HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

SECURITY
COMPETENCE

# Hacking-Lab 2.0

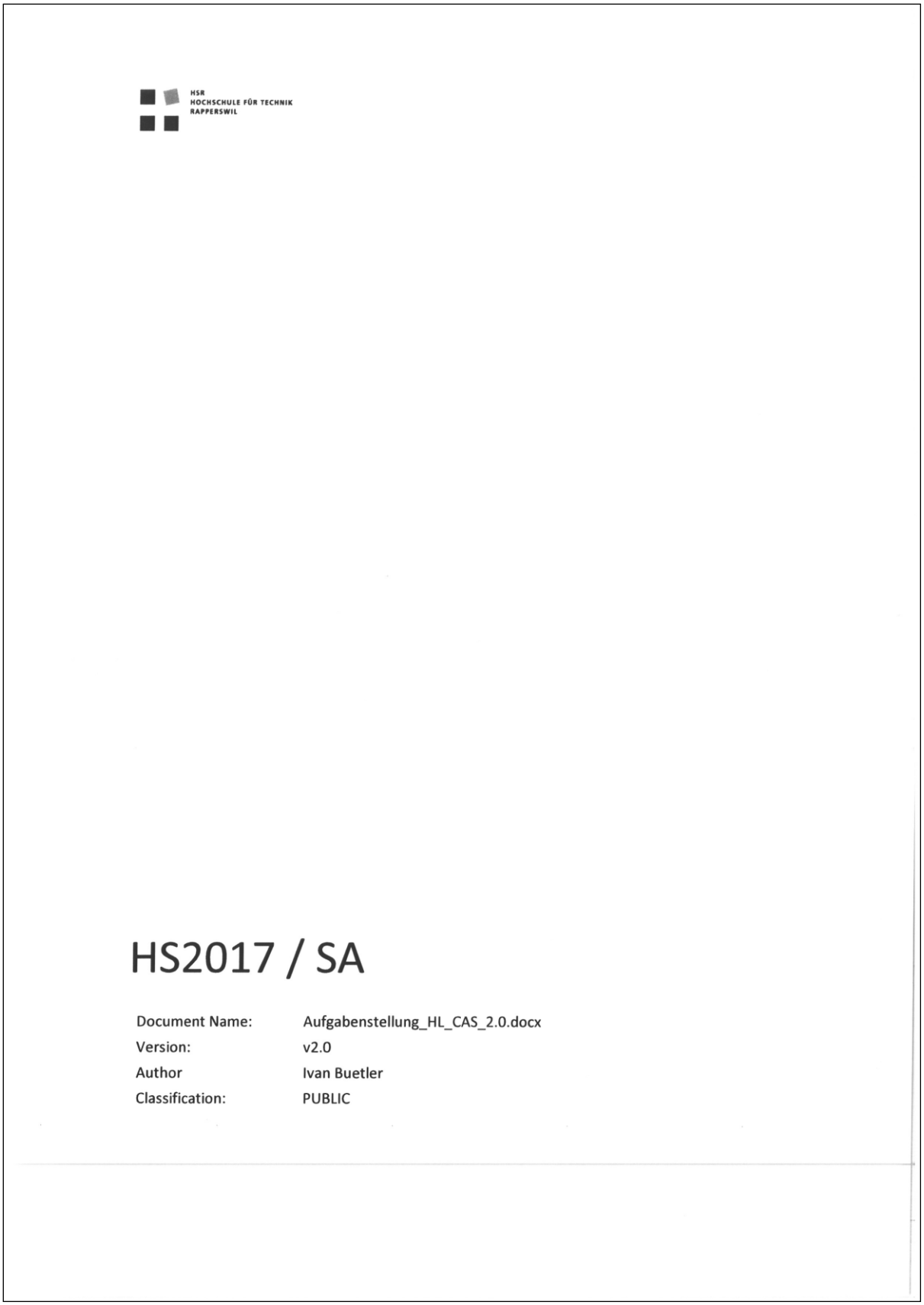
# Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2017


| | |
|---|---|
| Autor(en): | Janick Engeler, Yanick Gubler |
| Betreuer: | Ivan Bütler |
| Projektpartner: | Security Competence GmbH, 8645 Jona SG |
| Experte: | - |
| Gegenleser: | |

# 1 Aufgabenstellung



HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

HS2017 / SA

| | |
|---|---|
| Document Name: | Aufgabenstellung_HL_CAS_2.0.docx |
| Version: | v2.0 |
| Author | Ivan Buetler |
| Classification: | PUBLIC |

**Abb. 1 unterschriebene Aufgabenstellung - Seite 1**

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

## Table of Content





HSR Hochschule für Technik
Postfach 1475
CH-8640 Rapperswil
www.hsr.ch

**Abb. 2 unterschriebene Aufgabenstellung - Seite 2**

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

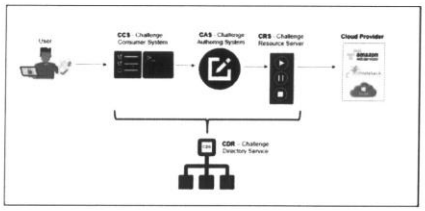
# 1 Ausgangslage

## 1.1 Einleitung

Die Security Competence GmbH ist für die Entwicklung und den Betrieb des Hacking-Lab verantwortlich. Das Hacking-Lab ist eine Übungsplattform für angehende Informatik und Security Spezialisten. Studierende können im Hacking-Lab eine Hands-On Übung auswählen und lösen. Dafür erhalten die Studierenden Punkte und Steigen im Ranking. Der Lehrer hat dabei den Überblick über den Fortschritt seiner Studierenden.

Das bisherige Hacking-Lab kann die neuen Anforderungen und Bedürfnisse nicht abdecken. So soll beispielsweise das Erfassen von Challenges einfacher werden. Lösungen der Studierenden sollen automatisch korrigiert werden können. Universitäten sollen das Hacking-Lab nach ihrem Geschmack aussehen lassen (Branding) können. Die Challenges sollen in mehreren Sprachen verfügbar sein. Das Hacking-Lab soll auch auf dem Mobile Phone nutzbar sein.

Die Security Competence GmbH hat sich daher für eine komplette Neuentwicklung entschieden. Grundsätzlich soll das Hacking-Lab 2.0 nicht mehr monolithisch, sondern ein weltweit verteiltes System nach den Grundsätzen des "Micro Service" Konzept werden. Das folgende Diagramm zeigt die wichtigsten Eckpfeiler von Hacking-Lab 2.0.




## 1.2 Auftrag

Die Studierenden sollen im Rahmen ihrer Studienarbeit ein mögliches Micro-Service Konzept der Zukunft entwerfen und darin enthaltend einen Prototyp für das Erfassen von Challenges implementieren. Da es die erste Arbeit im neuen Hacking-Lab 2.0 Kontext ist, gibt es noch keine bestehenden Schnittstellen und Interfaces zu anderen Micro-Services. Dies erschwert die Aufgabe für die Studierenden. Deshalb sollen und dürfen die Studierenden jene Systeme simulieren oder als quick-and-dirty Protoype bereitstellen, die für das Erfassen der Challenges absolut notwendig sind. Am Ende der Studienarbeit soll ein mögliches Konzept für das Hacking-Lab 2.0 vorliegen und einen funktionierenden Prototyp für das Erfassen von Challenges.



HSR Hochschule für Technik
Postfach 1475
CH-8640 Rapperswil
www.hsr.ch

**Abb. 3 unterschriebene Aufgabenstellung - Seite 3**

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

### 1.3 Anforderungen an die Analysephase

Es wird von den Studierenden erwartet, dass der Input zum Hacking-Lab 2.0 Konzept sowohl beim Industriepartner abgeholt, als auch mit dem HSR Betreuer abgestimmt werden. Viele Themen stehen zu Beginn der Arbeit noch nicht als konkrete Anforderung fest. Mittels den Workshops sollen die Studierenden ein Gefühl für die gewünschte Lösung erhalten und diese im Bericht entsprechend dokumentieren. Auf eine Analyse nach wissenschaftlichen Grundsätzen kann verzichtet werden.

### 1.4 Anforderungen an die Technik

Grundsätzlich erwartet der Industriepartner die Übergabe der Software als Source Code bei Github und in der Form von Docker Images. Sie erlauben es der Security Competence GmbH relativ einfach, die entwickelten Komponenten selbständig zu testen und den Studierenden entsprechendes Feedback zu geben.

### 1.5 Anforderungen an das Challenge Authoring System

Das System für die Erfassung und Bereitstellung von Challenges soll aus zwei getrennt voneinander funktionierenden Systemen bereitgestellt werden.

| System | Erklärung |
|---|---|
| **Challenge System** | RESTful API für Konsumenten von Challenges. Sie beziehen über dieses API die Aufgabenstellung, Hinweise, Level, Lösungen etc. |
| **Challenge Erfassung** | RESTful API und GUI Komponente für die Erfassung von Challenges. SSO Login via JWT Token. |

### 1.6 Vorgaben des Industriepartner

Die Security Competence GmbH macht den Studierenden folgende Technologievorgaben, welche bei der Entwicklung des Prototyp zu berücksichtigen sind.

| System | Erklärung |
|---|---|
| **Programmiersprache** | • Entwicklung in Java unter Verwendung des Play Framework |
| **Datenbanken** | • MySQL oder MariaDB |
| **Single-Sign-On** | • SSO mittels JWT |



HSR Hochschule für Technik
Postfach 1475
CH-8640 Rapperswil
www.hsr.ch

**Abb. 4 unterschriebene Aufgabenstellung - Seite 4**

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

## 1.7 Erwartetes Ergebnis

Neben den sonst üblichen Ergebnissen gemäss SA/BA Wegleitung der HSR, wird als konkretes Ergebnis folgende zwei Items erwartet

| System | Erklärung |
|---|---|
| **Analyse** | • Beschreibung Hacking-Lab 2.0 Konzept<br>• Micro-Service Architektur<br>• Überlegungen / Gedanken |
| **Software** | • Challenge Authoring System als Docker Container<br>• Funktionierender Prototyp zum Erfassen von Challenges<br>• Simulation der Umsysteme |



HSR Hochschule für Technik
Postfach 1475
CH-8640 Rapperswil
www.hsr.ch

**Abb. 5 unterschriebene Aufgabenstellung - Seite 5**

# 2 Abstract

## 2.1 Ausgangslage

Die Security Competence GmbH möchte im Rahmen der HSR Studienarbeiten HS2017 ein Konzept für ein neues Hacking-Lab 2.0 ausarbeiten und dazu einen ersten Prototyp erstellen. Das jetzige Hacking-Lab dient in erster Linie dazu Hacking Aufgaben (Challenges) zu lösen und wird vor allem in der HSR zum praktischen Lernen der Informationssicherheit und bei Compass Security für spezielle Events verwendet.
Probleme des jetzigen Systems sind unter anderem, dass das bestehende User Interface weder zeitgemäss noch mobilefähig ist. Weiter kann es nur einsprachig betrieben werden. Einer der grössten Nachteile für die Zukunft ist, dass es nicht einfach erweiterbar ist und man so keine Skalierbarkeit gewährleisten kann.
Ein weiterer Nachteil ist, dass es für die Betreiber einen grossen Mehraufwand generiert, der dadurch zustande kommt, dass erstens laufend neue Challenges erfasst und bearbeitet werden müssen und zweitens die Lösungen der durchgeführten Übungen mehrheitlich von Hand korrigiert werden.
Das jetzige System basiert auf einer sogenannten Live-CD (beinhaltet Hackingumgebung und /-tools), was einen unnötigen Aufwand für den Endanwender bedeutet.

## 2.2 Ziel

Das bestehende System soll nach und nach abgelöst und durch ein neues ersetzt werden. Für das neue System wird der Fokus bereits zu Beginn auf eine weltweite Nutzung gelegt. Dies führt jedoch zu einer neuen Problematik, da z.B. Länder wie China ihren verschlüsselten Internetverkehr drosseln und es somit schwierig wird aus diesem Land auf Server anderer Länder zuzugreifen. Dies würde zu regelrechten Performance Engpässen führen und so eine sinnvolle Nutzung verhindern. Deshalb wird eine hohe Skalierbarkeit angestrebt, was bedeutet, dass mehrere Instanzen des Hacking-Lab 2.0 parallel betrieben werden können und so der Verkehr über die Landesgrenzen hinaus minimal gehalten werden kann.
Eine weltweite Nutzung bedeutet auch, dass das System mehrsprachig betrieben werden können soll, so dass auch weniger sprachaffine Personen die Aufgaben ohne Übersetzungsprobleme lösen können.
Ein weiterer entscheidender Punkt ist der Mehraufwand. Diesem Problem soll dadurch begegnet werden, dass eine Community aufgebaut wird, welche selbstständig neue Challenges und Musterlösungen erfasst. Dadurch können auch die Übersetzungen direkt von einem Muttersprachler erstellt werden.
Damit die Qualität der angebotenen Inhalte nicht unter dem Community-Gedanken leidet, soll ein Review-System implementiert werden, wodurch neue oder veränderte Challenges durch einen Nutzer mit entsprechenden Rechten kontrolliert und freigegeben werden müssen, bevor sie für den Normalanwender zugänglich sind.
Damit die Live-CD des bestehenden Hacking-Labs in der Version 2.0 nicht mehr benötigt wird, sollen die benötigten Ressourcen (Docker-Container, VM’s, usw.) direkt einer Challenge zugewiesen werden können.

## 2.3 Ergebnis

Das Ergebnis dieser SA soll in erster Linie ein intelligentes Konzept sein, was alle zuvor genannten Probleme sinnvoll abdeckt bzw. verhindert. Deshalb wurde die erste Hälfte der Zeit dafür verwendet dieses Konzept im Gespräch mit dem Betreuer zu erarbeiten.
Nach der konzeptuellen Ausarbeitung wurde ein erster Prototyp des Challenge Authoring System (CAS) implementiert, der für die anderen verwendeten Systeme ein REST-API anbietet, um dort auf einfache Weise die entsprechenden Daten beziehen zu können. Das CAS besteht grundsätzlich aus drei Komponenten.

- CAS-Client welcher das GUI für die Challenge Erstellung und Übersetzung liefert
- CAS-Server welcher die Daten verwaltet und das API anbietet
- CAS-MySQL welches die Datenbank hält

Alle Subsysteme werden in einem eigenen Docker-Container implementiert. So kann schnell und einfach ein neues CAS hochgefahren werden, indem man lediglich das GIT-Repo klont und die darin enthaltenen Skripts ausführt.

# 3 Management Summary

## 3.1 Ausgangslage

Das bestehendes Hacking-Lab System, der Security Competence GmbH stellt die Betreiber nicht mehr vollends zufrieden. Es entspricht weder dem neusten Stand der Technik, noch kann man es mit den gegebenen Mitteln für eine weltweite Nutzung erweitern. Dazu kommt, dass kein wirtschaftlicher Nutzen vorhanden ist. Desweitern ist es für die Betreiber mit grossem Mehraufwand verbunden, die Daten des Systems zu pflegen und auf dem neusten Stand zu halten. Ein weiteres Defizit des aktuellen Systems ist, dass mit einer Live-CD gearbeitet werden muss. Dies führt dazu, dass die Anwender selbst dafür sorgen müssen, die Live-CD-Umgebung auf dem neusten Stand zu halten, sowie für deren Lauffähigkeit zu sorgen. Diesen Problemen sollte durch einen grundlegenden Neuentwurf der aktuellen Lösung entgegengewirkt werden.

## 3.2 Vorgehen/Technologien

In Zusammenarbeit mit der Security Competence GmbH wurde ein Konzept für ein neues Hacking-Lab 2.0 ausgearbeitet. In dieser konzeptuellen Analyse wurde versucht, den Fokus in die Zukunft zu richten und die Probleme der bestehenden Version auszumerzen. Die technische Machbarkeit wurde dann in einem zweiten Schritt durch kleinere Prototypen überprüft. Die daraus gewonnenen Erkenntnisse flossen darauf wieder in das Konzept und den endgültigen Prototyp mit ein. Damit nach dieser Studienarbeit ein allfälliges anderes Team die Arbeit weiterentwickeln kann, wurde die technische Dokumentation der Architektur derart ausführlich beschrieben, dass dies kein Problem darstellen wird. Auch wurde während der programmiertechnischen Arbeit Wert daraufgelegt, einen sauberen Stil zu verfolgen, sowie ein architektonisches Modell auszuarbeiten, welches mit zukünftigen Neuerung gut zurechtkommt. So kann die Einarbeitungszeit für ein anderes Team möglichst geringgehalten werden.

## 3.3 Ergebnisse

Das Ergebnis dieser Studienarbeit ist in erster Linie ein ausgereiftes Konzept, welches in Workshops zusammen mit dem Projektpartner und Betreuer erarbeitet wurde. Dieses beinhaltet die entscheidenden Fragen, die sich uns während der Analyse stellten, sowie die Probleme, welche dabei auftraten. Es wurde versucht alles in einem nachvollziehbaren Rahmen zu dokumentieren, damit später nicht erneut dieselben Fragen wieder beantwortet werden müssen. Das Konzept zeigt auf, wie man sich das neue Hacking-Lab 2.0 vorzustellen hat und welche wichtigen Komponenten bzw. Systeme es beinhaltet.

Im zweiten Teil der Arbeit wurde ein lauffähiger Prototyp erstellt, der für die Erfassung neuer Challenges benutzt werden kann. Dabei wurde grossen Wert daraufgelegt, dass das System beliebig erweitert und sehr einfach neue Instanzen hochgefahren werden können, damit auch in anderen Ländern solche Systeme verwendet werden können, ohne grosse Anpassungen zu tätigen.

## 3.4 Ausblick

Mit dieser Studienarbeit wurde eine gute Grundlage geschaffen, auf der das Hacking-Lab 2.0 weiterentwickelt werden kann. Die Systeme sind so voneinander gekapselt, dass es möglich wäre, mehrere parallele Teams mit der Weiterentwicklung zu beauftragen. Weiter wurde die Architektur und die entscheidenden Punkte des Konzepts so dokumentiert, dass auch einer unbeteiligten Person der Einstieg in das Projekt einfach ermöglicht werden kann.

# 4 Inhalt

# Hacking-Lab 2.0

## 5 Technischer Bericht

Janick Engeler
Yanick Gubler

## 5.1 Ausgangslage & Problembeschreibung

Das Hacking-Lab [1], welches momentan im Einsatz ist, stösst auf grossen Anklang. Es ist sowohl bei den Studierenden, welche dieses Tool im Modul Informationssicherheit 3 der HSR verwenden, wie auch bei der Compass Security [2] und weiteren Unternehmen sehr beliebt. Grundsätzlich funktioniert das System so wie es zurzeit eingesetzt wird relativ gut. Jedoch gibt es einige Defizite. So muss man zum Beispiel für gewisse Aufgaben eine Live-CD [3] starten, welche ein virtuelles Linux-System beinhaltet. Erst mit dieser virtuellen Maschine können dann bestimmte Challenges im Hacking-Lab gelöst werden. Somit ist jeder Benutzer selbst für die Aktualität und Lauffähigkeit des Systems verantwortlich.

Weiter ist das Hacking-Lab 1.0 nicht wirtschaftlich. So muss die Firma Compass Security stets selbst neue Challenges entwerfen, was mit grossem Aufwand verbunden ist. Nicht zuletzt, weil die Erstellung auf Dateiebene auf dem Webserver geschieht. Diese Challenges sind dann auch nur in der Sprache Englisch verfügbar. Dies ist nicht wirklich wirtschaftlich, da nicht davon ausgegangen werden kann, dass alle Englisch verstehen oder die Aufgaben in Englisch lösen können oder möchten und sich so möglicherweise potenzielle Kunden von diesem Produkt distanzieren.
Ein weiterer Punkt, der mit der Wirtschaftlichkeit des Produktes im Zusammenhang steht ist, dass ein potenzieller Kunde, der das Hacking-Lab verwenden möchte, dieses vielleicht auf seine eigenen Bedürfnisse bzw. auf seine internen Designrichtlinien anpassen möchte (sogenanntes Kunden-Branding), was mit der aktuellen Version nicht möglich ist.

Ein weiteres Problem der jetzigen Installation soll folgende Grafik visualisieren.

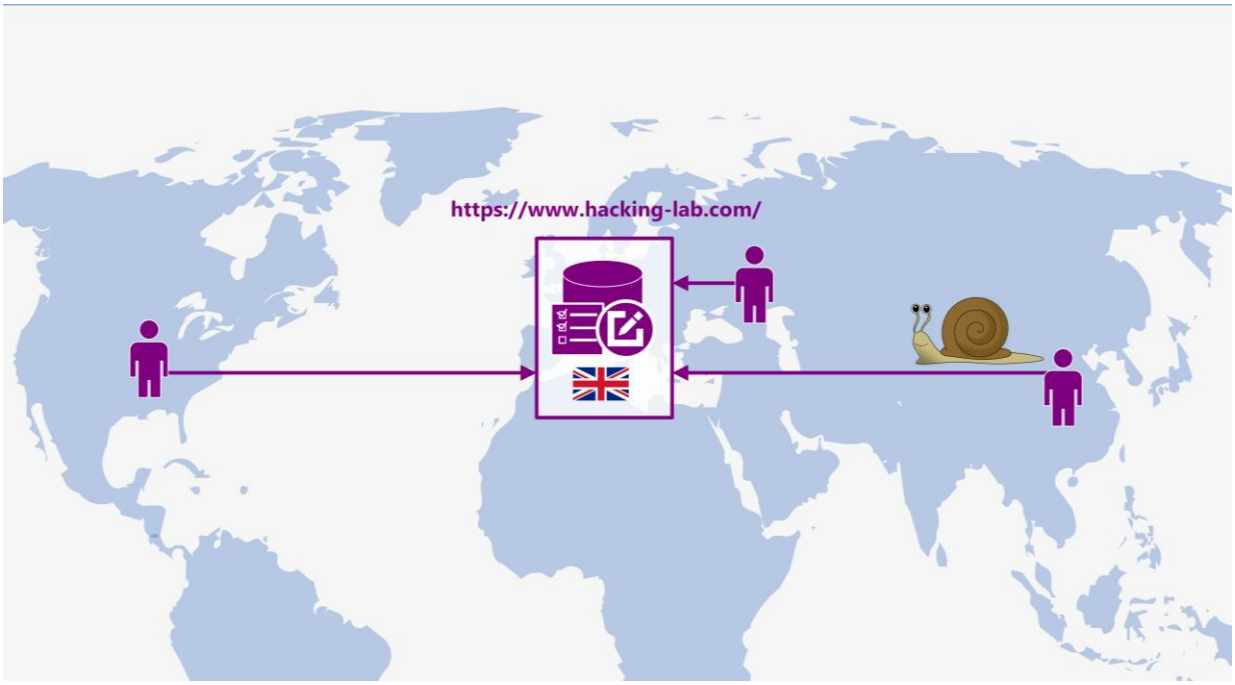



Abb. 6 Hacking-Lab 1.0

Darauf ist deutlich zu erkennen, dass es nur einen zentralen Dienst für das Hacking-Lab gibt. Somit greifen weltweit alle Hacking-Lab Benutzer auf einen zentralen Dienst zu. Besonders problematisch ist das bei den chinesischen Staatsbürgern. Denn die Volksrepublik China hat 2013 das Projekt «Goldener Schild» [4] ins Leben gerufen, welches zur Überwachung und Zensur des Internetverkehrs in China eingesetzt wird. Dieses blockiert unter anderem diverse Internetseiten, wie z.B. Google. Die Problematik kommt aber daher, dass verschlüsselte Verbindungen, welche die Landesgrenze von China verlas-

sen, mittels QoS (Quality of Service) gedrosselt werden, somit auch das Hacking-Lab, welches via verschlüsseltem Protokoll «https» mit seinen Benutzern kommuniziert. Somit ist die Verwendung zu wenig performant. Dieses Problem wird in dieser Arbeit verkürzt als «Chinaproblem» betitelt.

## 5.2 Lösungskonzept

Ein grosser Bestandteil dieser Studienarbeit ist die Ausarbeitung eines neuen Konzeptes für das Hacking-Lab, welches oben beschriebene Probleme löst und den Namen Hacking-Lab 2.0 tragen soll.

In der neuen Version soll dem Betreiber einiges an Arbeit abgenommen werden. Dies soll durch die Bildung einer Community erreicht werden. Diese soll selbstständig Challenges erfassen und bearbeiten können. Damit die Qualität der Challenges trotzdem hoch bleibt, soll ein Review-Prozess entwickelt werden. Somit werden neue Challenges nicht gleich veröffentlicht und müssen zuerst von anderen Community-Mitgliedern überprüft werden. Es ist jedoch anzunehmen, dass die Community solche Arbeiten nicht ohne eine Gegenleistung übernehmen wird. So wurde das Konzept eines Belohnungssystems entworfen. Dabei soll jeder Ersteller, Überprüfer und Übersetzer einer Challenge belohnt werden. Dies soll mit einer internen Währung dem HL$ (Hacking-Lab Dollar) erreicht werden. Die Währung kann dann später eingelöst werden, um beispielsweise neue Challenges zu erwerben.

Mithilfe des Community-Gedankens wird es nun auch dem einzelnen Benutzer möglich, selber Challenges in andere Sprachen zu übersetzen. Denn das Hacking-Lab 2.0 soll auch vollumfänglich in mehreren Sprachen verfügbar sein. Da die Übersetzungsarbeiten direkt durch die Community erledigt werden kann, wird auch die Qualität der Übersetzungen hoch sein, weil diese so direkt von Benutzern erstellt werden können, die diese Sprache sehr gut beherrschen oder diese als Muttersprache sprechen. Um jedoch auch hier den Vorteil des Vier-Augen-Prinzip vollumfänglich ausschöpfen zu können, wird der Review-Prozess auch für Übersetzungen angestrebt.

Mit diesem neuen Konzept sollen viele neue Challenges in guter Qualität und mehreren Sprachen angeboten werden können. Da dies nun ohne viel Aufwand der Betreiber geschehen wird, können diese sich anderen Aufgaben zuwenden. Wie zum Beispiel dem Vertrieb von Challenges oder Instanzen des Hacking-Lab 2.0. So kann sich beispielsweise ein Unternehmen entgeltlich den Luxus gönnen und eine eigene Infrastruktur betreiben. Das Unternehmen kann dann diese selber mit ihren eigenen Challenges befüllen oder alternativ von anderen Anbieter einkaufen. So können Kunden auch sicherheitskritische Challenges für ihr Unternehmen erstellen, um seine eigenen Mitarbeiter zu schulen. Dies bringt den Vorteil, dass potenzielle Sicherheitslücken einer Firma, auf die ihre Mitarbeiter geschult werden sollen, nicht nach aussen dringen oder besser gar nicht erst vorkommen. Mit all diesen neuen Konzepten soll die neue Version auch wirtschaftlichen Nutzen erbringen. Da das System in der verbesserten Ausführung auch mobilfähig sein wird und es so auf jedem belieben Smartphone wie auch Tablets funktionieren soll, wird auch das Marktpotenzial enorm gesteigert.

Um die Marktkapazität noch mehr auszureizen, gibt es ein Konzept, um das «Chinaproblem» zu bewältigen. Dies soll durch Dezentralisierung des Hacking-Lab Dienstes realisiert werden. Eben dieses Konzept der Dezentralisierung nahm den grössten Teil der Studienarbeit ein. Wie das dafür notwenige Konzept aussieht, soll folgende Grafik veranschaulichen.

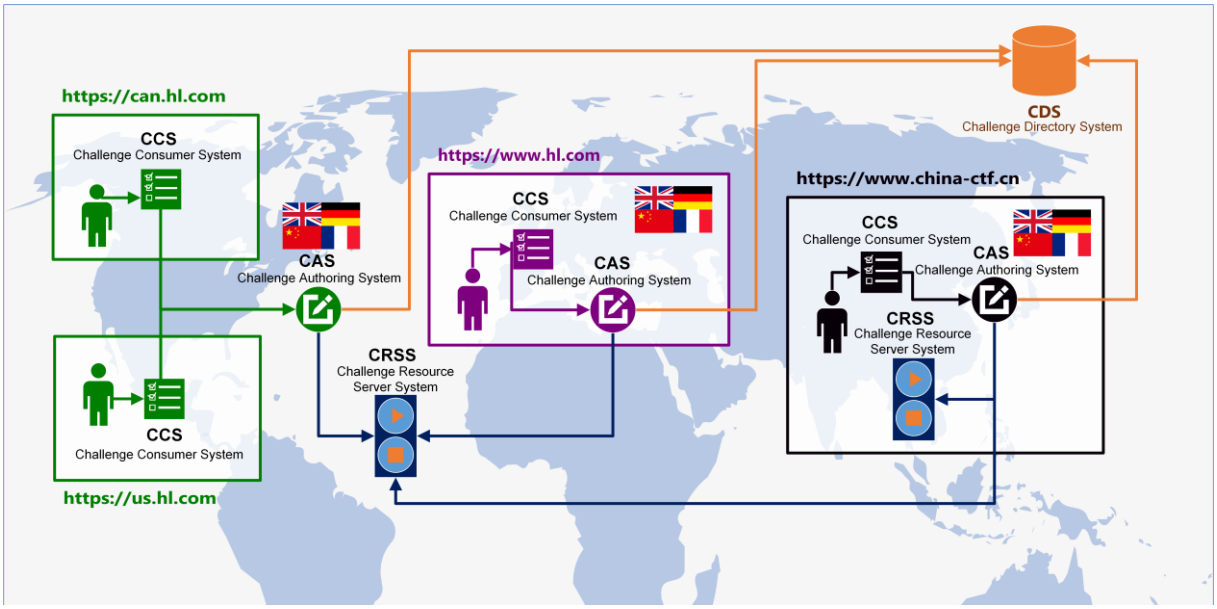



Abb. 7 Hacking-Lab 2.0

Wie in der Grafik ersichtlich ist, wird es global nur eine Komponente geben, welche als zentrale Anlaufstelle fungiert. Diese wird mit restdb.io [5] realisiert und nennt sich CDS (Challenge Directory System). Das CDS ist im Grunde genommen nur eine Datenbank, welche eine Schnittstelle nach aussen anbietet. Da das CDS mittels des genannten Dienstes realisiert wird, sind bereits alle REST-Schnittstellen [6] für das Abfragen, Eintragen, Aktualisieren und Löschen der Datensätze standardmässig vorhanden. Das Challenge Directory System wird benötigt, um eine Liste aller global verfügbaren Challenges mit einer global eindeutigen ID, realisieren zu können. Weiter sollen in der benannten Datensammlung auch alle Metadaten zu den Teilsystemen und Ressourcen des Hacking-Lab 2.0 gehalten werden. Somit weiss das CDS zu jeder Zeit, welche Challenges, Ressourcen und Teilsysteme (CCS, CAS, CRSS) des Hacking-Lab 2.0 zur Verfügung stehen und wo sich diese genau befinden. Da im Hacking-Lab 2.0 nicht mehr jeder Benutzer eine Verbindung zum zentralen Datenpunkt (CDS) aufbaut, sondern nur noch jedes CAS (Challenge Authoring System), wird es auf diese Weise kein Problem mehr darstellen, dass es global nur einmal zur Verfügung steht. Somit sollte auch das «Chinaproblem» gelöst sein, da der Datenverkehr der die Landesgrenzen der Volksrepublik verlässt minimal gehalten und die übermittelte Datenmenge sehr klein wird.

## 5.3 Umsetzung

Zu Beginn der Umsetzungsdokumentation soll die Analyse bzw. die Ausarbeitung des Konzeptes detailliert erläutert werden. Diese Analyse wird aufgrund der Übersichtlichkeit in kleinere Abschnitte aufgeteilt.

### 5.3.1 Datenmodell

In einem ersten Schritt überlegten wir uns anhand der ersten groben Aufgabenstellung, das Grundkonstrukt einer Challenge und ihrer dazugehörigen Komponenten. Wir stellten uns die Frage, welche Informationen eine Challenge überhaupt genau besitzen muss und wie diese sinnvoll definiert werden müssen, ohne Redundanzen zu produzieren. Um die wichtigsten Punkte dieses Analyse-Schrittes aufzuzeigen werden hier die einzelnen Punkte kurz aufgeschlüsselt.
Daraus wurde dann ein Datenmodell entwickelt, welches nach den Besprechungen laufend ergänzt wurde. So entstand schlussendlich das weiter unten auffindbare Domänenmodell.

#### 5.3.1.1 Challenge und deren Versionen

Eine Challenge ist ein simpler Behälter, welcher die statischen Metadaten beinhaltet. Diese verändern sich nie und werden einmal zu Beginn gesetzt. Es handelt sich dabei um die Attribute für die Kategorie einer Challenge, sowie ihres Typen und dem zugehörigen Mandanten. Ein weiteres statisches Attribut ist, ob die Challenge als Privat gesetzt wurde und so nur in der erstellenden CAS Anwendung auffindbar ist. Dies können Aufgaben sein, welche mit unternehmensinternen Daten arbeitet oder Übungen im Unternehmensumfeld sind. Diese sollen nicht öffentlich zugänglich sein, wodurch diese entsprechend im CDS gekennzeichnet sein müssen, oder gar nicht erst dorthin exportiert werden.

Die Hauptkomponente des Datenmodells sind die Versionen einer Challenge. Pro Challenge werden jeweils zehn Versionen gespeichert. Ältere werden gelöscht. Sie beinhalten alle dynamischen Daten einer Challenge, wie Level, Abstract, Name, Titel usw.
Damit nur jeweils eine Version aktiv ist und von den Anwendern gelöst werden kann, wurde ein Status eingefügt. Dieser kann sogleich dafür verwendet werden, den gesamten Review-Prozess abbilden zu können. Eine weitere Verwendung des Challenge Status kommt beim Export von Challenges zum Zuge. Hier soll es genau wie beim Lösen möglich sein, nur jeweils eine Version zu extrahieren und in einem anderen System einzufügen. Damit es keine Duplikate gibt und eine Challenge auf allen Systemen die gleiche ID besitzt, wird beim Erstellen einer Challenge auf dem globalen Directory ein Eintrag angelegt um eine eindeutige ID zu erhalten. So muss auf der lokalen Datenbank eines jeden CAS lediglich die ID aus dem CDS gesetzt werden, um die Eindeutigkeit sicherzustellen.

#### 5.3.1.2 Steps, Hints und Instructions

Im Zusammenhang des Datenmodells wurde sogleich das Konstrukt der Steps, welche Hints und Instructions beinhalten können, eingeführt.
Ein Step beschreibt einen Teil einer Aufgabe, die in einer Challenge gelöst werden soll. Er beinhaltet beliebig viele Hints und Instructions, welche als Hilfestellungen für das Lösen einer Challenge gesehen werden müssen. Der Unterschied zwischen den beiden ist, dass ein Hint dazu dient den Anwender in eine bestimmte Richtung zu lenken, wobei eine Instruction die genauen Schritte aufzeigt, welche vorgenommen werden müssen, um die Challenge zu lösen. Falls der Anwender eine Hilfestellung für das Lösen einer Challenge einblendet, werden ihm Punkte von der maximal für die Challenge möglichen Punktzahl abgezogen (Maximal Punktzahl / (2 * Anzahl Hilfestellungen)). Eine Challenge kann mehrere solcher Steps beinhalten.

#### 5.3.1.3 Maximal Punktzahl

Zu der maximalen Punktzahl für eine Challenge gibt es hier zu sagen, dass die Punkte zumeist durch das Level der Challenge gezogen werden. Je grösser der Schwierigkeitsgrad der Challenge, desto mehr Punkte werden für deren Lösung vergeben. Es gibt hier jedoch den Spezialfall, dass für einige Events, die maximale Punktzahl der Challenge explizit gesetzt werden möchte. Dazu wurde direkt auf der Challenge ein neues Feld angedacht, welches den Wert des Levels überschreibt. Dies kann dann der Fall sein, wenn man z.B. für das Lösen einer Challenge nur einen Punkt vergeben möchte und ansonsten keinen.

#### 5.3.1.4 Challenge Lösung (Goldnugget)

Im Zusammenhang mit der Lösung einer Challenge seien hier auch die Überlegungen dazu kurz erwähnt. Zu Beginn haben wir definiert, dass die Musterlösung einer Challenge direkt auf dem CAS pro Challenge gespeichert wird. Dadurch resultiert jedoch die Problematik, dass wenn ein User die Lösung gefunden hat, sie einfach an seine Kollegen weitergeben kann, ohne dass diese die Challenge lösen müssen. Um dieses Problem zu umgehen wurde angedacht, dass pro User und Challenge eine personalisierte Lösung gespeichert wird, welche einmalig ist und so nicht weitergegeben werden kann. Diese als «Goldnugget» bezeichnete Lösung wird durch das Challenge Resource Server System kurz CRSS generiert und beim Anfordern einer Ressource zuerst an das CAS und danach innerhalb der Ressource zum Consumer übermittelt.

#### 5.3.1.5 Medien

Dieses Konstrukt wird zur Speicherung verschiedenster Medien, wie Tondateien, Bilder oder Videos gebraucht. Für jedes dieser Dateien wird die dazu passende URL in der Datenbank gespeichert, damit diese einfach wieder auf dem Server gefunden werden können. Das wichtige hierbei ist, dass Medien in allen grösseren Konstrukten zur Anwendung kommen (Abstract, Step, Hint, Instruction) und so überall korrekt dargestellt werden müssen.

#### 5.3.1.6 Rating

Um die Qualität einer Challenge messen zu können, wurde ein Rating-System angedacht, bei dem ein Anwender eine Challengeversion bewerten kann. Dazu wird eine fixe Anzahl Bewertungspunkte vergeben, sowie optional ein beschreibender Text hinzugefügt. Zusätzlich wird der Ersteller der Bewertung hinterlegt, damit ein User nicht mehrmals eine Challenge bewerten kann. Ansonsten könnte der Ersteller einer Challenge, seine eigens erstellte Challenge so lange gut bewerten, bis sie für andere Anwender immer zuoberst auf der Liste erscheint. Solch ein Verhalten soll zu Beginn damit verhindert werden.

#### 5.3.1.7 Issues

Falls ein Anwender oder ein Reviewer ein Fehler in einer Challenge entdeckt, soll es möglich sein, dass er einen Issue zu dieser Challenge erfasst. Damit nicht ein gesamtes Issue System zusätzlich implementiert werden muss, wurde überlegt die Issues auf GitHub [7] zu speichern und im CAS nur eine Referenz darauf zu hinterlegen und welcher Benutzer den Issue erfasst hat. So hätte man die Möglichkeit über das GitHub interne API die Issues abzuholen oder neue anzulegen.

#### 5.3.1.8 UserCAS

Damit eine erfasste Challenge, ein Rating oder ein Issue einem Benutzer zugeordnet werden kann, wird ein Benutzer Objekt auf dem CAS benötigt. Es wird dabei der Name des Benutzers, sowie seine bevorzugte Sprache und sein Salary gespeichert. Das Benutzer Objekt wird durch das mitgelieferte JWT [8] auf der Datenbank angelegt.

Zu Beginn war eine Überlegung den Benutzer im globalen Directory zu speichern, damit er nur dort und nicht auf allen anderen Systemen wie CAS oder Consumer gespeichert werden muss. Dies hätte jedoch zur Folge gehabt, dass bei jedem Anmelden eine Anfrage auf diesen globalen Knotenpunkt getätigt werden müsste, wodurch das «Chinaproblem» wieder zum Zuge käme. Eine andere Überlegung in diesem Zusammenhang war, dass die Benutzerdaten aus Datenschutzgründen besser nur lokal und nicht auf einem globalen System gespeichert werden sollten.

#### 5.3.1.9 Rollen

Da unter anderem ein rollenbasiertes Review-System angestrebt wird, ist es unerlässlich diese korrekt auf der Datenbank des CAS zu persistieren. Die Schwierigkeit hierbei ist, dass ein Benutzer mehrere Rollen haben kann und ein Benutzer auf mehreren Mandanten bzw. Konsumenten vorkommen kann. So musste eine Zwischentabelle erstellt werden, wo dieser Zusammenhang abgebildet werden konnte. Analog zum Benutzer werden auch mittels JWT die einzelnen Rollen als JSONArray übertragen und falls nicht vorhanden angelegt. Initial werden die Rollen: Admin, Reviewer, Author, Contributor, Translator und User in der Datenbank angelegt.

#### 5.3.1.10 Mandanten Fähigkeit

Da eine entscheidende Anforderung des Industriepartners das Beibehalten der Mandantenfähigkeit sein sollte, musste auch dieses in das Datenmodell mit einfliessen. Da es jedoch möglich sein kann, dass man pro Mandant (was schlussendlich ein anderes Wort für den Konsumenten des CAS ist) mehrere unterschiedliche Rollen pro Benutzer einnehmen kann, musste das entsprechend überlegt und abgebildet werden.

#### 5.3.1.11 Sprache und Übersetzung

Da bei der neuen Implementierung die Mehrsprachigkeit im Vordergrund stehen sollte, wurde bereits zu Beginn ein eleganter Weg gesucht, um diese verschiedenen Übersetzungen sauber speichern zu können. Dazu wurden einige Versionen ausgearbeitet und im Team besprochen. Schlussendlich haben wir uns auf die beste Version geeinigt. Diese sieht eine Tabelle für die Definition der eigentlichen Sprache vor, sowie eine für das Speichern einer Attribut-ID mit einem zusätzlichen Typen, zur einfacheren Selektion und einer dritten für das Speichern der eigentlichen Übersetzung pro Sprache. Der Vorteil dieser Lösung ist, dass sehr schnell über die Attribut-ID alle verfügbaren Übersetzungen für genau ein Attribut gesucht werden können oder für jeden Typ eine Liste generiert werden kann, welche dann extern übersetzt und wieder in die Datenbank eingefügt werden kann. Die Attribut-ID wird dann jeweils auf den anderen Tabellen verwendet, um Redundanzen zu verhindern, wie im Datenmodell unten unschwer zu erkennen ist.

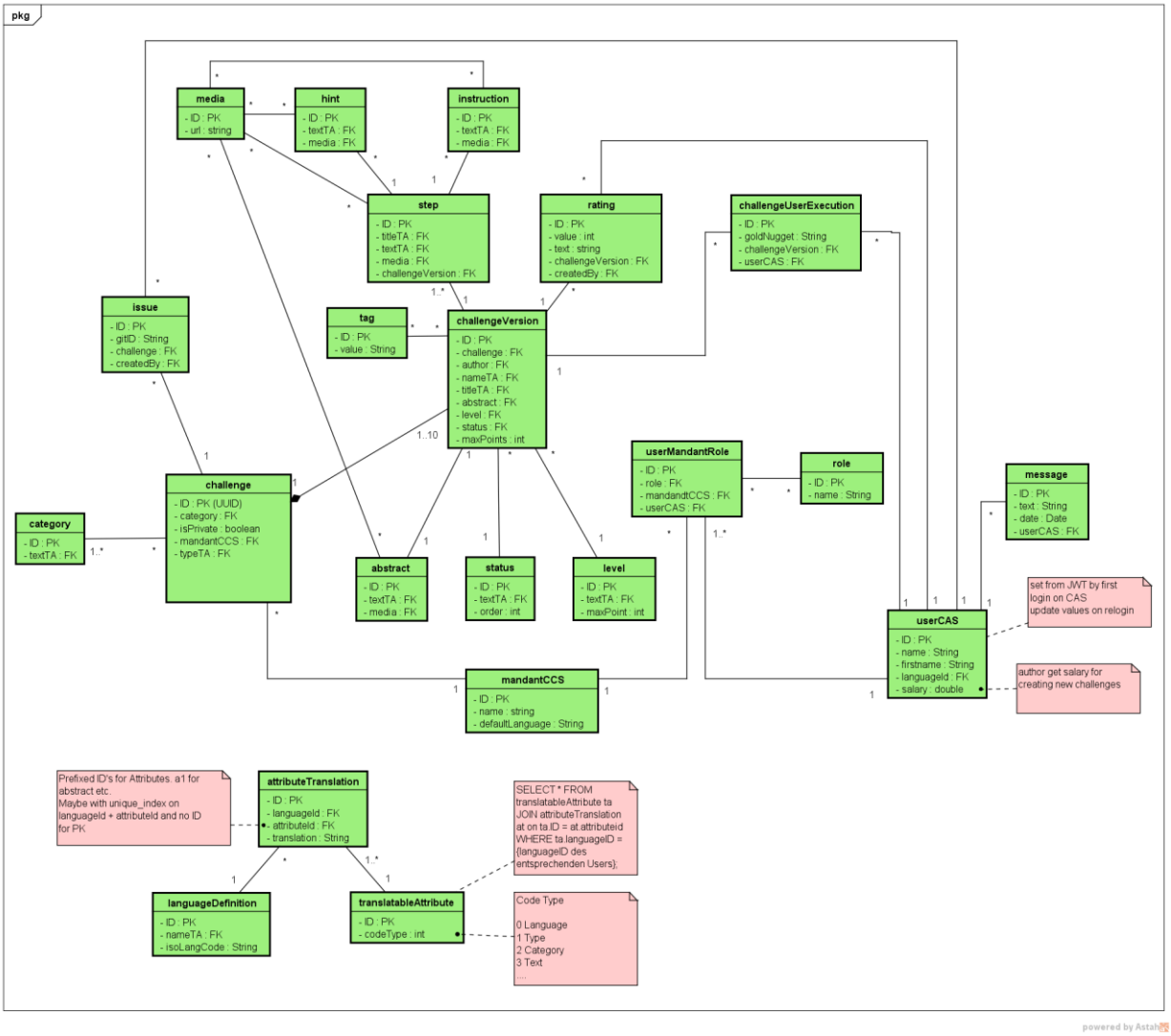



**Abb. 8 Domain Model**

## 5.3.2 Weitere Konzeptentscheidungen

Neben den konzeptuellen Entscheidungen zu den verwendeten Daten, wurden auch weiterführende Entscheidungen getroffen. Diese sollen in diesem Abschnitt beschrieben werden.

### 5.3.2.1 Review-Prozess

Der Review-Prozess stellt eine wichtige Komponente innerhalb des neuen Hacking-Lab 2.0 dar, wie oben bereits mehrfach angetönt wurde. Er sollte folgendermassen aufgebaut werden. Nachdem eine neue Challenge oder Übersetzung erstellt wurde, wird sie mit dem Status «New» in der Datenbank persistiert. Wenn sich nun ein Benutzer mit der Rolle Reviewer anmeldet, werden ihm die neuen Challenges seines Systems angezeigt. Die Übersetzungen hingegen werden nur angezeigt, wenn sie in der Standardsprache des Reviewers als Übersetzung vorliegen. Der Reviewer kontrolliert das Objekt und entscheidet sich am Ende, ob die Challenge oder die Übersetzung in der ihm vorliegenden Version veröffentlicht werden kann und veröffentlicht sie im positiven Fall (Status wird zu «Published»). Falls er nicht mit der Challenge oder Übersetzung einverstanden ist, wird eine Nachricht auf der Datenbank für die Challenge hinterlegt, die dem Challenge-Ersteller beim nächsten Anmelden anzeigt wird und ihm so mitteilt, was genau noch zu verbessern ist. Falls der Reviewer sich nicht sicher ist oder nicht

selber entscheiden möchte, ob das Objekt bereits die Qualitätsbedingungen für die Öffentlichkeit erfüllt, setzt er es auf den Status «Reviewed», wodurch ein anderer Reviewer das endgültige Urteil darüber fällen muss. Nur so kann innerhalb einer Community eine möglichst hohe Qualität von Übersetzungen und Challenges erzielt werden.

#### 5.3.2.2 Multi-Sprachunterstützung

Damit es möglich wird die Benutzeroberfläche für die Erfassung und Mutation von Challenges und deren Übersetzungen in der gewünschten Sprache anzuzeigen und um die Challenges, welche zum Lösen an das CCS geliefert werden, direkt in der für den Anwender angepassten Sprache über das API zurückzugeben, wurde eine weitere konzeptuelle Entscheidung getroffen. Dazu wird im ersten Schritt über das JWT die Standardsprache des Benutzers übermittelt und gespeichert, wie oben bereits erwähnt wurde. Falls die gewünschte Challenge in dieser Sprache vorhanden ist, wird sie einfach zurückgegeben. Falls diese Sprache nicht verfügbar ist, wird die hinterlegte Standardsprache des CCS für die Selektion verwendet. Falls auch diese nicht vorhanden ist, soll dem Benutzer eine Auswahl aller verfügbaren Sprachen geliefert werden, sodass er eine dieser auswählen kann. So wird dem Benutzer jeweils die beste Variante zum Gebrauch zur Verfügung gestellt.

#### 5.3.2.3 Übersetzungen

Neben einer einfachen Benutzeroberfläche für die Übersetzungen der einzelnen Challenge Attribute und dem nachfolgenden Review-Prozess derselben, war eine weitere Anforderung an das neue Konzept, dass der Übersetzer neben dem eigentlichen Übersetzen der Challenge auch direkt Änderungen am Originaltext vornehmen können muss. Dies führt jedoch zum Problem, dass beim Ändern des Originals eigentlich alle davon abgeleiteten Übersetzungen als ungültig markiert werden müssten. Als Lösung dieses Problems haben wir uns überlegt, dass bevor überhaupt das Original geändert werden kann, eine Checkbox auf dem Übersetzungsfenster angezeigt wird, welche man zuvor aktivieren muss. Dadurch wird der ausgegraute Originaltext aktiviert und Änderungen möglich. So kann ausgeschlossen werden, dass fälschlicherweise beim Übersetzen etwas geändert wird ohne, dass man sich dessen bewusst ist. Weiter soll es zwei Checkboxen geben, wo der Übersetzer angeben kann, ob es sich um eine kleine Änderung, wir stellen uns dabei die Korrektur eines Schreibfehlers vor oder um eine grössere Änderung handelt. Falls es sich nur um eine kleine Änderung handelt, müssten nicht alle Übersetzungen als ungültig markiert werden und wieder den ganzen Review-Prozesses durchlaufen. Anders bei grösseren Veränderungen bei denen der gesamte Prozess für alle Übersetzungen erneut durchlaufen werden muss.

#### 5.3.2.4 Issue Tracking

Wie oben bereits erklärt wurde, soll für das eigentliche Issue Tracking das bereits von GitHub vorhandene System verwendet werden. Damit dies korrekt funktioniert und das API verwendet werden kann, muss pro CAS ein GitHub Account angelegt werden und der zugehörige API Key gespeichert werden. Zusätzlich zum Account muss auch ein privates Repository angelegt werden, wo die Issues dann erstellt und bearbeitet werden können. In der CAS internen Datenbank wird dann pro Challenge die Referenz darauf gespeichert. Da eine Challenge jeweils mit einem Ersteller verbunden ist, wird diesem beim nächsten Anmelden angezeigt, dass neue Issues für eine seiner Challenges vorhanden sind, welche er dann abrufen kann. So kann er die Challenge verbessern und den Issue mit einem Kommentar versehen. Der Ersteller des Issues bekommt dann seinerseits eine Nachricht und kann den Issue gegebenenfalls schliessen.
Im Folgenden wird das Konzept des Issue-Systems anhand eines Sequenzdiagramms verbildlicht. Es gibt zwei alternative Pfade, um einen Issue zu Erfassen. Entweder befindet man sich im Consumer und erfasst über den «Report a Problem» Button einen Issue oder man befindet sich direkt auf dem CAS und überprüft dort eine Challenge und kann dann einen Issue erfassen.

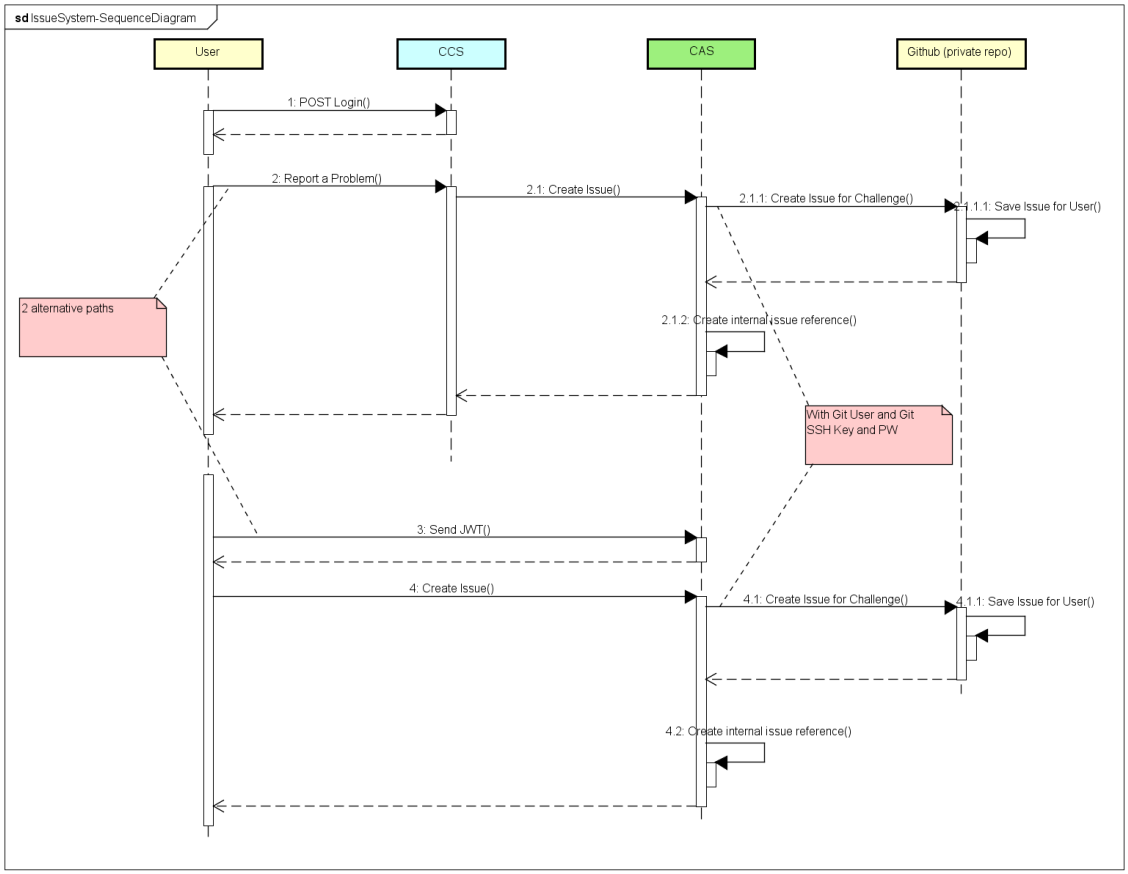



**Abb. 9 Sequenzdiagramm Issue-System**

#### 5.3.2.5 Bezahlsystem

In der konzeptuellen Vorarbeit wurde auch ein Bezahlsystem thematisiert, was jedoch noch nicht vollends ausgereift ist, da die Priorität nicht allzu hoch eingestuft wurde. Die getroffenen Überlegungen beginnen bei einer Bezahlung für erstellte Challenges, wenn diese den Review-Prozess durchlaufen haben und als gut befunden wurden. Anschliessend wird dem Benutzer eine Gutschrift auf sein Salary Konto gutgeschrieben. Dieses Salary wird nur zwischenzeitlich im CAS gespeichert. Es bleibt nur solange im CAS bis der Ersteller sich erneut anmeldet und den Betrag abholt, wodurch er dann auf seinem CCS Account gutgeschrieben wird. Hier gibt es jedoch noch offene Fragen, wie verhindert wird, dass auf dem CCS sein Guthaben verändert wird, wenn das CCS nicht von unserem Auftraggeber implementiert wird, sondern vom Kunden selber.

#### 5.3.2.6 JWT Ticket

Wie oben bereits beschrieben wird anhand des gelieferten signierten JWT ein Benutzer Objekt mit seinen gelieferten Rollen beim ersten Anmelden angelegt. Falls sich derselbe Benutzer erneut mit einem JWT anmeldet, dessen Werte sich jedoch verändert haben, wird dies automatisch in der Datenbank nachgetragen.
Da so jede Anfrage durch ein JWT abgesichert wird, ist es möglich eine Historie über die Aktivitäten des Benutzers zu führen. Auf diese Weise könnte das Problem, welches oben geschildert wurde betreffend des veränderten Guthabens nachvollzogen und verhindert werden. Dies da immer klar ist, wodurch ein Benutzer sein Vermögen angesammelt hat. Auch weiss man so genau, welche Challenges ein Benutzer angeschaut hat und welche er löste und falsch oder korrekt beantwortet hat.

#### 5.3.2.7 Rollenbasierte Autorisierung

Um auf dem Server eine weitere Autorisierungsschicht (Authorization Layer) für die einzelnen Funktionen zu implementieren, wurde angedacht diese rollenbasiert aufzubauen. Dazu wird neben der Authentifizierung mittels JWT eine weitere Abstraktionsebene aufgebaut, welche nach einer erfolgreichen Verifikation des JWT wirksam wird. Da dies programmiertechnisch mittels einer simplen Annotation auf der entsprechenden Route-Implementierung sehr einfach möglich ist, stellt dies kein Problem dar. Damit dies jedoch korrekt funktioniert, muss ein Mapping zwischen den bestehenden Rollen und der durch das API angebotenen Funktionalität aufgebaut werden. Man müsste hierfür eine eigene Administrationsseite erstellen, wo man dies mittels simplem Drag and Drop einfach bewerkstelligen könnte.

#### 5.3.2.8 Export / Import von Challenges

Damit Challenges auch zwischen unterschiedlichen Systemen ausgetauscht werden können, müssen diese in ein entsprechendes Format exportierbar sein. Dabei sollen jedoch dynamische Daten wie das Rating oder Issue vernachlässigt werden, da die für andere Benutzer möglicherweise auch unterschiedlich ausfallen würden. Es soll immer nur die aktive Version exportierbar sein. Damit dies zukünftig relativ einfach und ohne Probleme mit Duplikaten und dergleichen bewerkstelligt werden kann, wurde eine global eindeutige ID verwendet, welche aus dem globalen Directory stammt. So können Challenges im System, wo sie importiert werden sollen, einfach zusammengefügt oder überschrieben werden.

#### 5.3.2.9 Filtern von Challenges

Damit nicht immer die gesamte Liste der Challenges beim User angezeigt wird und er entsprechend nach Typ oder anderen Merkmalen filtern kann, soll eine Route zur Verfügung gestellt werden, wodurch dies bewerkstelligt werden kann. Dies muss so gelöst werden, dass mit jeder Eingabe ein Request an den Server getätigt wird und jeweils die entsprechende Liste zurückgeliefert wird.

### 5.3.3 Technische Konzeptentscheide

Neben den oben getroffenen allgemeinen Entscheiden und den Entscheiden zu der Datengrundlage, wurden weitere eher technisch angehauchte Dinge besprochen und weiterentwickelt. Es geht hier bereits tiefer in die technische Umsetzung als bei den Themen, die oben besprochen wurden. Es soll hier vermehrt um die eigentlichen Komponenten bzw. die einzelnen noch nicht beschriebenen Server gehen. Diese sollten hier als abschliessender Punkt zum Konzeptanalyse Teil beschrieben werden:

Das Challenge Authoring System (CAS) ist die entscheidende Komponente für den Prototyp dieses Projektes. Das Konzept desselben wurde im Rahmen dieser Studienarbeit entwickelt und teilweise umgesetzt. Dieses Teilsystem kann beliebig oft betrieben werden, was eine sehr gute Skalierbarkeit zur Folge hat. Der Hauptteil der Arbeit am CAS war der sogenannte Medium Editor. Dieser Editor für die Challenge Erfassung, ist stark an den von Medium [9] verwendeten Editor angelehnt und sollte auch die entsprechenden Funktionalitäten liefern können. Er sollte im Mindesten Formatoptionen anbieten, sowie «Embedded Code» und Links unterstützen und Bilder oder Filme einfügen können. Eine Anforderung mit kleiner Priorität war es auch, Bilder aus der Zwischenablage zu unterstützen. Damit jeweils alle Eingaben auf dem Server gespeichert werden können, sollte er kontinuierlich die aktuellen Änderungen an den Server liefern. Dies sollte asynchron geschehen, damit der Editor in der Zwischenzeit nicht dadurch blockiert wird. Näheres zu der technischen Umsetzung dieses Editors kann dem Kapitel «Software Architektur» entnommen werden.
Das Challenge Consumer System ist für die eigentliche Benutzerverwaltung zuständig. Weiter konsumiert es die Daten, welche vom CAS geliefert werden. Wenn nun von einem CCS eine Challenge gestartet werden soll, wird allenfalls das Challenge Resource Server System benötigt, welches die Ressource einer Challenge initialisiert und gegebenenfalls startet.
Das Challenge Resource Server System (kurz CRSS) hält unter anderem Ressourcen in Form von Docker-Containern. Wenn nun ein Benutzer eine Challenge durchführen möchte, welche eine Ressource dazu benötigt, macht das CAS eine Abfrage an das globale Directory (CDS). Als Antwort darauf wird dem CAS mitgeteilt, welche CRSS diese Ressource überhaupt halten. Im Anschluss daran soll das CAS entscheiden, welches CRSS das örtlich am nächsten liegende zum anfragenden CCS ist. Wenn dies bestimmt wurde wird auf diesem Challenge Resource Server System die entsprechende Ressource gestartet und dem Client zur Verwendung übergeben. Die Einstiegspunkte zu einem CRSS werden wiederum auf dem globalen Directory verwaltet und können vom CAS angefragt werden.
Damit das Challenge Resource Server System eine Ressource, überhaupt sinnvoll erstellen und ausliefern kann, benötigt er Daten des Benutzers, um für ihn einen Account in der Ressource anzulegen. Diese wird ihm vom Challenge Consumer System über das CAS mitgeteilt. Der Grund weshalb das CCS nicht direkt mit dem CRSS kommunizieren kann, sondern das CAS dazwischensteht, ist, damit das CAS Metadaten abfangen kann, um das Goldnugget für die entsprechende Ressource zu generieren.

Eine Ressource ist z.B. ein Bild, oder ein Docker-Container, welcher ein gewisses Betriebssystem mit vordefinierten Tools installiert hat oder einfach eine Amazon Cloud, welche von einem externen Dienst zur Verfügung gestellt wird. Während dem Builden der Ressource wird auch gleich das generierte Goldnugget eingepflegt, sodass die Lösung der Challenge pro Benutzer eindeutig wird.
Eine Idee im Zusammenhang mit Ressourcen und dem CRSS ist, dass dieses System bereits vorrätig Ressourcen mit definierten Goldnuggets besitzt, welche dann lediglich noch gestartet und ausgeliefert werden müssen.

Mit dieser vorhergehenden Überleitung mittels der technischen Konzeptentscheidungen, soll hier nun das teilweise entwickelte HL-CAS beschrieben werden, welches im Rahmen dieser Studienarbeit als Prototyp zur Challenge Erfassung bezeichnet wird. Eine wichtige Einschränkung ist, dass die Realisierung des ganzen Hacking-Lab 2.0 den Rahmen einer solchen Arbeit deutlich sprengen würde. Aus diesem Grund hat man sich auf die Entwicklung des Konzepts für das Hacking-Lab 2.0 und einen HL-CAS-Prototypen beschränkt.
Bei der Umsetzung wurde das Prinzip einer Mikroservicearchitektur [10] angewandt. Dies soll mit folgender Illustration veranschaulicht werden.

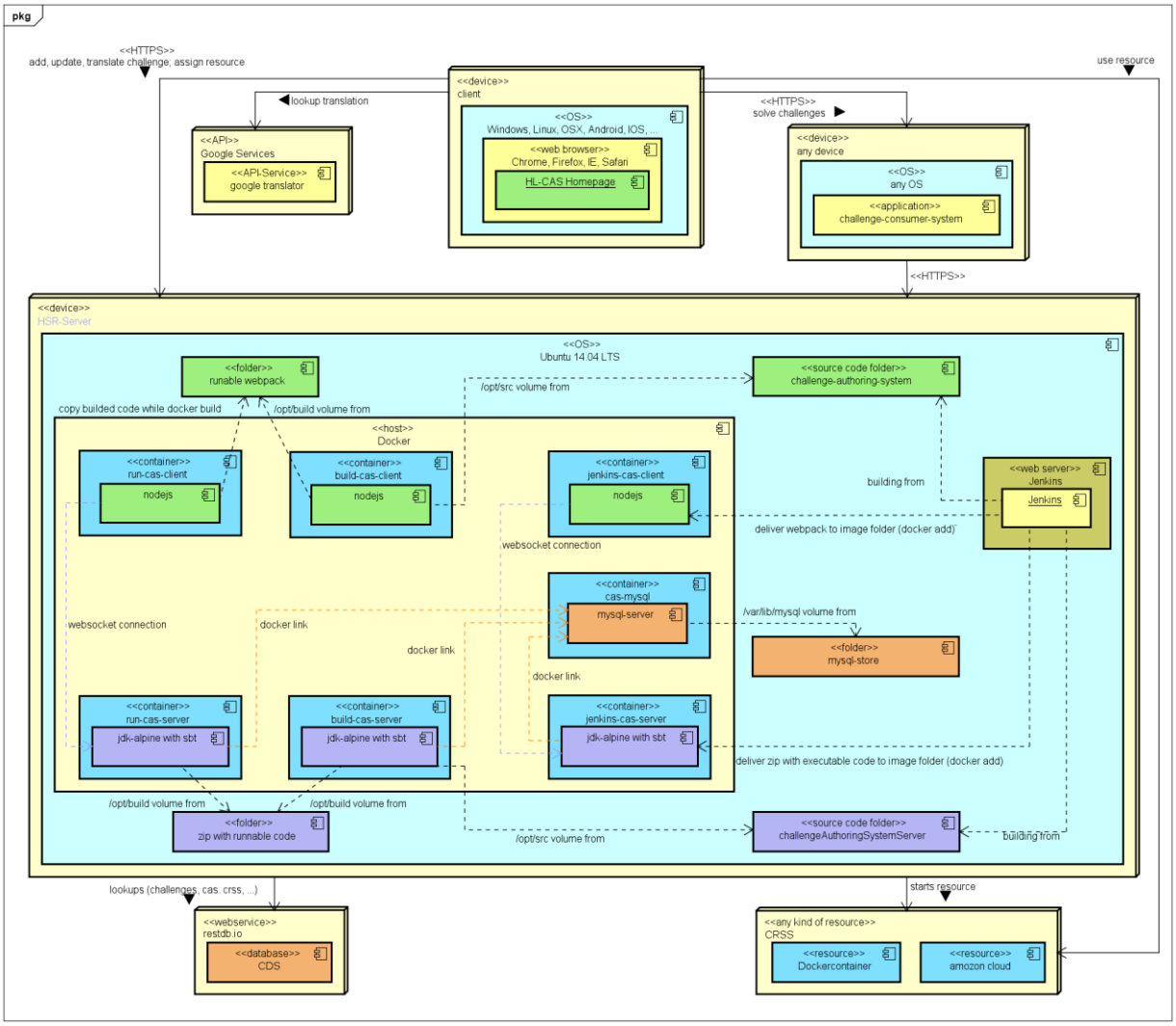



**Abb. 10 Deployment Diagramm**

Es ist klar ersichtlich, dass der HL-CAS-Client und HL-CAS-Server je in einem eigenen Docker-Container ausgeführt werden. Dabei ist der HL-CAS-Client für die Auslieferung der Benutzeroberfläche an einen HL-Benutzer zuständig, der neue Challenges erfassen oder bereits bestehende übersetzen oder korrigieren will. Für das Lösen von Challenges wird das Frontend jedoch von einem Challenge Consumer System kurz CCS geliefert, welches aber nicht Bestandteil dieser Arbeit ist. Der HL-CAS-Server stellt das Backend des CAS dar und wird für alle Verwaltungsaufgaben eingesetzt. Somit werden die Anforderungen an das System, auf welchem ein solcher CAS läuft, minimal gehalten. Der Grundgedanke dahinter ist, dass man ohne grossen Aufwand ein neues CAS in Betrieb nehmen kann. Dies indem man das GitHub-Repository des HL-CAS klont, dann eine Konfigurationsdatei mit Angaben zum neuen CAS füttert und zum Abschluss nur noch die Docker-Buildscripts ausführen muss. Als erstes wird ein Skript gestartet, welches ein Docker-Netzwerk initialisiert, dass für die Kommunikation unter den Containern

selbst zuständig ist. Zugleich wird auch gleich noch eine MySQL-Instanz [11] in einem Docker-Container gestartet. Hierbei ist zu beachten, dass eine Anforderung war, dass dessen Dateien ausserhalb des Containers liegen sollen. Somit kann der Container beliebig gestartet, gestoppt oder sogar neu gebaut werden, ohne dass irgendwelche Daten verloren gehen.
Im Anschluss daran müssen die Build-Skripts der Container «build-cas-client» und «build-cas-server» gestartet werden. Diese erstellen die Buildumgebung in einem Docker-Container. Als nächstes wird der jeweilige Source-Code aus dem Repository in den Container gemounted und gebuildet. Die fertig kompilierten Programme werden dann wiederum in einem anderen containerextern gemounteten Ordner verschoben. Somit ist sichergestellt, dass an diesem Speicherort immer ein lauffähiges Programm gehalten wird. Denn wenn beim Bauen ein Fehler auftritt, werden diese Dateien nicht verändert.
Abschliessend müssen nur noch die Skripts für die Container «run-cas-server» und «run-cas-client» gestartet werden. Diese bestehen aus Minimalsystemen, welche für die Ausführung der jeweiligen Applikation gebraucht werden. Auch diese Systeme können nach Belieben neu gebuildet werden, da alle zu persistierenden Daten in der MySQL-Datenbank gehalten werden. Weitere Informationen diesbezüglich sind dem Unterdokument «Software Architektur» zu entnehmen. Dort wird auch beschrieben, wie Jenkins verwendet wird und welche Aufgaben die Jenkins-Docker-Container haben.

Die Folgende zwei Illustrationen sollen dabei helfen, das Verständnis für das Zusammenspiel aller beteiligten Teilsysteme zu stärken.

Als erstes widmen wir uns der Ausführung einer Challenge.

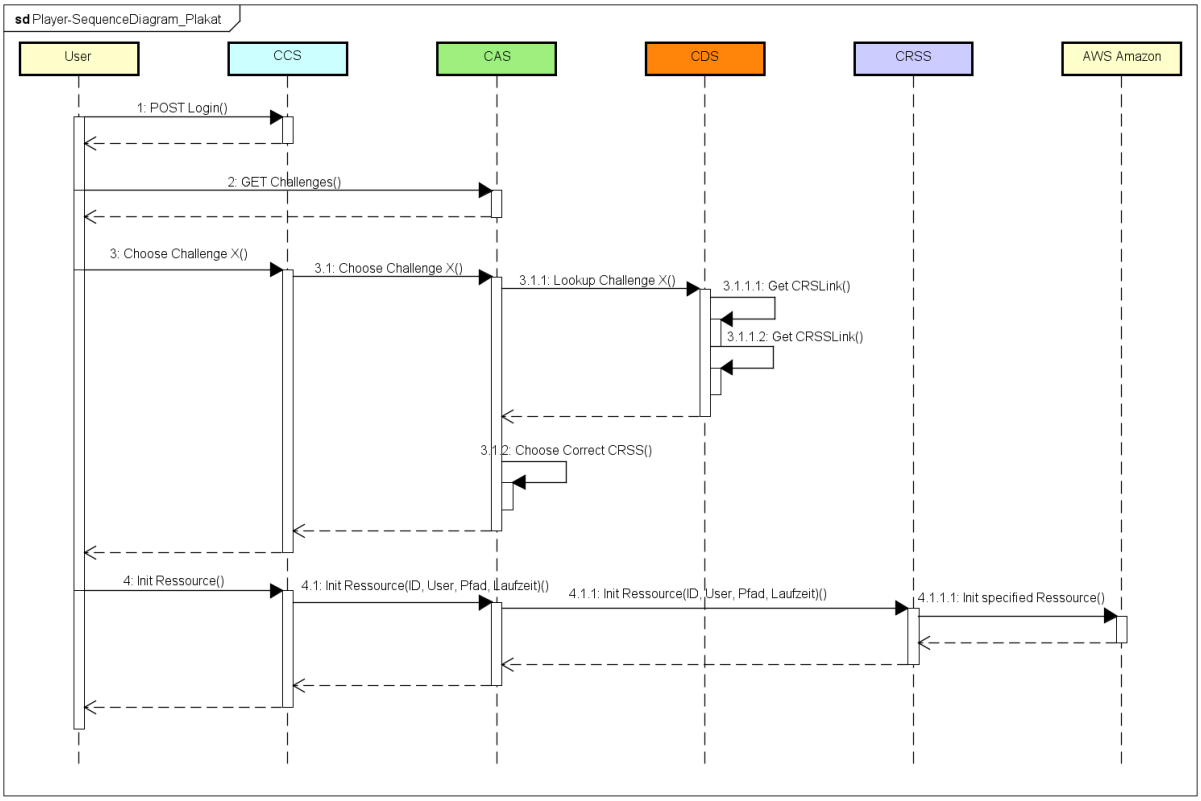



**Abb. 11 Sequenzdiagramm Challenge-Ausführung**

1. Ein Benutzer meldet sich bei seinem Challenge Consumer System an, welches die Benutzerverwaltung in sich hält und dem Benutzer die grafische Oberfläche zum Lösen einer Challenge zur Verfügung stellt.
2. Der Benutzer bezieht vom Challenge Authoring System eine Liste aller für ihn verfügbaren Challenges, welche ihm dann zur Auswahl angezeigt werden. Bei seiner Anfrage wird auch noch ein signiertes JWT [8] mitgeschickt, welches dem CAS die Benutzerverifikation erlaubt, da dieses nur vom ihm bekannten CCS erstellt werden kann. Ohne dieses Token ginge das nicht, da ein CAS nicht für die Benutzerverwaltung zuständig ist.
3. Nun wählt er eine Challenge aus und teilt die damit verbundene global eindeutige ID dem CCS mit, welcher diese Information wiederum dem CAS mitteilt. Dies muss dem CCS mitgeteilt werden, damit dieses den Status für eine Challenge-Ausführung einem Benutzer zuweisen kann. Dann wird durch das Challenge Authoring System auf dem Challenge Directory System nachgesehen, ob diese Challenge eine oder mehrere Ressourcen benötigt und auf welchen Challenge Resource Server System diese verfügbar ist. Danach wählt das CAS den für das CCS örtlich am nächsten liegenden CRSS aus und gibt alle nun zusammengetragenen Informationen (Challenge welche auf dem CAS persistiert ist und dazugehörige CRSS und CRS) nun bis zum Benutzer weiter.
4. Im letzten Schritt kann der Benutzer die für die Challenge benötigte Ressource über CCS→CAS→CRSS initialisieren. Das neue Konzept sieht für die Zukunft auch eine automatische Überprüfung von Challenge-Lösungen vor. Daher muss das Initialisieren der Ressource über den CAS geschehen. Somit wird es zu einem späteren Zeitpunkt möglich sein, für jede neue Instanz einer Ressource, die einem Benutzer zugeordnet wird, eine eigene eindeutige Musterlösung für die Korrektur zu erstellen. So wird es auch nicht mehr möglich sein, dass Benutzer durch Austauschen von fertigen Lösungen betrügen bzw. ihren Kollegen helfen können.

Damit man aber überhaupt eine Challenge lösen kann, muss diese zuerst erstellt werden. Dies soll im folgenden Diagramm knapp gezeigt werden.

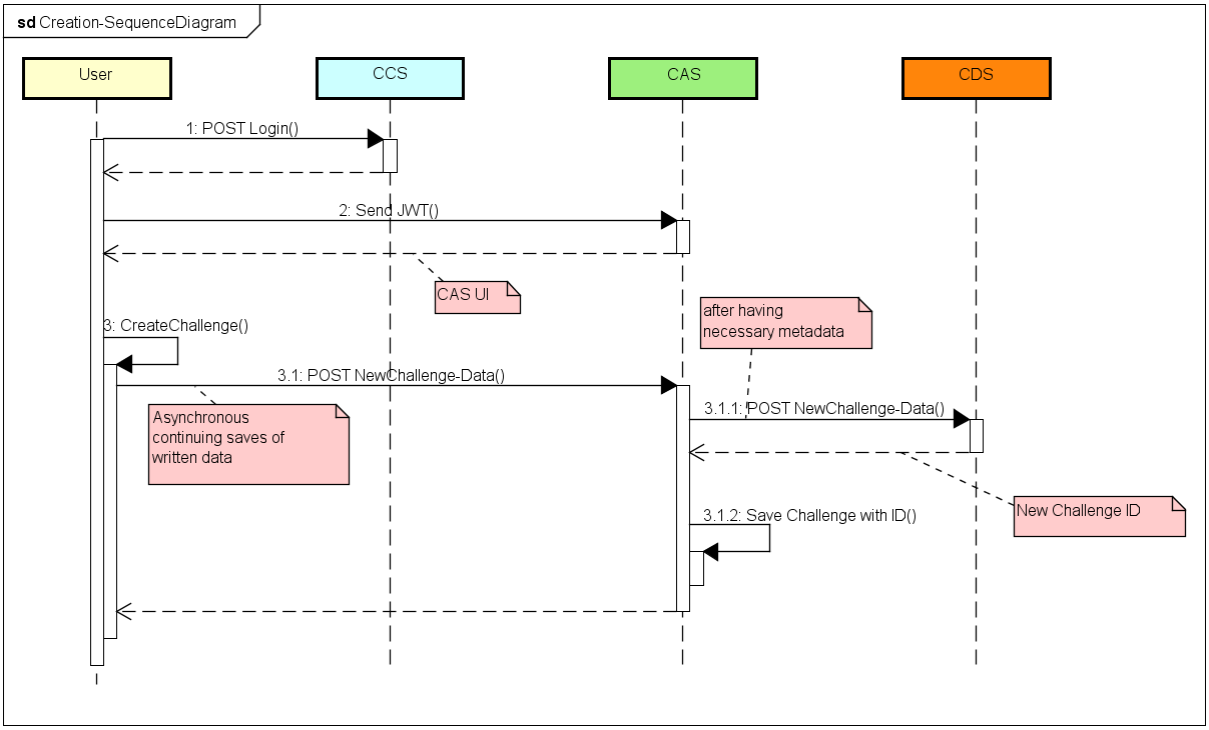



**Abb. 12 Sequenzdiagramm Challenge-Erstellung**

1. Als erstes loggt sich der Benutzer beim CCS ein.
2. Anschliessend wird dem CAS das signierte JWT gezeigt, worauf dieser dem Benutzer die grafische Oberfläche zur Erfassung neuer Challenges zur Verfügung stellt.
3. Die Challenge-Erstellung wird iterativ anhand eines Wizard abgearbeitet. Im Hintergrund wird bereits beim Starten des Wizard eine global eindeutige ID beim CDS eingetragen und für die neue Challenge verwendet. Als erstes muss der Benutzer einen Challenge-Titel vergeben, welcher gleich wieder beim CDS eingetragen wird. So geht es dann Schritt für Schritt weiter, indem er nach dem Titel ein Abstract, eine Kategorie, eine Schwierigkeitsstufe, benötigte Ressourcen usw. zur Challenge angibt. Für das Schreiben der Texte (z.B. die Aufgabenbeschreibung einer Challenge), wurde ein Editor, welcher an den von Medium [9] angelehnt ist, entwickelt. Abschliessend wird die Challenge auf dem CAS persistiert und dem Benutzer eine Statusmeldung zur Erstellung geschickt.

## 5.4 Ergebnisdiskussion mit Ausblick

In den vorhergestellten Kapiteln wurde nun das Konzept des Hacking-Lab 2.0 vermittelt. Als Ergebnis der Studienarbeit geht hauptsächlich eben genau diese Konzeptausarbeitung hervor. Denn zu Beginn der Arbeit wurden wir vor die komplexe Herausforderung gestellt, alle genannten Anforderungen in den Griff zu bekommen. In den ersten sechs Wochen lag der Fokus somit auf der Konzeptentwicklung und der Einarbeitung in die benötigten Technologien, von welchen die meisten für uns neu waren. Dies hatte zur Folge, dass die Entwicklung eines HL-CAS etwas auf der Strecke geblieben ist. Die Grundfunktionalitäten, wie zum Beispiel der Editor, konnten in einer ersten Version implementiert werden. Es konnten leider nicht alle dafür notwendigen Komponenten wie Steps, Hints und Instructions fertig gestellt werden. Ein weiteres Defizit ist, dass einige Funktionalitäten wie der Übersetzer aus Zeitmangel nicht umgesetzt werden konnten. Trotzdem bildet diese Studienarbeit eine sehr gute Grundlage für eine neue Version des Hacking-Lab. Da das ganze Konzept über das komplette Hacking-Lab 2.0 steht und eine gut durchdachte logische Struktur vorhanden ist.

Die Inbetriebnahme neuer HL-CAS wurde soweit optimiert / vereinfacht, dass nur das GitHub-Repository geklont werden muss. Um im Anschluss eine Konfigurationsdatei anzupassen und die Docker-Buildskripts nacheinander auszuführen. Abschliessend muss das neue System nur noch im CDS eingetragen werden und somit ist bereits ein neuer CAS-Knoten im Betrieb. Anforderungen an den für den Betrieb verwendeten Host sind minimal, da alle Schritte für die Inbetriebnahme (mit Ausnahme des Eintrags im CDS) in einem Docker-Container ausgeführt werden. Somit muss der Host minimal als Docker-Host fungieren.

Als nächste Schritte stehen die Vollendung des HL-CAS und Implementierung der restlichen Teilsysteme des Hacking-Lab 2.0, wie das Challenge Consumer System und dem Challenge Ressource System Server, an. Beim CAS müssen mindestens folgende Arbeiten noch gemacht werden:

- Fertigstellung des Challenge Editors
- Styling der Benutzeroberfläche
- Challenge Mutation
- Übersetzungstool
- Auslieferung der Übersetzungen für die Anzeige im Client
- Issue Tracking System Integration
- Einbinden des Review-Systems
- Rating von Challenges
- Goldnugget Generierung pro Benutzer
- Rollenbasierte Autorisierungsschicht
- Ressourceninstanziierung und Erörterung des am nächsten liegenden CRSS zu einem CCS

Als weiteren Schritt kann auch die Automatisierung der Korrektur von Challenge-Lösungen der Benutzer in Angriff genommen werden. Die Grundlagen dafür wurden in dem erarbeiteten Konzept berücksichtigt. So wäre es möglich, einen Vorrat an vorbereiteten Ressourcen auf den CRSS zu halten. Somit gäbe es zu einer Ressource, welche für eine bestimmte Challenge benötigt wird, mehrere Instanzen dieser Ressource. Jedoch unterscheiden sich diese im «Lösungswort» (Goldnugget), welches der Benutzer bekommt, wenn er die Challenge korrekt gelöst hat. Somit können mehrere Benutzer die gleiche Challenge lösen und erhalten jedoch bei korrekter Durchführung ein anderes Lösungswort für die Überprüfung.

Als Fortsetzung dieser Arbeit widmen wir unsere Bachelorarbeit ebenso dem Hacking-Lab 2.0. Wir erhoffen uns damit einen brauchbaren Stand der neuen Version erstellen zu können. Welches als ganzheitliches System funktioniert.

# Hacking-Lab 2.0

## 6 Anforderungsspezifikation & Domänenanalyse

Janick Engeler
Yanick Gubler

## 6.1 Änderungsgeschichte

| Datum | Version | Änderung | Autor |
|---|---|---|---|
| **12.10.17** | 1.0 | Dokument erstellt | Yanick Gubler |
| **13.10.17** | 1.1 | UseCases erfasst + Anforderungen verlinkt + div. Anpassungen | Team |
| **26.10.17** | 1.2 | UC Diagramm ausgetauscht + Datenmodell reinkopiert | Yanick Gubler |
| **27.10.2017** | 1.3 | UC's angepasst + Diagramme (UC + Domain Model angepasst) | Yanick Gubler |
| **08.12.2017** | 1.4 | UC_Systemgrenzen und Component Diagramm angepasst | Yanick Gubler |
| **14.12.2017** | 1.5 | Alle Diagramme aktualisiert | Yanick Gubler |

Tab. 1 Änderungsgeschichte - Anforderungsspezifikation & Domänenanalyse

# 6.2 Einführung

## 6.2.1 Zweck

Ziel dieses Dokuments, ist eine möglichst exakte Anforderungsdefinition für das HL-CAS festzuhalten. Ein Grossteil der Anforderungen wird aber bereits durch die Aufgabenstellung vorgegeben. Dieses Dokument dient uns als Nachschlagewerk bei internen Unklarheiten. Des Weiteren wird eine Domänenanalyse durchgeführt. Das Ziel der Domänenanalyse ist eine möglichst genaue Anforderungsdefinition zu erstellen.

## 6.2.2 Gültigkeitsbereich

Dieses Dokument dient als Grundlage für den gesamten Ablauf des Projekts und ist während diesem gültig. Das Dokument wird laufend ergänzt und in der Änderungsgeschichte nachgeführt. Die aktuellste Version ist jeweils die gültige.

## 6.2.3 Übersicht

Dieses Dokument umfasst sämtliche Anforderungen für das Produkt. Beginnend mit einer allgemeinen Beschreibung vom HL-CAS, wird ein Überblick über die Applikation verschafft.

Weiter werden Use Cases beschrieben. Danach werden nicht funktionale und weitere Anforderungen beschrieben. Zudem wird die Domäne analysiert und beschrieben.

## 6.2.4 Referenzen

- Aufgabenstellung

## 6.2.5 Glossar

| Begriff | Erklärung |
|---|---|
| **CAS** | Challenge Authoring System |
| **CCS** | Challenge Consumer System |
| **CD** | Continuous Delivery |
| **CDS** | Challenge Directory System |
| **CI** | Continuous Integration |
| **CRS** | Challenge Resource System |
| **CRSS** | Challenge Resource Server System |
| **HL** | Hacking Lab |
| **JWT** | JSON Web Token |
| **SBT** | Scala Build Tool |

Tab. 2 Glossar

## 6.3 Allgemeine Beschreibung

### 6.3.1 Produkt Perspektive

Das Hacking-Lab ist ein weltweites IT Security Labor, in welchem die User praktische Security Challenges (Übungen) lösen, um damit ihre Fähigkeiten hinsichtlich Angriff und Abwehr von Hacker Attacken zu trainieren.

Es soll ein Challenge Authoring System (CAS) entwickelt werden, um die Challenges im Hacking-Lab besser verwalten und nutzen zu können. CAS ist ein modernes System für die Erstellung, Bearbeitung und Pflege von Challenges, welche manchmal auch Übungen genannt werden. Das Hacking-Lab soll in diesem Kontext als Consumer des CAS Systems verstanden werden. Es soll möglich sein, dass auch andere Consumer das CAS System nutzen (Challenge Store)

### 6.3.2 Benutzer Charakteristik

Das Produkt wird vor allem von Schulen oder Unternehmen dazu verwendet, interne Wettkämpfe, Schulungen oder Kurse durchzuführen. Damit können die Informatikkenntnisse ihrer Schüler bzw. Mitarbeiter im Bereich IT-Sicherheit geschult oder geprüft werden. Da es sich bei den Anwendern um eher Informatik affines Klientel handelt, wird besonders darauf geachtet, dass die Oberfläche den aktuellsten Standards entspricht, damit ein möglichst sauberes Benutzererlebnis ermöglicht werden kann.

### 6.3.3 Einschränkungen

Da die Anwender mehrheitlich Informatiker oder Personen mit Kenntnissen in der IT sind, kann davon ausgegangen werden, dass jeweils die aktuellsten Browserversionen verwendet werden. Deshalb soll hier primär die Priorität der Umsetzung auf den modernsten Implementierungen der jeweiligen Browser liegen.

### 6.3.4 Abhängigkeiten

Das Produkt besitzt drei wichtige externe Abhängigkeit. Die erste dieser wäre das Challenge Directory System (CDS), welches alle Challenge-Metadaten zu den entsprechenden CAS gespeichert hat, damit man diese auch vom jeweiligen System exportieren kann. Die zweite ist das Challenge Resource Server System (CRSS), welches die von der Challenge benötigten Ressourcen verwaltet und für diese freigibt. Die letzte Abhängigkeit kommt durch das Challenge Consumer System (CCS) zu Stande. Dieses fungiert als Konsument unserer verwalteten Challenges und stellt diese unter anderem seinen Benutzern zum Lösen zur Verfügung.

# 6.4 Funktionale Anforderungen

## 6.4.1 Anforderungsbeschreibung

Die Anforderungsbeschreibung kann dem Dokument «Aufgabenstellung» entnommen werden.

## 6.4.2 Use Cases

Die folgende Illustration soll die Systemgrenzen aller Komponenten und Aktoren visualisieren. Weiter sieht man die Abhängigkeiten aller Systeme und welche Benutzerrollen es auf diesen jeweils gibt.

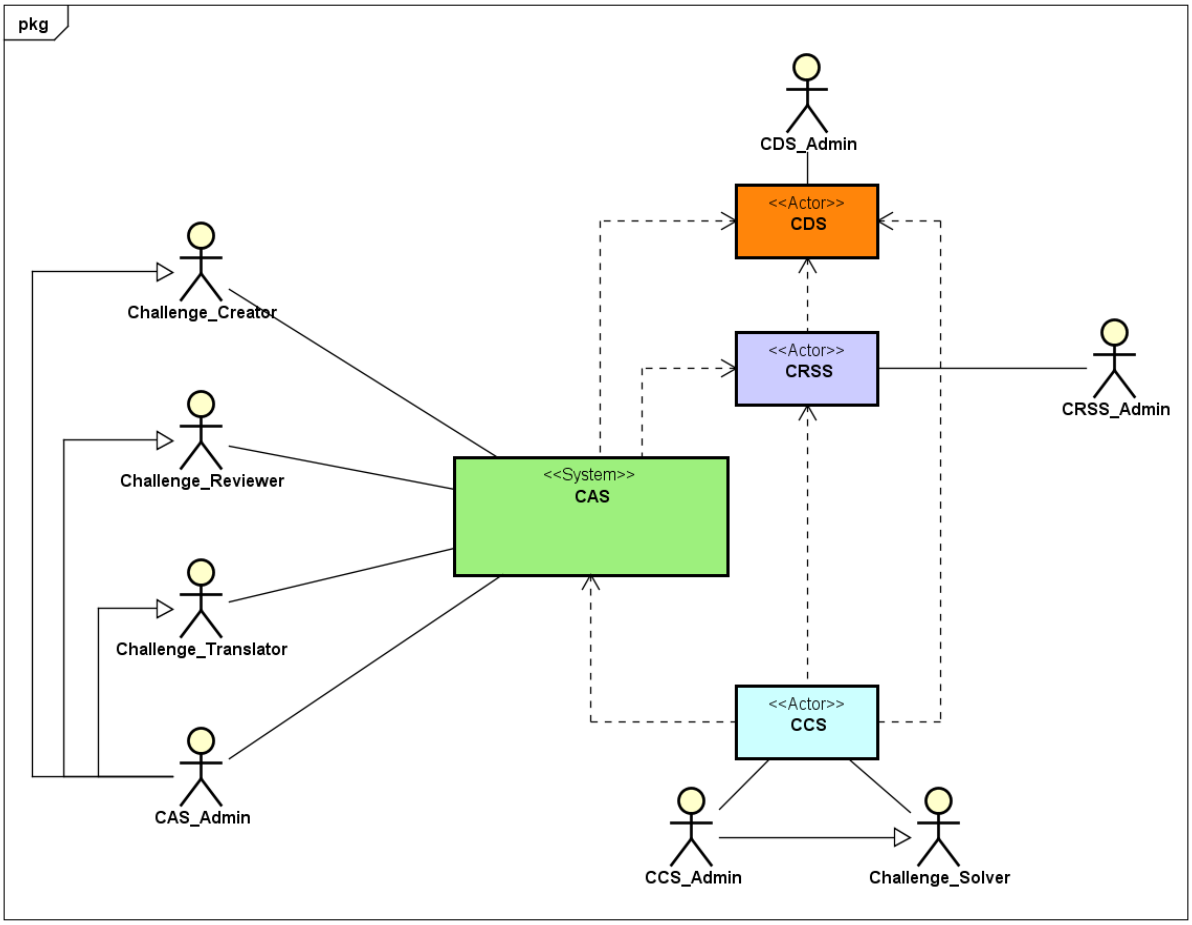



Abb. 13 Use Case Systemgrenzen

Die nachfolgende Grafik ist ein Zoom-In der oberen. Hier werden alle Use Cases des HL-CAS gelistet und mit den entsprechenden Aktoren verknüpft. Dazu ist noch zu erwähnen, dass der CAS_Admin auch alle Assoziationen zu den Use Cases der anderen CAS-Aktoren erbt.

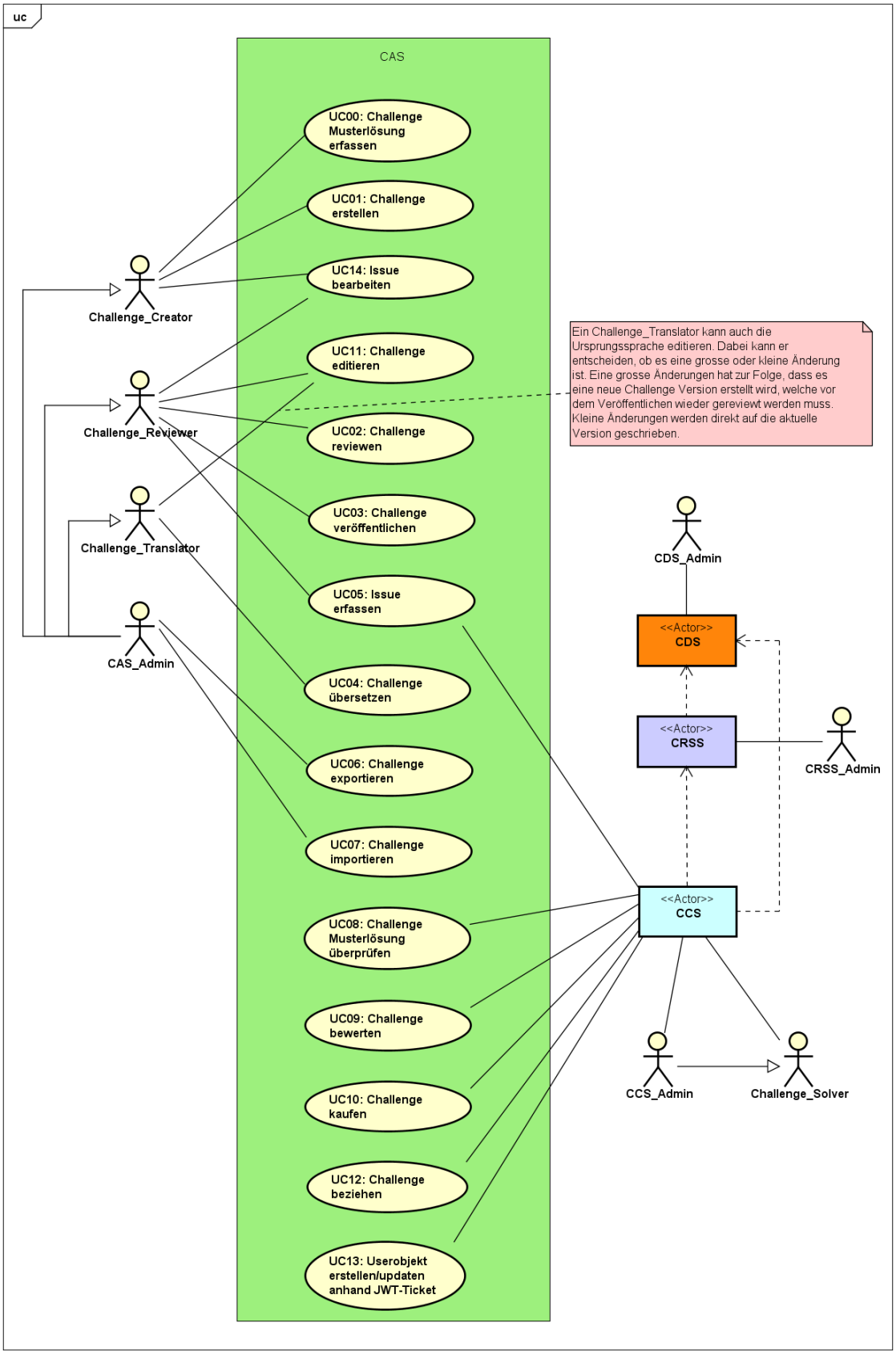



**Abb. 14 Use Case**

#### 6.4.2.1 UC00: Challenge Musterlösung erfassen

Für jede Challenge kann die entsprechende Musterlösung erfasst werden, welche dann über ein API abgefragt werden kann.

#### 6.4.2.2 UC01: Challenge erstellen

Der Autor kann in einem Formular mittels Markdown eine neue Challenge erstellen und diese für ein Review freigeben.

#### 6.4.2.3 UC02: Challenge reviewen

Eine erstellte Challenge kann durch einen Review-User (mit entsprechender Rolle) geprüft und falls alles korrekt ist freigegeben werden.

#### 6.4.2.4 UC03: Challenge veröffentlichen

Ein entsprechender Benutzer hat die Möglichkeit eine zuvor geprüfte Challenge zu veröffentlichen. Dadurch wird sie im CDS angezeigt.

#### 6.4.2.5 UC04: Challenge übersetzen

Ein entsprechender Benutzer kann die Challenge übersetzen, damit diese auch für Benutzer mit einer anderen Muttersprache verfügbar gemacht werden.

#### 6.4.2.6 UC05: Issue erfassen

Falls ein User beim Lösen einer Challenge einen Fehler entdeckt, kann er für diesen einen Issue erfassen. Damit kann dieser Fehler von einem User der entsprechenden Rolle behoben werden.

#### 6.4.2.7 UC06: Challenge exportieren

Der Admin des jeweiligen CAS kann die gewünschten Challenges exportieren, damit sie in einem anderen CAS wieder importiert werden können.

#### 6.4.2.8 UC07: Challenge importieren

Der Admin des jeweiligen CAS kann die angeforderten Challenges in sein eigenen CAS importieren, damit sie von seinen Consumern (CCS) wiederum ausgewählt werden können.

#### 6.4.2.9 UC08: Challenge Musterlösung überprüfen

Das CCS kann eine Lösung eines Solvers an den CAS schicken, dieser überprüft die Lösung anhand eines Goldnuggets und gibt dem CCS zurück, ob die Lösung des Solvers korrekt war.

#### 6.4.2.10 UC09: Challenge bewerten

Challenges können aus dem CCS bewertet werden.

#### 6.4.2.11 UC10: Challenge kaufen

Aus dem CCS kann ein Benutzer mit entsprechendem HL-$-Kontostand vorgefertigte Challenges kaufen.

#### 6.4.2.12 UC11: Challenge editieren

Challenges können bearbeitet werden. Dies kann z.B. aufgrund eines Reviews oder eines durch einen Benutzer erfassten Issue erfolgen.

#### 6.4.2.13 UC12: Challenge beziehen

Aus dem CCS kann ein Benutzer eine Challenge aus dem CAS beziehen, damit diese gelöst werden kann.

#### 6.4.2.14 UC13: Userobjekt erstellen/updaten anhand JWT-Ticket

Sobald ein Benutzer durch ein JSON Web Token auf ein CAS zugreift, wird ein Benutzerobjekt erstellt.

#### 6.4.2.15 UC14: Issue bearbeiten

Ein Benutzer mit der entsprechenden Rolle, kann sich einen zuvor erstellten Issue zuweisen und diesen bearbeiten.

## 6.5 Nicht funktionale Anforderungen

### 6.5.1 Funktionalität

#### 6.5.1.1 Sicherheit

Es darf nur authentifizierten Benutzern möglich sein, auf die Anwendung zuzugreifen. Falls man nicht eingeloggt ist, wird man auf das Anmeldefenster des CCS verwiesen. Die Passwörter der Accounts werden in der Datenbank verschlüsselt abgelegt. Die Felder zur Registrierung werden geprüft, damit keine SQL-Injection das System ungewollt verändern kann.

### 6.5.2 Zuverlässigkeit

#### 6.5.2.1 Wiederherstellbarkeit

Nach einem Absturz der Anwendung soll diese ohne irgendwelche Massnahmen zu treffen neu gestartet werden. Die aktuellen Sitzungen werden in einer ersten Version jedoch nicht wiederherstellbar sein.

#### 6.5.2.2 Fehlertoleranz

Bei einem Fehler in der Anwendung darf nur die aktuelle Operation des verursachenden Benutzers abgebrochen werden und somit dürfen die anderen Teilnehmer nicht davon betroffen sein. Es muss also darauf geachtet werden, dass die Applikation nicht unnötig abstürzt und die Fehlersituationen sauber behandelt werden.

#### 6.5.2.3 Verfügbarkeit

Da es sich bei der Anwendung (CAS) 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).

#### 6.5.2.4 Skalierbarkeit

Da die Anzahl CAS und deren CCS stark variieren können, muss die Anwendung einfach skalierbar sein. Somit ist einem potentiellen Wachstum nichts entgegenzusetzen.

### 6.5.3 Benutzbarkeit

#### 6.5.3.1 Verständlichkeit

Benutzer, welche das HL bereits kennen, müssen die Anwendung direkt ohne Tutorial oder ähnliches verwenden können. Die Verwendung muss mehrheitlich selbsterklärend sein, sodass auch für neue Benutzer keine Anleitung erforderlich ist.

#### 6.5.3.2 Bedienbarkeit

Die Applikation muss leicht und intuitiv bedienbar sein. Des Weiteren muss die Anwendung auch auf Mobilgeräten problemlos betrachtet werden können. Jedoch wird der Funktionsumfang auf den mobilen Endgeräten eingeschränkt sein.

#### 6.5.3.3 Robustheit

Fehler durch eine falsche Bedienung dürfen gar nicht erst möglich sein. Wenn trotzdem solche auftreten, müssen diese durch saubere Fehlerbehandlung abgefangen werden. Was ein robustes System gewährleistet.

#### 6.5.3.4 Wartbarkeit

Alle Komponenten müssen jeweils in einem Docker-Container ausgeführt werden, wodurch auch die Wartung relativ einfach möglich wird.

#### 6.5.3.5 Analysierbarkeit

Da sich die Fehleranalyse als relativ aufwendig gestalten könnte, da das System auf unterschiedliche Komponenten verteilt ist, muss es auf einfache Art möglich sein Fehler zu reproduzieren.

#### 6.5.3.6 Prüfbarkeit

Da die einzelnen Komponenten vor einem Build Unit Tests unterzogen werden, ist die Prüfbarkeit automatisch gegeben.

#### 6.5.3.7 Stabilität

Während der Ausführung sollten keine unerwarteten Situationen auftreten. Die Wahrscheinlichkeit dafür muss in diesem Projekt möglichst gering sein. Wenn die Datenbankanbindung implementiert wird, muss darauf geachtet werden, dass die ACID Eigenschaften nicht verletzt werden, was jedoch mit MySQL kein Problem sein sollte.

#### 6.5.3.8 Erweiterbarkeit

Bei der Implementierung wird viel Wert auf die Erweiterbarkeit gelegt, ohne dass grundlegende Architekturentscheide angepasst werden müssen.

### 6.5.4 Effizienz

Die Anwendung muss von mehreren CCS gleichzeitig verwendet werden können. Dabei sollen die Antwortzeiten aber nicht aus dem Rahmen fallen und unter einer Sekunde liegen.

### 6.5.5 Übertragbarkeit

Da alle Anwendungen des Systems jeweils in einem Docker-Container betrieben werden, wird das System einfach übertragbar sein. So kann z.B. ein Docker-Container, welcher auf einem UNIX System betrieben wird, auch auf einem Windows basierten System ausgeführt werden. Somit ist das System plattformunabhängig und kann sogar auf einem Cloudservice betrieben werden.

## 6.6 Domain Model

Im Folgenden soll die gesamtheitliche Struktur des neuen Hacking-Lab 2.0 verdeutlicht werden.

### 6.6.1 Component Diagramm

In der folgenden Abbildung soll das Zusammenspiel der verschiedenen Teilsysteme des neuen Konzepts dargestellt werden. Hauptmerkmale sind, dass es global nur ein CDS gibt und beliebig viele CRSS, CAS und CCS. Weiter soll verbildlicht werden, wie die CCS zu ihren Ressourcen kommen, welche anschliessend den Benutzern für die Lösung der Challenges zur Verfügung gestellt werden. Denn die CRSS sind unabhängig von den Systemen CAS und CCS. Somit kann von jedem CCS auf jedes CRSS zugegriffen werden.

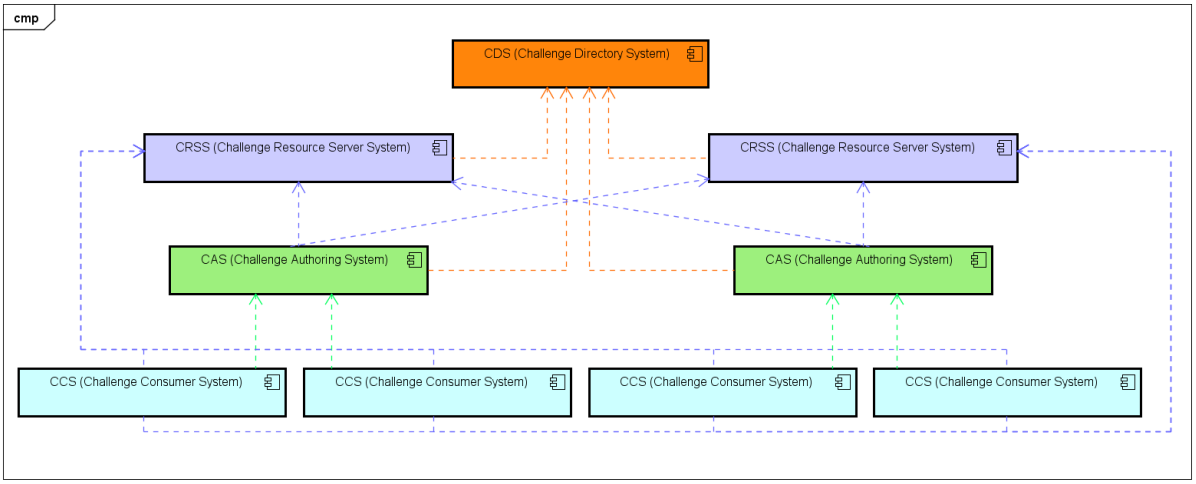



Abb. 15 Component Diagramm

### 6.6.2 HL-CDS

Nachfolgend wird ein Einblick in das CDS und deren gespeicherten Datenbankeinträge gegeben.

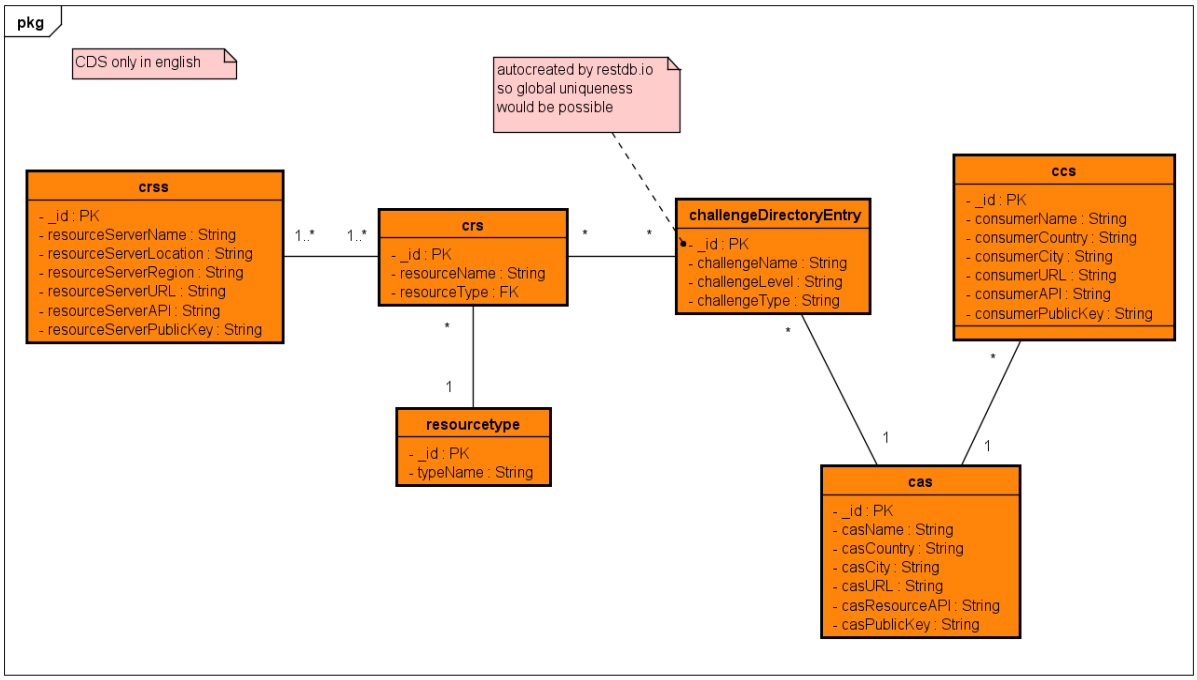



Abb. 16 Domain Model - HL-CDS

### 6.6.3 HL-CAS

In diesem Abschnitt soll ein Überblick über das Domain-Model der Kernapplikation dieser Studienarbeit vermittelt werden. Besonders erwähnenswert ist die Umsetzung der Mehrsprachigkeit. Die dafür zuständigen Tabellen (languageDefinition, attributeTranslation und translatableAttribute) sind ganz unten in der Grafik ersichtlich. Diese enthalten alle textuellen Daten, welche übersetzbar sein müssen. Damit die Abbildung übersichtlich bleibt wurde auf die Assoziationen zwischen den Übersetzungstabellen und dem Rest des Domain Models verzichtet. Damit die Assoziationsinformationen trotzdem erhalten bleiben, wurde jede Entität, welche auf eben diese Tabellen verweist, mit der Endung TA für «Translations Attribute» versehen.
Da dieses Diagramm bereits im «Technischen Bericht» detailliert erklärt wurde, sei hier auf dieses verwiesen.

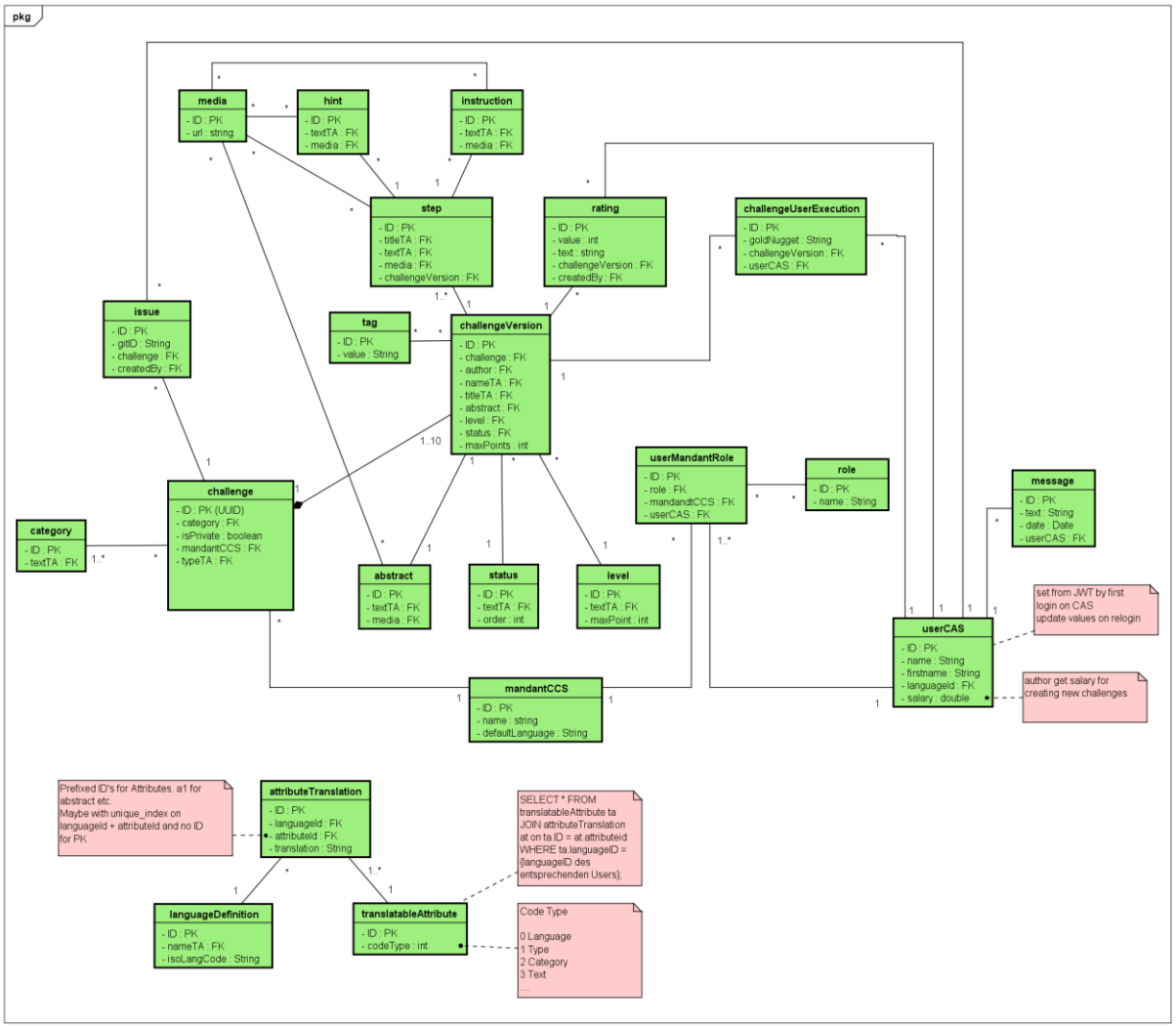



Abb. 17 Domain Model - HL-CAS

### 6.6.4 HL-CCS

Hier soll kurz illustriert werden, wie die Struktur auf einem CCS aussehen wird und welche Daten mindestens persistiert werden müssen.

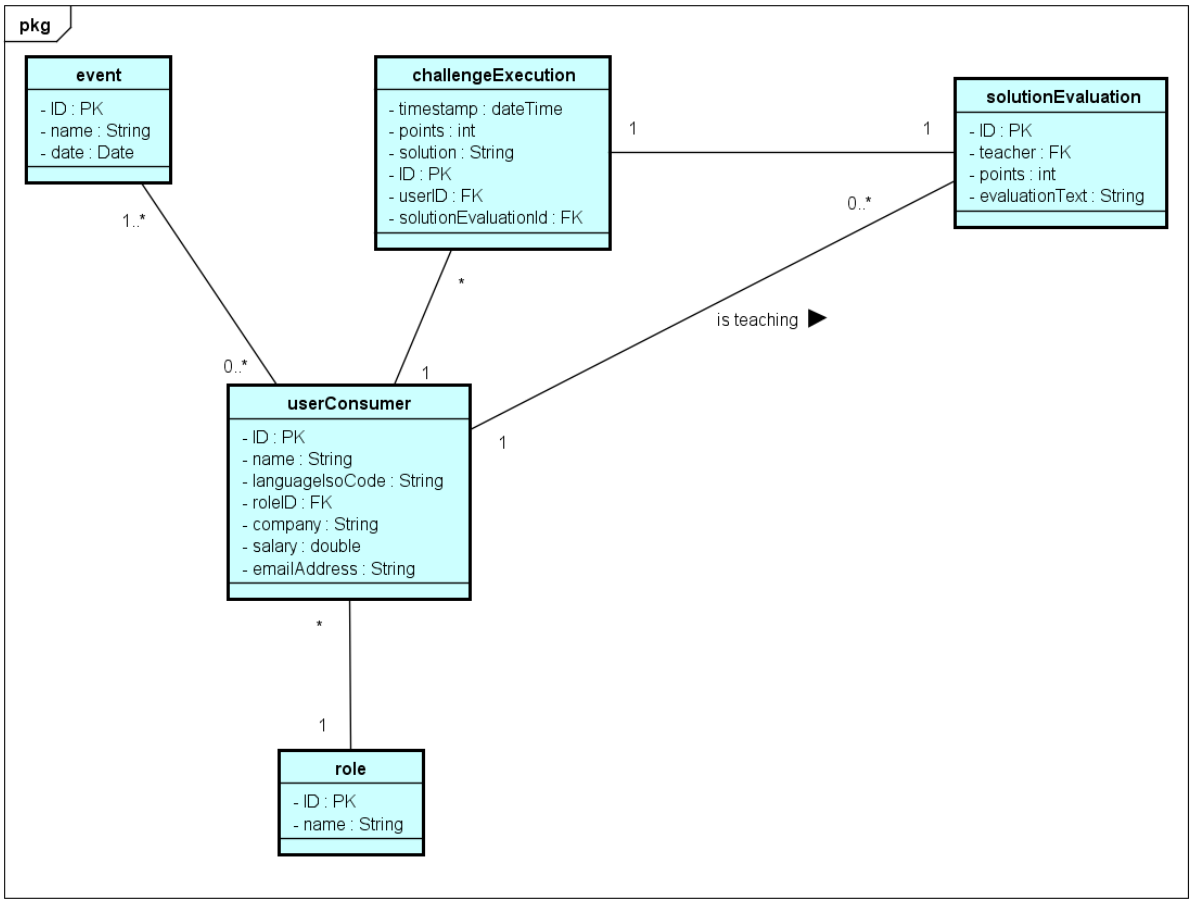



Abb. 18 Domain Model - HL-CCS

## 6.7 Systemanforderungen

### 6.7.1 Erstellung neuer Challenge

Darstellung des Ablaufs einer Challenge-Erstellung, welche bereits im Technischen Dokument erklärt wurde.

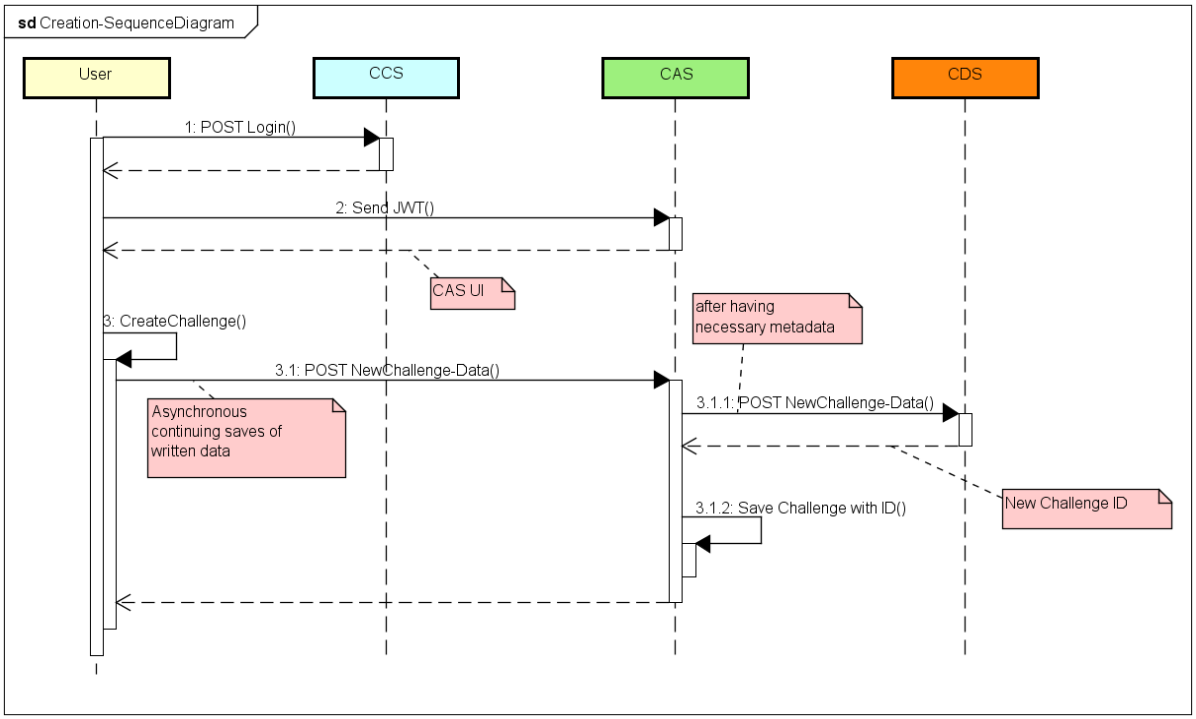



Abb. 19 Sequenzdiagram – Challenge-Erstellung

### 6.7.2 Ausführen einer Challenge

Es folgt eine sequentielle Abbildung der benötigten Anfragen und Antworten an die Teilsysteme des Hacking-Lab 2.0 für das Ausführen einer Challenge. Diese wurde bereits im Technischen Dokument genauer erläutert.

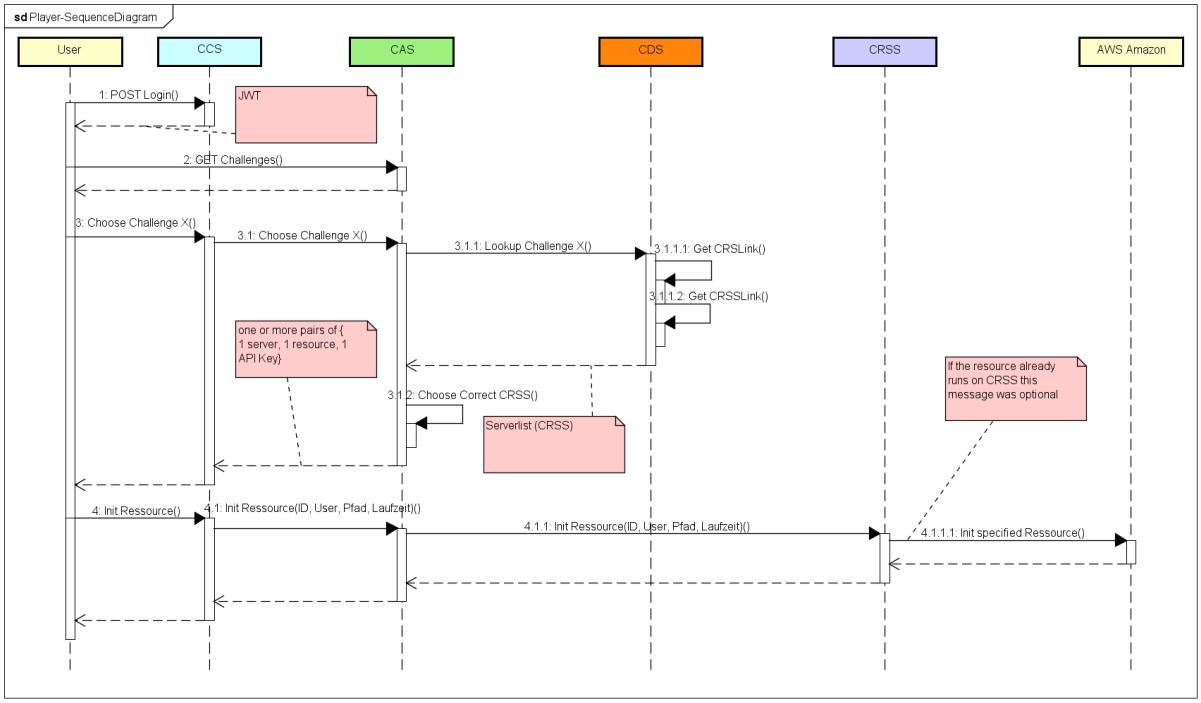



**Abb. 20 Sequenzdiagram – Challenge-Ausführung**

### 6.7.3 Erfassung eines Issue

Im Folgenden wird das Konzept des Issue-Systems anhand eines Sequenzdiagramms verbildlicht. Dieses wurde bereits im Technischen Bericht genauer erklärt.

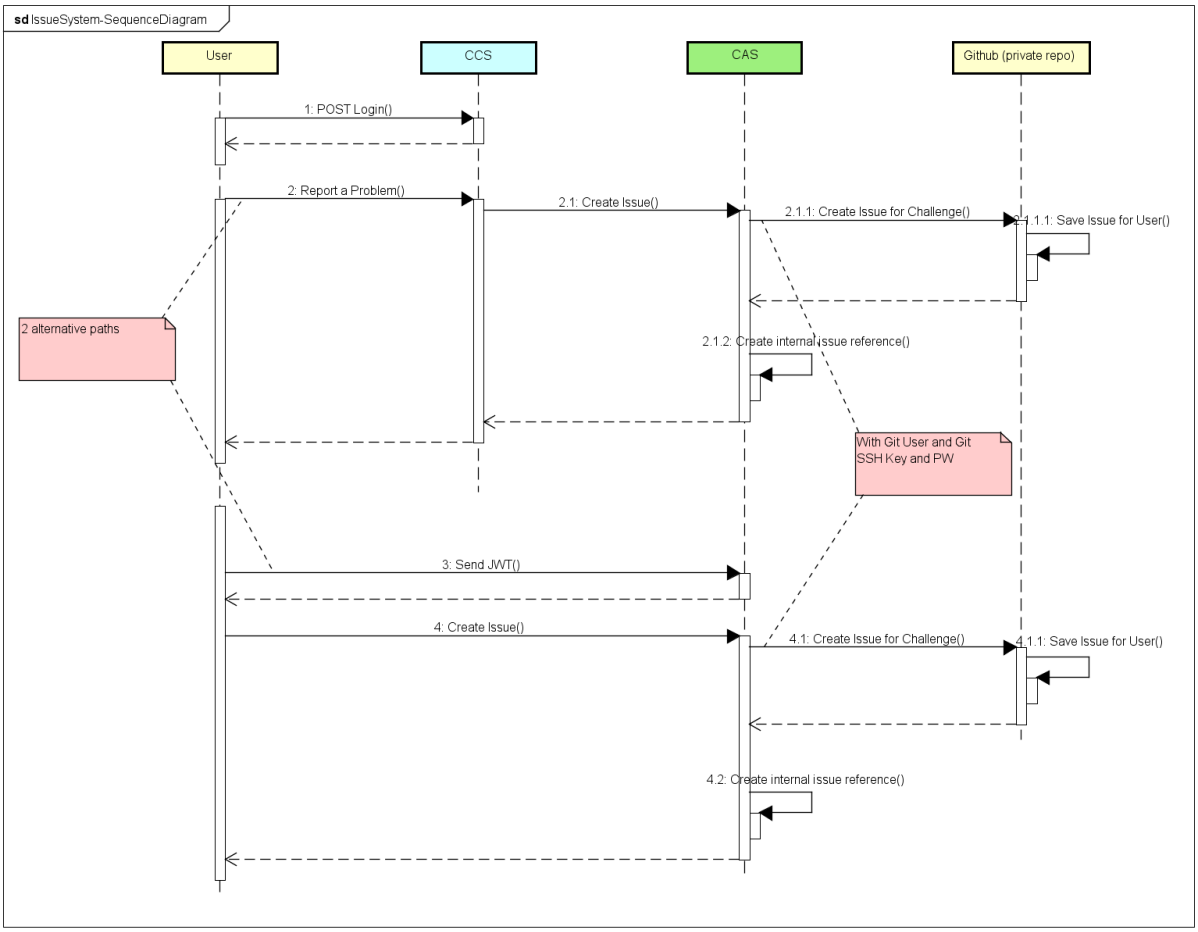



**Abb. 21 Sequenzdiagram – Issue-Erfassung**

# Hacking-Lab 2.0

## 7 Software Architektur Dokument

Janick Engeler
Yanick Gubler

## 7.1 Änderungsgeschichte

| Datum | Version | Änderung | Autor |
|---|---|---|---|
| **13.10.2017** | 1.0 | Erstellung Dokument | Yanick Gubler |
| **05.12.2017** | 1.1 | Ergänzung Systemübersicht + Deployment | Yanick Gubler |
| **08.12.2017** | 1.2 | Ergänzung Systemübersicht | Yanick Gubler |
| **18.12.2017** | 1.3 | Logische Architektur, Dokument überarbeitet | Janick Engeler |
| **19.12.2017** | 1.4 | Schematische Benutzeroberfläche + Ergänzungen | Janick Engeler |

Tab. 3 Änderungsgeschichte - Software Architektur Dokument

## 7.2 Einführung

### 7.2.1 Zweck

Dieses Dokument beschreibt die Software Architektur der Studienarbeit Hacking-Lab 2.0.

### 7.2.2 Gültigkeitsbereich

Dieses Dokument ist über die ganze Projektdauer gültig. Es wird fortlaufend ergänzt und die Änderungen in der Änderungshistorie dokumentiert. Gültig ist immer die aktuellste Version.

### 7.2.3 Referenzen

- Projektplan
- Anforderungsspezifikation-Domänenanalyse

### 7.2.4 Übersicht

Am Anfang des Dokumentes wird versucht, eine Übersicht über die gesamte Systemarchitektur zu geben. Beginnend mit einer Systemübersicht und den Zielen und Einschränkungen der Architektur. Anschliessend wird noch die logische Architektur erläutert. Zum Schluss wird dann noch das Deployment mit seinem Diagramm und der Grössen- und Leistungseinschränkungen und eine Übersicht der gemachten Tests erklärt.

Verwendete Abkürzungen können dem Dokument «Anforderungsspezifikation-Domänenanalyse» im Glossar-Kapitel entnommen werden.

## 7.3 Systemübersicht

In der folgenden Grafik wird die gesamte Systemarchitektur des HL 2.0 dargestellt. Dieses zeigt auch Komponenten, welche nicht Bestandteil dieser Studienarbeit sind.

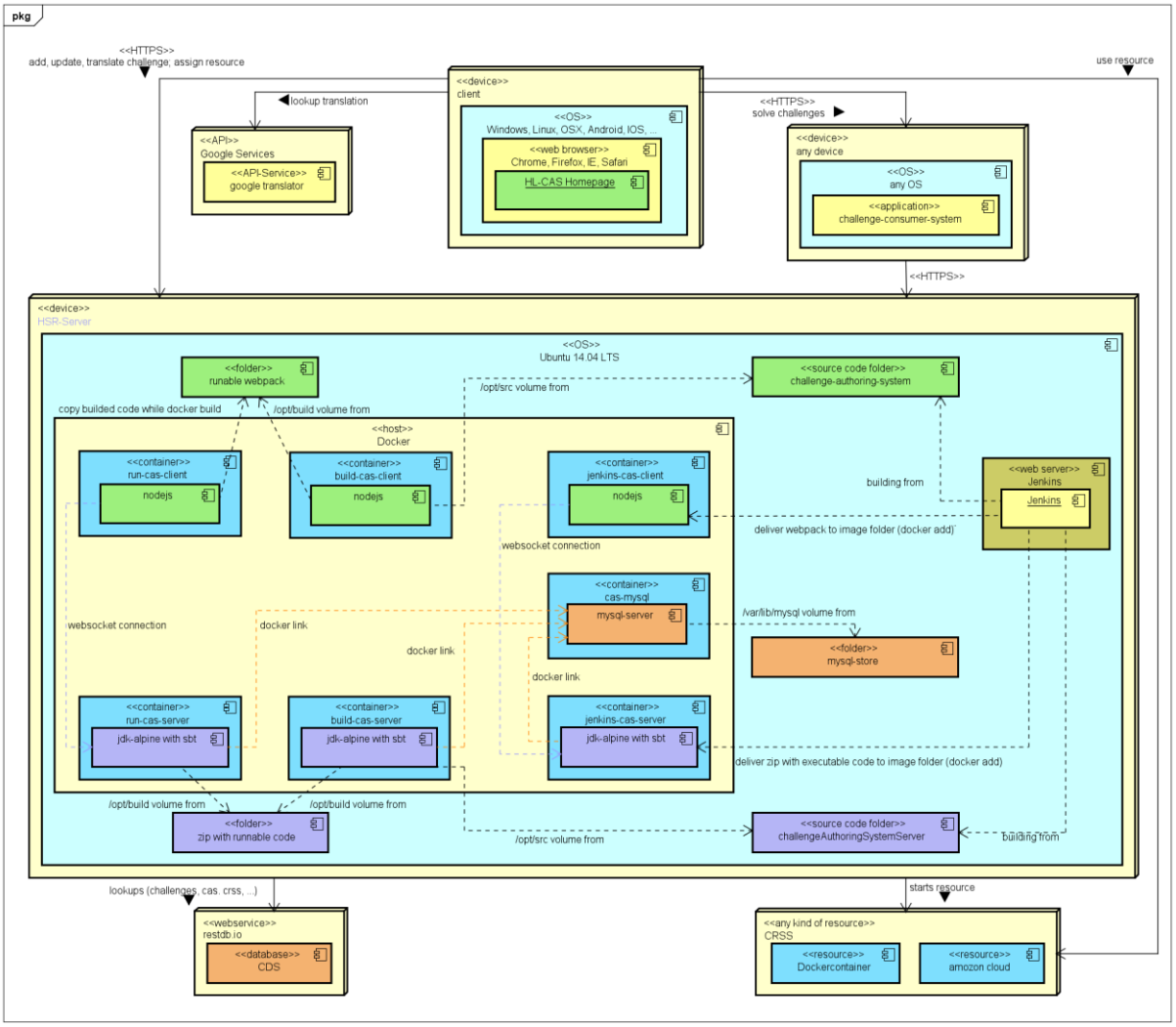



**Abb. 22 Deployment Diagramm**

Die Systemarchitektur dieses Projektes besteht aus einer Client-Applikation (challenge-authoring-system), welche in einem Node.js-Docker-Container zur Verfügung gestellt wird. Sowie einer Server Applikation (challengeAuthoringSystemServer). Die clientseitige Applikation (ReactJS) wird in einem Webbrowser, vorzugsweise Google Chrome ausgeführt. Diese dient dazu neue Challenges zu erfassen, diese zu editieren oder zu übersetzen und ihnen Ressourcen zuzuweisen. Die Serverapplikation, welche mittels des Play! Frameworks implementiert wird, läuft auf dem Framework internen Akka HTTP Webserver, welcher wiederum in einen jdk-alpine Docker-Container, mit zusätzlich installiertem SBT verpackt wurde.

Das HL-CAS läuft auf einem Ubuntu 14.04 LTS Server, welcher für die Dauer dieser Studienarbeit und der darauffolgenden Bachelorarbeit von der HSR virtuell zur Verfügung gestellt wird. Der Zugriff auf die Applikation kann über die Webadresse https://sinv-56043.edu.ch.ch:*jeweiliger_port* aufgerufen werden. Weitere Informationen zu den Ports und für den Zugriff von aussen, kann man aus dem Kapitel «Docker-Netzwerk» unter Deployment in diesem Dokument entnehmen.

### 7.3.1 Komponenten

Im Folgenden werden alle Komponenten des HL 2.0 kurz vorgestellt.

#### 7.3.1.1 Client

Der Client ist ein beliebiger Webbrowser. Dieser wird zum einen dazu verwendet, die Webseite des CCS darzustellen, welche in erster Linie dazu dient Challenges zu lösen. Zum anderen kann er auch die Webseite des CAS anfragen, welche primär dazu dient, Challenges zu erfassen, zu editieren oder zu übersetzen und Ressourcen zuzuweisen.

#### 7.3.1.2 Google Translator

Die CAS-Client Applikation greift direkt vom Client aus auf das Google Translator API zu. So können ganze Challenges automatisch in eine neue Sprache übersetzt werden. Der jeweilige Challenge-Bearbeiter muss dann nur noch die Korrektheit der Übersetzung bestätigen oder allfällige Korrekturen direkt im Editor vornehmen.

#### 7.3.1.3 Challenge Consumer System

Auch wenn das CCS nicht Bestandteil dieser Studienarbeit ist, wird hier kurz darauf eingegangen. Dieses System ist die Verbindung zwischen dem HL-CAS und dem Client. Die folgende Aufzählung soll die Hauptaufgaben des HL-CCS aufzeigen:

- Verwaltung der Benutzer, Gruppen, Rollen
- Darstellung von Challenges für die User (REST API CAS)
- Korrektur von Goldnugget basierten User Solutions (REST API CAS)
- Korrektur und Bewertung von User Lösungen
- Instanziierung, Löschung, Verwendung von Challenge Ressourcen (REST API CRS)
- Verwaltung Balance (Vermögen eines Benutzers in HL$)
- Verwaltung Points für gelöste Challenges
- Verwaltung Rangliste(n), Statistiken, Reports von Events

#### 7.3.1.4 Challenge Authoring System

Die folgende Aufzählung zeigt die Hauptaufgaben des HL-CAS:

- Verwaltung der Challenge (Abstract, Medien, Steps, Hints, Instructions)
- Verwaltung Challenge Status (active, archived, problem)
- Verwaltung Challenge Issues (git repo)
- Versionierung von Challenges (10 Versionen)
- Verwaltung verschiedener Übersetzungen von Challenges (Deutsch, Chinesisch, …)
- Verwaltung Challenge Level, (easy, medium, hard)
- Verwaltung Challenge Points (10, 20, 30)
- Verwaltung Challenge Type (Remote, Local, …)
- Verwaltung Bewertungen der User von einer Challenge (REST API für Consumer)
- Verwaltung Challenge Usage Policy (public, private, …)
- Bereitstellung API für Goldnugget-Check (autocheck)

Weiter ist das HL-CAS auf zwei Applikationen (Client: challenge-authoring-system und Server: challengeAuthoringSystemServer) aufgeteilt. Diese Aufteilung soll in den Folgekapiteln genauer dargestellt werden.

#### 7.3.1.4.1 challengeAuthoringSystemServer

Der ChallengeAuthoringSystem-Server (im weiteren Verlauf als CAS-Server bezeichnet) stellt das REST API für den Client und die zugreifenden Challenge Consumer Systeme bereit. Es werden verschiedenste Routes zur Verfügung gestellt, welche im Hintergrund durch Controllers behandelt werden.
Da das REST API verschiedenste Inhalte zu verarbeiten hat, gibt es jeweils auch unterschiedliche Controller, welche die einzelnen Themengebiete behandeln. So gibt es als Einstiegspunkt den SecurityController, welcher unter anderem Login Anfragen bearbeitet und das eingehende JWT überprüfen kann. Wichtig hierbei ist, dass man zu diesem Zeitpunkt noch nicht authentifiziert sein muss, damit eine Anfrage erfolgreich verarbeitet werden kann.
Der wichtigste Controller ist der ChallengeController, der die einzelnen Funktionen für die Erfassung einer Challenge bereitstellt. Im ChallengeController muss der Benutzer für jede nach aussen zugängliche Funktion ein entsprechendes Token im Header mitschicken, damit eine korrekte Antwort zurück geliefert werden kann. Auf die Controller und ihre Authentifizierung wird im entsprechenden Kapitel innerhalb dieses Dokumentes detaillierter eingegangen.
Als weiterer Punkt ist der Server für die Speicherung der Challenge-Daten verantwortlich, wozu er sich mit einer MySQL Datenbank verbindet. Beim Start des Servers werden initial Daten erstellt und bereitgestellt, welche für die Erfassung von Challenges und den dazugehörigen Erstellern benötigt werden. Dazu zählen Metadaten wie der Schwierigkeitsgrad von Challenges und Rollen für die Ersteller und Reviewer.

#### 7.3.1.4.2 Challenge-Authoring-System

Das Challenge-Authoring-System ist eine React Applikation, die das User Interface für das Erfassen von Challenges usw. zur Verfügung stellt. Sie wird bewusst vom Server getrennt, damit sie zu einem späteren Zeitpunkt leichter in das Challenge Consumer System oder ein anderes dafür vorgesehenes System integriert werden kann.
Die Daten, welche das CAS verwendet, werden über Anfragen an das REST API des Servers geholt. Es persistiert keine Daten selber, sondern sendet kontinuierlich Daten an den Server, damit bei einem Absturz des Browsers oder ähnlichen Problemen nichts verloren geht und danach an gleicher Ort und Stelle weitergearbeitet werden kann. Die genaueren Details dazu werden in einem späteren Kapitel erläutert.
Zurzeit wird der Client nur in einer Sprache angeboten. In einem späteren Schritt sollen auch die einzelnen Labels usw. vom Server geholt werden, damit nicht nur eine Mehrsprachigkeit der einzelnen Challenge-Attribute bewerkstelligt werden kann, sondern auch das gesamte User Interface je nach User Sprache andere Beschriftungen besitzt.

#### 7.3.1.4.3 Challenge Directory System

Das Challenge und Ressourcen Directory dient dem Auffinden von Challenges, Ressourcen, HL-CAS, HL-CCS und HL-CRSS. Das API ist nur über einen API Key nutzbar, sprich CCS und CAS Systeme müssen einzeln mit einem individuellen API Key versehen werden, um im Directory einen Eintrag zu lesen und auch erfassen zu können. Dieses System wurde mit Restdb.io umgesetzt, um den Verwaltungs- und Installationsaufwand möglichst gering zu halten. Somit wird ein globales Verzeichnis bereitgestellt, in welchem stets alle Challenges eines jeden HL-CAS aufgelistet sind.
Eine weitere Information, welche dieser Datenbank entnommen werden kann, ist welche Challenges in welchen Sprachen verfügbar sind. So können mittels der Standardsprache eines Challenge Consumer Systems oder eines einzelnen Benutzers die passenden Challenges für das CCS oder den einzelnen User herausgefiltert werden.
Damit man für jeden Benutzer immer die kürzeste Distanz zu einer Ressource anbieten kann, sind zudem die Standorte der Ressourcen-Server (HL-CRSS) in Restdb.io persistiert. Somit wird für jeden Benutzer stets der von ihm am wenigsten weit entfernte Ressourcen-Server geliefert, auf dem die von ihm benötigte Ressource verfügbar ist. Damit sollen Antwortzeiten minimiert werden.

#### 7.3.1.5 Challenge Resource Server System

Dieses System beinhaltet verschiedenste Ressourcen, welche via REST-API und API-Key an den Systemen HL-CAS und HL-CCS zur Verfügung gestellt werden. Die Hauptaufgaben sind folgende:

- Verwaltung von Challenge Ressourcen
- Bereitstellung REST API für Consumer und CAS
- Beispiele für Ressourcen
  - Docker
  - Amazon Cloud (AWS)
  - Images, Binaries
  - Virtual Machines
  - …

### 7.3.2 Client-Server Kommunikation

Im Challenge Authoring System gibt es zwei Arten von Kommunikation zwischen Server und Client. Die erste verwendet einfache HTTP Anfragen und bekommt dafür in einer HTTP Response das zugehörige JSON Objekt. Die zweite verwendet Websockets, damit die Eingaben jederzeit auf dem Server persistiert werden können. Die beiden Arten sollen nun kurz erläutert und deren Verwendung aufgezeigt werden.

#### 7.3.2.1 HTTP

Der grösste Teil der Kommunikation findet über einfache HTTP Anfragen statt. Dazu ruft der Client das REST API des Servers auf und bekommt dafür ein JSON Objekt in der Response zurückgeliefert, welches er dann entsprechend seinen Anforderungen verarbeiten oder direkt darstellen kann. Unter anderem werden auf der ersten Seite des Clients, alle verfügbaren CCS in einer DropDown-Liste dargestellt. Diese werden beim Laden der Seite über eine solche Anfrage angefordert und als JSON Array zurückgeliefert.
Beim Editor werden die einfachen Eingabefelder mittels dieser Kommunikationsart verarbeitet, da diese oftmals nicht sehr viel Text beinhalten und es deshalb nicht nötig war, auch hier eine bidirektionale Verbindung herzustellen. Einfache Eingabefelder sind in diesem Zusammenhang unter anderem das Feld für den Titel der Challenge, sowie Dropdown-Felder, welche z.B. die vorhandenen Levels einer Challenge darstellen.
Im Bild unten wird diese Kommunikation vereinfacht dargestellt:

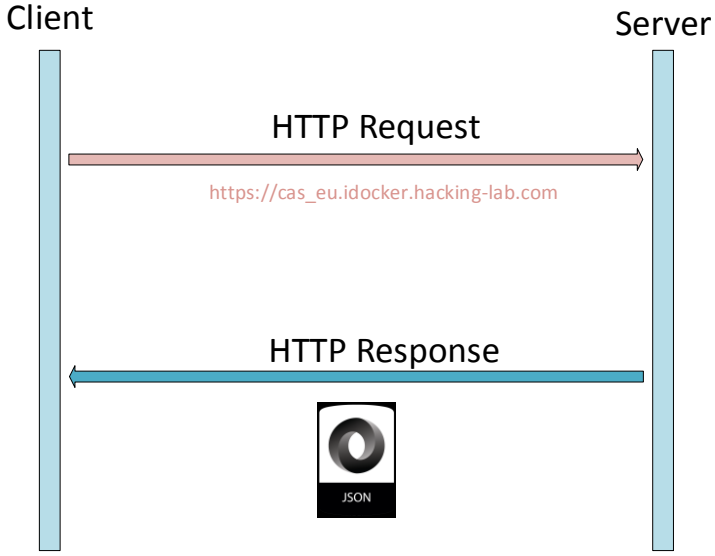



**Abb. 23 HTTP-Kommunikation**

#### 7.3.2.2 Websocket

Ein Websocket zwischen Client und Server wird dazu verwendet, dass analog des Medium Editors die eingegebenen Daten immer auf dem Server persistiert werden können, auch wenn der Browser abstürzt oder ein anderes Problem auftritt. Diese Kommunikationsart wird deshalb für die Felder verwendet, in welche relativ viel Text eingegeben wird, wie z.B. für die Abstracts oder Hints. So kann jede Tasteneingabe laufend auf den Server gesendet und dort direkt in der Datenbank abgelegt werden. Es ist klar, dass dadurch viel Overhead entsteht, da pro Eingabe eine Message über den Socket geschickt wird. Dies wird jedoch hingenommen, da eine Anforderung des Editors war, dass keine einzige Eingabe verloren gehen soll.

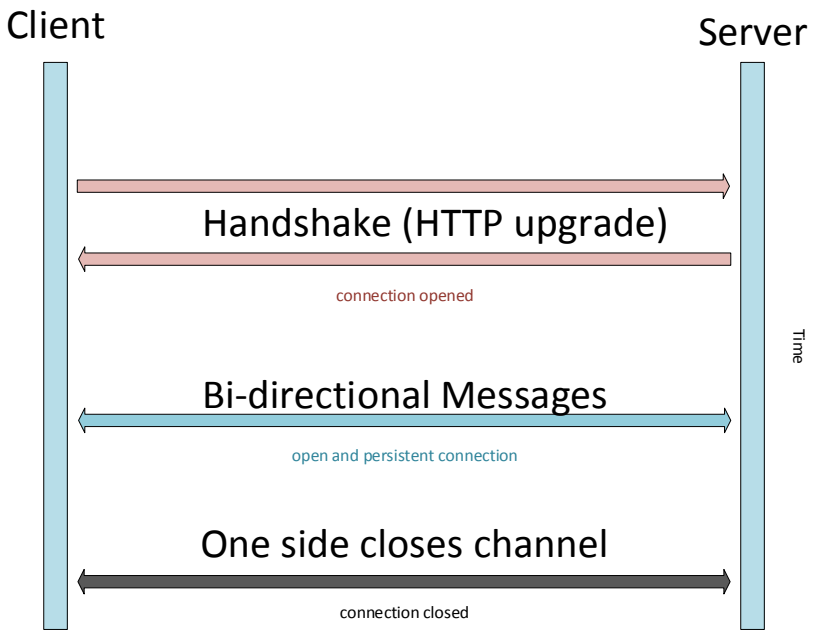



**Abb. 24 Websocket Kommunikation**

## 7.4 Architektonische Ziele & Einschränkungen

Die Applikation stützt sich auf eine MVC Architektur. Somit ist der grobe Aufbau in seinen Grundzügen definiert. Jedoch wurde die Architektur noch weiter unterteilt, um die Granularität noch feiner gestalten zu können. Dies konnte durch eine strikte Trennung zwischen Backend und Frontend erreicht werden. Die Trennung hat zur Folge, dass Codeduplizität minimiert wird und das Unit Testing auf kleine Bereiche beschränkt werden kann.

### 7.4.1 Backend

Die Aufteilung auf dem Server erfolgt folgendermassen:

#### 7.4.1.1 Controllers

Auf dem Server gibt es mehrere Controller, welche die Verbindung zwischen Client und Server sicherstellen. Die Controller stellen die eigentlichen Implementierungen für das REST API zur Verfügung. Damit der Umfang dieser Controller nicht den Rahmen sprengt, wurden eigene Service Klassen erstellt, damit Code ausgelagert werden kann und nicht die gesamte Logik in den Controller implementiert wird.

#### 7.4.1.2 Parsers

Hier werden einzelne selber implementierte Parser-Klassen abgelegt. Es wurde lediglich ein Parser für den Image-Upload benötigt, da der eingehende Byte-Stream vom Play internen MultipartFile Body-Parser nicht korrekt verarbeitet werden konnte.

##### 7.4.1.2.1 MyMultipartFileBodyParser

Dieser Parser wird dazu verwendet hochgeladene Bilder korrekt verarbeiten und auf dem Server ablegen zu können. Es wurde hier eine eigene Implementierung verwendet, damit mehr Kontrolle über das Wie und Wo der Speicherung von Bildern gewährleistet werden kann. Konkret soll hier Zugriff auf den akkumulierten Byte-Stream eines jeden Files ermöglicht werden, um daraus direkt eine Datei, ohne Verwendung einer temporären Ablage, zu generieren.

#### 7.4.1.3 JWTUtil

Es wird eine eigene statische Klasse erstellt, mit welcher ein JWT (JSON Web Token) generiert und dieses dann später wieder verifiziert werden kann. So kann die Verwendung von JWT sauber gekapselt werden.
Ein weiterer wichtiger Teil dieser Util-Klasse erstellt aus dem Payload des JWT's ein neues User-Objekt, welches dann später in der Datenbank über den entsprechenden Service persistiert werden kann. Der erstellte User beinhaltet unter anderem, die mit dem JWT übermittelten Rollen, welche sogleich einem Mandanten zugeordnet werden können. Dies soll eine spätere Mandantenfähigkeit des Systems ermöglichen.

#### 7.4.1.4 MarkdownConverter

Es wird eine MarkdownConverter Klasse erstellt, welche dazu verwendet wird, den vom Client generierten HTML-Code in das gewünschte Markdown-Format zu konvertieren. Es werden alle im Editor möglichen Styles, wie Fett, Kursiv und verschiedene Titelgrössen unterstützt. Als Parser wird die Open Source «jsoup» Java Library verwendet, welche Methoden für das extrahieren und manipulieren von HTML Dokumenten bereitstellt.

#### 7.4.1.5 Interfaces

Hier werden die für die Repositories definierten Interfaces erstellt. Diese Interfaces ermöglichen es die für die Speicherung verwendeten Repos ohne aufwendige Codeänderungen mit einer alternativen Implementierung zu ersetzen. So kann später ohne grossen Aufwand die Implementierung der Datenbank verändert werden, falls dies benötigt wird.
Die Interfaces müssen in einer eigenen Klasse auf eine explizite Implementierung gebunden werden, damit Google Guice die benötigten Dependencies automatisch korrekt injecten kann.

#### 7.4.1.6 Repos

Hier werden die Implementierungen der einzelnen Repository-Interfaces abgelegt, welche die Datenbankzugriffe kapseln. Die konkrete Implementierung verwendet Ebean für die Datenbankzugriffe. Es werden jeweils die CRUD Operationen implementiert, welche dann ein CompletionStage Attribut zurückliefern, damit diese Methoden asynchron ablaufen können. Teilweise ist es notwendig auf das Resultat einer Methode «zu warten», damit mit dem entsprechenden Datenbankobjekt weitergearbeitet werden kann. Dies wird dann in den Service Klassen entsprechend geregelt.

#### 7.4.1.7 Services

Die Services kapseln die Datenbankzugriffe der Repositories von den Controllern ab. So sollte es möglich sein in den Controllern nur die Services und keine Repositories zu verwenden. Somit wird die Dependency Injection durch Google Guice übersichtlicher und einfacher nachvollziehbar. Auch kann der Code dadurch sinnvoller getestet werden, da die Controller nur die Services verwenden, sodass fast ausschliesslich diese getestet werden müssen.

#### 7.4.1.8 Model

Das Model-Package wird in zwei unterschiedliche Bereiche unterteilt. Der erste Bereich besteht aus den Klassen, welche für das JSON-Handling verwendet werden. Es gibt hier also alle Klassen, welche als DataTransferObject fungieren.

Im zweiten Bereich des Models-Packages werden die im Anforderungsdokument beschriebenen Klassen definiert, welche für die Speicherung in der Datenbank verwendet werden. Aus diesem Grund wird hier für die Erklärung der einzelnen Klassen auf dieses verwiesen. Da die Klassen keine konkrete Logik enthalten, sondern nur auf einfachen Getter und Setter, welche mit Annotations versehen wurden, basieren, sollte diese allgemeine Beschreibung genügen. Der einzige erwähnenswerte Punkt in diesem Zusammenhang wäre, dass alle Klassen, bis auf die Challenge, eine abstrakte Basis-Klasse erweitern, welche die Attribute für die ID, sowie die Version der Daten beinhaltet, damit diese nicht auf jeder Klasse wieder neu implementiert werden müssen. Der Grund, weshalb die Challenge selber diese Basis-Klasse (BaseModel) nicht erbt, ist, dass sie als einzige Klasse für die ID einen String-Wert benötigt. Nicht wie in allen anderen einen einfachen Zahlenwert, der laufend hochgezählt wird.

#### 7.4.1.9 Modules

Es werden eigene Module erstellt, um die benötigten Actors für die Websockets, sowie konkrete Implementierungen für die Repository-Interfaces zu binden. Dem Model entsprechend gibt es für die drei Arten von Objekten (Translation, User und Challenges) jeweils separate Modules damit eine saubere Trennung erzielt werden kann.

### 7.4.2 Frontend

Das Frontend ist in diverse Teile gegliedert und besteht hauptsächlich aus JSX-Code. Es wurde hier eine Implementierung mit Redux angestrebt, damit einfacher getestet werden kann und State-Veränderungen prognostizierbar werden. Es werden hier die verschiedenen Ordner, durch welche die Applikationsteile gegliedert werden, kurz beschrieben.

#### 7.4.2.1 Actions

Im Actions Ordner werden die «Pure-Functions» für die Reducers definiert. Weiter befindet sich hier eine Definitionsdatei für die verwendeten Typen. Es können hier zwei Arten von Funktionen unterschieden werden. Als erstes seien die von aussen zugänglichen Methoden genannt, welche direkt von den Components aufgerufen werden, diese könne nicht direkt eine Zustandsänderung vollbringen. Dazu gibt es private Methoden, welche nur den Redux internen State verändern und ansonsten keine Logik implementieren. So wird der ganze Code sehr einfach zu testen und verursacht keine unvorhergesehenen Auswirkungen.

#### 7.4.2.2 Components

Im Components Ordner befinden sich die einzelnen Komponenten, aus welchen die Applikation zusammengebaut wird. Die Komponenten werden sauber gegliedert, um die Wiederverwendbarkeit zu erhöhen und den Code übersichtlicher gestalten zu können. Damit für die einzelnen Controls jeweils eigene Implementierungen für die Events sichergestellt werden können, wurden diese bis auf das einfachste Input-Feld selber implementiert. Auch kann so ein ungewünschtes Standardverhalten umgangen werden.

#### 7.4.2.3 Containers

Im Containers-Ordner wird die Verwendung des für die States verwendete Store in den einzelnen Root-Dateien gekapselt. Es gibt jeweils eine Implementierung für Entwicklung und eine für den produktiven Gebrauch. Der einzige Unterschied ist, dass speziell für die Entwicklung eine Devtool Komponente verwendet wird, womit sehr einfach der aktuelle Inhalt der States angezeigt werden kann. In den Root Dateien wird zusätzlich die eigentliche App gekapselt, in welcher die Login-Seite, sowie die anderen verwendeten Komponenten dynamisch gerendert werden. Weiter wird in der App.js Datei die Reducer-States auf Props gemappt, damit diese Werte in die Komponenten, wo sie benötigt werden, übergeben werden können.

#### 7.4.2.4 CSS

Hier werden, wie der Name bereits ahnen lässt, verschiedene Stylesheets definiert, welche für das korrekte Styling der Komponenten verantwortlich sind.

#### 7.4.2.5 Middleware

Es wird eine eigene Middleware Implementierung erstellt, welche für die Überprüfung der Authentifizierung für die Serveraufrufe dient. Weiter werden hier die benötigten Header gesetzt, damit dies nicht für jeden Aufruf separat getan werden muss.

#### 7.4.2.6 Reducers

Es werden drei unabhängige Reducers erstellt, welche ihre jeweiligen States separat behandeln und verwalten. Mittels combineReducers werden diese dann zu einem einzigen zusammengebaut und für die anderen Komponenten im App.js geladen.

#### 7.4.2.7 Store

Hier wird der in der Root-Datei verwendete Store definiert. Es wird zuerst die ThunkMiddleware mit der eigenen Middleware-Implementierung instanziiert und dann zurückgegeben. Auch hier gibt es wieder einen Store für die Entwicklung und einen für den produktiven Gebrauch. So muss hier für die Entwicklung das DevTool korrekt instanziiert und übergeben werden.

#### 7.4.2.8 Webpack

Die Konfiguration und die Erstellung des Client Builds wird mittels des Module Bundlers Webpack vollzogen. Es gibt hier eine Webpack-Datei für die Entwicklung und eine für den produktiven Gebrauch. Damit gemeinsam verwendete Plugins und Loaders nicht mehrmals definiert werden müssen, wurde eine dritte Webpack-Datei angelegt, welche diese Komponenten enthält. Mittels Webpack-Merge werden dann beim Build entweder die Entwicklungs-Version oder die produktive Version mit der gemeinsamen Datei zusammengefügt.
Ein weiteres in diesem Projekt entscheidendes Plugin ist Dotenv-Webpack. Dieses wird dazu verwendet, Umgebungsvariablen auszulesen und in die einzelnen Dateien zu injecten. Damit kann die Konfiguration sehr schön ausgelagert werden.
Um die Qualität des Codes sicherstellen zu können, wurde eslint verwendet, welches in IntelliJ entsprechende Warnungen oder Fehler ausgibt, wenn an der Syntax des Codes etwas nicht wie gewünscht formatiert ist. Man kann die Regeln, welche eslint verwendet sehr bequem über eine .eslintrc Datei, entsprechend seinen Anforderungen, konfigurieren.

#### 7.4.2.9 Server

Es werden zwei Webserver für das Ausliefern, des durch Webpack erstellten bundle.js, sowie der eigentlichen HTML-Seite (index.html) implementiert. Der entscheidende Unterschied zwischen den beiden Servern ist, dass für die Entwicklung der Webpack-Dev-Server und für den produktiven Gebrauch einen eigen entwickelten Node.js Express-Server verwendet wird. Grundsätzlich unterscheiden diese sich nur gering, da beide die gleichen Routes zur Verfügung stellen und in etwa gleich aufgebaut sind. Nur die Konfiguration unterscheidet sich.

#### 7.4.2.10 RequireJS

Um Imports zu handhaben und zugehörige Dateien zu finden, bietet RequireJS einen sehr guten Grundbaustein. Anders gesagt, handelt sich um einen Modularen Scriptloader, der die Scripts erst bei Bedarf abholt und ladet.

Weitere Informationen: http://requirejs.org/

#### 7.4.2.11 JQuery

JQuery ist eine sehr verbreitete JavaScript-Bibliothek, die sehr viele praktische Funktionen zur Verfügung stellt. Im Client wird JQuery hauptsächlich für den eigenen Medium Editor benötigt, damit dieser korrekt in den DOM eingefügt und verwendet werden kann.

Weitere Informationen: https://jquery.com/

## 7.5 Logische Architektur

Aufbau der Architektur, wie sie im Projekt umgesetzt wird.

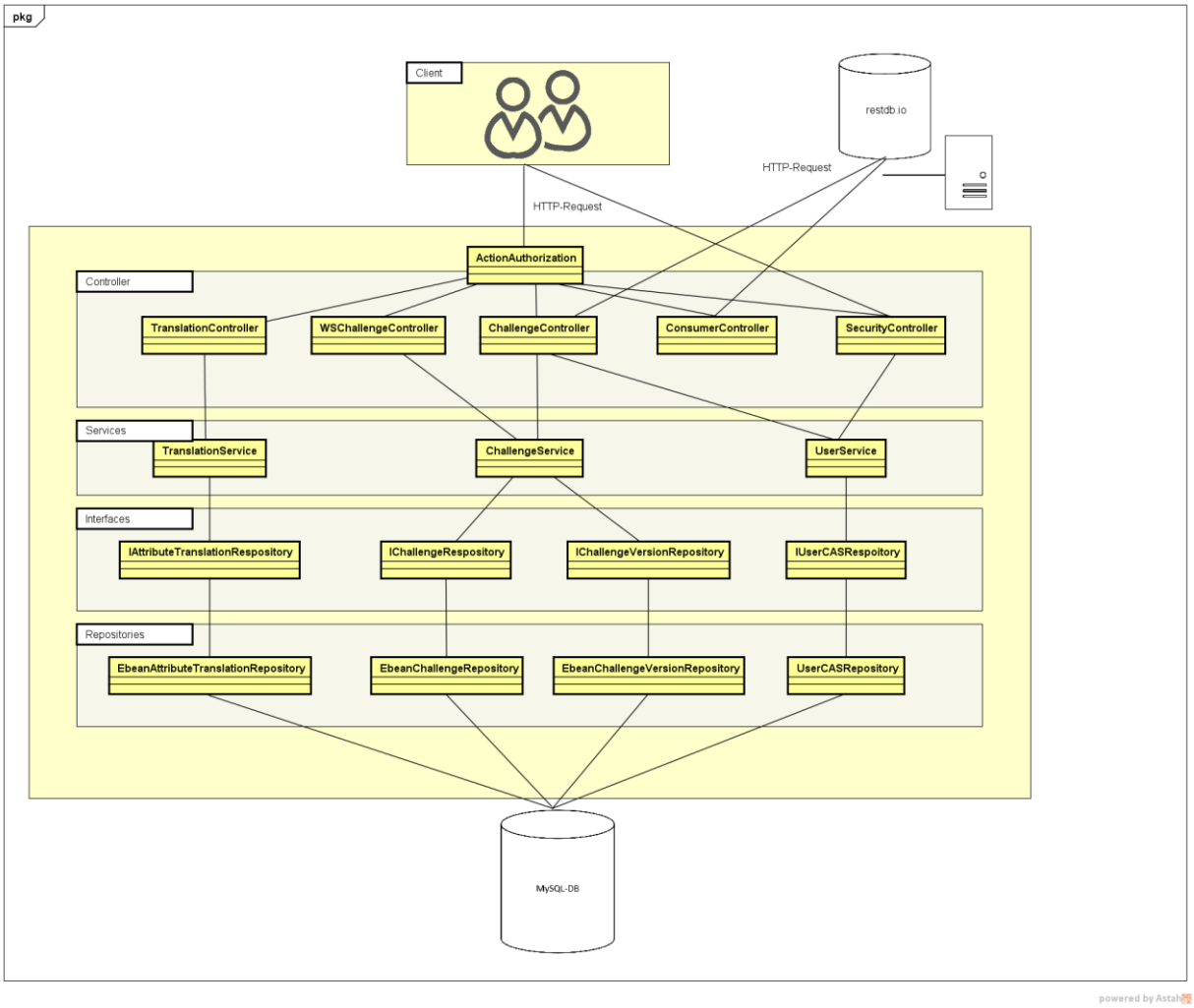



Abb. 25 Schichtenmodell HL-CAS

### 7.5.1 Client

Der Client ist im Prinzip die Webapplikation im Browser. Diese wird wie oben kurz erläutert, aufgeteilt. Die Verbindung zum Server erfolgt einerseits mit Websockets und andererseits mit einfachen HTTP API Aufrufen, welche über die verwendete Middleware sauber gekapselt werden.

### 7.5.2 Restdb.io

Der Datenbankserver von restdb.io fungiert in diesem Projekt als globales Repository, wo alle entscheidenden Informationen zentral und einmalig gespeichert werden. Diese Informationen beinhalten unter anderem die verfügbaren CCS mit ihren Public Keys für die Verifizierung des JWT, sowie die einzelnen CRSS, welche die expliziten Ressourcen verwalten. Weiter werden hier einzelne Attribute der Challenge gespeichert, damit sie von allen CAS angefragt werden können und man so Duplikate verhindern kann. Beim Erstellen einer neuen Challenge wird zuerst auf diesen Server eine POST Anfrage geschickt, damit die Challenge initial erstellt wird und man eine global eindeutige ID für die Speicherung auf der CAS internen Datenbank erhält.
Restdb.io stellt ein einfaches REST API zur Verfügung, welches mittels API Key angesprochen werden kann. So kann von jedem datenkonsumierenden System darauf zugegriffen und die benötigten Daten bezogen werden. Hauptsächlich wird dieser Server vom Consumer Controller, für die Anfrage der verfügbaren Consumer verwendet, sowie wie oben bereits beschrieben vom ChallengeController für das Erstellen einer neuen Challenge. Weiter benötigt das CAS Informationen über die Ressourcen Server, da die Aufgabe des CAS ist, einen geeigneten Server in der Nähe zu finden, wo die angefragte Ressource verfügbar ist.

### 7.5.3 Controller

Grundsätzlich gibt es zwei Eingangspunkte auf dem Server. Erstens der SecurityController und zweitens die ActionAuthorization Klasse. Ersterer wird dazu verwendet die Implementierung der Login Routes, sowie der JWT Verifizierung zu kapseln. Dieser Controller besitzt keine Methode, bei welcher die Anfrage mittels eines JWT authentifiziert sein müsste. Dies da man beim ersten Login noch nicht authentifiziert sein kann und noch keine Daten vom Server an den Client geliefert werden. Weiter wird im Security Controller die OPTIONS Anfrage einer jeden Route verwaltet, damit diese nicht mehrmals implementiert werden muss und einfachheitshalber so einfach wie möglich gehalten werden soll. Der zweite genannte Einstiegspunkt ist die ActionAuthorization Klasse, welche die Play interne Security.Authenticator Klasse erweitert. Diese kann vor jede zu authentifizierende Route-Implementierung annotiert werden, sodass bevor die eigentliche Methode aufgerufen wird, immer zuerst die getUsername Methode des ActionAuthenticators aufgerufen wird. In dieser Methode wird das Token aus dem Header, oder falls nicht vorhanden, aus dem Body entnommen und geprüft, dass das signierte JWT mit dem Public-Key des herkommenden Consumers überprüft werden kann. Falls die Verifizierung erfolgreich ist, wird der entsprechende Benutzer mit seinen Rollen und den anderen gelieferten Attributen auf der Datenbank persistiert und der Aufruf der Route-Implementierung zugelassen. Falls die Verifizierung fehlschlägt, wird dies entsprechend an den Aufrufer gemeldet und die Anfrage mit einem Fehler abgebrochen.

### 7.5.4 Services

In der Service-Schicht wird die eigentliche Logik, die in den Controllern verwendet wird, gekapselt. Hauptsächlich werden zurzeit die Repositories in den Services verwendet, damit diese nicht auf allen Controllern immer wieder von Google Guice injected werden müssen, sondern nur der jeweilige Service. Auch wurden Methoden aus den Controllern ausgelagert, welche von mehreren Controllern verwendet werden, damit kein duplizierter Code entsteht. Aus Gründen der Übersichtlichkeit wurden die Services selbst auf drei unterschiedliche aufgeteilt. Es gibt einen Service für Übersetzungen, einen für das Handling der Benutzer, sowie einen für die Verwaltung der Challenges. So wird verhindert, dass die Service Klassen zu unübersichtlich und überladen werden.

### 7.5.5 Interfaces

Die Interfaces kapseln eine Schicht weiter unten die eigentliche Implementierung der expliziten Repository Implementierungen. Der Grundgedanke, welcher hierbei verfolgt wurde ist, dass man eine alternative Implementierung zu Ebean sehr einfach erstellen kann, ohne dass eine grosse Codeänderung gemacht werden müsste. Im Bild oben wurden aus Gründen der Übersichtlichkeit auf die weniger entscheidenden Interfaces verzichtet. Für weitere Informationen sei auf das GIT Repo oder auf das Klassendiagramm im Anforderungsdokument verwiesen, da für jede dortige Klasse ein eigenes Interface erstellt wurde.

### 7.5.6 Repositories

Die Repositories sind mittels Ebean implementiert und stellen die einzelnen CRUD Operationen für die darunterliegende MySQL Datenbank zur Verfügung. Im Bild oben wurden aus Gründen der Übersichtlichkeit auf die weniger entscheidenden Repositories verzichtet. Für weitere Informationen sei auf das GIT Repo oder auf das Klassendiagramm im Anforderungsdokument verwiesen, da für jede dortige Klasse ein eigenes Repository erstellt wurde.

### 7.5.7 MySQL Datenbank

Die MySQL Datenbank stellt die konkrete Persistenz-Schicht dar. Sie wird durch Play anhand der Ebean-Annotationen auf den Model-Klassen generiert. Damit die Datenbank jeweils auf einem korrekten Stand ist, wird von Play im Evolutions Ordner eine *.sql Datei erstellt, welches mittels ALTER TABLE Statements die Datenbank entsprechend ändert oder wieder neu aufbaut. Falls sich eine solche Annotation ändert, ändert sich auch automatisch die *.sql Datei. Wenn der Server startet, wird eine von Play intern gerenderte View angezeigt, in der man mittels eines Buttons die Evolution durchführen kann.

## 7.6 Prozesse, Threads und Tasks

Um zu verstehen wie Parallelisierungstechniken eingesetzt werden können, muss ein Verständnis für die Funktionsweise des umliegenden Ökosystems existieren.
In unserem Fall hat der Einsatz des Play-Frameworks Auswirkungen auf das Thread-Modell.

### 7.6.1 Play

Intern ist Play von ganz unten an, asynchron aufgebaut. Jeder Request wird auf eine nicht-blockierende Art asynchron behandelt. Die Standardkonfiguration ist für asynchrone Controllers ausgelegt. Man sollte also darauf achten, dass verhindert werden kann, dass einzelne Controllerfunktionen auf Operationen warten müssen (z.B. JDBC).

Um zu gewährleisten, dass der Ausführungskontext im Scope bleibt, muss dieser in den Controller injected und dort verwendet werden. Dies ist jedoch erst die Hälfte der Problemlösung, da man sich so noch immer auf dem Standard Ausführungskontext von Play befindet und durch blockierende Aufrufe der Rendering Thread Pool von Play behindert wird. Deshalb sollten immer eigene Ausführungskontexte erstellt, korrekt konfiguriert und verwendet werden. Da in diesem Projekt die Rendering Engine Twirl jedoch nicht verwendet wird, stellt dies in diesem Umfeld kein Problem dar.

Genauere Informationen können aus der offizielle Dokumentation [12] von Play entnommen werden.

# 7.7 Deployment & Bildautomation

## 7.7.1 Deployment-Diagramm

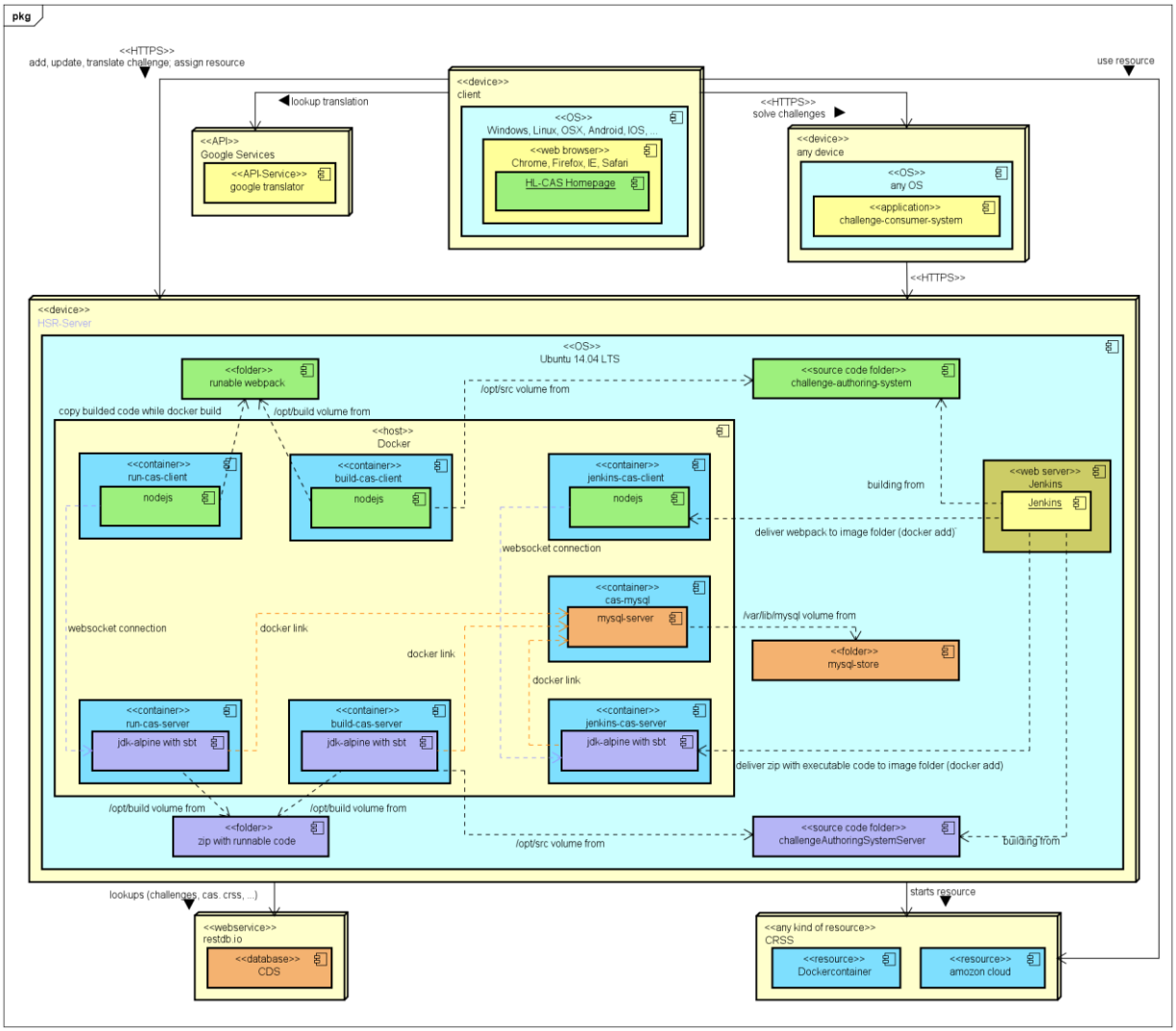



**Abb. 26 Deployment Diagramm**

Im Folgenden wird beschrieben, wie das Deployment des CAS gemacht wird. Damit das CAS schnell und einfach auf weitern Systemen zum Einsatz kommen kann, wurden sämtliche Komponenten in je einen Docker-Container verpackt. So ist es möglich innerhalb kürzester Zeit ein weiteres CAS in Betrieb zu nehmen. Voraussetzung dafür ist ein Betriebssystem, welches Bash Shell Scripts ausführen kann und mit einem Docker-Host ausgestattet ist. Denn um diese Docker-Container zu erstellen, zu starten u.Ä. werden Bash Scripts verwendet.

## 7.7.2 Allgemein

### 7.7.2.1 config.sh

Wenn man nun einen neuen CAS in Betrieb nehmen will, müssen hier einige Änderungen vorgenommen werden. Denn hier werden Variablen gesetzt, welche bei den doit.sh-Dateien als Docker-Umgebungsvariablen eingeschleust werden. So wird via Umgebungsvariable z.B. die Zeitzone, die Datenbank-URL/User/PW übergeben. So werden alle zu tätigen Einstellungen zentral an einem Ort gehalten.

### 7.7.2.2 Doit.sh

Um das Starten der Container so einfach wie möglich zu halten, gibt es für jeden Container ein doit.sh-Skript. Dieses mounted die jeweils benötigten Volumes, setzt Umgebungsvariablen, weisst dem Container eine IP-Adresse im Docker-Netzwerk zu, stoppt den alten Container, falls dieser noch ausgeführt wird und startet abschliessend den neu gebuildeten Container.

### 7.7.2.3 Dockerfile

Hier ist der Aufbau des jeweiligen Docker-Images beschrieben.

### 7.7.2.4 Image-Ordner

Jeder Docker-Container hat einen Ordner «Image». Dieser wird beim Docker-Buildprozess in den Container integriert. So enthält dieser beispielsweise beim run-cas-client-Container einen Ordner «certs», welcher die Serverzertifikatsdateien für den Node.js-Webserver beinhaltet, falls dieser als HTTPS-Dienst ausgeführt wird. Weiter enthält dieser Ordner bei benanntem Container auch die Datei server.js, welche die gesamte Node.js-Serverkonfiguration enthält.

### 7.7.2.5 startup.sh

Diese Datei ist in jedem «Image»-Ordner vorhanden. Bei jedem Dockerfile muss ein Einstiegspunkt (Entrypoint) angegeben werden. Dies wurde eben auf diese startup.sh gelegt. In dieser Datei wird dann die jeweilige Anwendung (CAS-Client, CAS-Server, Buildprozess, usw.) gestartet. Denn ein Docker-Container läuft nur so lange, wie der dazugehörige Einstiegspunkt.

## 7.7.3 cas-mysql

Dieser Container hat zwei primäre Zwecke. Einer ist die Initialisierung des Docker-Netzwerks und die andere ist den MySQL-Server zur Ausführung zu bringen.

Dieser Container erstellt nur eine leere Datenbank. Die Tabellen usw. werden erst während des Buildprozesses vom challengeAuthoringSystemServer erstellt.

#### 7.7.3.1 Docker Netzwerk

Die erste Komponente die zur Ausführung gebracht werden muss, ist der cas-mysql-Docker-Container. Da dieser in seinem doit.sh-Script ein Docker-Netzwerk initialisiert (Standardsubnetz 172.18.0.0/16 kann in der config.sh geändert werden). Jedem Container wird dann in seiner jeweiligen doit.sh-Datei beim docker run Befehl eine Fixe IP in eben diesem Subnetz gegeben. Somit ist die Erreichbarkeit der Container untereinander immer gewährleistet. Denn Docker weisst standardmässig IP-Adresse per DHCP zu, was die Erreichbarkeit nicht gewährleistet.

In der folgenden Tabelle werden alle Container mit ihren Adressen und Ports gelistet. Dabei sind alle Container mit einem externen Port zwischen 40000 – 40010 vom Internet über die URL: https://sinv-56043.edu.hsr.ch:jeweiligen_Port erreichbar.

| Container | Docker-Netzwerk-IP | Port extern | Port Intern |
|---|---|---|---|
| **cas-mysql** | 172.18.0.10 | 3306, 33060 | 3306, 33060 |
| **build-cas-server** | 172.18.0.11 | - | - |
| **jenkins-cas-server** | 172.18.0.12 | 40001 | 9000 |
| **run-cas-server** | 172.18.0.13 | 40002 | 9000 |
| **build-cas-client** | 172.18.0.14 | - | - |
| **run-cas-client** | 172.18.0.15 | 40004 | 3001 |
| **jenkins-cas-client** | 172.18.0.16 | 40005 | 3001 |

Tab. 4 Docker-Netzwerk

#### 7.7.3.2 mysql-Server

Damit die Daten auch nach Beendigung/Absturz des Containers noch verfügbar sind, werden diese nicht intern im Container persistiert. Der Speicherort der Datenbank wird im doit.sh angegeben. Dieser kann beliebig verändert werden. Wir haben diesen für uns auf */opt/cas-mysql-db* gesetzt.

Weiter wird die Datei «createCASUser.sql» via Dockerfile in den Container nach */docker-entrypoint-initdb.d* kopiert. Denn alle Dateien die unter diesem Pfad abgelegt sind, werden automatisch von MySQL zur Ausführung gebracht, sobald der Container gestartet wird. In dieser Datei wird ein Datenbankbenutzer «cas» erstellt, sein Passwort und seine Berechtigungen gesetzt. Weiter enthält das Dockerfile den Namen und das Root-Passwort der zu erstellenden Datenbank.

### 7.7.4 Jenkins

Unser Jenkins-Server enthält einen GitHub-Hook. Somit wird der Jenkins Buildprozess bei jedem neuen Commit auf den master-Branch ausgelöst. Bei jeder Auslösung wird der Arbeitsbereich von Jenkins gesäubert, somit ist gewährleistet, dass keine lokalen Abhängigkeiten mit im Spiel sind. So kann man sicher sein, dass der Prozess auch auf einem anderen System funktioniert. Mit Prozess ist gemeint, dass das GitHub-Repository geklont wird und das Projekt und alle Docker-Container zur Ausführung gebracht werden können.

Jenkins verwendet bei uns keinen Docker für den Buildprozess. Dies geschieht direkt mit einer Shell Ausführung oder einem Plugin. Sobald der Build fehlschlägt (z.B. auch durch eine nicht bestandene Testmethode) bricht der Buildprozess sofort ab und der Build erscheint rot. So wird immer nur lauffähiger Code in einen Docker-Container gepackt.

Nachdem ohne Fehler gebuildet werden konnte, werden die gebuildeten Anwendungen in die Container jenkins-cas-client/-server gepackt und automatisch gestartet.

## 7.7.5 challenge-authoring-system

### 7.7.5.1 Build-Erstellung

Für die Build-Erstellung wird der Container build-cas-client verwendet. Dieser holt sich den aktuellen SourceCode aus dem lokalen GitHub-Repository per Volume-Mount. Im Anschluss wird diese ReactJS-Clientapplikation mit dem Node Package Manager gebaut. Weiter wird dafür Webpack eingesetzt. Somit werden beim Builden alle Dateien zu einigen wenigen zusammengefasst. Diese werden nach einem erfolgreichen Buildprozess in ein gemountetes Volume kopiert, welches ausserhalb des Containers im Ordner «build» am selben Ort wie das doit.sh des build-cas-client liegt. Somit ist an dem erwähnten Ort immer eine lauffähige Version der Applikation vorhanden.

### 7.7.5.2 Build-Publikation

Für die Publikation ist der Container run-cas-client verantwortlich. Wie schon oben erwähnt, liegt im Verzeichnis des build-cas-client im build-Ordner immer ein lauffähiges kompiliertes Programm. Sobald nun das doit.sh des run-cas-client ausgeführt wird, werden die kompilierten Daten in einen Unterordner «app» in dem Image-Ordner des run-cas-client kopiert. Wie schon erwähnt wird der Image Ordner, welcher auch die Zertifikate und die Node.js-Serverdatei enthält, via Dockerfile dem Container zur Verfügung gestellt (reinkopiert).

## 7.7.6 challengeAuthoringSystemServer

### 7.7.6.1 Build-Erstellung

Für die Build-Erstellung wird der Container build-cas-server verwendet. Dieser holt sich den aktuellen SourceCode aus dem lokalen GitHub-Repository per Volume-Mount. Im Anschluss wird diese Play Framework-Serverapplikation mit SBT (Scala Build Tools) gebaut. Damit gebaut werden kann, muss der cas-mysql Container in Ausführung sein. Denn während des Vorgangs werden die Tabellen usw. in der Datenbank erstellt, sofern diese noch nicht vorhanden sind. Nach dem Buildprozess ist eine ZIP-Datei vorhanden. Diese enthält alles was gebraucht wird, um die Serveranwendung zu starten. Selbst der Webserver (Akka) ist darin enthalten. Diese wird nach einem erfolgreichen Buildprozess in ein mit dem run-cas-server geteiltes, gemountetes Volume kopiert, welches ausserhalb des Containers im Ordner «build» am selben Ort wie das doit.sh des build-cas-server liegt. Somit ist an dem erwähnten Ort immer eine lauffähige Version der Applikation vorhanden.

### 7.7.6.2 Build-Publikation

Für die Publikation ist der Container run-cas-server verantwortlich. Wie schon oben erwähnt, liegt im Verzeichnis des build-cas-server im build-Ordner immer ein lauffähiges kompiliertes Programm in Form einer ZIP-Datei. Sobald nun das doit.sh des run-cas-server zur Ausführung kommt, wird der Container gestartet und aus dem geteilten Volume die ZIP-Datei in den Container entpackt. Anschliessend wird noch das Startskript, welches im ZIP vorhanden war gestartet.

## 7.8 Datenspeicherung

Die Daten werden auf einer serverseitigen MySQL Datenbank persistiert und über ein REST API für den Client zugänglich gemacht. Der heutige Stand ist wie weiter oben erwähnt mit einzelnen Repositories aufgebaut, welche ein bestimmtes Interface implementieren. Dieser modulare Aufbau hat den Vorteil, dass ein Repository sehr einfach durch eine alternative Implementierung ersetzt werden kann. So muss lediglich in einem Repository, welches das Interface implementiert eine andere Datenbank eingebunden werden und dieses dann im Code verwendet werden, ohne dass in der Verwendung eine Anpassung von Nöten ist. Die Repositories sind Thread sicher implementiert, da die Daten ansonsten in einen inkonsistenten Zustand geraten könnten, weil von mehreren Orten darauf zugegriffen wird.

## 7.9 REST API

Im folgender Tabelle sieht man, wie das REST API aufgebaut ist:

| Methode | URL | Link to |
|---|---|---|
| GET | /api | controllers.Assets.at(path="/public", file="index.html") |
| POST | /api/login | controllers.SecurityController.login |
| POST | /api/logout | controllers.SecurityController.logOUT |
| OPTIONS | /api/authoritation | controllers.SecurityController.options |
| POST | /api/authoritation | controllers.SecurityController.PROVEjwt |
| OPTIONS | /api/challenges | controllers.SecurityController.options |
| GET | /api/challenges | controllers.SecurityController.getChallenges |
| OPTIONS | /api/challenges/ | controllers.SecurityController.options |
| GET | /api/challenges/*filter | controllers.ChallengeController.getFilteredChallenges(filter) |
| OPTIONS | /api/challenge/createNewChallengeGUI | controllers.SecurityController.options |
| GET | /api/challenge/createNewChallengeGUI | controllers.ChallengeController.createNewChallengeGUI |
| OPTIONS | /api/challenge/createNewChallenge | controllers.SecurityController.options |
| GET | /api/challenge/createNewChallenge | controllers.ChallengeController.createNewChallenge |
| OPTIONS | /api/challenge/wsCreateNew | controllers.SecurityController.options |
| GET | /api/challenge/wsCreateNew/*queryString | controllers.WSChallengeController.ws(queryString) |
| OPTIONS | /api/challenge/modify | controllers.SecurityController.options |
| GET | /api/challenge/modify/:challengeId | controllers.ChallengeController.modifyChallenge(challengeId) |
| OPTIONS | /api/challenge/translate | controllers.SecurityController.options |
| GET | /api/challenge/translate/:challengeID | controllers.ChallengeController.modifyChallenge(challengeId) |
| OPTIONS | /api/challenge/rate | controllers.SecurityController.options |
| GET | /api/challenge/rate/:challengeId | controllers.ChallengeController.rateChallenge(challengeId, rateValue, description) |
| OPTIONS | /api/consumers | controllers.SecurityController.options |
| GET | /api/consumers | controllers.ConsumerController.getConsumer |
| OPTIONS | /api/consumer | controllers.SecurityController.options |
| GET | /api/consumer | controllers.ConsumerController.getSpecificConsumer |
| OPTIONS | /api/levels | controllers.SecurityController.options |
| GET | /api/levels | controllers.ChallengeController.getLevel |
| OPTIONS | /api/categories | controllers.SecurityController.options |
| GET | /api/categories | controllers.ChallengeController.getCategories |
| OPTIONS | /api/upload_image | controllers.SecurityController.options |
| GET | /api/upload_image | controllers.ChallengeController.upload(imageId) |
| GET | /api/assets/*file | controllers.Assets.versioned(path="/public", file: Asset) |

Tab. 5 REST-API

Das wichtigste hierbei ist, dass es für jede einzelne Route eine OPTIONS Route gibt, mit welcher die verschiedenen CORS Headers ausgehandelt werden. Jede Anfrage auf den Server wird immer zuerst mittels einer solchen OPTIONS Anfrage erfolgen. Wenn dieser nicht vorhanden ist, schlägt der Request aufgrund verletzter CORS Regeln fehl. Damit nicht für jeden Request eine eigene OPTIONS Route definiert werden muss, gibt es genau eine, welche im Security Controller implementiert wurde. Diese wurde in der heutigen Version nur sehr rudimentär definiert und muss für neue Anforderungen entsprechend angepasst werden.
Ein weiterer wichtiger Punkt ist, dass Play mit einem eingebauten Controller daherkommt, welcher die «Public Assets» liefert. Diese Public Assets sind verschiedenste Objekte, welche im public Ordner von Play vorhanden sind. Dazu gehören JavaScript Datei, Stylesheets, sowie Bilder, welche wie die anderen Request behandelt werden und an die anfordernden Clients geliefert werden. Da in diesem Projekt, jedoch der clientseitige Code nicht von Play zur Verfügung gestellt wird, sondern als eigenstehende Applikation implementiert wurde, haben sie hier eine eher nebensächliche Rolle.

## 7.10 Benutzeroberflächengestaltung

Da die Benutzeroberfläche mit React aufgebaut wurde, macht es an dieser Stelle wenig Sinn mit Screenshots zu arbeiten. Aus diesem Grund wird eine schematische Visualisierung des User Interface für die Challenge Erfassung verwendet, um die Gestaltung möglichst sauber auf einer Seite aufzeigen zu können. Danach folgt die erklärende Ergänzung dazu.

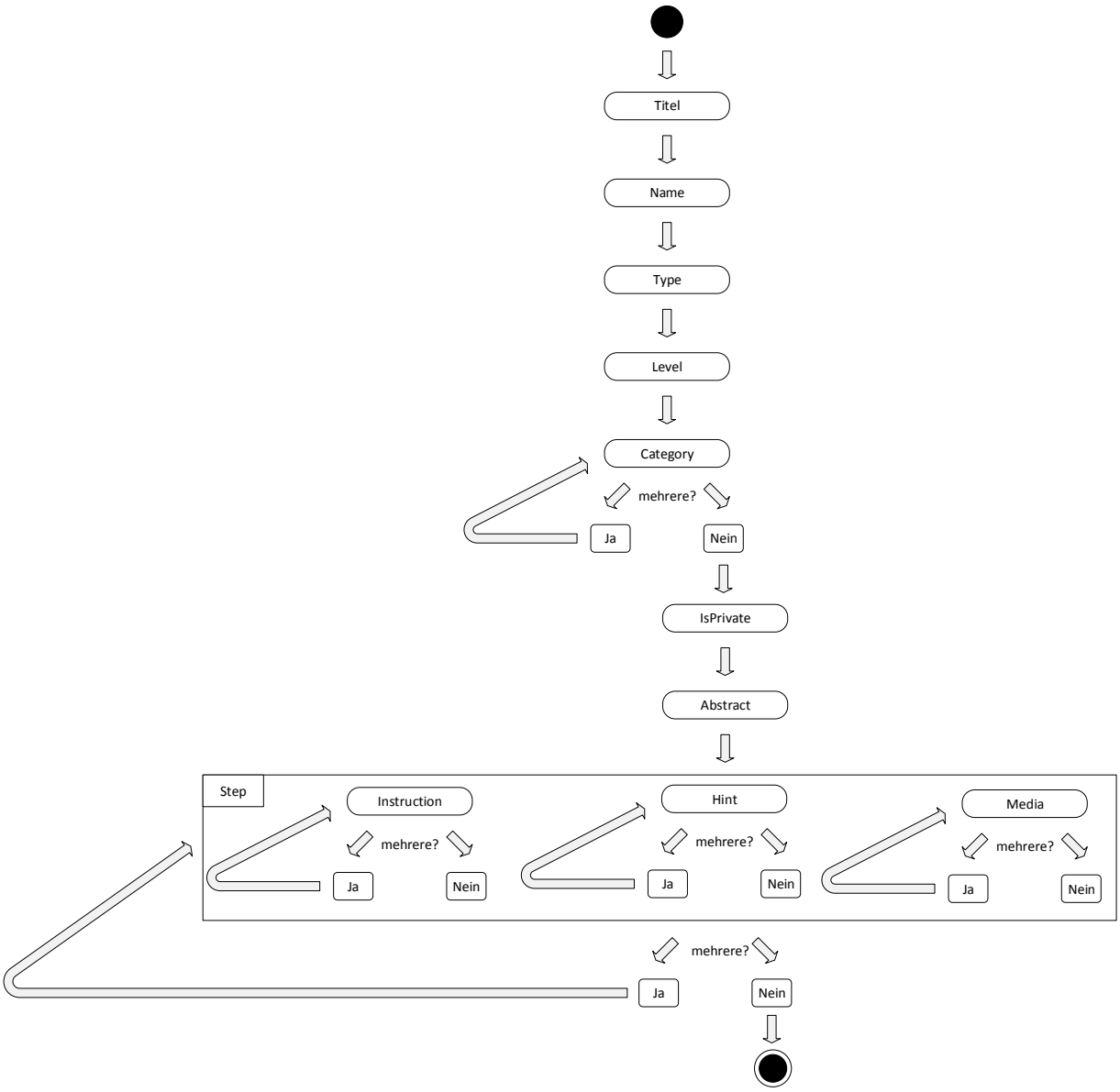



Abb. 27 schematische Darstellung der Benutzeroberfläche

Die ersten drei React Komponenten, welche für die Erfassung des Titels, des Namens und des Typen verwendet werden, sind einfache Eingabefelder. Man kann hier lediglich auf einer Zeile einen kleinen Text angeben, was jedoch in diesem Kontext genügen sollte.
Die vierte Komponente ist eine Eigenentwicklung eines Dropdown Controls. Über das REST API des Servers werden alle verfügbaren Levels geholt und hier zur Auswahl gegeben.
Die nächste Komponente ist wieder ein eigenentwickeltes Dropdown. Der Unterschied zu oben ist, dass man zusätzlich zu den vom Server gelieferten Kategorien eine neue erfassen kann, wenn die gewünschte nicht dabei ist. Diese wird dann sogleich auf dem Server angelegt und für die nächste Challenge Erfassung in der Auswahl angezeigt. Ein weiterer Unterschied zur simplen Dropdown-Auswahl für die Levels ist, dass man hier mehrere Kategorien auswählen können muss.

Die nächste Komponente ist eine gestylte Checkbox, welche als Toggle-Button dargestellt wird. Damit wird es einfach eine Challenge als privat zu kennzeichnen, damit diese nur auf dem aktuellen CAS vorhanden ist und nicht nach aussen zugänglich ist.
Die wichtigste Komponente ist die des Abstracts. Diese wird als Eigenentwicklung des Medium Editors dargestellt, wo neben Styling des Textes auch Bilder und Videos hochgeladen werden können. Es ist hier möglich einen grossen Fliesstext zu erfassen, welcher zuerst in Form von HTML Tags an den Server übermittelt wird und dort mittels eines Converters als Markdown gespeichert wird. Das Abstract fungiert als beschreibende Komponente der Challenge, wobei die nächste Komponente die eigentliche Aufgabenstellung beschreibt.
Wie bereits angetönt dient die Step Komponente als Teil der Aufgabenstellung. Es soll möglich sein mehrere solche Steps pro Challenge zu erfassen. Weiter ist angedacht, dass jeder Step mehrere Instructions, Hints und / oder Medien beinhaltet. Ein Step besitzt neben Instructions, Hints und Medien auch einen eigenen Text, welcher analog zum Abstract wie der Medium Editor verwendet wird. Medien können hierbei Bilder oder Filme sein. Eine Instruction umfasst mehrere Schritte, welche zum Lösungsweg führen. Anders werden die Hints aufgebaut, wo man lediglich einen kleinen Hinweis für das Finden der Lösung erhält. Falls ein Hint oder eine Instruction für das Lösen einer Challenge verwendet wird, werden dem Benutzer Punkte abgezogen, damit er nicht mehr die maximale Punktzahl für eine Challenge erhalten kann.

### 7.10.1 Tools

### 7.10.2 Entwicklungsumgebung

Sämtliche Teammitglieder verwenden IntelliJ IDEA 2017.2.15 Ultimate mit folgenden Plug-Ins:

| Plug-In | Einsatzgebiet |
|---|---|
| **Scala** | Scala Plug-In, welches für das Play Framework verwendet wird. Es stellt Unterstützung für Scala, sbt und Play Framework zur Verfügung |
| **JUnit** | Assistiert für die Navigation in und aus Testmethoden und Klassen und hilft Testmethoden zu erstellen und zu verwalten |
| **NodeJS** | Node.js Integration für die clientseitige Implementierung |
| **PlayframeworkSupport** | Unterstützung für die serverseitige Implementierung |
| **Git Integration** | Stellt die Integration mit der Git Versions Kontrolle zur Verfügung |
| **Guice** | Stellt die Integration mit dem Dependency Injection Tool von Google zur Verfügung |
| **Java EE** | Verschiedenste Java Plug-Ins, welche für Datenbankzugriffe, sowie Websockets verwendet werden |

Tab. 6 IntelliJ IDEA Plug-Ins

## 7.11 Test

Im Folgenden werden die verwendeten Testmethoden genauer unter die Lupe genommen. Die verschiedenen Tests sind an Barry Boehms V-Modell angelehnt, wobei zur besseren Benutzerakzeptanz auch noch ein Usability Test durchgeführt werden soll.

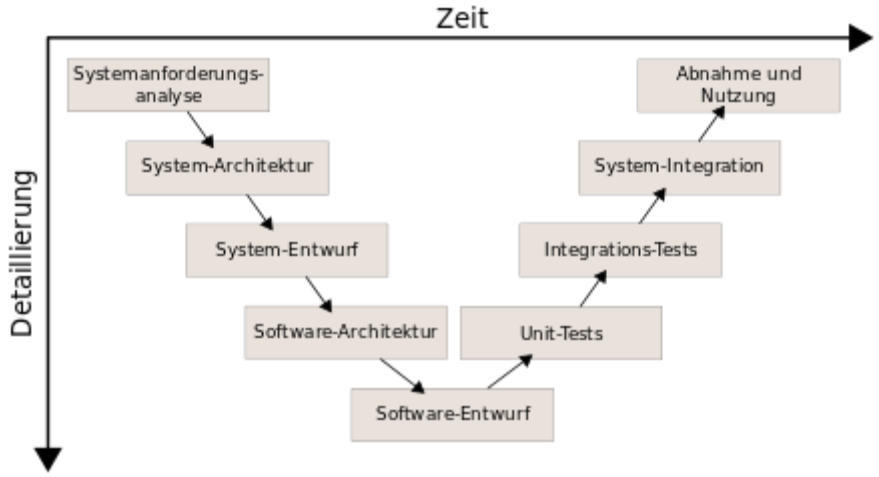



Abb. 28 V-Modell [13]

### 7.11.1 Unit Test

Die Unit Tests beschränken sich zurzeit auf Kernfunktionalitäten, welche für die Applikation entscheidend sind. Deshalb wurden vor allem die Utils überprüft. Aufgrund der verwendeten Architektur ist es jedoch sehr einfach möglich, alle verwendeten Komponenten zu testen. Sinnvollerweise könnte man sich dabei auf die Service Implementierungen beschränken, da hier die wichtigsten Funktionalitäten der Applikation abgebildet wurden. Das Play Framework stellt jedoch einfache Mechanismen zur Verfügung, um auch die Controller einfach testen zu können. Es wird dazu eine Applikation gemockt, an welche Request verschickt werden können und dann mittels Assertions die zurückgelieferten Results überprüfen kann.

### 7.11.2 Usability Test

Es wurden ganz rudimentäre Usability Tests durchgeführt, indem die Erstellung der Challenge durchgespielt wurde. Aufgrund fehlender Zeit, konnten diese jedoch nur mit den beiden Entwicklern und nicht mit einem grösseren unabhängigen Publikum durchgeführt werden, was die Aussagekräftigkeit dieser in Frage stellt. Wenn die Applikation später einer grösseren Community zur Verfügung steht, werden die Rückmeldungen der Anwender sicher dazu beitragen, dass die Bedienbarkeit immer wieder verbessert und sich so die bestmögliche Variante herauskristallisiert.

### 7.11.3 System- und Integrationstest

Damit die Applikation innerhalb der Docker-Container garantiert korrekt gebaut und lauffähig ist, wurden mehrere Integrationstests durchgeführt, in welchen das Zusammenspiel aller Komponenten durchgespielt wurde. Weiter wurden diese dann auf unterschiedlichen Systemen wie Mac, Linux und Windows ausgeführt, um ausschliessen zu können, dass auf einem dieser irgendwelche Probleme auftreten können. So wurde schnell klar, dass auf dem Mac noch zusätzliche Konfigurationen notwendig sind, damit alle Komponenten korrekt miteinander interagieren können. So müssen z.B. die Volume-Mounts speziell eingerichtet/autorisiert werden. Um ausschliessen zu können, dass die Docker-Container auf einem System nur wegen globalen Installationen von Abhängigkeiten korrekt funktionieren, wurde mehrmals auf einem neuen Server auf der grünen Wiese begonnen, da dies zu Beginn Probleme bereitet hat.

# 8 Verzeichnisse

## 8.1 Abbildungsverzeichnis



## 8.2 Tabellenverzeichnis

## 8.3 Quellverzeichnis

[1] S. C. GmbH, «Hacking-Lab 1.0,» Security Competence GmbH, [Online]. Available: Security Competence GmbH. [Zugriff am 2017].

[2] «Compass Security,» [Online]. Available: https://www.compass-security.com/. [Zugriff am 2017].

[3] «LiveCD Release 10-15,» Security Competence GmbH, [Online]. Available: https://media.hacking-lab.com/. [Zugriff am 2017].

[4] «Projekt Goldener Schild - Wikipedia,» [Online]. Available: https://de.wikipedia.org/wiki/Projekt_Goldener_Schild#Unterwanderung. [Zugriff am 18 12 2017].

[5] «restdb.io,» [Online]. Available: https://restdb.io/. [Zugriff am 2017].

[6] M. Fowler, «steps toward the glory of REST,» [Online]. Available: https://martinfowler.com/articles/richardsonMaturityModel.html. [Zugriff am 2017].

[7] Github, «GitHub Help - About issues,» [Online]. Available: https://help.github.com/articles/about-issues/. [Zugriff am 2017].

[8] «JWT - Introduction to JSON Web Tokens,» [Online]. Available: https://jwt.io/introduction/. [Zugriff am 2017].

[9] Medium, «Welcome to Medium, where words matter.,» [Online]. Available: https://medium.com/about. [Zugriff am 2017].

[10] M. Fowler, «Microservices,» [Online]. Available: https://martinfowler.com/articles/microservices.html. [Zugriff am 2017].

[11] mysql, «mysql - Official Docker Repository,» [Online]. Available: https://hub.docker.com/_/mysql/. [Zugriff am 2017].

[12] Play, «Play 2.6.x documentation,» [Online]. Available: https://www.playframework.com/documentation/2.6.x/Home. [Zugriff am 2017].

[13] «webpack - Production,» [Online]. Available: https://webpack.js.org/guides/production/. [Zugriff am 2017].

[14] «trainingdevops - Continuous Integration with Jenkins and Docker,» [Online]. Available: https://www.trainingdevops.com/training-material/advance-docker-training/using-jenkins-with-docker-container. [Zugriff am 2017].

[15] «Traefik,» [Online]. Available: https://traefik.io/. [Zugriff am 2017].

[16] «thegeekstuff - Bg, Fg, &, Ctrl-Z – 5 Examples to Manage Unix Background Jobs,» [Online]. Available: http://www.thegeekstuff.com/2010/05/unix-background-job/. [Zugriff am 2017].

[17] «stackoverflow - Repack of Git repository fails,» [Online]. Available: https://stackoverflow.com/questions/4826639/repack-of-git-repository-fails. [Zugriff am 2017].

[18] «stackoverflow - how to delete all commit history in github?,» [Online]. Available: https://stackoverflow.com/questions/13716658/how-to-delete-all-commit-history-in-github. [Zugriff am 2017].

[19] «stackoverflow - How to create an HTTPS server in Node.js?,» [Online]. Available: https://stackoverflow.com/questions/5998694/how-to-create-an-https-server-in-node-js. [Zugriff am 2017].

[20] «stackoverflow - How can I clean my .git folder? Cleaned up my project directory, but .git is still massive,» [Online]. Available: https://stackoverflow.com/questions/5277467/how-can-i-clean-my-git-folder-cleaned-up-my-project-directory-but-git-is-sti. [Zugriff am 2017].

[21] «severalnines - MySQL Docker Containers: Understanding the basics,» [Online]. Available: https://severalnines.com/blog/mysql-docker-containers-understanding-basics. [Zugriff am 2017].

[22] «Restdb.io,» [Online]. Available: https://restdb.io/. [Zugriff am 2017].

[23] «Redmine,» [Online]. Available: https://www.redmine.org/. [Zugriff am 2017].

[24] «Play! Framework,» [Online]. Available: https://www.playframework.com/. [Zugriff am 2017].

[25] «npm - dotenv-webpack,» [Online]. Available: https://www.npmjs.com/package/dotenv-webpack. [Zugriff am 2017].

[26] «MySQL - Dockercontainer,» [Online]. Available: https://hub.docker.com/_/mysql/. [Zugriff am 2017].

[27] «Medium - Triggering a Jenkins build from a push to Github,» [Online]. Available: https://medium.com/@marc_best/trigger-a-jenkins-build-from-a-github-push-b922468ef1ae. [Zugriff am 2017].

[28] «Jenkins.io - sbt plugin,» [Online]. Available: https://wiki.jenkins.io/display/JENKINS/sbt+plugin.

[29] «jansipke.nl - Enable HTTPS on Jenkins,» [Online]. Available: https://jansipke.nl/enable-https-jenkins/. [Zugriff am 2017].

[30] «GitHub Help - Generating a new SSH key and adding it to the ssh-agent,» [Online]. Available: https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/#platform-linux. [Zugriff am 2017].

[31] «GitHub,» [Online]. Available: https://github.com/. [Zugriff am 2017].

[32] «docker,» [Online]. Available: https://www.docker.com/. [Zugriff am 2017].

[33] «bitnami Docs - How To Start The Stack Automatically On Boot?,» [Online]. Available: https://docs.bitnami.com/installer/faq/linux-faq/#how-to-start-the-stack-automatically-on-boot. [Zugriff am 2017].

[34] «bitnami - Redmine,» [Online]. Available: https://bitnami.com/stack/redmine. [Zugriff am 2017].