Domänennamen als jener, welcher standardmässig in der .env-Datei enthalten ist. Dabei wird die Umgebung im Swarm Mode betrieben (ENABLE_DOCKER_SWARM=true in .env)
Ausgeführte Schritte	Für diesen Test wurde das Git-Repository in `/opt/hl2/` geklont und alle Schritte gemäss der Anleitung im Kapitel 6.5.5 ausgeführt. Dabei wurde der Domänenname hl2.ygubler.ch verwendet. Der Host-Teil der FQDNs der Dienste wurde so belassen wie er standardmässig eingestellt ist. (z.B. cas-client.hl.ygubler.ch → cas-client.hl2.ygubler.ch). Um dies zu realisieren wurde auf dem DNS-Server zwei weiterer DNS-A-Record (*.hl2.ygubler.ch und hl2.ygubler.ch) erfasst, welcher auf die gleiche Adresse wie hl.ygubler.ch zeigen.
Erwartetes Ergebnis	Die Umgebung ist mit dem neuen Domänennamen vollumfänglich verfügbar. Die zu überprüfenden Schritte sind der Anleitung zu entnehmen.
Tatsächliches Ergebnis	Die Umgebung konnte mit der neuen Umgebung hochgefahren werden. Dabei stimmt diese mit der Anleitung überein.
Testergebnis	OK

Tab. 20 Systemtest - Step by Step Anleitung – für neue Domänennamen - Swarm Mode

6.4.2 Dokumentation der UI Tests

Die UI Tests werden anhand der vier Use Cases für den Editor durchgeführt. Die zusätzlichen Use Cases werden wegen ihrer Simplizität anhand von Unit Tests direkt im Server Code geprüft und finden aus diesem Grund in diesem Kapitel keinen Platz.

Beschreibung	Wert
Test-Nummer	1
Beschreibung	Ein Benutzer mit der hl_editor Rolle kann eine neue Challenge erstellen.
Ausgeführte Schritte	1. Der Tester meldet sich mit dem Benutzernamen «test_editor» und dem Passwort «test» auf der Seite https://cas-client.hl.ygubler.ch an. 2. Der Tester wählt aus dem Dropdown-Control den Consumer «Test» aus und bestätigt seine Auswahl mit einem Klick auf «Submit». 3. Der Tester klickt auf den «Create New Challenge» Button. 4. Der Tester gibt in das «Name»-Feld den Wert «Dies ist eine Test-Challenge» ein und bestätigt seine Eingabe mit einem Klick auf «Submit». 5. Der Tester gibt in das «Titel»-Feld den Wert «SQL Injection» ein. 6. Der Tester wählt ein Bild aus seiner Galerie aus befördert es per Drag & Drop in den Bereich «Drop files here to upload» und bestätigt seine Eingabe mit einem Klick auf «Submit». 7. Der Tester wählt aus dem DropDown den Typ «jeopardy» aus. 8. Der Tester wählt das Level «Medium» aus. 9. Der Tester klickt auf das «Hantel»-Symbol und bestätigt seine Eingabe mit einem Klick auf «Submit». 10. Der Tester wählt aus dem Dropdown-Control die Kategorie «Reverse Engineering», sowie «Network Security» aus und bestätigt seine Eingabe mit einem Klick auf «Submit». 11. Der Tester gibt das Keyword «testchallenge» ein bestätigt seine Eingabe mit «Enter» und klickt auf den «Submit» Button. 12. Der Tester klickt auf das «Yes» Label. 13. Der Tester gibt in den Editor für das Abstract «Dies ist eine Test-Challenge» ein und bestätigt seine Eingabe mit einem Klick auf «Submit». 14. Der Tester klickt auf den «Add Instruction» Button. 15. Der Tester gibt den Text «Dies ist eine Instruction» ein und bestätigt seine Eingabe durch einen Klick auf «Finish Section». 16. Der Tester klickt auf den «Select None»-Button und bestätigt seine Eingabe mit einem Klick auf «Submit». 17. Der Tester gibt in den Editor «Solution» ein und bestätigt seine Eingabe mit einem Klick auf den «Submit»-Button. 18. Der Tester kontrolliert die Werte im Summary und bestätigt die Eingabe mit einem Klick auf den «Check Summary»-Button. 19. Der Tester klickt auf das grüne Symbol
Erwartetes Ergebnis	Die Challenge kann mit dem Editor ohne Fehler erstellt werden. Die erstellte Challenge wird danach in der Übersicht angezeigt.
Tatsächliches Ergebnis	Die Challenge konnte korrekt erstellt werden und wurde auf dem Consumer in der Übersicht angezeigt
Testergebnis	OK

Tab. 21 UI-Test - Challenge erstellen

Beschreibung	Wert
Test-Nummer	2
Beschreibung	Ein Benutzer mit der hl_editor Rolle kann eine bestehende Challenge bearbeiten.
Ausgeführte Schritte	1. Der Tester meldet sich mit dem Benutzernamen «test_editor» und dem Passwort «test» auf der Seite https://cas-client.hl.ygubler.ch an. 2. Der Tester wählt aus dem Dropdown-Control den Consumer «Test» aus und bestätigt seine Auswahl mit einem Klick auf «Submit». 3. Der Tester klickt auf eine bestehende Challenge in der Auswahlliste. 4. Der Tester gibt in das «Name»-Feld den Wert «Dies ist eine bearbeitete Test-Challenge» ein und bestätigt seine Eingabe mit einem Klick auf «Submit». 5. Der Tester wählt in der Navigation den Eintrag für «Abstract» aus. 6. Der Tester bearbeitet den Text innerhalb des Markdown-Editors und bestätigt seine Eingabe mit einem Klick auf den «Submit»-Button. 7. Der Tester wählt in der Navigation den Eintrag für «Goldnugget» aus. 8. Der Tester klickt auf den «Select None»-Button und bestätigt die Veränderung mit einem Klick auf «Submit». 9. Der Tester wählt in der Navigation den Eintrag für «Level» aus. 10. Der Tester klickt auf das Level «leet». 11. Der Tester geht zurück zum Consumer
Erwartetes Ergebnis	Die bestehende Challenge kann mit dem Editor ohne Fehler bearbeitet werden. Die bearbeitete Challenge wird danach in der Übersicht mit dem Level «leet», sowie dem Namen «Dies ist eine bearbeitete Test-Challenge» angezeigt.
Tatsächliches Ergebnis	Die bestehende Challenge kann mit dem Editor ohne Fehler bearbeitet werden. Die bearbeitete Challenge wird danach in der Übersicht mit dem Level «leet», sowie dem Namen «Dies ist eine bearbeitete Test-Challenge» angezeigt.
Testergebnis	OK

Tab. 22 UI-Test - Challenge bearbeiten

Beschreibung	Wert
Test-Nummer	3
Beschreibung	Ein Benutzer mit der hl_translator Rolle kann eine bestehende Challenge übersetzen.
Ausgeführte Schritte	1. Der Tester meldet sich mit dem Benutzernamen «test_translator» und dem Passwort «test» auf der Seite https://cas-client.hl.ygubler.ch an. 2. Der Tester wählt aus dem Dropdown-Control den Consumer «Test» aus und bestätigt seine Auswahl mit einem Klick auf «Submit». 3. Der Tester klickt auf die deutsche Flagge der bestehenden Challenge mit der ID (24634bab-e068-463e-820c-c5f650198f08) in der Auswahlliste. 4. Der Tester klickt auf den «Translate Challenge»-Button. 5. Der Tester klickt auf den «Save Challenge»-Button.
Erwartetes Ergebnis	Die bestehende Challenge konnte erfolgreich übersetzt und in der Datenbank gespeichert werden.
Tatsächliches Ergebnis	Die bestehende Challenge konnte erfolgreich übersetzt und in der Datenbank gespeichert werden.
Testergebnis	OK

Tab. 23 UI-Test - Challenge übersetzen

Beschreibung	Wert
Test-Nummer	4
Beschreibung	Ein Benutzer mit der hl_translator Rolle kann eine bestehende Challenge Übersetzung bearbeiten.
Ausgeführte Schritte	1. Der Tester meldet sich mit dem Benutzernamen «test_translator» und dem Passwort «test» auf der Seite https://cas-client.hl.ygubler.ch an. 2. Der Tester wählt aus dem Dropdown-Control den Consumer «Test» aus und bestätigt seine Auswahl mit einem Klick auf «Submit». 3. Der Tester klickt auf die deutsche Flagge der bestehenden Challenge mit der ID (24634bab-e068-463e-820c-c5f650198f08) in der Auswahlliste. 4. Der Tester sieht die bestehende deutsche Übersetzung 5. Der Tester klickt auf den «Modify Translation»-Button. 6. Der Tester ändert das Abstract der Challenge 7. Der Tester klickt auf den «Save Challenge»-Button.
Erwartetes Ergebnis	Die bestehende Challenge Übersetzung konnte erfolgreich angepasst und in der Datenbank gespeichert werden.
Tatsächliches Ergebnis	Die bestehende Challenge Übersetzung konnte erfolgreich angepasst und in der Datenbank gespeichert werden.
Testergebnis	OK

Tab. 24 UI-Test - Challenge Übersetzung bearbeiten

Beschreibung	Wert
Test-Nummer	5
Beschreibung	Ein Benutzer mit der hl_translator Rolle kann eine bestehende Challenge Übersetzung mit der automatischen Übersetzung überschreiben, wird aber davor gewarnt.
Ausgeführte Schritte	1. Der Tester meldet sich mit dem Benutzernamen «test_translator» und dem Passwort «test» auf der Seite https://cas-client.hl.ygubler.ch an. 2. Der Tester wählt aus dem Dropdown-Control den Consumer «Test» aus und bestätigt seine Auswahl mit einem Klick auf «Submit». 3. Der Tester klickt auf die deutsche Flagge der bestehenden Challenge mit der ID (24634bab-e068-463e-820c-c5f650198f08) in der Auswahlliste. 4. Der Tester sieht die bestehende deutsche Übersetzung 5. Der Tester klickt auf den «Translate Challenge»-Button. 6. Der Tester klickt auf den «Save Challenge»-Button. 7. Es wird ein Pop-up angezeigt, dass einzelne Werte der Challenge bereits von Hand übersetzt wurden. 8. Der Tester klickt auf OK 9. Es wird ein zweites Pop-up angezeigt, wo der Tester das Überschreiben mit der automatischen Übersetzung bestätigen kann. 10. Der Tester klickt auf «OK».
Erwartetes Ergebnis	Die bestehende Challenge Übersetzung konnte erfolgreich mit der automatischen Übersetzung überschrieben werden.
Tatsächliches Ergebnis	Die bestehende Challenge Übersetzung konnte erfolgreich mit der automatischen Übersetzung überschrieben werden.
Testergebnis	OK

Tab. 25 UI-Test - Challenge Übersetzung Warnung

Beschreibung	Wert
Test-Nummer	6
Beschreibung	Ein Benutzer mit der hl_user Rolle kann eine bestehende Challenge Übersetzung nicht anschauen.
Ausgeführte Schritte	1. Der Tester meldet sich mit dem Benutzernamen «test_user» und dem Passwort «test» auf der Seite https://cas-client.hl.ygubler.ch an. 2. Der Tester wählt aus dem Dropdown-Control den Consumer «Test» aus und bestätigt seine Auswahl mit einem Klick auf «Submit». 3. Der Tester klickt auf die deutsche Flagge der bestehenden Challenge mit der ID (24634bab-e068-463e-820c-c5f650198f08) in der Auswahlliste. 4. Der Tester sieht eine Meldung «Unauthorized Request!»
Erwartetes Ergebnis	Der Tester kann eine bestehende Übersetzung aufgrund mangelnder Berechtigung nicht anschauen. Es wird die Meldung «Unauthorized Request!» angezeigt
Tatsächliches Ergebnis	Der Tester kann eine bestehende Übersetzung aufgrund mangelnder Berechtigung nicht anschauen. Es wird die Meldung «Unauthorized Request!» angezeigt
Testergebnis	OK

Tab. 26 UI-Test - bestehende Challenge Übersetzung anschauen

6.5 Installationsanleitung und Benutzerhandbuch

Um spätere Betreiber der neu entwickelten Hacking-Lab 2.0 Umgebung bei der Installation wie auch dem Betrieb zu unterstützen, werden im Folgenden die zu tätigenden Schritte beschrieben. Diese wurden wie folgt gegliedert und die unter den Abschnitten «Installation» und «Benutzung» aufgeführten, beziehen sich jeweils auf eine vom Admin Tool zur Verfügung gestellte Option.

- Anforderungen
    - Hardware
    - Software
        - Docker-Host
        - KeycloakUserImporter
- Admin Tool
- Installation
    - Grundkonfiguration
    - Erstellung der Zertifikate
    - Anpassung der Konfigurationsdateien
        - docker-compose.common-services_template.yml
        - docker-compose.build_template.yml
        - docker-compose.run_template.yml
        - .env
        - traefik.toml
        - createCASUser.sql
        - createKeycloakUser.sql
        - create_docker_secrets.sh
        - default.startup.yaml
        - proxy-cas-server-cors-config.yml
        - proxy-cas-client-cors-config.yml
    - Vorbereitung des Docker-Hostsystems
        - init_swarm
        - create_docker_network
        - create_docker_secrets
- Benutzung
    - build_CAS
    - build_CAS_hard
    - run_CAS
    - run_CAS_hard
    - removeStacks
    - remove_Containers
    - show_logs_build_CAS
    - show_logs_run_CAS
    - show_logs_cas-traefik

6.5.1 Anforderungen

Da das neue Konzept auf Docker in einer Microservice-Architektur aufgebaut ist, sind die Anforderungen minimal gehalten.

6.5.1.1 Hardware

Die Anforderungen an die Docker-Host Hardware sind minimal. Um jedoch gute Performance in der Run- und Build-Umgebung gewährleisten zu können, sollten mindestens 2 GB Memory und 2 Prozessorkerne zur Verfügung stehen. Zwar würden auch weniger Ressourcen ausreichen, jedoch hat sich im Verlaufe des Projekts gezeigt, das mit weniger Rechenleistung mit erheblichen Geschwindigkeitseinbussen zu rechnen ist. Damit genügend Speicherplatz für die Docker-Images und den Zuwachs der Datenbank vorhanden ist, sollten mindestens 15 GB freier Speicherplatz zur Verfügung stehen.

6.5.1.2 Software

Zum einen gibt es Anforderungen an das System, auf welchem die Container zum Laufen gebracht werden sollen und zum andern an das System, auf welchem der KeycloakUserImporter ausgeführt wird.

6.5.1.2.1 Docker-Host

Um die Docker-Infrastruktur betreiben zu können, müssen Docker und Docker-Compose installiert sein. Da die Compose-Dateien in der Version 3.6 erstellt wurden, wird es notwendig, dass die Docker Engine mindestens in der Version 18.02.0+ [38] und Compose in der Version 1.20.0 [39] betrieben wird.

Wie man die aktuellste Version der Docker Engine und Compose installiert wird in der Docker-Dokumentation ausführlich beschrieben.

- Installation Docker Engine:
  https://docs.docker.com/install/linux/docker-ce/ubuntu/#set-up-the-repository
- Installation Compose:
  https://docs.docker.com/compose/install/#install-compose

Weiter wird ein Linux auf dem Host benötigt, da alle verwendeten Skripte mittels Bash und weiteren Linux-Programmen aufgebaut sind. Für den Betrieb während der Bachelorarbeit wurde ein von der HSR gestellter Ubuntu 16.04.4 LTS Server verwendet.

6.5.1.2.2 KeycloakUserImporter-Host

Um diese Software erfolgreich zu betreiben, wird Java in der Version 8 benötigt. Es wurde auch versucht, diese in der Java Version 10 zu betreiben. Jedoch ist der Keycloak Java Client noch nicht mit dieser Version kompatibel. Weiter wird empfohlen, dass die Applikation ein einer Eclipse-IDE gebaut wird, da dies den Vorgang erheblich vereinfacht.

6.5.2 Admin Tool

Im Weiteren wird häufig auf das Admin Tool verwiesen. Dieses ist im Repository unter «docker/admin_tool.sh» zu finden. Dieses kann auf zwei verschiedene Arten verwendet werden. Wenn es ohne Parameter gestartet wird, erscheint ein interaktives Menü, welches folgende Optionen zur Verfügung stellt:

```
 1) init_swarm
 2) create_docker_network
 3) create_docker_secrets
 4) createLDAPCerts.sh
 5) build_CAS
 6) build_CAS_hard
 7) run_CAS
 8) run_CAS_hard
 9) removeStacks
10) removeContainers
11) show_logs_build_CAS
12) show_logs_run_CAS
13) show_logs_cas-traefik
14) edit_docker-compose.common-services_template.yml
15) edit_docker-compose.build_template.yml
16) edit_docker-compose.run_template.yml
17) edit_.env
18) edit_traefik.toml
19) edit_cas-mysql_createCASUser.sql
20) edit_keycloak-mysql_createKeycloakUser.sql
21) edit_create_docker_secrets.sh
22) edit_ldap/image/environment/default.startup.yaml
23) edit_keycloak-proxy/configs/proxy-cas-server-cors-config.yml
24) edit_keycloak-proxy/configs/proxy-cas-client-cors-config.yml
25) Quit
```

Abb. 72 admin_tool.sh Optionen

Beim Drücken der Entertaste erscheinen die Auswahlmöglichkeiten erneut. Dies kann hilfreich sein, wenn sich nach der Ausführung eines Skripts diese Anzeige aus dem sichtbaren Bereich verschiebt.

Die zweite Variante das Admin Tool zu verwenden, bezieht sich auf das Starten der Container und die Anzeige deren Logs. Dies lässt sich wie folgt bedienen:

```
Usage: admin_tool.sh [option] [OPTIONAL_DOCKER_TO_BUILD (default: all) docker-service]
 option:  -b        to build containers and build CAS in docker cas_build_stack
          -bh       to build containers and build CAS in docker cas_build_stack (build --force-rm --no-cache --pull)
          -r        to build containers and run CAS in docker cas_run_stack
          -rh       to build containers and run CAS in docker cas_run_stack (build --force-rm --no-cache --pull)
          -lb       to docker service logs from cas_build_stack
          -lr       to docker service logs from cas_run_stack
```

Abb. 73 admin_tool.sh mit Parameter

So wird es möglich, dieses Tool zu verwenden, ohne dass gleich immer alle Container neu gebuildet werden müssen. Wenn man beispielsweise nur Keycloak und einen der Server Proxy komplett neu erstellen will und anschliessend deren Logs betrachten möchte, geht dies folgendermassen:

```
./admin_tool.sh -rh cas-keycloak proxy-cas-server
./admin_tool.sh -lr cas-keycloak proxy-cas-server
```

- Der Parameter r steht für das Builden aller Container für die Run-Umgebung.
- Der Parameter b steht für das Builden alle Container für die Build-Umgebung.
- Der Parameter h steht für hard, welcher dem Build-Befehl «--force-rm -no-cache –pull» anfügt.
- Der Parameter l steht für log, welcher die Logs anzeigt.

6.5.3 Installation

6.5.3.1 Grundkonfiguration

Die Grundkonfiguration ist bereits im Git-Repository enthalten. So wird es den Betreibern möglich, innert kürzester Zeit ein neues CAS-System in Betrieb zu nehmen. Es sind jedoch noch einige pro System spezifische Anpassung notwendig, welche in dieser Anleitung beschrieben werden.

6.5.3.2 Erstellung der Zertifikate

In der Grundkonfiguration sind alle benötigten Zertifikate vorhanden. Jedoch sind diese auf die Domain hl.ygubler.ch ausgestellt. Um selbstsignierte Zertifikate für eine andere Domain auszustellen, werden zwei verschieden Zertifikatsgeneratoren benötigt (für OpenSSL und GnuTLS).

Um das Traefik Zertifikat zu generieren, wurde ein kleiner Zertifikatsgenerator entwickelt. Dieser ist im Hauptverzeichnis des Repositories im Ordner «certGenerator» zu finden. Dabei gibt es zwei verschiedene Varianten, eine mit Erstellung einer CA und eine ohne. Als erstes soll die dort aufzufindende Datei «req.conf» auf die zu betreibende Domäne angepasst werden. Im Anschluss daran kann das Skript «doCert.sh» ausgeführt werden, welches die Zertifikate erstellt und im Ordner «out» persistiert. Diese neu erstellten Zertifikate (Private und Public) müssen nach der Erstellung an den folgenden Ort «docker/traefik/acme/» mit den Namen «cert.key» und «cert.pem» kopiert/verschoben werden.

Die Erstellung des LDAP-Zertifikats kann mit dem Menüpunkt vier «createLDAPCerts.sh» im Admin Tool durchgeführt werden. Nach dem Starten des Tools werden nacheinander alle benötigten Parameter von der Konsole abgefragt und am Schluss automatisch an die korrekte Stelle kopiert. Dieser Menüpunkt sollte aber nur ausgeführt werden, wenn auch ein neues Zertifikat erstellt werden soll. Ansonsten wird das aktuelle überschrieben.

6.5.3.3 Anpassung der Konfigurationsdateien

Alle anzupassenden Konfigurationsdateien sind im Admin Tool enthalten. Diese sind mit den Optionen 14 bis 24 des Admin Tools zur Bearbeitung erreichbar.

6.5.3.3.1 docker-compose.common-services_template.yml

Diese Datei enthält den cas-mysql, welcher in der Run- wie auch in der Build-Umgebung verwendet wird. Diese Datei kann so belassen werden, da alle Konfigurationen aus der «.env»-Datei entnommen werden.

6.5.3.3.2 docker-compose.build_template.yml

Diese Datei enthält die Container build-cas-server, build-cas-client und build-consumer, welche zusammen mit dem cas-mysql die Build-Umgebung bilden und für das Bauen der Services gebraucht wird. Diese Datei kann so belassen werden, da alle Konfigurationen aus der «.env»-Datei entnommen werden.

6.5.3.3.3 docker-compose.run_template.yml

Diese Datei enthält alle Container, welche zusammen mit dem cas-mysql die Run-Umgebung bilden, welche für den Betrieb des Hacking-Lab 2.0 benötigt wird. Diese Datei kann so belassen werden, da alle Konfigurationen aus der «.env»-Datei entnommen werden.

6.5.3.3.4 .env

Diese Datei wird von allen Services gebraucht und dient als globale Konfigurationsdatei. Es werden hier nicht alle Werte erklärt, da diese mittels Kommentaren in der genannten Datei beschrieben werden. Im vorliegenden Abschnitt werden nur die wesentlichen Variablen genauer betrachtet, die beim Wechseln der Domain gebraucht werden oder für den initialen Betrieb der Umgebung entscheidend sind.

- KEYCLOAK_MIGRATION_ACTION=<export|import|>
  - Beim ersten Start auf «import» festlegen, somit wird die im Repository vorhandene Keycloak Konfiguration importiert
  - Nach dem erfolgreichen Import sollte die Variable auf «export» gesetzt werden, um bei jedem Start von Keycloak die in der Datenbank gespeicherte Konfiguration nach «docker/keycloak/export» zu exportieren. Wird diese Variable leer gelassen, wird beim Starten von Keycloak kein Import oder Export durchgeführt.
  - **Schlägt die Migration Action fehl, kann der Container nicht gestartet werden. Dies kann auch der Fall sein, wenn keine Verbindung mit dem LDAP-Server aufgebaut werden kann. Denn in der Grundkonfiguration sind die Benutzer, Rollen und Gruppen im LDAP persistiert.**
- ENABLE_DOCKER_SWARM=<false|true>
  - False Docker läuft im Standalone Mode
  - True Docker läuft im Swarm Mode
- DATABASE_PATH=<Pfad an welchem die CAS-Datenbank gespeichert werden soll>
- KEYCLOAK_DATABASE_PATH=<Pfad an welchem die Keycloak-DB gespeichert werden soll>
- ADDRESS_CAS_SERVER=<FQDN, mit welchem der CAS-Server erreicht werden soll>
- ADDRESS_CAS_CLIENT=<FQDN mit welchem der CAS-Client erreicht werden soll >
- ADDRESS_CONSUMER=<FQDN mit welchem der Consumer erreicht werden soll>
- ADDRESS_KEYCLOAK=<FQDN mit welchem Keycloak erreicht werden soll>
- ADDRESS_PHPLDAPADMIN=<FQDN mit welchem phpLDAPadmin erreicht werden soll>
- CAS_LDAP_HOSTNAME=<FQDN des Docker-Hosts auf welchem LDAP erreichbar sein soll>
- CAS_PHPLDAPADMIN_BIND_ID=<cn=admin,dc=hl,dc=ygubler,dc=ch>
  - Hier müssen die dc Werte an den LDAP-Hostnamen angepasst werden.
- CAS_SERVER_EDITOR_ROLES=<realm roles welche Editorberechtigungen auf dem CAS-Server bekommen sollen (mehrere Roles durch «,» trennen, ohne Abstand>
- CAS_SERVER_TRANSLATOR_ROLES=<realm roles welche Translatorberechtigungen auf dem CAS-Server bekommen sollen (mehrere roles durch «,» trennen, ohne Abstand>
- CAS_SERVER_USER_ROLES=<realm roles welche Userberechtigungen auf dem CAS-Server bekommen sollen (mehrere Roles durch «,» trennen, ohne Abstand>
- CAS_SERVER_ADMIN_ROLES=<realm roles welche Administratorberechtigungen auf dem CAS-Server bekommen sollen (mehrere Roles durch «,» trennen, ohne Abstand>

6.5.3.3.5 traefik.toml

Da die Traefik-Regeln via Docker-Labels erstellt werden, gibt es in dieser Datei nur wenig anzupassen.

Im Abschnitt «[docker]» muss der neue Domainname gesetzt werden.

Im Abschnitt «[entryPoints]» könnte man optional noch den Pfad/Namen der zu verwendenden Zertifikate deklarieren.

6.5.3.3.6 createCASUser.sql

Beim Starten des Containers cas-mysql wird mittels dieser SQL-Datei der Datenbankbenutzer für den CAS-Server erstellt und dessen Berechtigung vergeben. In dieser Datei kann falls erwünscht der Datenbank-Benutzer, sein Passwort und der Name der Datenbank angepasst werden. Falls hier eine Änderung getätigt wird, muss diese auch in den Dateien «.env» und «create_docker_secrets.sh» durchgeführt werden. Dies da die anderen Container keinen Zugriff auf diese Datei haben.

6.5.3.3.7 createKeycloakUser.sql

Beim Starten des Containers cas-keycloak-mysql wird mittels dieser SQL-Datei der Datenbankbenutzer für den cas-keycloak erstellt und dessen Berechtigung vergeben. In dieser Datei kann falls erwünscht der Datenbank-Benutzer, sein Passwort und der Name der Datenbank angepasst werden. Falls hier eine Änderung getätigt wird, müssen diese auch in den Dateien «.env» und «create_docker_secrets.sh» durchgeführt werden. Dies da die anderen Container keinen Zugriff auf diese Datei haben.

6.5.3.3.8 create_docker_secrets.sh

In dieser Datei werden die Passwörter, wie auch die Proxy Encryption-Keys und Client Secrets, welche in den Containern benötigt werden gehalten. Das Client-Secret der Proxys muss mit dem hinterlegten Wert innerhalb von Keycloak (Keycloak-Adminwebseite → Clients → gewünschter Client → Credentials → Secret) übereinstimmen. In der Grundkonfiguration stimmen diese Werte bereits überein. Ansonsten können hier die Passwörter nach eigenem Ermessen geändert werden.

6.5.3.3.9 default.startup.yaml

Diese Datei wird nur für den ersten Start des cas-ldap benötigt und wird anschliessend innerhalb des Containers gelöscht. Wenn die Grundkonfiguration verwendet wird, wurde diese bereits angewandt und hat keinen Einfluss. Falls jedoch die Inhalte der Ordner «docker/ldap/config» und «docker/ldap/database» gelöscht werden, kommt diese Datei wieder zum Zug. Ist dies der Fall, müssen die OUs für die Keycloak User, Gruppen und Rollen manuell mittels phpldapadmin erstellt werden.

Bei einer neuen Konfiguration gilt es folgende Werte anzupassen:

- LDAP_ORGANISATION: <Name der Organisation>
- LDAP_DOMAIN: <Domänenname auf dem LDAP operieren soll, muss mit CAS_LDAP_HOSTNAME aus der «.env»-Datei übereinstimmen>
- LDAP_ADMIN_PASSWORD: <Administratorpasswort>
- LDAP_CONFIG_PASSWORD: <Konfigurationspasswort>
- LDAP_READONLY_USER_PASSWORD: <Readonly-Benutzerpasswort>
- LDAP_TLS_CRT_FILENAME: <Name des zu verwendenden Zertifikats>
    - Wenn der Name angepasst wird, stimmt dieser nicht mehr mit dem vom createLDAPCerts.sh generierten überein. → Anpassung notwendig

- LDAP_TLS_KEY_FILENAME: ldap.key
  - Wenn der Name angepasst wird, stimmt dieser nicht mehr mit dem vom createLDAPCerts.sh generierten überein. → Anpassung notwendig
- LDAP_TLS_CA_CRT_FILENAME: ca.crt
  - Wenn der Name angepasst wird, stimmt dieser nicht mehr mit dem vom createLDAPCerts.sh generierten überein. → Anpassung notwendig

6.5.3.3.10 proxy-cas-server-cors-config.yml

Unter «cors-origins» gilt es die FQDNs aller CAS-Clients und Consumers einzutragen, damit die CORS-Header auf dem Proxy des CAS-Servers richtig gesetzt werden.

6.5.3.3.11 proxy-cas-client-cors-config.yml

Unter «cors-origins» gilt es die FQDNs aller Consumers einzutragen, damit die CORS-Header auf dem Proxy des CAS-Clients richtig gesetzt werden.

6.5.3.4 Vorbereitung des Docker-Hostsystems

Die nun folgenden Vorbereitungen können allesamt aus dem Admin Tool heraus gestartet werden.

6.5.3.4.1 init_swarm

Bei der Ausführung dieser Option wird der Docker Swarm Modus initiiert. Dies wird notwendig, sobald in der «.env»-Datei die Variable «ENABLE_DOCKER_SWARM» auf «true» gesetzt wird. Wurde der Wert auf «false» gesetzt und «init_swarm» wird ausgeführt, hat dies keinen Einfluss auf den Betrieb.

6.5.3.4.2 create_docker_network

Diese vom Admin Tool zur Verfügung gestellte Option erstellt das von allen Containern benötigte Docker-Netzwerk, dessen Namen in der «.env»-Datei festgelegt werden kann.

6.5.3.4.3 create_docker_secrets

Nach dem Start dieses Skripts werden die Passwörter entweder in einzelne Dateien in dem Ordner «docker/secrets/<secretalias>.txt» oder in einen Docker Secret-Store geschrieben, dies hängt davon ab, in welchem Modus Docker betrieben wird.

- Swarm Mode → Secret-Store
- Standalone Mode → Passwortdateien

6.5.4 Benutzung

Die folgenden Befehle zur Benutzung stammen aus dem Admin Tool.

Für das reine Starten der Container wurde kein Befehl im Administrationstool erfasst, da das Builden der Container ohne Änderungen sehr schnell geht. Falls trotzdem ein Container im Standalone-Mode nur gestartet werden soll, kann dies mit folgendem Befehl bewerkstelligt werden.

```
docker-compose -f docker-compose.stack_run.yml -f docker-compose.stack_build.yml \
-f docker-compose.stack_common-services.yml up -d <containername>
```

6.5.4.1 build_CAS

Buildet die seit dem letzten Mal veränderten Container der Build-Umgebung und startet alle Container der Build-Umgebung.

Um dies zu erreichen, wird im Hintergrund der Befehl `./admin_tool.sh -b` ausgeführt, welcher alle Container der Build-Umgebung anspricht. Daher wird empfohlen den Befehl nur auf die gewünschten Container anzuwenden. Um dies zu erreichen, kann der Befehl auf der Konsole mit der Angabe der gewünschten Container als Parameter ausgeführt werden.

```
./admin_tool.sh -b <container name> <container name>
```

6.5.4.2 build_CAS_hard

Diese Option unterscheidet sich von der normalen build_CAS Option dadurch, dass sie dem Docker Build-Befehl die Parameter «`--force-rm --no-cache –pull`» übergibt. Es hat sich in der Praxis gezeigt, dass teils Altlasten im Cache vorhanden sind. Mit dieser Methode wird das Verwenden eines Cache verhindert und sichergestellt, dass das neuste Docker-Image heruntergeladen wird.

«Build-CAS_hard» verwendet im Hintergrund den Befehl `./admin_tool.sh -bh`. Da dieser aber immer alle Container betrifft, dauert dieser Vorgang sehr lange, weil alle Images neu heruntergeladen werden. Daher wird empfohlen den Befehl nur auf die Container anzuwenden, die das erneute Bauen erfordern. Um dies zu erreichen, kann der Befehl auf der Konsole mit der Angabe der gewünschten Container als Parameter ausgeführt werden.

```
./admin_tool.sh -bh <container name> <container name>
```

6.5.4.3 run_CAS

Buildet die seit dem letzten Mal veränderten Container der Run-Umgebung und startet alle Container der Run-Umgebung.

Um dies zu erreichen, wird im Hintergrund der Befehl `./admin_tool.sh -r` ausgeführt, welcher alle Container der Run-Umgebung anspricht. Daher wird empfohlen den Befehl nur auf die gewünschten Container anzuwenden. Um dies zu erreichen, kann der Befehl auf der Konsole mit der Angabe der gewünschten Container als Parameter ausgeführt werden.

```
./admin_tool.sh -r <container name> <container name>
```

6.5.4.4 run_CAS_hard

Diese Option unterscheidet sich von der normalen «run_CAS» dadurch, dass sie dem Docker Build-Befehl die Parameter «`--force-rm --no-cache –pull`» übergibt. Es hat sich in der Praxis gezeigt, dass teils Altlasten im Cache vorhanden sind. Mit dieser Methode wird das Verwenden eines Cache verhindert und sichergestellt, dass das neuste Docker-Image heruntergeladen wird.

«Run_CAS_hard» verwendet im Hintergrund den Befehl `./admin_tool.sh -rh`. Da dieser aber immer alle Container betrifft, dauert dieser Vorgang sehr lange, weil alle Images neu heruntergeladen werden. Daher wird empfohlen den Befehl nur auf die Container anzuwenden, die das erneute Bauen erfordern. Um dies zu erreichen, kann der Befehl auf der Konsole mit der Angabe der gewünschten Container als Parameter ausgeführt werden.

```
./admin_tool.sh -rh <container name> <container name>
```

6.5.4.5 removeStacks

Entfernt alle Services (Container) im Swarm Modus.

6.5.4.6 remove_Containers

Entfernt alle Container im Standalone Modus.

6.5.4.7 show_logs_build_CAS

Zeigt alle Logs der Container in der Build-Umgebung und folgt diesen (nur im Standalone Mode).

«show_logs_build_CAS» verwendet im Hintergrund den Befehl `./admin_tool.sh -lb`. Da dieser jedoch die Logs aller Build-Container ausgibt, kann es schnell unübersichtlich werden. Um dieser Problematik entgegenzuwirken, wird empfohlen das Kommando auf der Konsole mit der Angabe der gewünschten Container als Parameter zu starten.

```
./admin_tool.sh -lb <container name> <container name>
```

6.5.4.8 show_logs_run_CAS

Zeigt alle Logs der Container in der Run-Umgebung und folgt diesen (nur im Standalone Mode).

«show_logs_run_CAS» verwendet im Hintergrund den Befehl `./admin_tool.sh -lr`. Da dieser jedoch die Logs aller Run-Container ausgibt, kann es schnell unübersichtlich werden. Um dieser Problematik entgegenzuwirken, wird empfohlen das Kommando auf der Konsole mit der Angabe der gewünschten Container als Parameter zu starten.

```
./admin_tool.sh -lr <container name> <container name>
```

6.5.4.9 show_logs_cas-traefik

Zeigt den Inhalt der Datei «docker/traefik/log/traefik.log» und folgt den neuen Einträgen die in sie geschrieben werden. Der Log-Level kann in der Datei «traefik.toml» deklariert werden. In der Grundkonfiguration steht dieser auf «INFO». Wird der Log-Level «DEBUG» eingesetzt, werden alle Anfragen an den Traefik in die Logdatei geschrieben, was zu einem enormen Zuwachs der Datei führt.

6.5.5 Step by Step – für neuen Domänennamen

1. Anforderung überprüfen
    a. Hardwareanforderungen überprüfen
        i. min. 2 GB Memory vorhanden
        ii. min. 2 Prozessorkerne vorhanden
        iii. min 15 GB freier Speicherplatz vorhanden
    b. Softwareanforderungen überprüfen von
        i. Docker-Host
            1. Ist ein Linux-System inkl. Bash und vi
            2. Docker Engine min. Version 18.02.0+ installiert
            3. Compose min. Version 1.20.0 installiert
        ii. KeycloakUserImporter-Host
            1. Java Version 8 installiert
            2. Vorzugsweise Eclipse zum Builden installiert
2. Installation
    a. Grundkonfiguration durch klonen des Git-Repositories
    b. Erstellung neuer Zertifikate für die neue Domäne
        i. Traefik Zertifikat für den neuen Domainnamen mittels «certGenerator» aus dem Git-Hauptverzeichnis (beim certGenerator die Datei req.conf anpassen)
        ii. Zertifikat (cert.key und cert.pem aus dem out-Ordner des certGenerators) nach «docker/traefik/acme» kopieren
        iii. LDAP Zertifikat für den neuen Domainnamen mittels Zertifikatsgenerator aus dem Admin Tool «createLDAPCerts.sh» generieren
    c. Anpassung der Konfigurationsdateien
        i. .env
            1. KEYCLOAK_MIGRATION_ACTION auf «**import**» festlegen
            2. DATABASE_PATH festlegen und Ordner auf Host erstellen
            3. KEYCLOAK_DATABASE_PATH festlegen und Ordner auf Host erstellen
            4. ADDRESS_CAS_SERVER neuen FQDN für den CAS-Server festlegen
            5. ADDRESS_CAS_CLIENT neuen FQDN für den CAS-Client festlegen
            6. ADDRESS_CONSUMER neuen FQDN für den Consumer festlegen
            7. ADDRESS_KEYCLOAK neuen FQDN für den Keycloak festlegen
            8. ADDRESS_PHPLDAPADMIN neuen FQDN für phpLDAPadmin festlegen
        ii. traefik.toml
            1. Neuen Domainnamen setzen
            2. Evtl. muss der Zertifikatsname angepasst werden (bezieht sich auf das Zertifikat, welches im «acme»-Ordner liegt
        iii. createCASUser.sql
            1. Neues Datenbankbenutzerpasswort vergeben
        iv. createKeycloakUser.sql
            1. Neues Datenbankbenutzerpasswort vergeben
        v. create_docker_secrets.sh
            1. Passwörter (**ausgenommen Keycloak Client-Secrets**) neu vergeben
            2. «database_password» muss mit dem im «createCASUser.sql» übereinstimmen
            3. «keycloak_database_password» muss mit dem im «createKeycloakUser.sql» übereinstimmen
        vi. proxy-cas-server-cors-config.yml

1. Unter «cors-origins» Werte der Variablen «ADDRESS_CAS_CLIENT» und «ADDRESS_CAS_CONSUMER» aus der «.env»-Datei eintragen.

vii. proxy-cas-client-cors-config.yml
1. Unter «cors-origins» Werte der Variable «ADDRESS_CAS_CONSUMER» aus der «.env»-Datei eintragen.

d. Vorbereitung des Docker-Hostsystems mithilfe des Admin Tool
i. Docker-Netzwerk mittels «create_docker_network» erstellen
ii. Docker Secrets mittels «create_docker_secrets» generieren

3. Benutzung mithilfe des Admin Tools
a. `admin_tool.sh -bh` ausführen
b. Warten bis alle Build-Container wieder beendet wurden, dies kann mit `docker ps` überprüft werden
Aktueller Status ist mit `admin_tool.sh -lb` überprüfbar.
c. `admin_tool.sh -lb build-cas-server` ausführen und überprüfen ob keine Fehler angezeigt werden und die ZIP-Datei erstellt werden konnte.
d. `admin_tool.sh -lb build-cas-client` ausführen und überprüfen ob keine Fehler angezeigt werden und die von Webpack generierten Dateien erstellt werden konnten
e. `admin_tool.sh -lb build-consumer` ausführen und überprüfen ob keine Fehler angezeigt werden und die von Webpack generierten Dateien erstellt werden konnten
f. `admin_tool.sh -rh` ausführen
g. `admin_tool.sh -lr` ausführen und der Ausgabe folgen bis alle Container gestartet sind
h. `docker ps` ausführen, es sollten folgende Container angezeigt werden
i. cas-keycloak
ii. cas-keycloak-mysql
iii. cas-ldap
iv. cas-mysql
v. cas-phpldapadmin
vi. cas-traefik
vii. proxy-cas-client
viii. proxy-cas-server
ix. proxy-consumer
x. run-cas-client
xi. run-cas-server
xii. run-consumer
i. In der «.env»-Datei «KEYCLOAK_MIGRATION_ACTION» wieder auf «**export**» stellen oder leer lassen
j. Auf cas-keycloak Administrationswebseite (https://ADDRESS_KEYCLOAK) verbinden (Benutzer: admin; Passwort: bHLkeycloak18), um folgende Einstellungen zu ändern
i. Administratorpasswort der Realm «master» gleich wie in der «create_docker_secrets»-Datei
ii. Für die drei Clients (cas_client_eu, cas_server_eu und consumer_eu) der «Hacking-Lab» Realm
1. Neue Secrets unter Clienteinstellungen → Credentials generieren und in «create_docker_secrets»-Datei eintragen
2. «Root URL», «Valid Redirect URIs» und «Web Origins» an neue Domäne anpassen
iii. create_docker_secrets ausführen, damit die Secrets generiert werden
iv. `admin_tool.sh -rh proxy-cas-server proxy-cas-client proxy-consumer` ausführen

v. `admin_tool.sh -lr proxy-cas-server proxy-cas-client proxy-consumer` ausführen und überprüfen, ob diese erfolgreich gestartet wurden.
vi. Unter «User Federation → ldap → Settings» folgende Werte an neue Domäne anpassen und speichern
    1. Users DN
    2. Bind DN
    3. Bind Credential (neues LDAP-Admin PW definieren)
vii. Unter «User Federation → ldap → Settings» auf «Unlink users» klicken
viii. Unter «User Federation → ldap → Mappers» die Mapper DNs an neue Domäne anpassen
    1. «myGroupMapper» die Variable «LDAP Groups DN»
    2. «myRoleMapper_cas-client-eu» die Variable «LDAP Roles DN»
    3. «myRoleMapper» die Variable «LDAP Groups DN»

k. LDAP neu aufsetzen (**Achtung** Keycloak währenddessen nicht neu starten)
i. cas-ldap mit `docker stop cas-ldap` stoppen
ii. cas-phpldapadmin mit `docker stop cas-phpldapadmin` stoppen
iii. Inhalt von «docker/ldap/config» und «docker/ldap/database» löschen
iv. .env anpassen
    1. CAS_LDAP_HOSTNAME neuen FQDN für den LDAP festlegen
    2. CAS_PHPLDAPADMIN_BIND_ID neue Bind_ID festlegen
v. default.startup.yaml anpassen
    1. LDAP_ORGANISATION Name der Firma eintragen
    2. LDAP_DOMAIN gleichen Wert wie CAS_LDAP_HOSTNAME (aus .env)
    3. LDAP_ADMIN_PASSWORD neues Passwort festlegen (gleiches wie bei «Bind Credentials» beim cas-keycloak
    4. LDAP_READONLY_USER_PASSWORD neues Passwort festlegen
    5. LDAP_CONFIG_PASSWORD neues Passwort festlegen
vi. cas-ldap mit `admin_tool.sh -rh cas-ldap cas-phpldapadmin` neu erstellen
vii. Auf cas-phpldapadmin mit «https://ADDRESS_PHPLDAPADMIN» zugreifen und mit CAS_PHPLDAPADMIN_BIND_ID und LDAP_ADMIN_PASSWORD anmelden, um die OUs «users», «groups», «roles» und «roles_cas-client-eu» mit objectClass «organizationalUnit» und «top», analog zu den Bindings im Keycloak, zu erstellen
viii. Nun auf der Keycloak-Adminwebseite einen LDAP Sync unter «User Federation/ldap» starten, sollte eine Erfolgsmeldung anzeigen
    1. Synchronize all users (die drei Benutzer der Grundkonfiguration werden nicht synchronisiert, gilt nur für ab hier neu erstellte Benutzer)
    2. Unter den drei Mappern ein Sync von Keycloak nach LDAP starten

4. Überprüfen der Services
a. Neue Benutzer im Keycloak erfassen oder mittels KeycloakUserImporter (Rolle «hl_users» vergeben damit diese den Zugriff auf die Services haben)
b. Auf Traefik-Dashboard mit «http://<neue Domäne>:8080» zugreifen, welches fünf Frontend und Backend Regeln anzeigen sollte
c. Auf phpLDAPadmin mit «https://ADDRESS_PHPLDAPADMIN» zugreifen und mit CAS_PHPLDAPADMIN_BIND_ID und LDAP_ADMIN_PASSWORD anmelden. Hier sollten nun die synchronisierten Keycloak-Benutzer ersichtlich sein.
d. Zugriff auf CAS-Server, CAS-Client und Consumer mittels «https://<Adresse aus .env-Datei>» und mit einem neu erstellten Keycloak-Benutzer, welcher mindestens die Rolle «hl_users» besitzt, testen.

6.5.6 Step by Step – Einstellungen der Bachelorarbeit

1. Anforderung überprüfen
    a. Hardwareanforderungen überprüfen
        i. min. 2 GB Memory vorhanden
        ii. min. 2 Prozessorkerne vorhanden
        iii. min 15 GB freier Speicherplatz vorhanden
    b. Softwareanforderungen überprüfen von
        i. Docker-Host
            1. Ist ein Linux-System inkl. Bash und vi
            2. Docker Engine min. Version 18.02.0+ installiert
            3. Compose min. Version 1.20.0 installiert
        ii. KeycloakUserImporter-Host
            1. Java Version 8 installiert
            2. Vorzugsweise Eclipse zum Builden installiert
2. Installation
    a. Grundkonfiguration durch klonen des Git-Repositories
    b. Anpassung der Konfigurationsdateien
        i. .env
            1. KEYCLOAK_MIGRATION_ACTION auf «**import**» festlegen
            2. DATABASE_PATH festlegen und Ordner auf Host erstellen
            3. KEYCLOAK_DATABASE_PATH festlegen und Ordner auf Host erstellen
    c. Vorbereitung des Docker-Hostsystems mithilfe des Admin Tool
        i. Docker-Netzwerk mittels «create_docker_network» erstellen
        ii. Docker Secrets mittels «create_docker_secrets» generieren
3. Benutzung mithilfe des Admin Tools
    a. `admin_tool.sh -bh` ausführen
    b. Warten bis alle Build-Container wieder beendet wurden, dies kann mit `docker ps` überprüft werden
    Aktueller Status ist mit `admin_tool.sh -lb` überprüfbar.
    c. `admin_tool.sh -lb build-cas-server` ausführen und überprüfen ob keine Fehler angezeigt werden und die ZIP-Datei erstellt werden konnte.
    d. `admin_tool.sh -lb build-cas-client` ausführen und überprüfen ob keine Fehler angezeigt werden und die von Webpack generierten Dateien erstellt werden konnten.
    e. `admin_tool.sh -lb build-consumer` ausführen und überprüfen ob keine Fehler angezeigt werden und die von Webpack generierten Dateien erstellt werden konnten
    f. `admin_tool.sh -rh` ausführen
    g. `admin_tool.sh -lr` ausführen und der Ausgabe folgen bis alle Container gestartet sind
    h. In der «.env»-Datei «KEYCLOAK_MIGRATION_ACTION» wieder auf «**export**» stellen oder leer lassen
    i. `docker ps` ausführen, es sollten folgende Container angezeigt werden
        i. cas-keycloak
        ii. cas-keycloak-mysql
        iii. cas-ldap
        iv. cas-mysql
        v. cas-phpldapadmin
        vi. cas-traefik
        vii. proxy-cas-client

viii. proxy-cas-server
ix. proxy-consumer
x. run-cas-client
xi. run-cas-server
xii. run-consumer

4. Überprüfen der Services
    a. Auf Traefik-Dashboard mit «http://hl.ygubler.ch>:8080» zugreifen, welches fünf Front- und Backend Regeln anzeigen sollte
    b. Auf phpLDAPadmin mit «https://ldapadmin.hl.ygubler.ch» zugreifen und mit Login DN «cn=admin,dc=hl,dc=ygubler,dc=ch» und Passwort «bHLldap18» anmelden. Nun sollte die LDAP-Struktur aus der Grundkonfiguration ersichtlich sein
    c. Zugriff auf CAS-Server, CAS-Client und Consumer mittels «https://<Adresse aus .env-Datei>» testen. Zum Testen folgende Keycloak-Benutzer verwenden:
        i. «test_user» mit Passwort «test»
        ii. «test_editor» mit Passwort «test»
        iii. «test_translator» mit Passwort «test»

7 Verzeichnisse

7.1 Abbildungsverzeichnis

7.2 Tabellenverzeichnis

7.3 Quellverzeichnis

[1] «Standard Markdown Syntax,» [Online]. Available: https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet.

[2] «Docker-Compose Secrets,» [Online]. Available: https://docs.docker.com/compose/compose-file/#secrets.

[3] «Docker Swarm Modus,» [Online]. Available: https://docs.docker.com/engine/swarm/.

[4] «Play Framework,» [Online]. Available: https://www.playframework.com/.

[5] «Medium-Editor Github Repo,» [Online]. Available: https://github.com/yabwe/medium-editor.

[6] «Medium,» [Online]. Available: https://medium.com/.

[7] «Dropbox Paper Webseite,» [Online]. Available: https://www.dropbox.com/paper.

[8] «Demarcate Github Repo,» [Online]. Available: https://github.com/will-hart/demarcate.js.

[9] «Dillinger Github Repo,» [Online]. Available: https://github.com/joemccann/dillinger.

[10] «Stackedit.io Guthub Repo,» [Online]. Available: https://github.com/benweet/stackedit.

[11] «TOAST UI Editor Github Repo,» [Online]. Available: https://github.com/nhnent/tui.editor.

[12] «Woofmark Github Repo,» [Online]. Available: https://github.com/bevacqua/woofmark.

[13] «SimpleMD Markdown Editor Github Repo,» [Online]. Available: https://github.com/sparksuite/simplemde-markdown-editor.

[14] «Pen Editor Github Repo,» [Online]. Available: https://github.com/sofish/pen.

[15] «Turndown Github Repo,» [Online]. Available: https://github.com/domchristie/turndown.

[16] «upndown Github Repo,» [Online]. Available: https://github.com/netgusto/upndown.

[17] «to-markdown npm Website,» [Online]. Available: https://www.npmjs.com/package/to-markdown.

[18] «html-to-markdown Github Repo,» [Online]. Available: https://github.com/thetutlage/html-to-markdown.

[19] «Markdown-it Github Repo,» [Online]. Available: https://github.com/markdown-it/markdown-it.

[20] «markdown-js Github Repo,» [Online]. Available: https://github.com/evilstreak/markdown-js.

[21] «marked Github Repo,» [Online]. Available: https://github.com/markedjs/marked.

[22] «showdown Github Repo,» [Online]. Available: https://github.com/showdownjs/showdown.

[23] «Deepl Translator,» [Online]. Available: https://www.deepl.com.

[24] «Jackson,» [Online]. Available: https://github.com/FasterXML/jackson-docs.

[25] «Docker Compose File,» [Online]. Available: https://docs.docker.com/compose/compose-file/.

[26] «Medium - Docker environment variables expanded from secrets,» [Online]. Available: https://medium.com/@basi/docker-environment-variables-expanded-from-secrets-8fa70617b3bc.

[27] «Traefik,» [Online]. Available: https://traefik.io/.

[28] «Docker Compose File Labels,» [Online]. Available: https://docs.docker.com/compose/compose-file/#labels.

[29] «Keycloak,» [Online]. Available: https://www.keycloak.org/.

[30] «codecentric - Access Management mit Keycloak,» [Online]. Available: https://blog.codecentric.de/2016/06/accessmanagement-mit-keycloak/.

[31] «Keycloak Issue 4828 - LDAP default groups not working,» [Online]. Available: https://issues.jboss.org/browse/KEYCLOAK-4828?_sscc=t.

[32] «Github - offizieller Keycloak Auth Proxy - Java,» [Online]. Available: https://github.com/keycloak/keycloak/tree/master/proxy.

[33] «DockerHub-Rebository Osixia,» [Online]. Available: https://hub.docker.com/u/osixia/.

[34] «Github - openLDAP issue 28,» [Online]. Available: https://github.com/osixia/docker-openldap/issues/28. [Zugriff am 02 05 2018].

[35] «openldap-backup Docker-Image Osixia,» [Online]. Available: https://hub.docker.com/r/osixia/openldap-backup/.

[36] «Keycloak Java API,» [Online]. Available: https://www.keycloak.org/docs-api/4.0/javadocs/index.html. [Zugriff am 30 05 2018].

[37] «Studienarbeit Hacking-Lab 2.0,» [Online]. Available: https://eprints.hsr.ch/628/.

[38] «Docker Compose Kompatibilitätsmatrix,» [Online]. Available: https://docs.docker.com/compose/compose-file/compose-versioning/#compatibility-matrix.

[39] «Docker Compose Releases,» [Online]. Available: https://github.com/docker/compose/releases.

[40] «JWT - Introduction to JSON Web Tokens,» [Online]. Available: https://jwt.io/introduction/.

[41] «JBoss - Keycloak Issue 4828,» [Online]. Available: https://issues.jboss.org/browse/KEYCLOAK-4828. [Zugriff am 02 08 2018].

[42] «docker-ce Installation docker.com,» [Online]. Available: https://docs.docker.com/install/linux/docker-ce/ubuntu/#set-up-the-repository.

[43] «install docker-compose docker.com,» [Online]. Available: https://docs.docker.com/compose/install/#install-compose.

[44] «Konzept SSO-Service,» [Online]. Available: https://blog.leog.me/discourse-sso-with-auth0-e49486d0294a.

7.4 Glossar

Abkürzung	Begriff
API	Application Programming Interface
CA	Certificate Authority
CAS	Challenge Authoring System
CCS	Challenge Consumer System
CD	Continuous Delivery
CDS	Challenge Directory System
CI	Continuous Integration
CORS	Cross-Origin Resource Sharing
CRS	Challenge Resource System
CRSS	Challenge Resource Server System
DN	Distinguished Name
FQDN	Fully-Qualified Domain Name
HL	Hacking Lab
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JWT [40]	JSON Web Token
LDAP	Lightweight Directory Access Protocol
OIDC	OpenID Connect
PAM	Linux Pluggable Authentication Modules
POSIX	Portable Operating System Interface
RDBMS	Relational Database Management System
REALM	Bereich oder Territorium
SAML	Security Assertion Markup Language
SBT	Scala Build Tool
SMTP	Simple Mail Transfer Protocol
SOP	Same-Origin-Policy
SSO	Single Sign On
UI	User Interface
VPN	Virtual Private Network
WWW	World Wide Web
XSS	Cross-Site-Scripting

Tab. 27 Glossar