Bachelorarbeit

HSR Live Survey

Frühjahressemester 2019

Autoren
Marcel Keller
Quoc Tin Tran

Betreuer
Manuel Bauer

Experte
Pascal Schäfer

Gegenleser
Prof. Beat Stettler

HSR Hochschule für Technik Rapperswil
Abteilung Informatik
CH-8640 Rapperswil, Schweiz

Abstract

Aus pädagogischer Sicht helfen Repetitionen von Lerninhalten das Gelernte zu verankern. Solche Wiederholungen während der Unterrichtssequenz können die Studierenden mental aktivieren und dazu animieren, sich rege am Unterricht zu beteiligen und bieten zusätzlich Abwechslung zum üblichen Unterricht. Im Rahmen der Bachelorarbeit soll eine Plattform entwickelt werden, welches den Dozierenden erlaubt, eine Reihe von Fragen zu verfassen und zu präsentieren. Studierende als Teilnehmer sollen einfach über ihr Smartphone oder Notebook die gestellten Fragen beantworten können. Die abgegebenen Antworten sollten in Echtzeit aktualisiert und angezeigt werden.

Beim Start der Bachelorarbeit mussten zuerst die Anforderungen an eine solche Plattform erhoben werden. Die Anforderungsanalyse resultiert zum einen darin, verschiedene Technologien zu evaluieren, da nur geringe Einschränkungen bestanden. Zum anderen war parallel die Konzipierung der Microservice-Architektur ein wichtiger Schritt zum Endprodukt. Die User Interfaces wurden als Single Page Applications in React umgesetzt. Im Backend agieren mehrere Microservices als ASP.NET Core Web APIs oder als SignalR Hubs, um die jeweiligen Anfragen der Webanwendungen zu behandeln. Die ermittelten Anforderungen wurden während des Projekts mit einem agilen Ansatz umgesetzt, wobei fortlaufend Feedback vom Auftraggeber in die Umsetzung miteingeflossen ist.

Es ist eine Plattform mit drei Webanwendungen entstanden. Mit der ersten App (Presentation Designer) lassen sich neue Präsentationen, Slides und Fragen erstellen und verwalten. Um solche Präsentationen durchzuführen, existiert eine weitere App (Presenter Client) für den Dozierenden, welcher darüber die Durchführung steuern kann. Damit Studierende die angezeigten Fragen beim Presenter Client beantworten können, ist dafür die dritte Anwendung (Voter Client) verantwortlich. Über einen einfachen Zugangscode ist es möglich, an der Durchführung teilzunehmen und Antworten abzugeben.

Management Summary

Ausgangslage

In der Pädagogik und Didaktik stellt sich insbesondere im Bereich Wissensvermittlung die Frage, wie man den Unterricht dynamischer und interessanter gestalten kann. Ausserdem ist die Verankerung des behandelten Lernstoffes auf allen Schulstufen inklusive Teritiärstufe wie der Fachhochschule ein zentraler Punkt in der Wissensvermittlung. Klassisch in den Vorlesungen könnten Repetitionsfragen in die bereits meist vorhandenen PowerPoint-Präsentationen eingebaut oder schon vorhandene Plattformen genutzt werden. Ersteres ist unkomfortabel zu handhaben und letzteres ist oft sehr teuer, so dass Lizenzen pro Nutzer bzw. Lehrbeauftragten erworben werden müssen. Aus diesem Grund soll eine Plattform erstellt werden, auf der man einfach neue Fragen erfassen und von den Studierenden beantworten lassen kann. Ergebnisse sollen in Echtzeit angezeigt werden, so dass Rückschlüsse auf den Lernerfolg seitens Lehrkraft möglich sind.

Vorgehen und Technologien

Bei der Vorgehensweise wurde ein agiler und iterativer Ansatz in Kombination mit einem weiteren Modell gewählt, welches Züge vom Wasserfallmodell aufzeigt. Es existieren die vier Phasen Inception, Elaboration, Construction und Transition. Jede Phase wurde jeweils in zweiwöchige Iterationen unterteilt. In der Elaboration-Phase wurden Analysen durchgeführt, so dass aufgrund der resultierenden Anforderungen die Evaluationen zu Technologien und Systemarchitekturen in Absprache mit dem Auftraggeber vollzogen werden konnten. Um einem grossen Andrang auf die Plattform standzuhalten, wurde die Plattform in der Construction-Phase so entwickelt, dass sie sich bei Bedarf der Last anpassen kann. In der letzten Phase Transition wurden alle Dokumente für die Abgabe erstellt und die Übergabe des Endprodukts an den Auftraggeber vorbereitet.

Ergebnisse

Entwickelt wurde eine Plattform mit drei Webapplikationen und mehreren Services. Mit einer Applikation lassen sich Präsentationen, Slides und Fragen erfassen und bearbeiten. Die erstellen Präsentationen bilden die Basis für die Beantwortung der Fragen, denn eine zweite Anwendung dient dazu, solche Präsentationen durchzuführen und Resultate anzuzeigen. Mit der letzten Applikation lassen sich die gestellten Fragen über jedes internetfähige Gerät beantworten.

Inhaltsverzeichnis

Teil I.

Technischer Bericht

1. Einleitung und Übersicht

1.1. Problemstellung

In der Schule und im Studium wird in kurzer Zeit viel Wissen vermittelt. Um den Lerninhalt schneller und besser beherrschen zu können, bieten sich Rekapitulierungen während dem Unterricht an. Diese sollen helfen das vermittelte Wissen zu festigen und die Unterrichtssequenz lebhafter und interessanter zu gestalten. Oft sind selber erstellte Fragesequenzen mühsam zu erstellen und bieten keine einfache Möglichkeit, Ergebnisse gebündelt anzuzeigen. Zwar existieren bereits Anwendungen, welche die erwähnten Punkte erfüllen, diese kosten oft sehr viel, da pro Nutzer bzw. Lehrkraft Lizenzen erworben werden müssen.

1.2. HSR Live Survey

Die Plattform HSR Live Survey soll es Dozierenden erlauben, alle verfassten Fragen zentral zu verwalten und diese im Unterricht live zu präsentieren. Logisch gesehen soll eine Frage genau einem Slide zugeordnet werden können, so dass ein Slide wie als Container für eine Frage fungiert. Übergeordnet sollen die Slides und Fragen zu einer Präsentation zusammengefasst werden. Durch die Trennung von Präsentationsdefinition- und durchführung, soll es möglich sein die gleiche Präsentation mehrmals durchzuführen.

Um während den Rekapitulierungen Einschränkungen zu verhindern und Abwechslung zu bieten, sollten nach Möglichkeit die gängigsten Arten von Fragen zur Verfügung stehen. Weitere Fragetypen sollten einfach hinzugefügt werden können. Neben dem Aspekt der Erweiterbarkeit ist auch die Skalierbarkeit wichtig. Aus diesem Grund sollten sich die einzelnen Teile des System einzeln deployen und skalieren lassen, was im Endeffekt einer Microservice-Architektur entspricht. In Abbildung 1 ist das grobe Konzept des Systems aufgezeigt. Darin zu sehen sind ein Presenter, mehrere Voters und Services sowie vier Webanwendungen, wobei der Presenter Remote als optional angesehen werden kann und deswegen schwarz gefärbt ist.

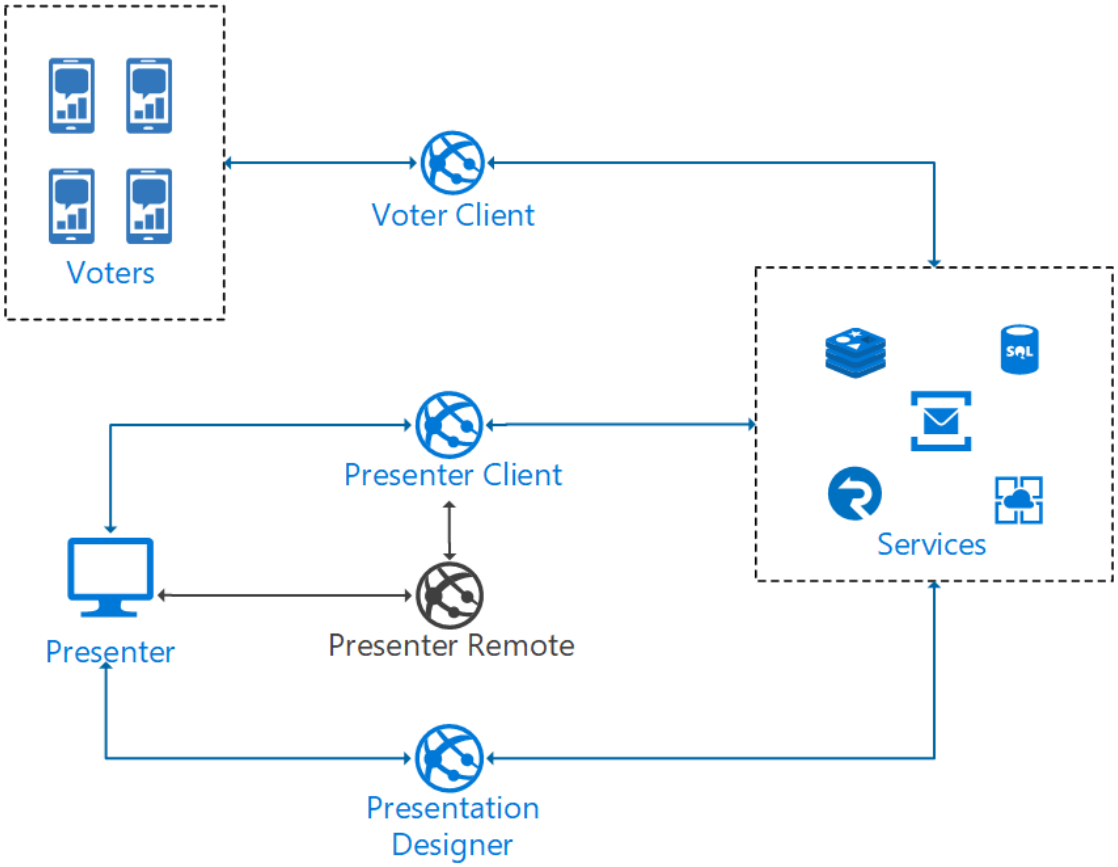


Abbildung 1.: Konzept
Quelle: Eigene Darstellung

2. Vorgehen und Technologien

2.1. Vorgehensmodell

In der Bachelorarbeit kam ein gemischten Vorgehensmodell zum Einsatz, welches sich Scrum++ nennt und eine Kombination aus Scrum und dem Rational Unified Process darstellt. Durch Letzteres ergibt sich eine grobe vorgegebene Struktur aus den vier Phasen *Inception, Elaboration, Construction* und *Transition*. Durch den Punkt *End of Elaboration* wird sichergestellt, dass vor der Construction-Phase sämtliche Analysen und Abklärungen bezüglich Anforderungen und Architektur einen Stand erreichen, dass mit der Entwicklung reibungslos gestartet werden kann. Um eine grosse Flexibilität beizubehalten, wird während der gesamten Projektzeit Scrum eingesetzt.

2.2. Zeitliches Vorgehen

Für die Bachelorarbeit standen 15 Semesterwoche zu je 20 Stunden pro Person und am Schluss zwei Wochen zu je 30 Stunden pro Person zur Verfügung. Mit Scrum++ wurden zweiwöchige Sprints definiert und entsprechende Meilensteine jeweils am Ende eines Sprints angesetzt, um zum Start des Projektes bereits grobe Ziele zu definieren. Weitere Informationen sind im Projektplan unter Kapitel 1 zu finden.

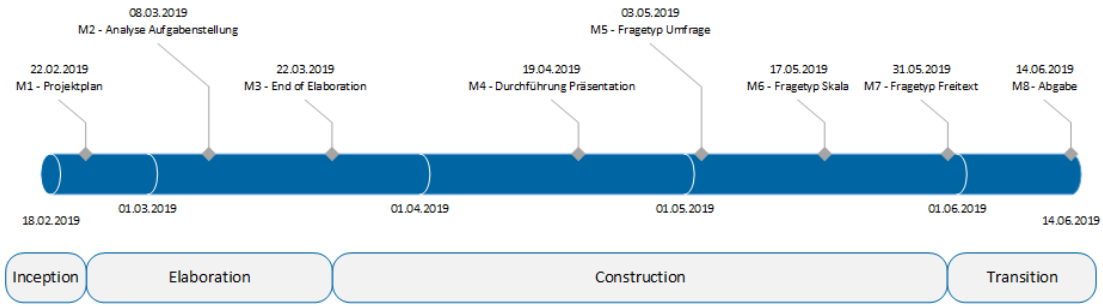


Abbildung 2.: Übersicht zeitliches Vorgehen
Quelle: Eigene Darstellung

2.3. Technologien

Es wurden verschiedene Technologien eingesetzt, die insbesondere zu Beginn des Projektes ausgiebig analysiert und evaluiert wurden.

2.3.1. Frontend

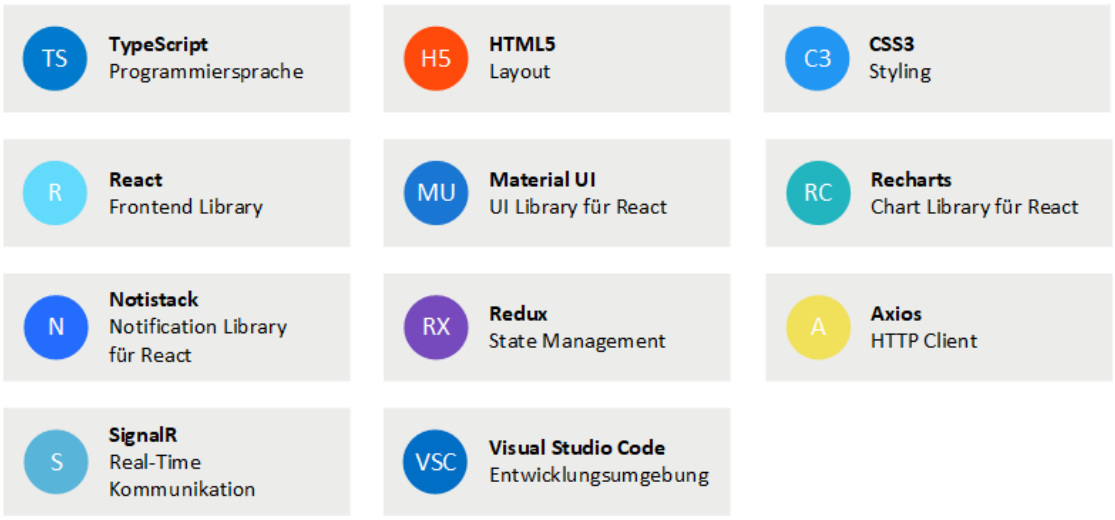


Abbildung 3.: Frontend-Technologien
Quelle: Eigene Darstellung

2.3.2. Backend

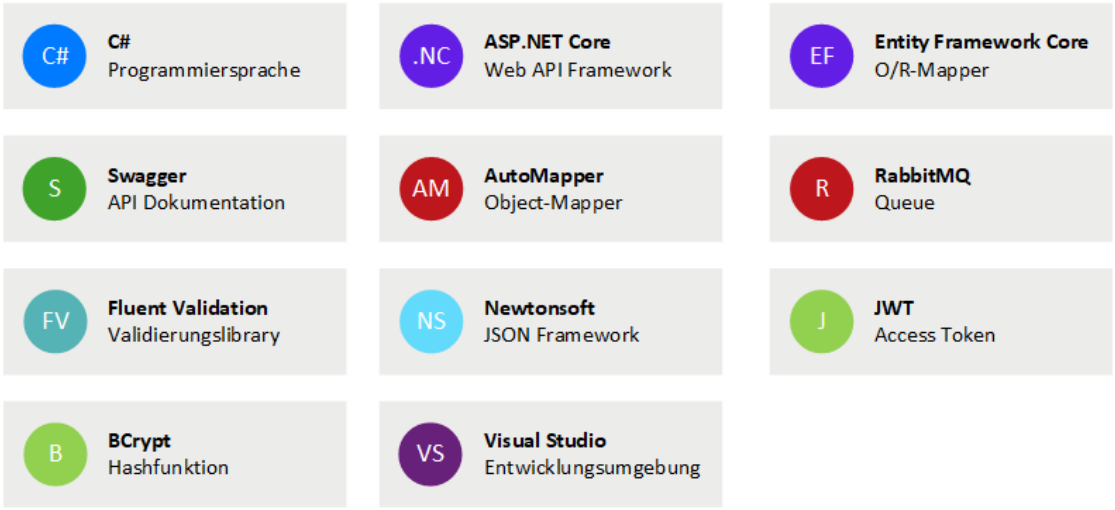


Abbildung 4.: Backend-Technologien
Quelle: Eigene Darstellung

2.3.3. Infrastruktur

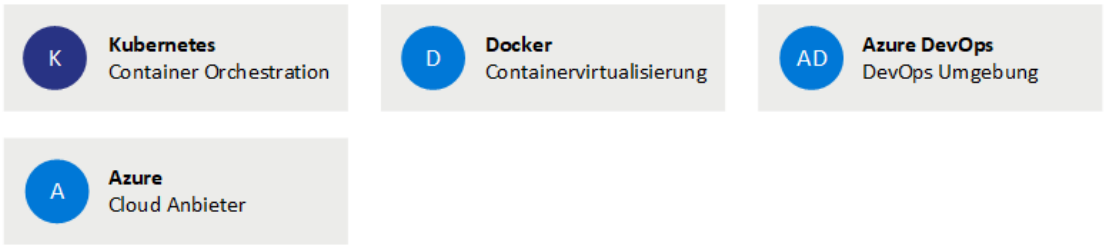


Abbildung 5.: Infrastruktur-Technologien
Quelle: Eigene Darstellung

3. Ergebnisse

3.1. Frontend

Für die Benutzerinteraktion wurden drei Webapplikationen mit React und TypeScript entwickelt.

Presentation Designer Bei dieser Webanwendung handelt es sich um ein Frontend, das als Erfassungsmaske dient. Im Vergleich mit den anderen beiden Anwendungen enthält der Presentation Designer viel Logik, da die Erfassungs- und Bearbeitungsmöglichkeiten sehr ausgeprägt sind.
Im Presentation Designer lassen sich konkret folgende Tätigkeiten ausüben:

- Registration und Login
- Eigenes Benutzerprofil verwalten
- Präsentationen erstellen, bearbeiten und löschen
- Slides erstellen, bearbeiten und entfernen
- Fragen erfassen, bearbeiten und löschen
- Auswahlmöglichkeiten definieren, anpassen und entfernen

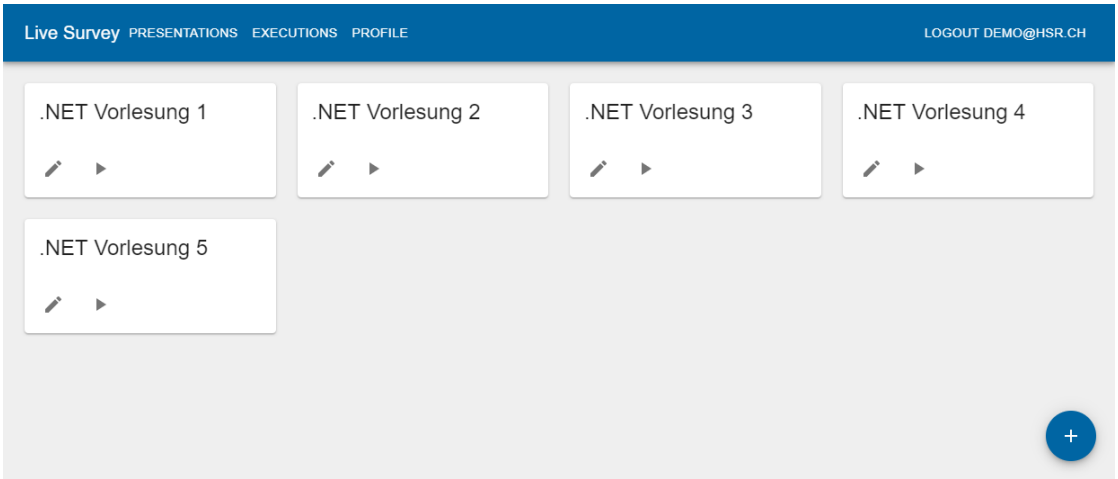


Abbildung 6.: Präsentationsübersicht im Presentation Designer
Quelle: Eigene Darstellung

Presenter Client Der Presenter Client ist für die Präsentationsdurchführung zuständig und dient als Anzeige der Antwortresultate. Es werden nur minimale Benutzereingaben in Form von Buttons entgegengenommen um die Präsentationsdurchführung zu steuern. Ausserdem werden die Anzahl abgegebenen Antworten angezeigt, sowie die noch zur Verfügung stehende Antwortzeit, falls eine definiert wurde. Für die Echtzeitkommunikation wird SignalR[1] genutzt. Die Daten werden als Chart mit der Library Recharts[2] generiert. Folgende Steuerungselemente sind verfügbar:

- Präsentationdurchführung starten
- Nächste Frage anzeigen
- Vorherige Frage anzeigen
- Antwortabgabe bei allen Teilnehmern deaktivieren/aktivieren
- Fullscreen-Anzeige
- Richtige Resultate anzeigen, wenn dies der Fragetyp erlaubt

[1] `https://dotnet.microsoft.com/apps/aspnet/real-time`
[2] `http://recharts.org/`

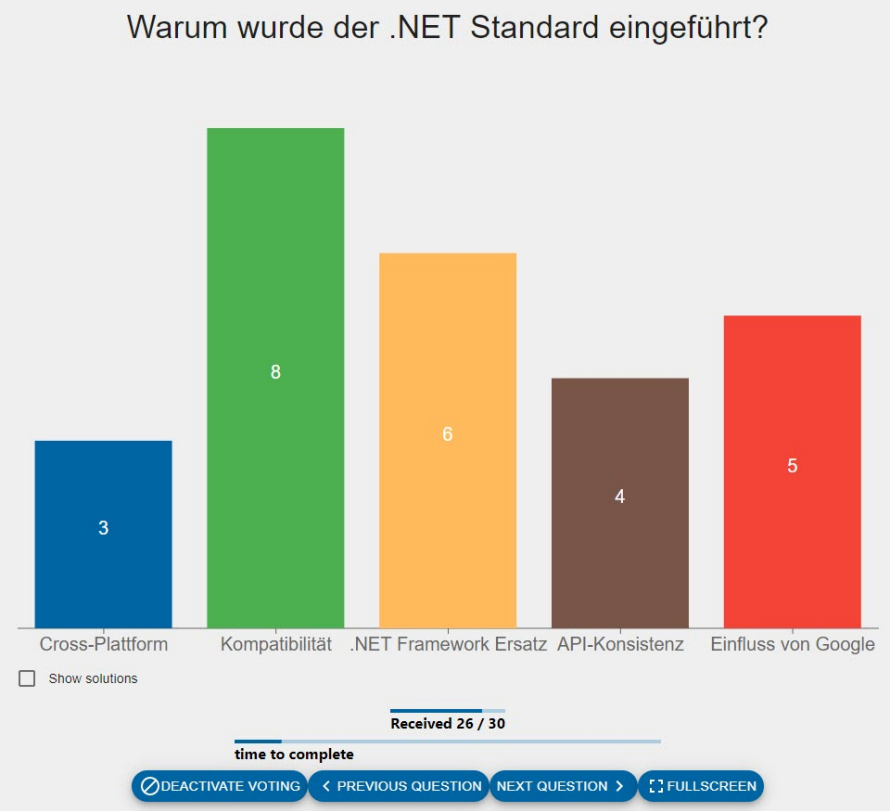


Abbildung 7.: Resultatansicht eines Quiz' im Presenter Client
Quelle: Eigene Darstellung

Voter Client Für die Teilnehmenden steht der Voter Client als Schnittstelle zur Verfügung. Dieser zeigt die Fragen und die entsprechenden Auswahlmöglichkeiten an. Falls eine Antwortzeit definiert wurde, ist diese ebenfalls in Form eines Fortschrittsbalken ersichtlich. Auch beim Voter Client ist eine Echtzeitkommunikation nötig und deswegen wird hier ebenfalls SignalR[3] eingesetzt.
Über den Voter Client kann der Benutzer folgende Aktionen durchführen:

- Zugangscode für eine Präsentationsdurchführung eingeben
- Antworten abgeben

[3] `https://dotnet.microsoft.com/apps/aspnet/real-time`

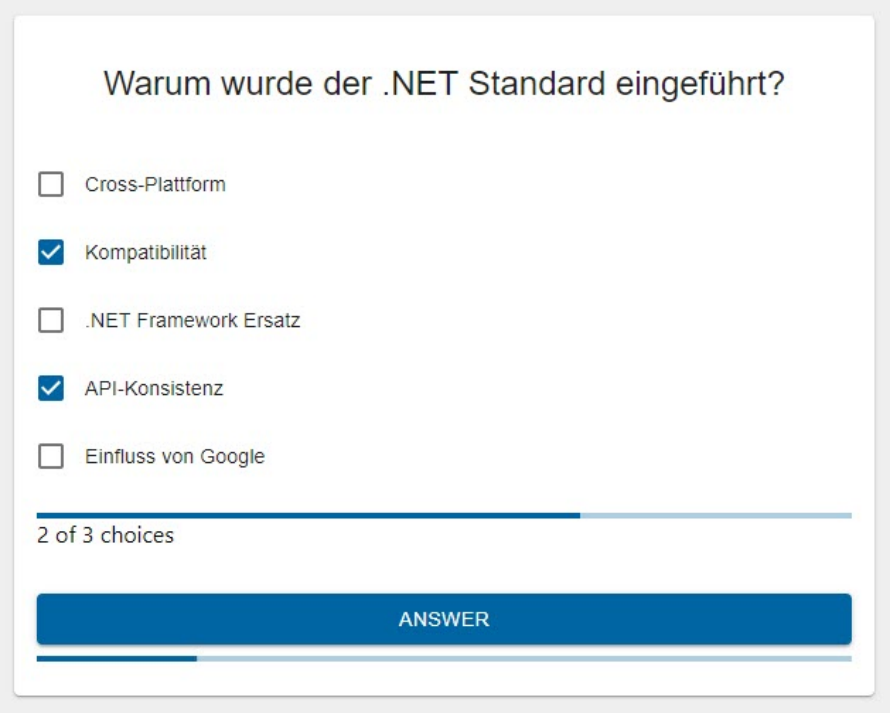


Abbildung 8.: Antwortansicht eines Quiz' im Voter Client
Quelle: Eigene Darstellung

3.2. Backend

Das Backend besteht aus mehreren Microservices, die sich zusammen mit den drei Frontends in einem Kubernetes Cluster befinden. Jeder Service ist für einen bestimmte Teilbereich der Aufgabenstellung verantwortlich. Grob lassen sich zwei Typen von Services unterscheiden. Zum einen gibt es RESTful Web APIs, welche über HTTP angesprochen werden können. Als Datenformat wird das im Web verbreitete Format JSON eingesetzt. Zum anderen existieren SignalR-Hubs, die für die Echtzeitkommunikation zuständig sind und über Websockets-Verbindungen mit den Frontends kommunizieren. Durch das Einhalten des Single Responsibility Principle ist es möglich, Änderungen im Normalfall nur an einem Service durchführen zu müssen.

Datastore In einer "klassischen" Microservices-Architektur besitzt jeder Service seinen eigenen Datastore. Bei Änderungen werden die jeweiligen Services, die daran interessiert sind, benachrichtigt. Da dies den Umfang der Projektarbeit gesprengt hätte, besitzt zwar jeder Service ebenfalls einen eigenen Datastore, aber nur in Form von privaten Tabellen.

3.3. Erweiterbarkeit

Für den Fall, das neue Fragetypen hinzukommen, ist das System so aufgebaut, dass Änderungen an den Services und Frontends möglichst nur an einem Ort stattfinden müssen.

3.4. Skalierbarkeit

Durch die Aufteilung in mehrere Komponenten ist jeder Service oder auch jede Webapp individuell skalierbar, so dass auf sich ändernde Systemumstände schnell und gezielt reagiert werden kann.

4. Zeitauswertung

Das Resultat der Bachelorarbeit wurde in ingesamt 720 Arbeitsstunden erreicht, was genau der Vorgabe entspricht.

4.1. Aufteilung Stunden nach Teammitglied

Die Aufteilung der Total Stunden von 720 auf die Teammitglieder beträgt in etwa 50%/50%. Dem Zugrunde liegt, dass wir praktisch ausschliesslich an den gleichen Tagen zusammen gearbeitet haben. Ebenso wurden gegenseitige Unterstützungen geboten, falls ein Teammitglied bei einer Arbeit länger als gedacht hatte oder angestanden ist.

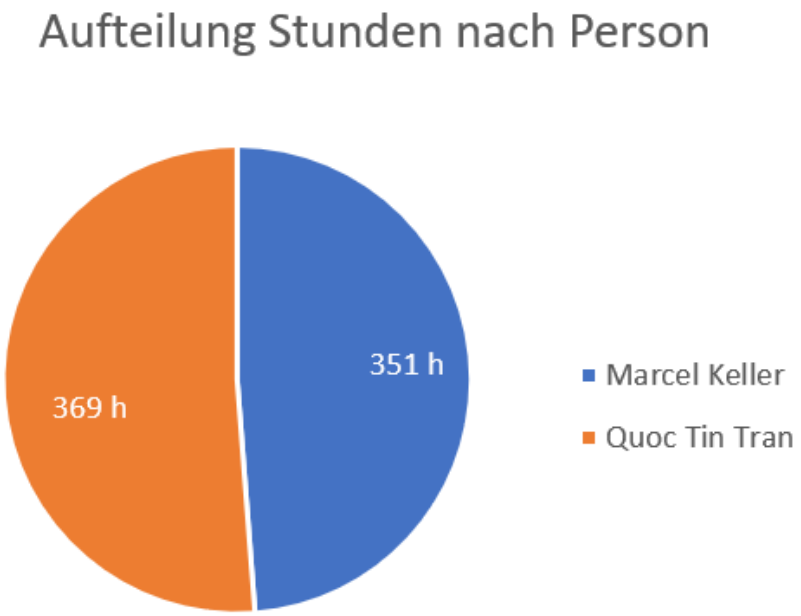


Abbildung 9.: Aufteilung Stunden nach Person
Quelle: Eigene Darstellung

4.2. Aufteilung Stunden nach Aktivität

Die Aufteilung der Total Stunden nach Aktivität kommt wie erwartet mit einem grossen Anteil von Entwicklung daher. Herausstechend ist, dass der Anteil der Infrastruktur sehr klein ist. In Realität war der Aufwand für die Infrastruktur höher, diese Arbeiten wurden aber direkt im Rahmen der Entwicklung erledigt und auch so verbucht.

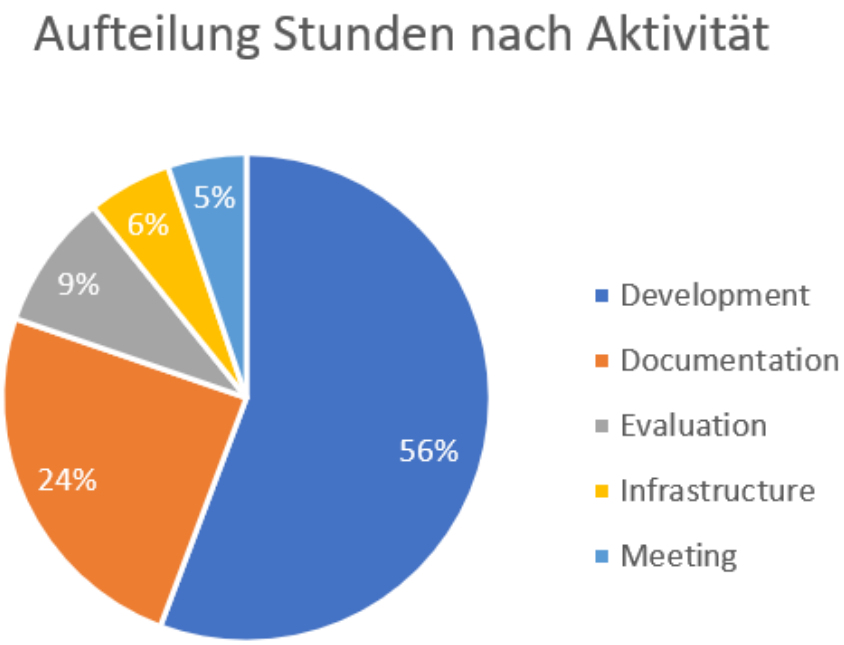


Abbildung 10.: Aufteilung Stunden nach Aktivität
Quelle: Eigene Darstellung

5. Schlussfolgerung

Es wurde ein solides System in den 17 Wochen entwickelt, welches bereits einen Einsatz im Unterricht erlaubt. Die zwingenden funktionalen Anforderungen wurden mit einigen Ausnahmen (siehe Kapitel 5.2) erfüllt. In dieser Zeit konnte der Fokus nicht auf weitere Funktionen gelegt werden, da die Arbeiten im Bereich DevOps sehr zeitintensiv waren. Es wurden deswegen in Absprache mit dem Auftraggeber keine optionalen Funktionalitäten umgesetzt.

5.1. Umgesetzte Arbeiten

Eigene Benutzeradministration (UC01 - 04) Auf der Plattform ist es möglich sich selbständig zu registrieren und einzuloggen. Die angegebenen Daten bei der Registrierung können im Nachhinein geändert werden.

Präsentationserstellung (UC05 - 10) Präsentationen lassen sich nach dem Login erstellen und wieder bearbeiten. Zum Bearbeiten gehört auch, dass man Slides und die dazugehörigen Fragen anlegen und wieder editieren kann. Dabei sind verschiedene Fragetypen verfügbar, welche der Benutzer vor dem Anlegen frei auswählen kann.

Präsentationsdurchführung (UC11) Um die erstellten Fragen von Teilnehmenden beantworten zu lassen, kann der Benutzer eine vorher angelegte Präsentation mehrmals durchführen. Bereits durchgeführte Präsentation lassen sich erneut inklusive bestehender Resultate anschauen.

Teilnahme an einer Präsentation (UC12 - 13) Teilnehmende können über einen automatisch generierten Zugangscode an einer bestimmten Präsentationsdurchführung teilnehmen und die angezeigten Fragen beantworten.

5.2. Nicht umgesetzte Arbeiten

Präsentation bearbeiten Im Use Case 06 wurde die Funktion, die Slidereihenfolge zu ändern, aufgrund Zeitmangels nicht umgesetzt.

Präsentation bearbeiten Beim Use Case 11 wurde keine Funktion implementiert, welche es erlaubt, eine Durchführung zu beenden. Der Grund dafür ist, dass durch ein Schliessen des Browsers das gleiche Ziel erreicht werden kann.

Präsentationsexport Die Exportfunktion wurde von den optionalen Anforderungen als höchstes priorisiert, aber nicht umgesetzt, da zuerst die zwingenden Anforderungen erfüllt werden mussten.

Teilen von Präsentationen Das Teilen von Präsentationen stellt eines von den wichtigeren Funktionen dar bei den optionalen Anforderungen, wurde ebenfalls aufgrund mangelnder Zeit nicht umgesetzt.

Presenter Remote Eine weitere Applikation, um die Präsentation von Extern zu steuern wurde aufgrund der Priorität und Zeitmangels nicht umgesetzt.

Individuelle Slides Die Anpassung der Slidedesigns ist eine Funktion, die noch mehr Individualität pro Präsentation miteinbringen kann. Dies wurde in Absprache mit dem Auftraggeber nicht prioritär behandelt.

5.3. Erfahrungen

5.3.1. Microservices

Mit dem Projekt HSR Live Survey wurde zum ersten Mal für das Team eine Microservices-Architektur entworfen und implementiert. Dies war durchgehend eine grosse Herausforderung. Durch die geringe Erfahrung wurde viel Zeit in der Architekturphase verwendet, da keine bereits durchgeführten Projekte als Ansatz oder Hilfsmittel verwendet werden konnten. Auch während der Implementation war der Zeitaufwand für das Einrichten der Infrastruktur und der Prozesse für Continuous Integration und Deployment enorm hoch verglichen mit bekannten, ähnlichen Projekten.

Als grösste Herausforderung während der gesamten Projektzeit ist die Aufteilung der einzelnen Services anzusehen. Diese Aufteilung konnte gemäss bekannten Patterns auf zwei Wege geschehen (siehe Kapitel 4.11.1), doch beide wurden nicht komplett korrekt angesehen. Diese Aufteilung führte zu vielen Diskussionen und Ungewissheit, da keine Erfahrung vorhanden war. Im Endprodukt der Arbeit sind einige Services vorhanden. Diese Services kommunizieren oft untereinander, was zu einer hohen Kopplung führt.

Diese hohe Kopplung möchte eine Microservices-Architektur eigentlich vermeiden.

Ein grosser Vorteil von Microservices sind die kleinen, sehr übersichtlichen Services. Neue Features oder Anpassungen konnten so ohne grosse Sorgen über Nebenwirkungen oder grosses Testing anderer Komponenten durchgeführt und gezielt released werden. Nach dem Verstehen der Systemarchitektur sind diese Services für neue Entwickler leicht zu verstehen und Änderungen schnell durchzuführen.

Erfahrungen mit der Microservices-Architektur werden positive und negative aus dem Projekt genommen. Zum Ende würde aber das Team ein Projekt in einer solchen Domäne, wo viele Features grundsätzlich nur das Lesen und Speichern von Daten sind, nicht mehr in einer Microservices-Architektur umsetzen. Für andere Projekttypen könnte der Einsatz von Microservices eben wegen der Schlankheit und Einfachheit der einzelnen Services definitiv wieder zum Einsatz kommen.

5.3.2. Kubernetes

In der Bachelorarbeit wurden viele neue Technologien, unter anderem Kubernetes als Containerorchestrator, eingesetzt. Mit Kubernetes wurden einige Aspekte wie die Skalierung von Containern stark vereinfacht. Um diese Einfachheit erst zu erreichen, musste sehr viel Aufwand bei der Installation und Konfiguration der ganzen Infrastruktur betrieben werden.

Sobald die ersten Konfigurationsdateien für die Services vorhanden waren, funktionierte das Deployment sehr zügig. Durch *helm* konnten einige Zeilen Code gespart werden, da dort die Möglichkeit besteht pro Service ein Template für das Deployment zu erstellen. So mussten pro Service und Umgebung nur noch die entsprechenden Umgebungsvariablen gesetzt werden.

Als weitere Nodes über Azure zum Kubernetes Cluster hinzugefügt wurden, entstand ein sehr merwürdiges Phänomen. Es wurden sehr viele Pods von allen Services erstellt, obwohl etwas anderes konfiguriert wurde. Weiter wurde beim Load Test bemerkt, dass der Datenbank-Pod immer wieder abstürzte und dadurch neue Pods erstellt wurden aufgrund des definierten Replica Sets. Die fehlerhaften Pods wurden nie automatisch gelöscht, so dass immer ein manuelles Intervenieren nötig war.

Kubernetes kann sehr gut im produktiven System eingesetzt werden, nur muss man sehr viel Erfahrung und Kenntnisse über die Konfigurationsmöglichkeiten haben, um die Systeme zuverlässig betreiben zu können.

5.4. Ausblick

Der Auftraggeber erhält zum Abschluss der Bachelorarbeit ein funktionierendes System, welches bereits im Unterricht benutzt werden kann. Mit der Übergabe der Dokumentation und des Source Codes ist die Entwicklung an der HSR Live Survey Plattform beendet. Ob das System vollumfänglich produktiv eingesetzt wird, ist unklar. Es müssen die Vereinbarungen getroffen werden, wo und von wem das System betrieben wird und wer für die Betriebskosten aufkommt.

5.4.1. Einsatz in weiteren Schulen

Die Plattform ist primär nur für den Unterricht an der HSR vorgesehen. In Zukunft könnte sich dies ändern, wenn andere Schulen daran Interesse zeigen. Vorstellbar ist eine Lösung, welche das System als Software as a Service Lösung (SaaS) auch für andere Interessenten bereitstellt.

5.4.2. HTTPS

Für einen produktiven Einsatz müsste HTTPS konfiguriert werden, damit die übertragenden Daten sicher sind vor Dritten.

5.4.3. Erweiterungen

Fragetypen Durch die ausgearbeitete Architektur ist es möglich, einfach weitere Fragetypen hinzuzufügen.

Bewertungen & Auswertungen Zum aktuellen Zeitpunkt sind keine Bewertungen und Auswertungen der Antworten möglich. In Zukunft könnten Antworten beispielsweise mit einer Anzahl Punkte pro richtig abgegebene Antwort bewertet werden. Im Datenmodell wurde dies berücksichtigt.

Teil II.

Projektplanung

1. Projektplan

Dieser Abschnitt beschreibt die Planung und das Projekt HSR Live Survey, welches im Rahmen der Bachelorarbeit an der HSR umgesetzt wird.

1.1. Projektübersicht

1.1.1. Ziel des Projekts

Aus didaktischer Sicht ist es sinnvoll, anfangs einer Unterrichtssequenz eine kurze Rekapitulation des in der Vorwoche behandelten Stoffes vorzunehmen. Um gleichzeitig die Studierenden mental zu aktivieren, bietet es sich an, ein interaktives Medium zu verwenden.

Mit HSR Live Survey soll eine Plattform geschaffen werden, welche es dem Dozierenden erlaubt, eine Reihe von Fragen zu formulieren. Diese können durch die Studierenden über eine mobile Website beantwortet werden. Die Anmeldung erfolgt über einen einfachen Zugangscode.

Gleichzeitig kann die Lehrperson die Website im Präsentationsmodus auf dem Beamer anzeigen. Die von den Teilnehmenden eingegebenen Resultate werden in Real-Time aktualisiert und dargestellt.

1.1.2. Lieferumfang

Zur Plattform HSR Live Survey gehören folgende Applikationen:

- Voter Client
- Presenter Client
- Presentation Designer
- Presenter Remote (optional)

Diese werden in Form von Source-Code bereitgestellt.

Folgende Dokumente gehören ebenfalls zum Lieferumfang:

- Evaluationsbericht

- Anforderungsspezifikation
- Architekturspezifikation
- Testdokumentation
- Installationsanleitung
- Technischer Bericht
- Persönliche Berichte pro Teammitglied

1.2. Projektorganisation

Das Projekt HSR Live Survey wird innerhalb des Moduls Bachelorarbeit umgesetzt. Dabei besteht das Projekt-Team aus zwei Personen und wird von einem Dozenten seitens HSR betreut, welcher gleichzeitig die Rolle des Auftraggebers einnimmt.

1.2.1. Organisationsstruktur

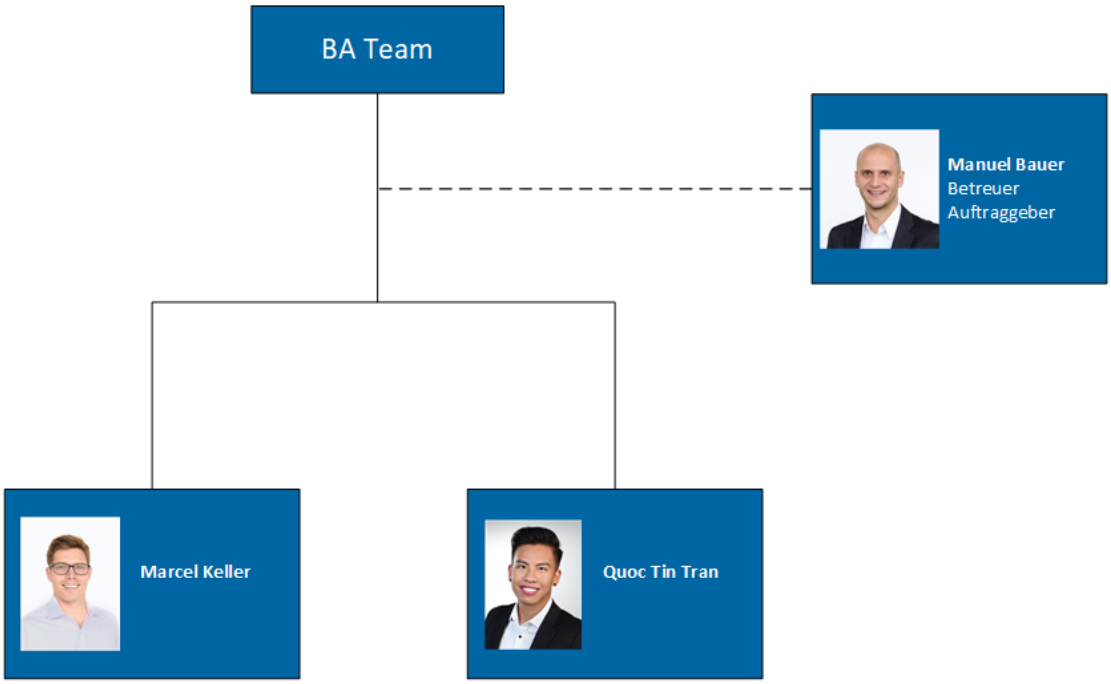


Abbildung 11.: Organigramm
Quelle: Eigene Darstellung

1.2.1.1. Teammitglieder

Nachfolgend sind alle Teammitglieder mit ihren Zuständigkeiten und Kontaktangaben aufgelistet. Trotz zugeordneten Zuständigkeiten, arbeitet jedes Teammitglied in jedem Bereich mit.

Quoc Tin Tran

Kürzel	qtran
E-Mail	qtran@hsr.ch
Zuständigkeiten	Lead Frontend Lead Infrastructure + Azure DevOps

Tabelle 1.: Angaben Quoc Tin Tran
Quelle: Eigene Darstellung

Marcel Keller

Kürzel	m2keller
E-Mail	m2keller@hsr.ch
Zuständigkeiten	Lead Backend Lead Azure

Tabelle 2.: Angaben Marcel Keller
Quelle: Eigene Darstellung

1.2.2. Externe Schnittstellen

Auftraggeber und Betreuer Auftraggeber und Betreuer im Rahmen der Bachelorarbeit ist Manuel Bauer.

Infrastruktur Für technische Belangen wie Server-Bestellung oder Fragen zur Infrastruktur, die von der HSR betrieben werden, ist der HSR Service Desk zuständig.

1.3. Management Abläufe

1.3.1. Kostenvoranschlag

Das Projekt wird im Frühjahressemester 2019 umgesetzt. Dieses dauert vom 18.02.2019 bis am 14.06.2019. Dabei stehen 15 Semesterwochen zu je 20 Stunden pro Person zur Verfügung. Anschliessend ist eine Abschlussphase von zwei Wochen zu je 30 Stunden pro Person geplant, welche nur für die Bachelorarbeit vorgesehen ist. Dies ergibt ein Total von 360 Stunden pro Person (12 ECTS-Credits * 30 Stunden pro ECTS-Credit).

1.3.2. Meilensteine

Name	Datum	Work Products
M1 – Projektplan	22.02.2019	• Initialer Projektplan erstellt
M2 – Analyse Aufgabenstellung	08.03.2019	• Domainanalyse • Anforderungsanalyse • Entwurf der UI-Komponenten
M3 – End of Elaboration	22.03.2019	• Prototyp erstellt • Technologieentscheidungen gefällt • Architekturspezifikation • Continuous Integration aufgesetzt • Entwicklungsumgebung eingerichtet
M4 – Durchführung Präsentation	19.04.2019	• Präsentation kann ohne Slides erstellt und durchgeführt werden • Benutzer kann mit Code an Präsentation teilnehmen
M5 – Fragetyp Umfrage	03.05.2019	• Fragetyp Umfrage umgesetzt • Login und Registrierung möglich
M6 – Fragetyp Skala	17.05.2019	• Fragetyp Skala umgesetzt • Bearbeiten von Benutzerprofil möglich
M7 – Fragetyp Freitext	31.05.2019	• Fragetyp Freitext umgesetzt
M8 – Abgabe	14.06.2019	• Alle Abgabedokumente erstellt

Tabelle 3.: Meilensteine
Quelle: Eigene Darstellung

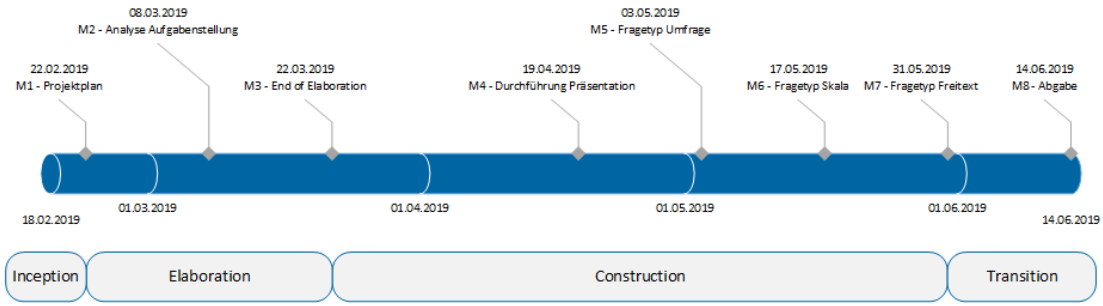


Abbildung 12.: Meilensteinübersicht
Quelle: Eigene Darstellung

1.3.3. Besprechungen

Zwischenbesprechung Die Zwischenbesprechungen mit dem Betreuer finden jeweils wöchentlich nach Absprache statt. Über eine Durchführung wird am Vortag entschieden, die Besprechung sollte aber mindestens alle zwei Wochen stattfinden.

Teambesprechungen Teambesprechungen finden jeweils wöchentlich am Montagnachmittag statt.

1.4. Risikomanagement

1.4.1. Risiken

Das Risikomanagement wird in einem separaten Excel Dokument Risiken.xlsx geführt. Die Risiken werden nach jeder zweiten Iteration neu evaluiert und die Planung gegebenenfalls angepasst.

1.4.2. Umgang mit Risiken

Für alle Risiken, welche im Risikomanagement aufgelistet wurden, wird eine entsprechende Reserve eingeplant. Diese Reserve berechnet sich aus der geschätzten Eintrittswahrscheinlichkeit und dem maximalen Schaden. Die errechnete Reservezeit wird für die Gesamtlaufzeit des Projekts berücksichtigt, mit dem Ziel dabei Schätzungen einzelner Arbeitspakete nicht anpassen zu müssen.

Treten geplante Risiken in einer immer höheren Dichte ein oder sollten neue Risiken auftreten, wodurch Arbeitspakete oder Meilensteine gefährdet würden, so wird mit der Beratungsstelle ein adäquates Szenario zum weiteren Vorgehen ausgearbeitet.

1.5. Projektmanagement

Für das Projektmanagement wird auf Microsoft Azure DevOps gesetzt, welches einen breiten Funktionsumfang bietet, um das gesamte Projekt im Überblick zu behalten. In Azure DevOps werden die Arbeitspakete, Meilensteine, Dokumentationen, Source Code sowie Builds, Releases und Deploys verwaltet.

User Stories Die User Stories stellen die einzelnen konkreten Funktionen und Arbeiten dar, die umgesetzt werden sollen. Einer User Story können bei Bedarf Tasks angehängt werden.

1.5.1. Arbeitspakete

Die detaillierte Ansicht der Arbeitspakete ist in Azure DevOps[1] zu finden. Es wird ein persönlicher Microsoft Account benötigt um im Projekt mitzuwirken. Der Betreuer seitens HSR benutzt seinen persönlichen Microsoft Account.

1.5.1.1. Schätzung

Im Rahmen des Sprint Planning werden die Arbeitspakete in Mannstunden geschätzt. Wir haben uns für diese Variante entschieden, da es zu lange dauern würde, den Teamspeed in Story Points vernünftig berechnen zu können.

1.6. Qualitätsmassnahmen

In diesem Kapitel wird definiert, mit welchen Massnahmen die Qualität des Projekts so hoch wie möglich gehalten wird.

1.6.1. Dokumentation

Die Dokumente werden in einem eigenen Git-Repository[2] abgelegt und unterstehen dem Vier-Augen-Prinzip. Dadurch werden Fehler minimiert und die Qualität entsprechend gesteigert.

1.6.2. Entwicklung

Der Source Code ist pro Teilapplikation in einem eigenen Repository zu finden. Massnahmen zur Erhöhung der Qualität des Source Code siehe 1.6.2.4 Code Reviews. Ebenso wird bei jedem Build der Code gegen den Styleguide überprüft, wobei Verstösse während dem Build im Continuous Integration zu einem Buildfehler führen.

[1] `https://dev.azure.com/HSRLiveSurvey`
[2] `https://dev.azure.com/HSRLiveSurvey/HSRLiveSurvey/_git/HSRLiveSurveyDocuments`

1.6.2.1. Definition of Done

Mit der «Definition of Done» wird festgelegt mit welchem Zustand eine Arbeit als abgeschlossen betrachtet werden kann.

User Stories

- produzierter Code entspricht den Anforderungen der User Stories
- Code übersetzt ohne Fehler und ohne Warnungen
- Unit Tests laufen fehlerfrei durch
- Review des Codes wurde durchgeführt
- Code entspricht den Coding Guidelines
- Architektur wurde eingehalten

Sprints

- zeitliche Limite wurde eingehalten
- Product Backlog wurde angepasst
- Sprint Retrospektive wurde abgehalten
- Planung des nächsten Sprint wurde gemacht

1.6.2.2. Coding Guidelines

- Kein unaufgeräumter (z.B. auskommentierter) Code
- Alles Unfertige ist markiert mit «TODO:»

1.6.2.3. Code Style Guidelines

Als Vorlage dienen die Coding Style Guidelines von dofactory[3].

Backend Coding-Guidelines werden durch das Plugin StyleCop in Visual Studio abgedeckt und ebenfalls beim Build automatisch überprüft.

Frontend Im Frontend wird auf TSLint gesetzt. Da es im TypeScript keinen offiziellen Styleguide gibt, werden die empfohlenen Styleguides “tslint:recommended” und “tslint-react” verwendet, welche noch mit eigenen Vereinbarungen vom Team ergänzt oder abgeändert wurden. Ansonsten dient der C# Styleguide als Vorbild, wie zum Beispiel bei der Art der Benennung von Klassen und Variablen.

[3] `http://www.dofactory.com/reference/csharp-coding-standards`

1.6.2.4. Code Reviews

Zur Erhöhung der Qualität des Source Code wird mit «pull requests» innerhalb von Git gearbeitet, wodurch auch für den Source Code auf ein Vier-Augen-Prinzip gesetzt wird.

1.6.3. Testen

1.6.3.1. Unit Testing

Alle systemrelevanten Klassen und Komponenten werden systematisch und wiederkehrend durch automatisierte Unit Tests geprüft. Hierfür wird der Ansatz des Microtesting angestrebt.
In begründeten Ausnahmefällen kann auf Unit Tests verzichtet werden.

Die Unit Tests werden automatisiert und regelmässig durch den Azure DevOps - Continuous Integration ausgeführt. Dabei steht dem ganzen Team eine Übersicht zur Verfügung, welche die Testresultate darstellt und somit auf fehlgeschlagene Tests entsprechend reagiert werden kann. Es wird dadurch eine Zunahme der Code Qualität, sowie eine grössere Zuverlässigkeit während des kompletten Entwicklungsprozesse erwartet.

Backend Im Backend werden die Unit Tests mit xUnit geschrieben und bei jedem Build auf dem Build-Server ausgeführt.

Frontend Im Frontend werden auf Tests verzichtet. Dies wurde mit dem Betreuer so besprochen.

1.7. Infrastruktur

Für das Projekt verwendet jedes Teammitglied seinen eigenen Laptop oder PC. Das Hauptwerkzeug für die Projekt- und Versionsverwaltung inkl. Build und Deployment Management ist Microsofts Cloud Lösung «Azure DevOps». Die HSR stellt je einen virtuellen Server mit Windows Server 2016 R2 und Linux Ubuntu 18.04 zur Verfügung, auf dem die Build- und Deployagents eingerichtet sind.

1.7.1. Übersicht der Tools

Werkzeug	Version	Beschreibung	Links
Azure DevOps	n/a	Projektverwaltung Versionsverwaltung (Git) Continuous Integration Automatic Deployment	https://dev.azure.com/savzhsr/
Visual Studio	2017 / 2019	Entwicklungsumgebung	https://www.visualstudio.com/de/vs
Visual Studio Code	1.33.x	Entwicklungsumgebung	https://code.visualstudio.com/
ReSharper	2017	Refactoring	https://www.jetbrains.com/resharper
Astah Professional	7	UML	http://astah.net/editions/professional
Azure CLI	2.0.x	Azure Handling	https://docs.microsoft.com/en-us/cli/azure/
Docker for Windows	18.09.x	Container Management	https://docs.docker.com/docker-for-windows/
Kubernetes (kubectl)	1.14	Container Orchestrierung	https://kubernetes.io/docs/tasks/tools/install-kubectl/
helm	2.13.1	Kubernetes Package Manager	https://helm.sh/
Node.js	10.15.x (LTS)	JavaScript Runtime	https://nodejs.org/en/
NPM	6.4.1	Node Package Manager	https://www.npmjs.com/

Tabelle 4.: Übersicht der Tools
Quelle: Eigene Darstellung

Teil III.

SW-Engineering Dokumente

1. Anforderungsspezifikation

Dieser Abschnitt beschreibt die Anforderungen an das Projekt HSR Live Survey, welches im Rahmen der Bachelorarbeit an der HSR umgesetzt wird.

1.1. Allgemeine Beschreibung

1.1.1. Produktfunktion

Die allgemein beschriebenen Produktfunktionen sind sowohl in der Aufgabenstellung als auch im Projektplan zu finden. Die detaillierten Funktionalitäten sind im Kapitel 1.2 - Use Cases erfasst.

1.1.2. Benutzercharakteristik

Das HSR Live Survey richtet sich an alle Dozenten, die in ihren Vorlesungen den behandelten Stoff der Vorwoche auf interaktive Art rekapitulieren möchten. Als Teilnehmende der Präsentationen fungieren Studierende, die sich dadurch aktiv am Unterricht beteiligen.

1.2. Use Cases

1.2.1. Use Case Diagramm

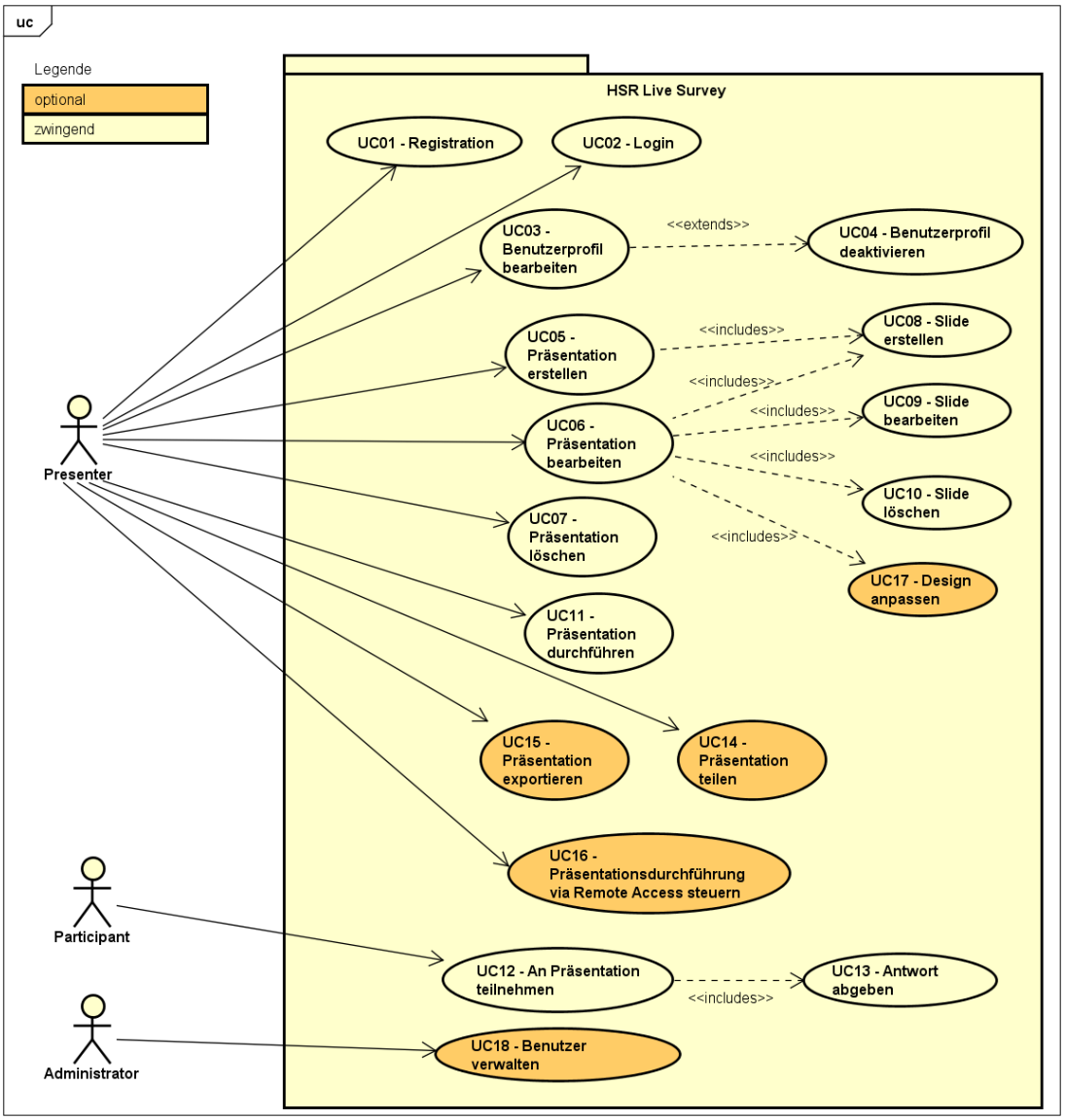


Abbildung 13.: Use Case Diagramm
Quelle: Eigene Darstellung

1.2.2. Aktoren

Aktor	Beschreibung
Presenter	Erstellt Präsentationen mit Fragen und Antworten und führt diese durch
Participant	Nimmt an durchzuführenden Präsentationen teil
Administrator	Verwaltet die Benutzer im System

Tabelle 5.: Aktoren
Quelle: Eigene Darstellung

1.2.3. UC01 - Registration

Um mich später im System als Presenter anmelden zu können, registriere ich mich. Dabei gebe ich

1. meinen Vor- und Nachnamen,
2. meine E-Mail Adresse,
3. und mein gewünschtes Passwort ein.

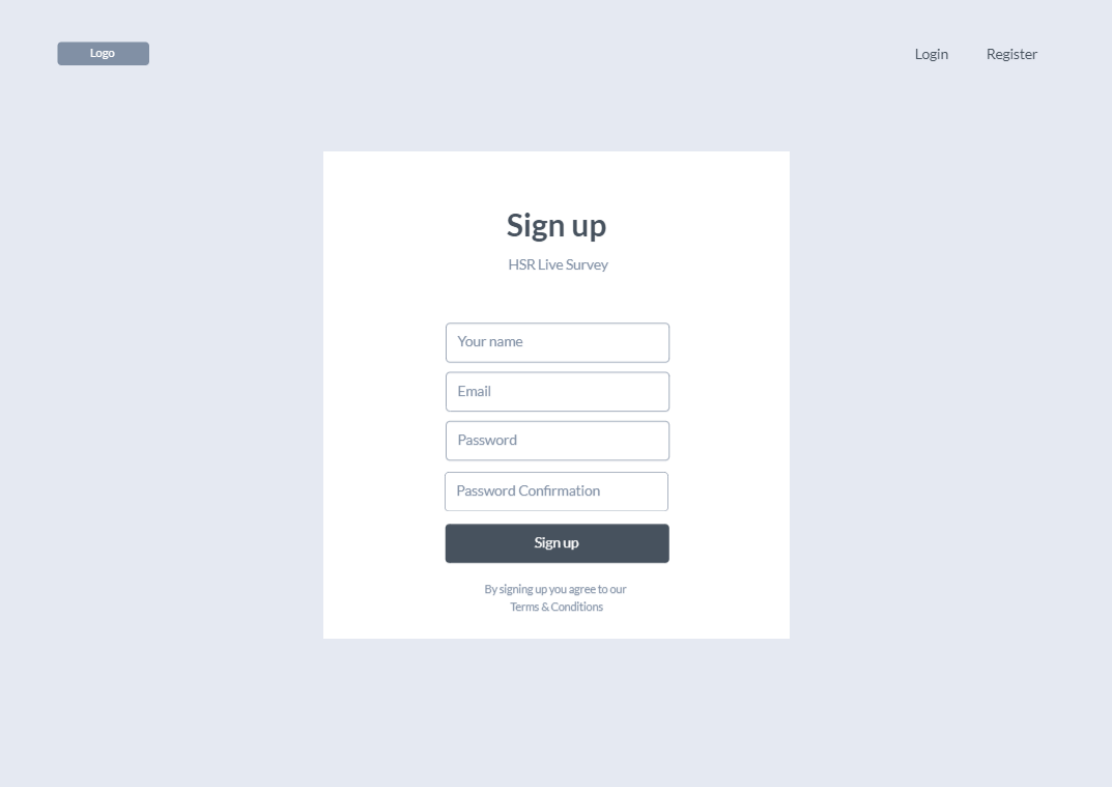


Abbildung 14.: UI Registration für UC01
Quelle: Eigene Darstellung

1.2.4. UC02 - Login

Als Presenter verwalte ich meine Präsentationen. Dabei logge ich mich im System ein und gebe

1. meine E-Mail Adresse und
2. mein Passwort ein.

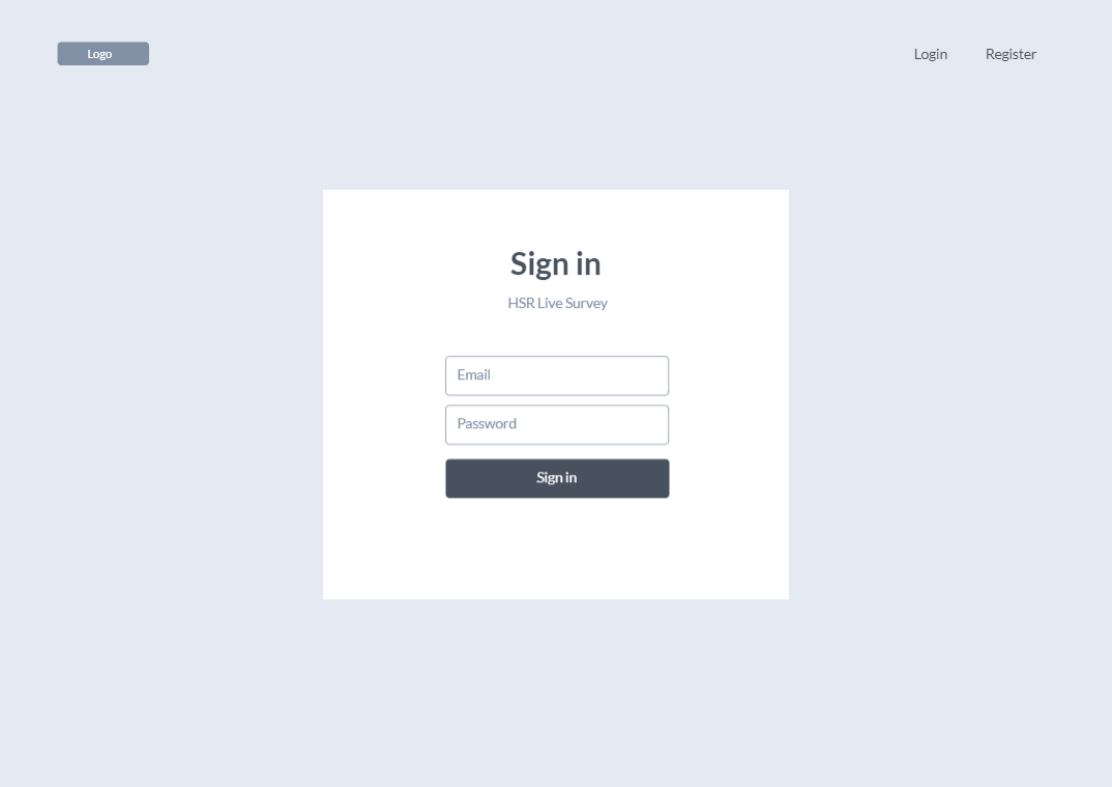


Abbildung 15.: UI Login für UC02
Quelle: Eigene Darstellung

1.2.5. UC03 - Benutzerprofil bearbeiten

Precondition Der Presenter ist im System angemeldet.
Als Presenter ändere ich meine bei der Registrierung angegebenen Daten. Ich bearbeite und speichere dabei

- meine angepassten Name,
- meine neue E-Mail Adresse oder
- mein neues Passwort.

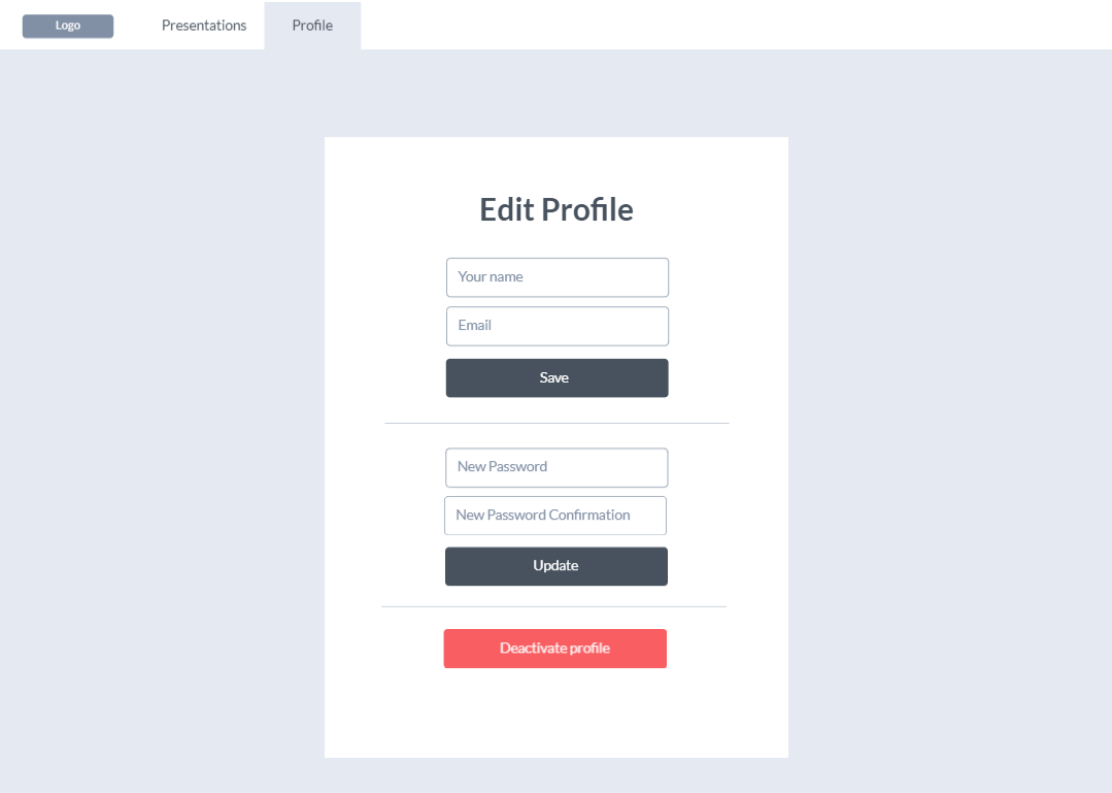


Abbildung 16.: UI Benutzerprofil bearbeiten für UC03 und UC04
Quelle: Eigene Darstellung

1.2.6. UC04 - Benutzerprofil deaktivieren

Precondition Der Presenter ist im System angemeldet.
Als Presenter deaktiviere ich endgültig mein Benutzerprofil inkl. meiner erstellten Präsentationen, weil ich mein Profil und die Präsentationen nicht mehr benötige. Meine Präsentationen, die ich mit anderen geteilt habe, stehen diesen immer noch zur Verfügung.

1.2.7. UC05 - Präsentation erstellen

Precondition Der Presenter ist im System angemeldet.
Als Presenter erstelle ich eine neue Präsentation und vergebe dieser einen Titel. Dabei erstelle ich den ersten Slide (siehe UC08 - Slide erstellen). Eine Präsentation besteht aus mehreren Slides.

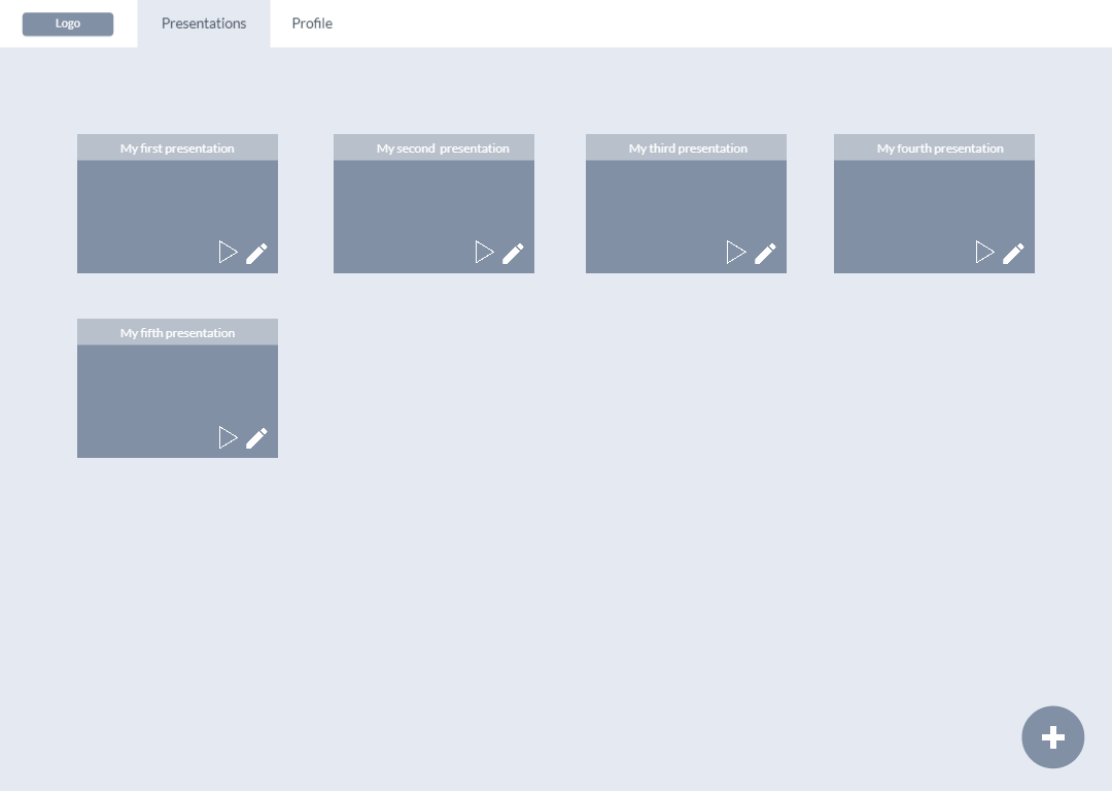


Abbildung 17.: UI Präsentationsübersicht für UC05 und UC07
Quelle: Eigene Darstellung

1.2.8. UC06 - Präsentation bearbeiten

Precondition Der Presenter ist im System angemeldet und Präsentation existiert.
Als Presenter bearbeite ich eine bestehende Präsentation und kann dabei

- die Reihenfolge der Slides ändern,
- einen neuen Slide erstellen (siehe UC08 - Slide erstellen),
- einen bestimmten Slide bearbeiten (siehe UC09 - Slide bearbeiten),
- einen bestimmten Slide löschen (siehe UC10 - Slide löschen) oder
- optional das Design für alle Slides anpassen (siehe UC17 - Design anpassen (optional)).

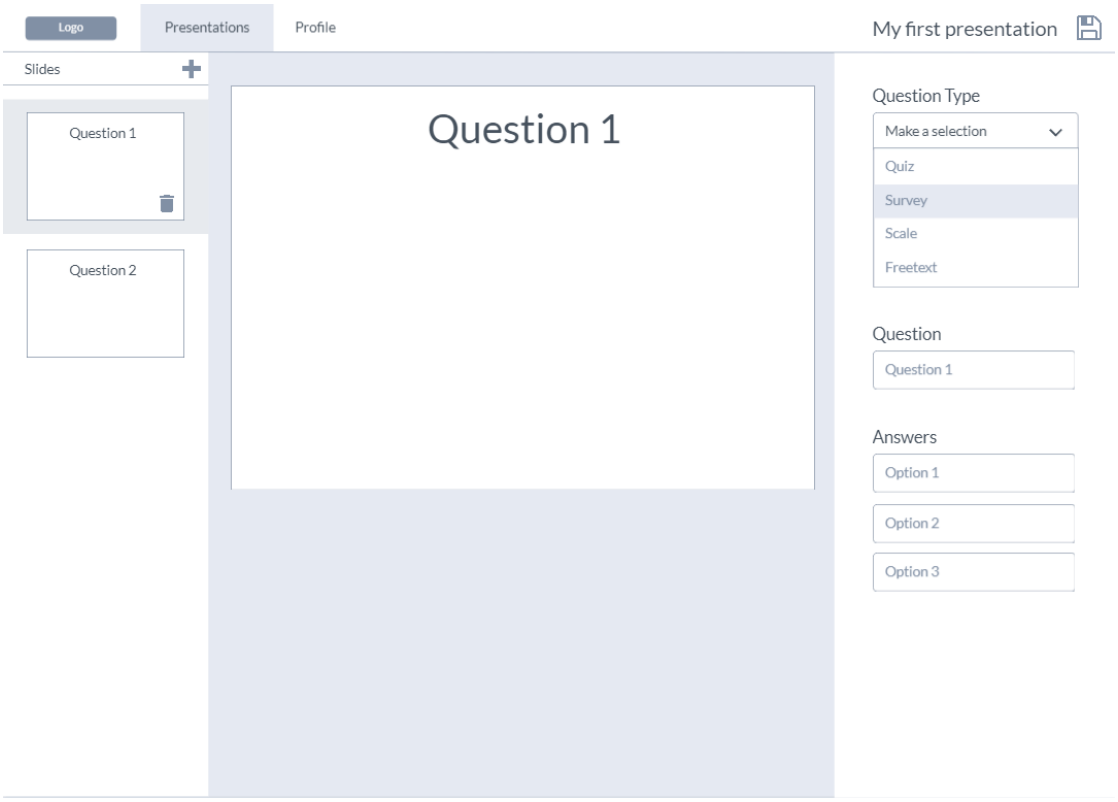


Abbildung 18.: UI Präsentation bearbeiten für UC06, UC08, UC09, UC10
Quelle: Eigene Darstellung

1.2.9. UC07 - Präsentation löschen

Precondition Der Presenter ist im System angemeldet und Präsentation existiert.
Als Presenter lösche ich eine Präsentation inkl. der Slides, weil ich diese nicht mehr benötige.

1.2.10. UC08 - Slide erstellen

Precondition Der Presenter ist im System angemeldet und Präsentation existiert.
Als Presenter erstelle ich einen neuen Slide und wähle einen Fragetyp aus. Dabei habe ich die Auswahl zwischen folgenden Fragetypen:

- Umfrage
- Quiz
- Skala
- Freitext

1.2.11. UC09 - Slide bearbeiten

Precondition Der Presenter ist im System angemeldet und Slide existiert.
Als Presenter bearbeite ich einen Slide. Im Slide definiere ich

1. eine Frage,
2. die dazugehörigen Antworten und
3. weitere Konfigurationseinstellungen wie die maximale Antwortzeit.

1.2.12. UC10 - Slide löschen

Precondition Der Presenter ist im System angemeldet und Slide existiert.
Als Presenter lösche ich einen bestimmten Slide aus der Präsentation.

1.2.13. UC11 - Präsentation durchführen

Precondition Der Presenter ist im System angemeldet.
Damit ich zu meiner Präsentation Feedback von Teilnehmern bekomme, führe ich eine bereits erstellte Präsentation durch. Dabei

1. starte ich eine Präsentation und
2. teile den automatisch generierten Teilnahmecode den Teilnehmern mit.
3. Sobald alle Teilnehmer dabei sind,
4. wechsle ich auf den ersten Slide.

Optional kann der Teilnahmecode als QR-Code generiert werden.

Angezeigt wird immer die Anzahl Votes und die Anzahl Teilnehmer. Während einer Durchführung kann ich folgende Aktionen in beliebiger Reihenfolge und Kombination ausführen:

- Wechseln in den Fullscreen-Modus
- Beenden des Fullscreen-Modus
- Navigation vor- und rückwärts durch die Präsentation
- Eingegebene Resultate und je nach Fragetyp korrekte Lösung zeigen/verbergen
- Durchführung beenden

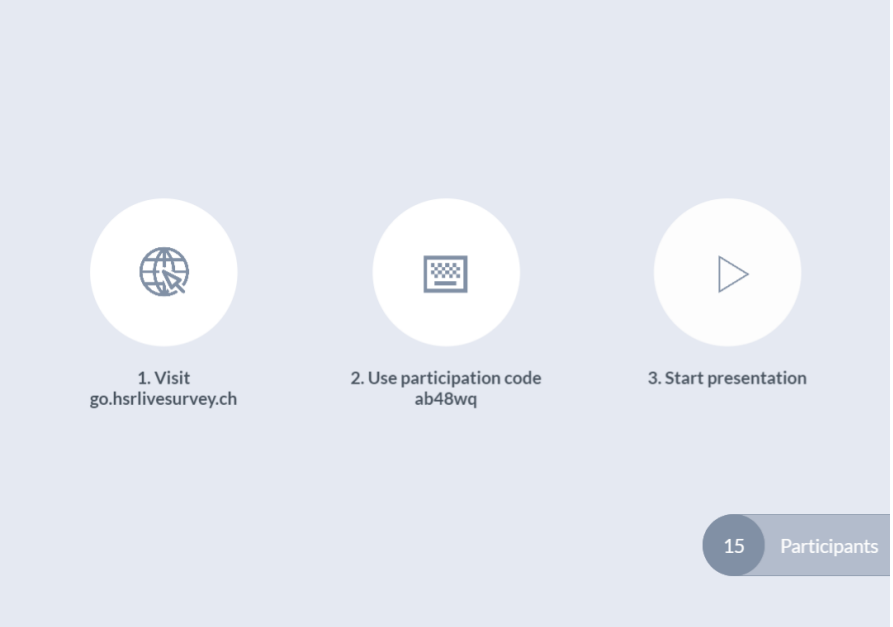


Abbildung 19.: UI Präsentationsstart für UC11
Quelle: Eigene Darstellung

Abbildung 20.: UI Präsentationsdurchführung für UC11
Quelle: Eigene Darstellung

1.2.14. UC12 - An Präsentation teilnehmen

Precondition Eine Präsentation ist in der Durchführung.
Als Participant möchte ich an Präsentationen teilnehmen können. Dabei gebe ich

1. den erhaltenen Teilnahmecode ein und
2. warte auf den Start der Präsentation.

Optional kann anstelle des Teilnahmecodes ein QR-Code verwendet werden.

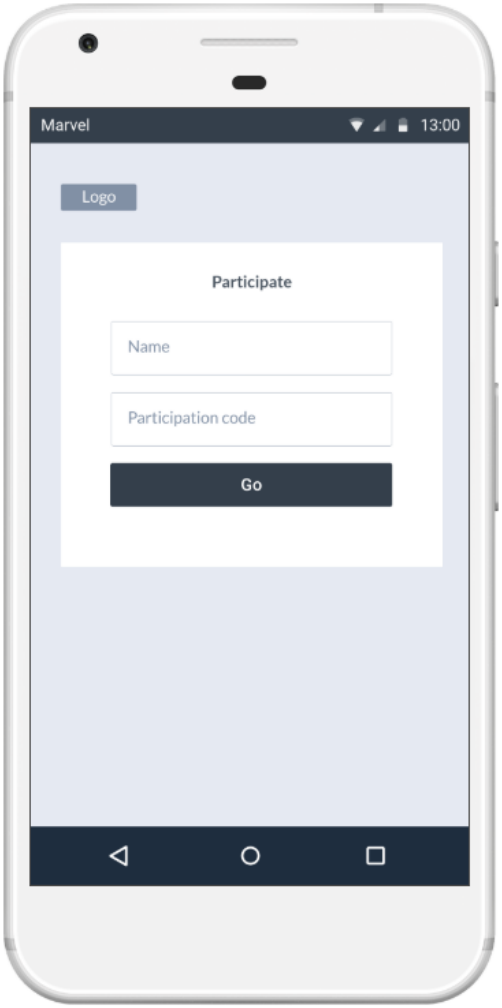


Abbildung 21.: UI Präsentationsteilnahme für UC12
Quelle: Eigene Darstellung

1.2.15. UC13 - Antwort abgeben

Precondition Participant nimmt an Präsentation teil.
Als Participant

1. wähle ich meine Antworten aus und
2. sende diese ab.

Optional kann das Resultat im Voter Client angezeigt werden.

(a) Antwortabgabe (b) Warteszenario

Abbildung 22.: UIs Antwortabgabe für UC13
Quelle: Eigene Darstellung

1.2.16. UC14 - Präsentation teilen (optional)

Precondition Der Presenter ist im System angemeldet.
Damit auch andere Presenter meine erstellten Präsentationen bearbeiten und durchführen können, teile ich eine Präsentation mit anderen. Dabei gebe ich die E-Mail Adresse des anderen Presenters ein und lade ihn ein.

1.2.17. UC15 - Präsentation exportieren (optional)

Precondition Der Presenter ist im System angemeldet.
Damit ich als Presenter Präsentationen mit Musterlösungen anderen Personen zur Verfügung stellen kann, exportiere ich diese. Dabei wähle ich

1. die Präsentation aus und
2. entscheide mich zwischen PDF und PowerPoint als Format.

1.2.18. UC16 - Präsentationsdurchführung via Remote Access steuern (optional)

Precondition Eine Präsentation ist in der Durchführung.
Damit ich als Presenter bei einer Durchführung einer Präsentation nicht auf mein Computer als Steuermittel angewiesen bin, kann ich auf einer dedizierten Seite mit meinem Smartphone die Präsentation steuern. Dafür

1. generiere ich ein Code für die Steuerung,
2. öffne die Steuerungsseite auf meinem Smartphone,
3. gebe den generierten Code ein und
4. steuere dann meine Präsentation.

Dabei sind folgende Aktionen möglich:

- Wechseln in den Fullscreen-Modus
- Beenden des Fullscreen-Modus
- Navigation vor- und rückwärts durch die Präsentation
- Eingegebene Resultate und je nach Fragetyp korrekte Lösung zeigen/verbergen

1.2.19. UC17 - Design anpassen (optional)

Als Presenter passe ich für alle Slides in der Präsentation das Design an. Dabei kann ich folgendes anpassen:

- Farbe
- Schrift
- Logo

1.2.20. UC18 - Benutzer verwalten (optional)

Precondition Der Administrator ist im System angemeldet.
Als Administrator verwalte ich die im System registrierten Benutzer. Dabei kann ich

- Benutzer editieren und
- Benutzer deaktivieren.

1.3. Fragetypen

Für verschiedene Fragen kommen verschiedene Möglichkeiten der Antwortabgabe und Anzeige in Frage. Diese verschiedenen Fragetypen wurden analysiert und enstsprechend umgesetzt oder verworfen.

1.3.1. Umfrage (Multiple Choice)

Eine der wichtigsten Fragen stellt die Umfrage dar. Dabei können aus einer vorgegebenen Liste an Auswahlmöglichkeiten eine oder mehrere ausgewählt werden. Dieser Fragetyp eignet sich beispielsweise für Fragen nach Bekanntem oder Vorlieben. Dieser Fragetyp wird umgesetzt.

1.3.2. Umfrage (Single Choice)

Eine leicht abgeänderte Version des Multiple Choice, wobei nur maximal eine Auswahlmöglichkeit angewählt werden kann. Dieser Fragetyp wird umgesetzt.

1.3.3. Quiz

Dieser Fragetyp funktioniert im Grundprinzip gleich wie die Umfrage, es kann jedoch bei den Auswahlmöglichkeiten angegeben werden, ob diese korrekt ist oder nicht. Diese Lösung kann dann im Rahmen der Präsentation der Antworten angezeigt werden. Bei dem Fragetyp Quiz kann konfiguriert werden, wie viele Antworten maximal abgegeben werden können. Dieser Fragetyp wird umgesetzt.

1.3.4. Freitext

Um nach Ideen oder Stichworten zu fragen eignet sich der Fragetyp Freitext. Bei diesem kann der Teilnehmer ein oder mehrere Texte als Antwort absenden. Dieser Fragetyp wird umgesetzt.

1.3.5. Freitext bewertet

Der Fragetyp Freitext bewertet ist eine Mischung aus Freitext und Quiz. Bei der Konfiguration der Frage kann der Ersteller angeben, welche Text als Antwort gezählt werden und welche falsch sind. Dieser Fragetyp wird nicht umgesetzt, da es zum einen bei der

Validierung der Korrektheit der Antworten Probleme mit Umlauten oder Abständen geben könnte. Zum anderen würde sich für solche Fragen eher der Fragetyp Quiz anbieten, da bei diesem die Auswahl klarer ersichtlich ist für die Teilnehmer.

1.3.6. Skala

Ähnlich wie eine Multiple Choice Frage, allerdings werden die Antwortmöglichkeiten bei diesem Fragetyp nicht vorgegeben sondern können in einem vordefinierten Bereich ausgewählt werden. Schätzfragen würden perfekt auf diesen Fragetyp passen. Dieser Fragetyp wird umgesetzt.

1.3.7. Skala bewertet

Die bewerte Version des Fragetyps Skala wird nicht umgesetzt, da zusammen mit dem Auftraggeber keine konkreten Use Cases gefunden werden konnten, die perfekt auf diesen Fragetyp passen. Die analysierten Fragen konnten besser mit dem Quiz umgesetzt werden.

1.4. UI Entwurf

Die UI Entwürfe für die entsprechenden Use Cases wurden mit Marvel, einem Online Tool für die Erstellung von Wireframes, erstellt. Dabei wurden nur die zwingenden Use Cases abgedeckt.

Presenation Designer und Presenter Client Für den Presentation Designer und den Presenter Client existiert ein minimaler UI Entwurf, der auf der Marvel Seite[1] angesehen und interaktiv durchgeklickt werden kann.

Voter Client Der Entwurf für den Voter Client kann auf der Marvel Seite[2] angesehen und interaktiv durchgeklickt werden.

[1] `https://marvelapp.com/3ec658g`
[2] `https://marvelapp.com/5bif71i`

1.5. Weitere Anforderungen

1.5.1. Nicht-Funktionale Anforderungen

ID	Anforderung
01	**Verfügbarkeit**
NFA01.01	Das System HSR Live Survey muss zu den Unterrichtszeiten an der HSR in 95% der Zeit verfügbar sein.
02	**Bedienbarkeit**
NFA02.01	Jeder Student an der Technischen Hochschule Rapperswil muss die Applikation Voter Client nach einer Einführung von 5 Minuten ohne Probleme bedienen können.
NFA02.02	Jeder Dozent an der Technischen Hochschule Rapperswil muss die Applikation Presenter Client nach einer Einführung von 1 Stunde ohne Probleme bedienen können.
NFA02.03	Ein Teilnehmer muss sobald die Voter Client Applikation gestartet ist mit maximal 4 Eingabeoperationen (Texteingabe oder Klick auf Button) an einer Präsentation teilnehmen können.
03	**Performance**
NFA03.01	Antwortzeit der Komponenten Presentation Designer, Presenter Client und Voter Client beim initialen Aufruf darf eine Sekunde nicht überschreiten.
NFA03.02	Antwortzeit von API Aufrufen in 90% der Fälle maximal 100 Millisekunden, in allen anderen Fällen maximal 500 Millisekunden.
04	**Sicherheit**
NFA04.01	Passwörter der Benutzer müssen mit einem im Januar 2019 als sicher angesehen Hash gehasht persistiert werden.
05	**Erweiterbarkeit**
NFA05.01	Ein Informatikstudent im 6 Semester muss nach einer Woche Einarbeitszeit einen neuen Fragetypen innerhalb von zwei Wochen implementieren können.
06	**Skalierbarkeit**
NFA06.01	Alle Komponenten des HSR Live Survey müssen so konzipiert werden, dass sie horizontal skaliert werden können.

Tabelle 6.: Nicht-Funktionale Anforderungen
Quelle: Eigene Darstellung

1.5.2. Randbedingungen

1.5.2.1. User-Interfaces

Alle UI-Applikationen müssen als Single Page Application entwickelt werden.

1.5.2.2. Backend

Alle Backend-Komponenten müssen Microsoft .NET Core verwenden.

1.5.3. Cloud-Anbieter

Als Cloud-Anbieter muss Microsoft Azure verwendet werden.

1.6. Anforderungen an den Ablauf

1.6.1. Frage bei späterem Eintritt einer Durchführung

Falls sich ein Participant bei einer Durchführung einwählt, die bereits gestartet ist, soll ihm nicht die aktuelle Frage angezeigt werden. Sollte diese angezeigt werden, wäre es möglich, dieselbe Frage in mehreren Browsern mehrfach zu beantworten und so das Resultat zu verfälschen. Der Nachteil dieser Variante ist, dass eventuell nicht alle Participants die aktuelle Frage beantworten können. Durch die Anzeige der Anzahl Participants vor dem Start der Präsentation beim Presenter kann dieses Problem aber entschärft werden.

1.6.2. Änderung an einer Präsentation

Sobald eine Präsentation einmal durchgeführt wurde und diese über Antworten von Teilnehmenden verfügt können Änderungen an der Präsentation diese Antworten verfälschen. Dies ist beispielweise beim Entfernen eines Slides oder einer Antwortmöglichkeit der Fall. Für spätere Auswertungen würden alle Antworten zu diesen gelöschten Entitäten nicht mehr erscheinen und so ein falsches Bild abgeben. Eine Möglichkeit wäre, alle zu einer Präsentation dazugehörigen Daten vor einer Durchführung zu Kopieren und diese zusammen mit der Präsentationsdurchführung abzulegen. Dies führt aber zu massiven Datenduplikaten, welche allenfalls nicht benötigt werden würden. Die umzusetzende Variante soll den Benutzer vor Änderungen an einer Präsentation warnen, sobald diese bereits einmal durchgeführt wurde. Mit der Anzeige der Warnung hat der Benutzer die Möglichkeit, die Änderungen auf eigene Verantwortung durchzuführen oder die Präsentation zu kopieren und die Änderungen an der Kopie durchzuführen. Mit dieser Variante werden Daten nur kopiert, falls es der Benutzer als sinnvoll erachtet. Eine Kopie kann beispielsweise beim Korrigieren eines Schreibfehlers in einer Antwortmöglichkeit nicht von nöten sein.

2. Domainanalyse

2.1. Domain Model

Das Domain Model wurde für die einfachere Lesbarkeit in zwei Diagramme unterteilt. Die Abbildung 23 zeigt das Domain Model als Übersicht ohne die konkreten Fragetypen. Diese sind in der Abbildung 24 ersichtlich. Im Abschnitt Wichtige Konzepte werden die im Domain Model gezeigten Konzepte beschrieben.

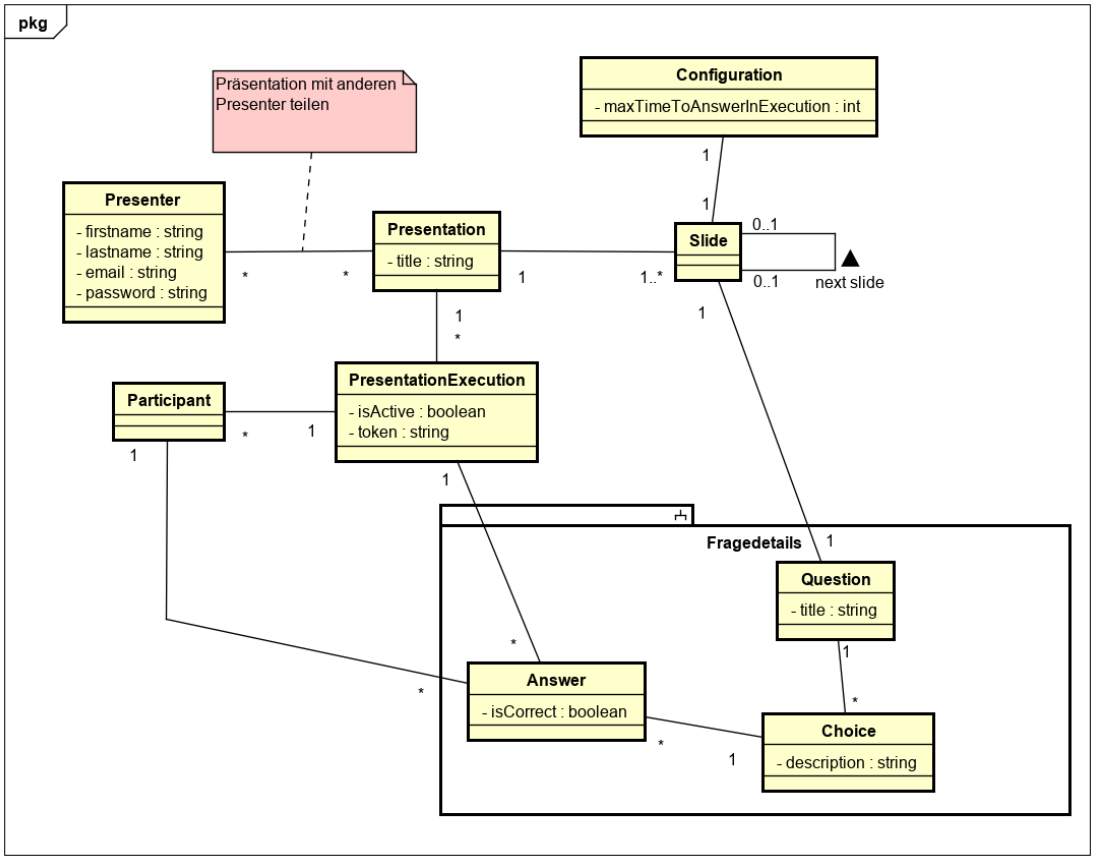


Abbildung 23.: Domain Model Übersicht
Quelle: Eigene Darstellung

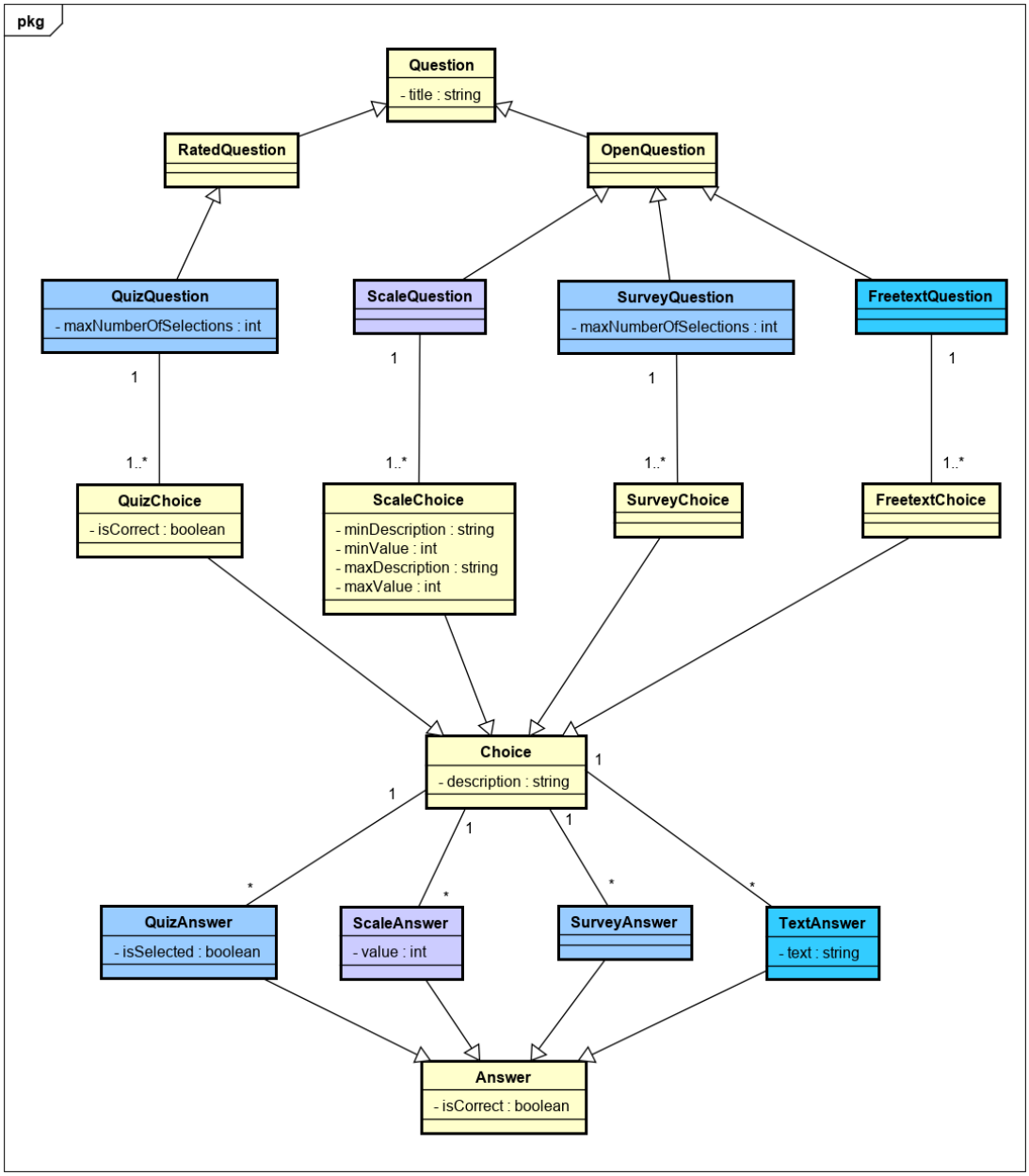


Abbildung 24.: Domain Model Fragedetails
Quelle: Eigene Darstellung

2.1.1. Wichtige Konzepte

2.1.1.1. Präsentationsdurchführung

Die PresentationExecution beschreibt die Durchführung einer Präsentation, an welcher Participants teilnehmen.

2.1.1.2. Presenter

Der Presenter muss sich in der Applikation registrieren und anmelden, um Präsentationen erstellen und durchführen zu können. Aktuell ist es nicht angedacht, dass ein Presenter mit seiner Identität an Präsentationen teilnehmen kann. Dies würde immer im Kontext eines Participants geschehen.

2.1.1.3. Participant

Teilnehmer einer Durchführung einer Präsentation sind Participants. Diese müssen sich im System nicht anmelden sondern nehmen via Token an einer Durchführung teil.

2.1.1.4. Fragetypen

Die Fragetypen sind unterteilt in die beiden Kategorien "bewertete Frage" (RatedQuestion) und "offene Frage" (OpenQuestion). Der Unterschied liegt in den Antworten. Bei den bewerteten Fragen gibt es richtige und falsche Antworten, bei den offenen Fragen nicht. Pro Fragetyp existieren je nach Typ ein Typ pro Kategorie. Diese sind in der Abbildung 24 farblich als Paar markiert.

Freitext-Fragen Bei einer Freitextfrage kann der Participant einen beliebigen Text eingeben und muss keine Wahl einer vordefinierten Antwortliste tätigen.

Skala-Fragen Bei einer Skala-Frage wählt der Participant einen Wert in einer vordefinierten Skala aus.

Umfrage / Quiz Bei einer Umfrage kann der Participant eine oder mehrere Antworten wählen, es gibt dabei aber keine richtigen Antworten. Beim Quiz gibt es ebenso eine oder mehrere mögliche Antworten, diese sind aber richtig oder falsch. Bei diesen Fragetypen gibt es für die Begrenzung der möglichen Antworten, die ein Teilnehmer anwählen kann, die Eigenschaft maxNumberOfSelections.

2.1.1.5. Auswahlmöglichkeiten (Choice)

Die Auswahlmöglichkeiten unterscheiden sich je nach Fragetyp in den Eigenschaften. Deshalb besitzt jeder Fragetyp einen eigenen Auswahlmöglichkeits-Typ.

2.1.1.6. Abgegebene Antworten (Answer)

Bei einer Durchführung gibt jeder Participant eine oder mehrere Antworten pro Frage ab. Diese werden für Auswertungszwecke bereits als richtig oder falsch markiert. Bei offenen Fragen werden diese immer als richtig markiert.

2.1.1.7. Slide

Ein Slide repräsentiert die Darstellung einer Frage inklusive Titel und gewissen Eigenschaften, die konfiguriert werden können. Um die Reihenfolge der Präsentation steuern zu können kennt zudem jeder Slide seinen Nachfolger.

2.1.1.8. Configuration

Pro Slide gibt es Einstellmöglichkeiten wie beispielsweise die maximale Antwortzeit. In Zukunft können mehr solche Eigenschaften hinzukommen, deshalb existiert eine eigene Klasse dafür.

2.1.1.9. Beziehung Presenter - Presentation

Die Beziehung von Presenter und Presentation bildet zum einen den Ersteller und Owner einer Präsentation ab. Zum anderen können Präsentationen geteilt werden, dies wird ebenfalls in dieser Beziehung abgebildet.

2.1.1.10. Beziehung Freetext - FreetextChoice

Eine Freetext-Frage kann mehrere Textanwortmöglichkeiten beinhalten. Diese sind implizit immer korrekt da es keinen Sinn ergibt eine falsche Freitextauswahlmöglichkeit zu formulieren.

2.1.1.11. Beziehung Answer - Choice

Eine abgegebene Anwort ist mit der Auswahlmöglichkeit verbunden um das Resultat eines Slides anzeigen zu können. Zur Vereinfachung sind die Beziehungen der Antworten direkt mit der Basisklasse der Auswahlmöglichkeiten verbunden, in Realtität sind diese aber direkt mit den entsprechenden Subtypen verbunden.

3. Evaluation

3.1. Vorgaben

Folgende Vorgaben wurden gemeinsam mit dem Auftraggeber vor dem Start der Bachelorarbeit definiert.

3.1.1. Backend-Technologie

Als Technologie für Backend-Teile des Projektes soll Microsoft .NET Core verwendet werden. Umgesetzt werden diese mit ASP.NET Core Web API.

3.1.2. Cloud-Anbieter

Als Cloud-Anbieter sollen Dienste von Microsoft Azure verwendet werden. Welche Dienste für welchen Teil des Projektes verwendet werden, muss evaluiert werden.

3.2. Datenbank

3.2.1. SQL oder NoSQL

Ein Grundsatzentscheid bei der Auswahl der Datenbanktechnologie stellt die Auswahl zwischen SQL oder NoSQL dar. Wir haben uns für einen Mix aus beiden Technologien entschieden. Nachfolgend sind die Entscheidungen pro Technologie aufgeführt.

SQL SQL ist ein bekannter und weit verbreiteter Standard. Der Hauptvorteil von SQL sind Relationen und deren Integrität. Als Nachteil kann das starre Schema angesehen werden. Unser Domain Model kann aus dieser Sicht in zwei Teile geteilt werden, ein Teil davon beinhaltet die Präsentation- und Userverwaltung. Dieser Teil wird sich in naher Zukunft kaum ändern oder es kommen nur einzelne neue Attribute hinzu. Deshalb haben wir uns entschieden, für diesen Teil der Applikation eine SQL Datenbank zu verwenden.

NoSQL NoSQL erlaubt es unter anderem, Dokumente in JSON-Format zu speichern. Diese Dokumente beinhalten nur ein indirektes Schema, welches in der Applikation, aber nicht auf der Datenbank definiert ist. Für den zweiten Teil useres Domain Models, die Fragen und Antworten, kommt die Freiheit des Schemas in Frage. Denn einzelne Fragetypen haben ein sehr unterschiedliches Schema, bei neuen Fragetypen können auch komplett andere Schemas hinzukommen. Für diesen Teil haben wir uns aus diesem Grund für eine NoSQL-Datenbank entschieden.

3.2.2. Produkte für SQL Server

Nachfolgend sind einige mögliche Produkte aufgelistet, die als Datenbank für unser System eingesetzt werden könnten.

Azure SQL Database Azure SQL Database bietet Microsoft SQL Server-Datenbanken an. Diese können wie eigen gehostete SQL Server Datenbanken verwendet werden. Ebenso werden Dienste wie Skalierung und Überwachung und Optimierung dieser Datenbanken angeboten. [25] Der Zugriff auf eine SQL Server Datenbank mit C# stellt kein Problem dar und lässt sich mit dem Entity Framework lösen.

Azure Database for MySQL, MariaDB und PostgreSQL Azure bietet Datenbankserver für gängige Open-Source Datenbanksysteme an. Diese Datenbankserver beinhalten eine Hochverfügbarkeit und eine sekundenschnelle Skalierung. Ebenso ist Sicherheit ein grosses Thema, was von Azure abgedeckt wird. [6] MariaDB ist in Azure verfügbar [5] und wird von den Entwicklern von MySQL vorangetrieben und als MySQL-Ersatz angepriesen [1]. Als weiteres Datenbankmanagementsystem wird PostgreSQL von Azure angeboten [7]. Für all diese drei Datenbanksysteme bietet das Entity Framework zusammen mit verschiedenen Providern eine einfache Möglichkeit, Zugriffe aus C# auf die Datenbank zu implementieren.

SQL Server in Docker Container Für den Zweck der HSR Live Survey Plattform genügen nach Vergleich der SQL Server 2017 Editions die Express Version. Der grössten Vorteile der Express Version ist, dass sie im Vergleich zu den anderen Editionen geringere Systemvoraussetzungen aufweist [13] und zudem kostenlos ist [39]. Der SQL Server lässt sich sowohl als Linux als auch als Windows Container betreiben [33]. Die einzige Einschränkung bei Windows Containern ist, dass die Express Version die am höchsten unterstützte Version darstellt [47]. Für die persistente Speicherung von Daten wird ein Storage benötigt, dieser kann in Azure mit “Disk Storage” umgesetzt werden [10].

3.2.3. Produkte für NoSQL

Azure Cosmos DB Hinter der Azure Cosmos DB steckt eine NoSQL-Datenbank mit globaler Verteilung und horizontaler Skalierung im Herzen. Cosmos DB bietet auch bestehende APIs an wie MongoDB, Cassandra und sogar SQL an [4]. Zugriffe aus C# Code sind mit der von Microsoft zur Verfügung gestellten Library “Microsoft.Azure.DocumentDB” [29] möglich.

VM mit NoSQL Eine weitere Möglichkeit ist das Aufsetzen einer virtuellen Maschine inklusive dem Installieren einem NoSQL-Server, beispielsweise MongoDB [28]. Sicherlich der grösste Vorteil dieser Lösung ist die Flexibilität. Diese bietet aber auch einige Nachteile, da alles selber installiert und konfiguriert werden muss. Dies von dem Betriebssystem bis zur Anwendung, dabei muss beispielsweise auch die Firewall konfiguriert werden. Diese Konfiguration macht es schwierig, eine solche VM schnell zu skalieren. Die Anbindung

eines modernen NoSQL-Server in C# sollte dank verfügbaren Libraries [42] kein Problem sein.

NoSQL in Docker Container Analog dem VM Server in Docker Container könnte auch ein NoSQL Server in einem Docker Container betrieben werden. Als Beispiel könnte der von MongoDB zur Verfügung gestellt Docker Container verwendet werden.

SQL Server JSON Datentyp Es gibt weiter die Möglichkeit, den relationalen Teil und den NoSQL-Teil in einer Datenbank zusammen zu halten. Dabei wird der NoSQL-Teil als herkömmlicher String im JSON-Format abgespeichert, welcher im Code über einen JSON-Converter serialisiert und deserialisiert werden kann. Ein Vorteil davon ist, dass über die entsprechende SQL Server Version normale SQL Queries mit zusätzlichen JSON-Funktionen möglich sind [15]. Weiter sind die Referenzen zwischen relationalem und NoSQL-Teil einfacher zu gestalten, da eine Document Collection im eigentlichen Sinne eine relationale Tabelle darstellt und so Contraints darauf definiert werden können. NoSQL ist bekannt für sehr gute Skalierbarkeit, welches sich aber negativ auf die Datenkonsistenz auswirkt. Diese Problem ist auch bekannt unter dem CAP-Theorem [43]. Falls der NoSQL-Teil auf einer anderen klassischen NoSQL-Datenbankinstanz betrieben wird, verliert man die Kontrolle über die Datenintegrität zwischen relationalem Teil und NoSQL-Teil bzw. sind solche Mechanismen für die referentielle Integrität selbst zu implementieren. In der Variante JSON Daten im MS SQL Server zu speichern, ergeben sich wahrscheinlich mehr Documents, die aber bezüglich der Datengrösse kleiner sind im Vergleich zur klassischen NoSQL-Datenbankinstanz, wo grössere Teile des Datenmodells als NoSQL abgespeichert werden und so die Documents schneller wachsen bzw. von Anfang an schon grösser sind. Der Zugriff vom Code auf die Datenbank kann in dieser Variante mit dem Entity Framework als Library umgesetzt werden.

3.3. Frontend-Technologie

Die beliebtesten und am häufig verwendetsten Frontend-Technologien sind Angular, React und Vue.js [51]. Mit allen drei Technologien ist es möglich Single Page Applications zu entwickeln.

3.4. Kommunikation

3.4.1. Azure Service Bus

Azure Service Bus ist der Messaging Service von Microsoft. Darin können Queues und Topics erstellt, über die Nachrichten versendet und gelesen werden können.

3.4.1.1. Lokale Entwicklungsumgebung

Es gibt keinen Emulator, der lokal verwendet werden kann. Bei der Entwicklung müsste immer mit einem Service Bus in Azure gearbeitet werden.

3.4.1.2. Kosten

Um Topics verwenden zu können muss mindestens das Standard Tier [48] verwendet werden. Die monatlichen Kosten des Azure Service Bus sind in der Abbildung 25 aufgezeigt. Diese Kosten beinhalten 1000 gleichzeitige Verbindung und 10 Millionen Messages.

Microsoft Azure Estimate

Your Estimate

Service type	Region	Description	Estimated Cost
Service Bus	North Europe	Standard tier: 10, 1,000 brokered connection(s), 0 Hybrid Connect listener(s) + 0 overage per GB, 0 relay hour(s), 0 relay message(s)	CHF9.66
Support		Support Licensing Program	CHF0.00 Microsoft Online Services Program
		Monthly Total	CHF9.66
		Annual Total	CHF115.87

Disclaimer

All prices shown are in Swiss Franc. (chf). This is a summary estimate, not a quote. For up to date pricing information please visit https://azure.microsoft.com/pricing/calculator/
This estimate was created at 3/13/2019 12:39:41 PM UTC.

Abbildung 25.: Preisübersicht Service Bus in Azure
Quelle: Export Preisrechner Azure

3.4.1.3. Bidirektionale Kommunikation

Eine bidirektionale Kommunikation wäre mit Azure Service Bus theoretisch möglich, dafür müssten aber pro Richtung je eine Queue oder ein Topic erstellt werden. Das hätte zur Folge, dass pro theoretische Verbindung zwei Verbindungen offen gehalten werden müssen.

3.4.2. Vorteile

Azure Service Bus verwaltet Verbindungen, ist verteilt und bietet verschiedene Messaging-Funktionen wie Auto-Forwarding, Dead-Lettering, Scheduled-Delivery und weiteres an

[49].

3.4.3. Nachteile

Azure Service Bus ist an Azure gebunden und kann nicht bei einem anderen Cloud-Anbieter oder lokal betrieben werden.

3.4.4. SignalR

Es gibt zwei Möglichkeiten, SignalR zu verwenden. In beiden Fällen muss die App selber entwickelt und betrieben werden. Der Unterschied liegt in der Verwaltung von Verbindungen. Werden die Verbindungen selber verwaltet, gibt es bei der Skalierung von SignalR-Backends das Problem, dass die Verbindungen nicht immer bei demselben Server landen. Azure SignalR Services bietet ein Dienst an, der diese Verbindungen verwaltet und als Proxy zwischen Client und Server dient.

3.4.4.1. Lokale Entwicklungsumgebung

Eine eigene SignalR App kann lokal entwickelt und betrieben werden, da eine SignalR App nichts anderes als ein Teil einer .NET Core Web Applikation ist und diese sich direkt von Visual Studio starten lässt.

3.4.4.2. Kosten

Eigene Komponente Wenn die ganze Infrastruktur selber betrieben wird, sind neben den Kosten für das Hosting der SignalR App auch die Kosten der weiteren Infrastruktur, wie z. B. die des Load Balancers zu beachten.

Azure SignalR Service Die monatlichen Kosten für den Azure SignalR Service sind in Abbildung 26 ersichtlich. Diese Kosten beinhalten ein SignalR Standard Tier welches 1000 gleichzeitige Verbindungen und 1 Million Messages pro Tag an.

Microsoft Azure Estimate

Your Estimate

Service type	Region	Description	Estimated Cost
Azure SignalR Service	North Europe	Standard Tier, 1 Unit(s)	CHF48.19
Support		Support	CHF0.00
		Licensing Program	Microsoft Online Services Program
		Monthly Total	CHF48.19
		Annual Total	CHF578.31

Disclaimer

All prices shown are in Swiss Franc. (chf). This is a summary estimate, not a quote. For up to
This estimate was created at 3/13/2019 12:49:26 PM UTC.

Abbildung 26.: Preisübersicht SignalR Services in Azure
Quelle: Export Preisrechner Azure

3.4.4.3. Bidirektionale Kommunikation

Durch Verwendung von Websockets in SignalR sind die benötigten bidirektionalen Kommunikationswege abgedeckt.

3.4.5. Vorteile

Ein Vorteil ist, dass bei der Entwicklung kein Emulator oder ähnliches verwendet werden muss.

3.4.6. Nachteile

SignalR lässt TCP Connections offen, obwohl einige nicht gebraucht werden und dabei auf Idle gesetzt werden könnten. Skalierung ist ein Problem bei der Eigenentwicklung, diese kann aber mithilfe von anderen Produkten wie beispielsweise Redis umgesetzt werden.

3.5. Architektur / Deployment / Hosting

3.5.1. Azure Service Fabric

Azure Service Fabric ist ein Service von Microsoft der das Orchestrieren von Applikation in Container ermöglicht [9]. Mit dem Einsatz von Service Fabric muss sich der Entwickler nicht mehr um die Verwaltung von Virtual Machine, Speicher oder Netzwerk kümmern sondern nur um seine Applikationen und die Konfiguration der Umgebung.

3.5.1.1. Konzepte

Service Fabric kennt Cluster, Nodes, Named Applications und Named Services. Ein Cluster fasst mehrere Nodes zusammen, dabei ist ein Rechner oder VM eine Node. Eine Named Application fasst einen oder mehrere Named Service zusammen, die Funktionalität anbieten. Ein Service erfüllt dabei eine eigenständige Funktionalität und ist nicht direkt von anderen Services abhängig. Dieser besteht aus Code, Konfiguration und Daten. Ebenso können Container-Applikationen wie beispielsweise Docker-Container in Service Fabric eingebunden werden, was erlaubt beinahe alle möglichen Applikationen zu betreiben. [38]

3.5.1.2. Lokale Entwicklungsumgebung

Microsoft bietet die Möglichkeit, lokal ein Service Fabric Cluster zu betreiben. Dieser bietet auch eine einfache Möglichkeit, Applikation direkt aus dem Visual Studio zu deployen und auch zu debuggen. [31] Aktuell können aber nur Windows Container verwendet werden, was einen Einfluss auf das Deplyoment und Hosting hat.

3.5.1.3. Kosten

In Azure existiert mit Azure Service Fabric ein Dienst der die Verwaltung von Service Fabric Applikationen erlaubt. Für diesen Dienst fallen keine Kosten an, es müssen nur die verwendeten Nodes bezahlt werden. In der Abbildung 27 sind die monatlichen Kosten in einem Betrieb mit 3 Nodes und einem Storage von 32 GB, Transaktionskosten kommen hinzu und kosten pro 10'000 Transaktionen $0.0005 pro Monat [45].

Microsoft Azure Estimate

Your Estimate

Service type	Region	Description	Estimated Cost
Service Fabric	North Europe	Deployment Option: Service Fabric Cluster, Windows OS; Scale Set 1: 3 Virtual Machine(s); Pay as you go; Storage for Logs: 1 GB LRS data redundancy, 1 Service Fabric cluster(s) x 1 Months	CHF33.00
Storage	North Europe	Managed Disks, Standard HDD, S4 Disk Type 1 Disks	CHF1.51
Support		Support	CHF0.00
		Licensing Program	Microsoft Online Services Program
		Monthly Total	CHF34.51
		Annual Total	CHF414.11

Disclaimer

All prices shown are in Swiss Franc. (chf). This is a summary estimate, not a quote.
For up to date pricing information please visit https://azure.microsoft.com/pricing/calculator/
This estimate was created at 3/4/2019 3:06:03 PM UTC.

Abbildung 27.: Preisübersicht Service Fabric in Azure
Quelle: Export Preisrechner Azure

3.5.1.4. Vorteile

Wie Azure ist auch Azure Service Fabric von Microsoft und lässt sich deshalb sehr gut in die Welt von Azure einbinden. Zusätzlich kann eine Service Fabric Umgebung auch selber oder bei anderen Cloud-Anbietern betrieben werden, das ein späterer Umzug theoretisch möglich macht. Dadurch dass alle Container-Applikationen betrieben werden können, können auch Datenbanken so ausgeführt werden.

3.5.1.5. Nachteile

Service Fabric wird hauptsächlich von Microsoft entwickelt. Eine Integration in andere Cloud-Anbieter ist grundsätzlich möglich, dort müssen aber eigene virtuelle Maschinen installiert und betrieben werden, was einen grossen Nachteil bezüglich Konfiguration und Skalierbarkeit mit sich bringt.

3.5.2. Kubernetes

Kubernetes ist ein Open-Source Plattform für das Verwalten und Orchestrieren von Applikationen in Containern. Initial wurde Kubernetes von Google entwickelt und wird

auch von Google eingesetzt. [32]

3.5.2.1. Konzepte

Kubernetes kennt Services, Pods, Cluster und Nodes. Ein Pod ist die kleinste Einheit und kann als Prozess verstanden werden, der im "Betriebssystem" Kubernetes läuft. Als bekanntestes Modell des Deployments, wird in einem Pod nur ein Container laufen gelassen, es gibt aber die Möglichkeit, zusammengehörige Container im gleichen Pod zu betreiben. Services sind den Pods übergeordnet und bieten diesen ein Interface nach aussen an. Nodes sind die kleinste Einheit einer Computer-Hardware in Kubernetes. Eine Node ist oft ein physischer oder virtueller Rechner, theoretisch kann aber fast alles verwendet werden, also beispielsweise auch ein Raspberry Pi. Nodes werden zusammengefasst in einem Cluster. Als Entwickler muss man sich eher auf die Cluster konzentrieren, da Kubernetes die Nodes in einem Cluster selber verwaltet und laufende Pods dynamisch auf Nodes verteilt. [16]

3.5.2.2. Lokale Entwicklungsumgebung

Kubernetes kann lokal mit Minikube [37] aufgesetzt und betrieben werden. Diese Installation kann nur mit einen Cluster mit einem Node eingerichtet werden, dies hat auf die Entwicklung aber keinen direkten Einfluss.

3.5.2.3. Kosten

In Azure existiert mit dem Azure Kubernetes Service (AKS) ein Angebot, mit welchem Kubernetes betrieben werden kann [8]. Dieses Angebot ist gratis, Kosten entstehen nur für die verwendeten Nodes. In der Abbildung 28 sind die monatlichen Kosten aufgelistet, mit denen gerechnet werden muss. In dieser Rechnung sind drei Nodes enthalten, welche den ganzen Monat laufen und ein Harddisk-Speicher von 32 GB besitzen. Transaktionskosten kommen hinzu und kosten pro 10'000 Transaktionen $0.0005 pro Monat. [45]

Microsoft Azure Estimate

Your Estimate

Service type	Region	Description	Estimated Cost
Azure Kubernetes Service (AKS)	North Europe	3 B1S (1 vCPU(s), 1 GB RAM) nodes x 1 Months; Pay as you go; 1 managed OS disks – S4	CHF25.87
Support		Support	CHF0.00
		Licensing Program	Microsoft Online Services Program (MOSP)
		Monthly Total	CHF25.87
		Annual Total	CHF310.38

Disclaimer

All prices shown are in Swiss Franc. (chf). This is a summary estimate, not a quote.
For up to date pricing information please visit https://azure.microsoft.com/pricing/calculator/
This estimate was created at 3/4/2019 3:08:29 PM UTC.

Abbildung 28.: Preisübersicht Kubernetes in Azure
Quelle: Export Preisrechner Azure

3.5.2.4. Vorteile

Kubernetes ist kein Produkt von Microsoft und bereits weit verbreitet. Dadurch wäre ein Wechsel von Azure auf einen anderen Anbieter in Zukunft einfach möglich.

3.5.2.5. Nachteile

Die Integration von Kubernetes in Azure ist relativ gut und funktionierte in der Evaluation auch. Es ist aber denkbar, dass Microsoft vorzugsweise seine eigenen Produkte vorantreiben wird und Kubernetes deshalb etwas ausser Acht lassen wird.

3.6. Entscheidungen

3.6.1. Datenbank

SQL Damit die Datenbank nicht als eigene Komponente in Azure betrieben werden muss, sondern in den Kubernetes-Cluster eingebaut werden kann, fiel die Entscheidung auf den SQL Server als Docker Container.

NoSQL Alle Anforderungen können mit dem SQL Server JSON Datentyp umgesetzt werden, deshalb wir diese Technologie für den NoSQL-Teil der Datenbank verwendet.

Diese Entscheidung erleichtert auch das Deployment, denn es muss nur eine Datenbank betrieben werden.

3.6.2. Frontend-Technologie

In Absprache mit Manuel Bauer wurde React als Frontend-Technologie ausgewählt.

3.6.3. Kommunikation

Da besonders viele bidirektionale Kommunikation verwendet wird, fällt der Azure Service Bus weg, da bei diesem eine solche Kommunikation komplizierter ist und selber verwaltet werden muss. Damit SignalR ebenfalls in das Hosting-Model von Docker Container passt und keine direkte Abhängigkeit zu Azure mit SignalR Services besteht, wird die SignalR-Komponente selber entwickelt und als Docker Container ausgeführt. Damit eine Skalierung dieser Komponente möglich ist, wird Redis zur Verteilung und Verwaltung der Connections verwendet.

3.6.4. Architektur / Deployment / Hosting

Die Entscheidung fällt auf Kubernetes zusammen mit Azure Kubernetes Service. Kubernetes existiert bereits einige Zeit länger als Service Fabric und ist auch bekannter und aufstrebender. Dies ist als Beispiel bei Google Trends ersichtlich, ein Chart daraus ist in der Abbildung 29 abgebildet. Auch bei den Kosten ist Kubernetes auf Azure etwas günstiger als Service Fabric.

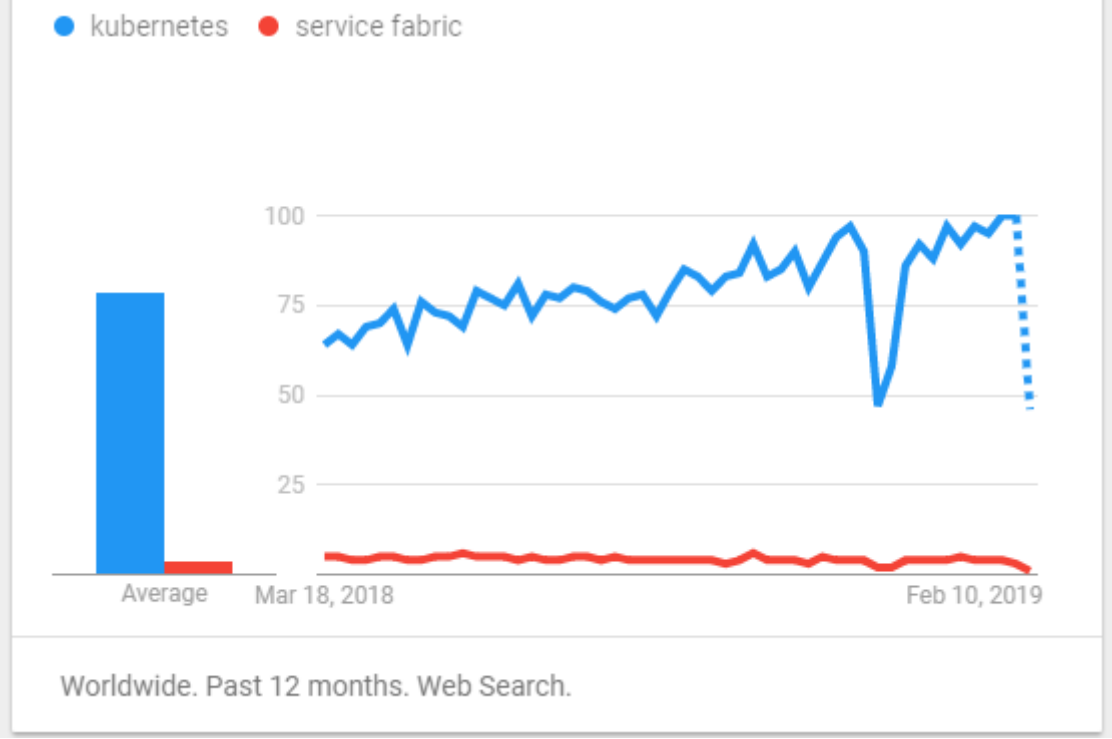


Abbildung 29.: Google Trend Analyse zwischen Kubernetes und Service Fabric
Quelle: Printscreen Google Trends

3.6.5. Kosten

Um ein ungefähres Gefühl für die endgültigen Kosten zu bekommen, wurden diese im Azure Price Calculator berechnet, das Resultat ist in der Abbildung 30 ersichtlich. Es sind zwei Kostenpunkte zu berücksichtigen, zum einen die Nodes für den Kubernetes Cluster und zum anderen der Speicher für die Daten der Datenbank. In der Rechnung wurde ein Node ausgewählt, da dieser die anfängliche Last alleine tragen kann. Beim Speicher wurde eine 32 GB HDD ausgewählt mit 100 Millionen Transaktionen im Monat, dies sollte zu Beginn ausreichen.

Microsoft Azure Estimate

Your Estimate

Service type	Region	Description	Estimated Cost
Azure Kubernetes Service (AKS)	West Europe	1 B2S (2 vCPU(s), 4 GB RAM) nodes x 1 Months; Pay as you go; 0 managed OS disks – S4	CHF34.48
Storage	West Europe	Managed Disks, Standard HDD, S4 Disk Type 1 Disks	CHF6.43
Support		**Support**	CHF0.00
		Licensing Program	**Microsoft Online Services Program**
		Monthly Total	**CHF40.91**
		Annual Total	**CHF490.98**

Disclaimer

All prices shown are in Swiss Franc. (chf). This is a summary estimate, not a quote. For up to date pricing information please visit https://azure.microsoft.com/pricing/calculator/
This estimate was created at 3/18/2019 12:08:09 PM UTC.

Abbildung 30.: Total monatliche Kosten für Azure Services
Quelle: Export Preisrechner Azure

4. Architekturspezifikation

4.1. Einführung

Dieser Abschnitt beschreibt die Architektur der Umsetzung des Projektes HSR Live Survey, welches im Rahmen der Bachelorarbeit an der HSR umgesetzt wird.

4.2. Systemübersicht

Das Gesamtsystem in Abbildung 31 ist grundlegend in drei Kontexte unterteilt, wobei nur die beiden letzteren architektonisch relevant sind und der erste Kontext nur zum Verständnis in der Übersicht vorhanden ist:

- Browser
- Kubernetes Cluster
- Azure

Browser Die Benutzer greifen via Browser auf die jeweiligen User Interfaces der Plattform zu.

Kubernetes Cluster Der Kern des System befindet sich im Kubernetes Cluster, in der die verschiedenen Microservices jeweils als Docker Container betrieben werden.

Azure Die einzige Komponente, die sich ausserhalb des Clusters befindet, ist der Storage von Azure für die SQL Datenbank.

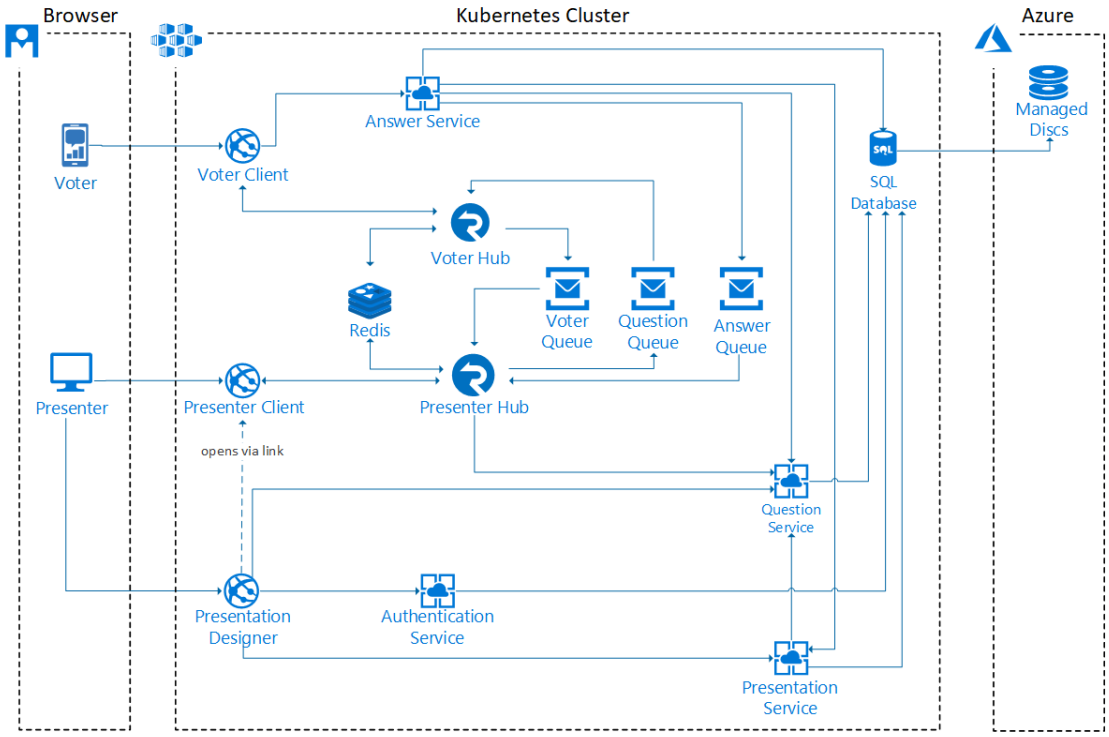


Abbildung 31.: Systemübersicht
Quelle: Eigene Darstellung

4.2.1. Frontend

Es existieren drei User Interfaces, die jeweils als Single Page Applications im Browser laufen.
Folgende Frontends sind für die Präsentationsdurchführung zuständig:

- Voter Client
- Presenter Client

Für die Verwaltung der Präsentationen ist folgende SPA verantwortlich:

- Presentation Designer

Die Verbindung zwischen Presentation Designer und Presenter Client ist als indirekte Abhängigkeit zu betrachten, da die Ausführung einer Präsentation aus dem Designer gestartet wird.
Die jeweils verwendeten Libararies sind unter Kapitel 4.8.1 zu finden.

4.2.2. Backend

Presentation Service Dieser Service übernimmt Aufgaben, welche die Verwaltung von Präsentationen betreffen.

Authentication Service Der Authentication Service ist verantwortlich für alle relevanten Operationen wie Login oder Registrierung, die den Benutzer betreffen.

Question Service Dieser Microservice kümmert sich darum, Fragen zu laden oder zu speichern.

Answer Service Der Service persistiert die abgegebenen Antworten und verschickt Messages mit den Antworten, die in einer Queue landen.

Voter Hub Der Voter Hub liest Fragen aus einer Queue. Diese werden an die entsprechenden Participants verschickt.

Presenter Hub Diese SignalR Komponente liest die nächste Frage aus der Datenbank aus und verschickt sie an den Presenter und Voter Client. Erhaltene Antworten werden aus einer Queue ausgelesen und entsprechend verarbeitet.

Redis Eine Redis Datenbank speichert und verwaltet im Hintergrund die jeweiligen Hub-Connections, damit die SignalR Hubs skalierbar sind.

Die Dependencies sind unter Kapitel 4.8.2 zusammengestellt.

4.3. Architektonische Ziele & Einschränkungen

4.3.1. Ziele

Mit dem Einsatz einer Microservice-Architektur sind die Ziele zu einem Teil vorgegeben. Services sollen klein, modular und nicht abhängig von anderen Services sein. Sollte eine solche Abhängigkeit trotzdem benötigt werden, muss diese möglichst asynchron implementiert werden. Sollten Daten sofort verfügbar sein, so muss dieser Zugriff abstrahiert implementiert werden, damit dieser in Zukunft ohne grossen Aufwand abgeändert werden kann.

Die Frontends sollen möglichst modular entwickelt werden, damit die einzelnen Komponenten wiederverwendbar sind. Komplexe Logik wie beispielsweise das Validieren von Antworten soll nicht von den Frontends erledigt werden sondern durch die Services. So bleiben die Frontends schmal und beinhalten keine komplexe Logik.

4.3.2. Einschränkungen

4.3.2.1. Nicht-Funktionale Anforderungen

4.3.2.2. Skalierbarkeit

Jeder Service soll so implementiert werden, dass dieser skalierbar ist. Eine Skalierung soll vertikal, aber besonders auch horizontal möglich sein. Das bedeutet, dass die Services wenn immer möglich Stateless sein sollen. Sollte dies nicht möglich sein, muss eine Lösung dafür gesucht oder eine Alternative verwendet werden.

4.3.2.3. Performance

Daten gehören einem einzelnen Service. Dieser bietet ein API an, um an diese Daten zu gelangen. Es werden also einige Requests benötigt, um in den einzelnen Services die Arbeit erledigen zu können. Bei der Entwicklung soll von Beginn an auf mögliche Performance-Probleme geachtet, aber noch keine Lösung eingebaut werden. Denn Lösungen wie Caching von Daten sind komplex und sollen erst implementiert werden, wenn es nicht ohne geht. So kann zum einen der Aufwand für die Implementation, zum anderen die Komplexität des Codes reduziert werden.

4.3.2.4. Security

HSR Live Survey wird zu Beginn nur innerhalb der HSR von Dozenten und Studenten verwendet. Es kann also davon ausgegangen werden, dass niemand dem anderen Schaden möchte. Trotzdem sollen grundlegende Sicherheitsaspekte wie das Speichern von Passwörtern als Hash oder die Autorisierung von Benutzern umgesetzt werden. Es werden folgende Grundsätze verfolgt:

- Keine sensiblen Benutzerdaten als Klartext gespeichert
- Keine sensiblen Benutzerdaten als Klartext in Log
- Zugriff nur auf eigene Daten möglich
- Jeder Zugriff auf Daten nur mit Autorisierung, sofern dieser von extern möglich ist
- Validierung und Bereinigung von Input-Daten

4.4. Logische Architektur

4.4.1. Allgemeine Architektur

Die Architektur vieler Services sind sehr ähnlich, da alle im Grundsatz dieselbe Funktion erfüllen müssen, nur für einen anderen Teil der Domäne. Deshalb wird die Beschreibung dieser Architektur in diesem Abschnitt zusammengefasst und in den servicespezifischen

Unterkapiteln nur auf die Eigenheiten eingegangen.

Im allgemeinen kann zwischen drei Arten von Applikationstypen unterschieden werden:

- Frontend Applikation
- SignalR Hub
- Backend Service

Jede einzelne Art hat seine eigene Architektur. Diese sind nachfolgend beschrieben.

4.4.1.1. Allgemeines

Layering Alle Applikationen, insbesondere aber die Backend Applikationen, wurden nach dem Layering-Prinzip konzipiert. Dabei wurde auf ein Strict-Layering verzichtet, es dürfen gewisse Layers also auch übersprungen werden. Dies um unnötigen Code, der nur als Durchlauf verwendet wird, zu vermeiden. Der Zugriff auf einen höher angesiedelten Layer ist aber verboten. Die Frontend Applikation beinhalten ebenfalls ein gewisses Layering, dieses ist aber nicht so klar wie in den Services.

4.4.1.2. Frontend Applikation

Ein beispielhaftes Package-Diagram für eine Frontend Applikation ist in der Abbildung 32 ersichtlich. Nachfolgend werden einige wichtige, allgemeingültige Konzepte beschrieben, welche mit den Best Practices von Mirko Stocker (Verantwortlicher React im Modul Web Engineering & Design 3) abgestimmt sind [36].

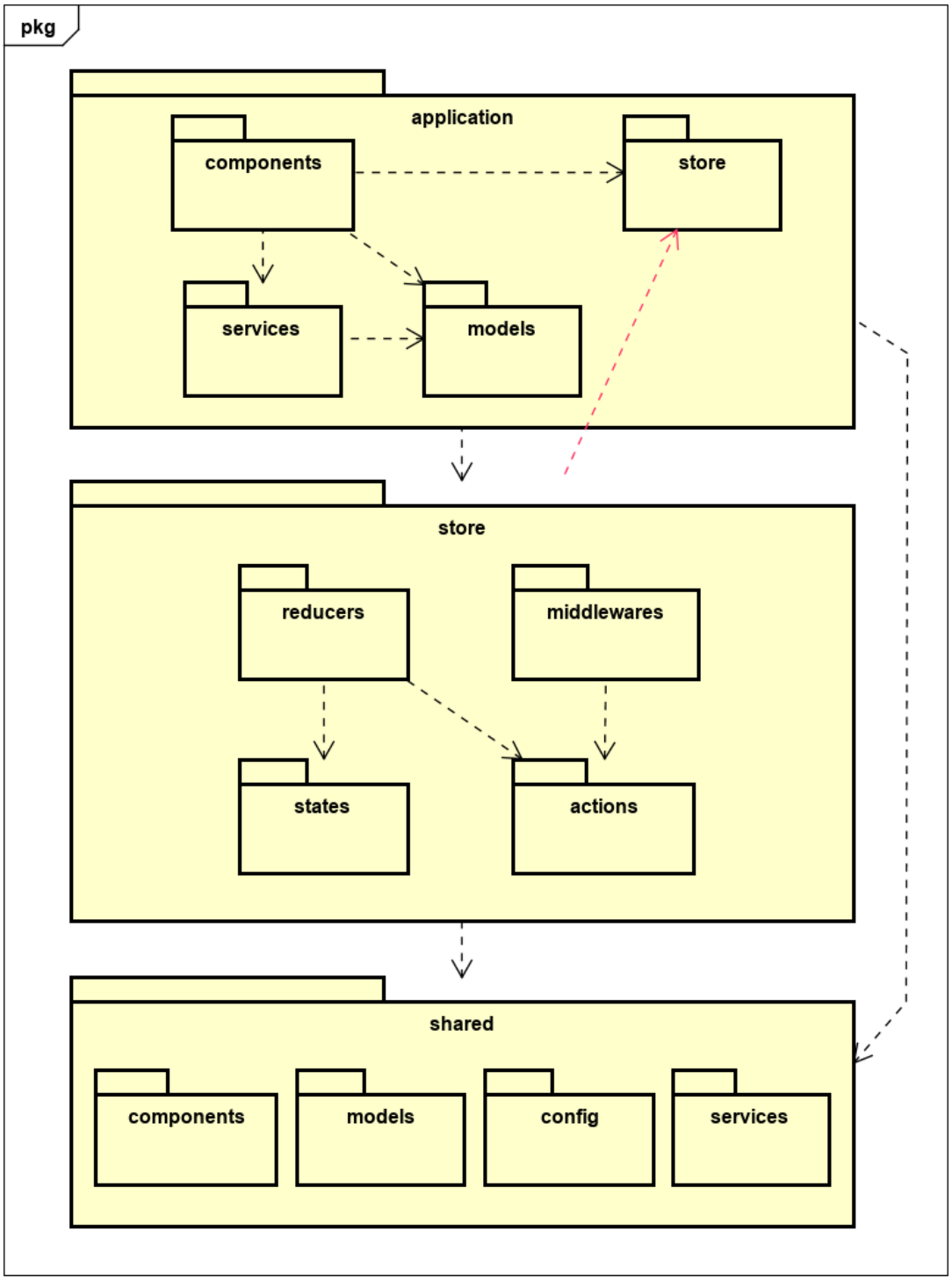


Abbildung 32.: Package-Diagramm einer Frontend Applikation
Quelle: Eigene Darstellung

application In diesem Package befinden sich alle relevanten Anzeigekomponenten und Container, welche spezifisch für Teile der Applikation sind.

store in application Jedes Feature besitzt ein store-Package, welches die zusammengehörigen Redux-Komponenten zusammenfasst. Dabei wird dieses vom globalen Store nur eingebunden, da der Root-Reducer den entsprechenden Reducer deklarieren muss. Dies wird im Diagramm der Abbildung **??** durch den roten Pfeil symbolisiert.

store In diesem Package sind alle Actions, Reducers und States von Redux zusammengefasst. Ausserdem befindet sich hier die SignalR Middleware, welche für die Connection zum SignalR Hub verantwortlich ist.

shared Dieses Package beinhaltet alle wiederverwendbaren Komponenten, Models und zusätzlich die Konfigurationseinstellungen der App.

Layering Ein Layering findet konkret in jedem Applikationsspezifischem Package statt. Dort existiert mit den Components ein UI-Layer gemischt mit dem Business Layer, ein DataAccess-Layer mit den Services und ein Domain-Layer mit den Models. Der UI und Business-Layer ist gemischt, da in React der Code für die View und die Logik in einem File gehandhabt wird.

4.4.1.3. SignalR Hub

Die Architektur der SignalR Hubs unterscheidet sich besonders im Applikations-Layer von den Backend Services. Hier existieren keine Controller sondern Hubs. Das Package Diagramm eines solcher SignalR Hubs ist in Abbildung 33 abgebildet. Die Layers *Business* und *DataAccess* sind in den SignalR Hubs sehr dünn, da die Hubs mehr als Durchlauf zwischen Services und Clients dienen und an sich keine grosse Logik besitzen. Nachfolgend werden einige wichtige, allgemeingültige Konzepte beschrieben.

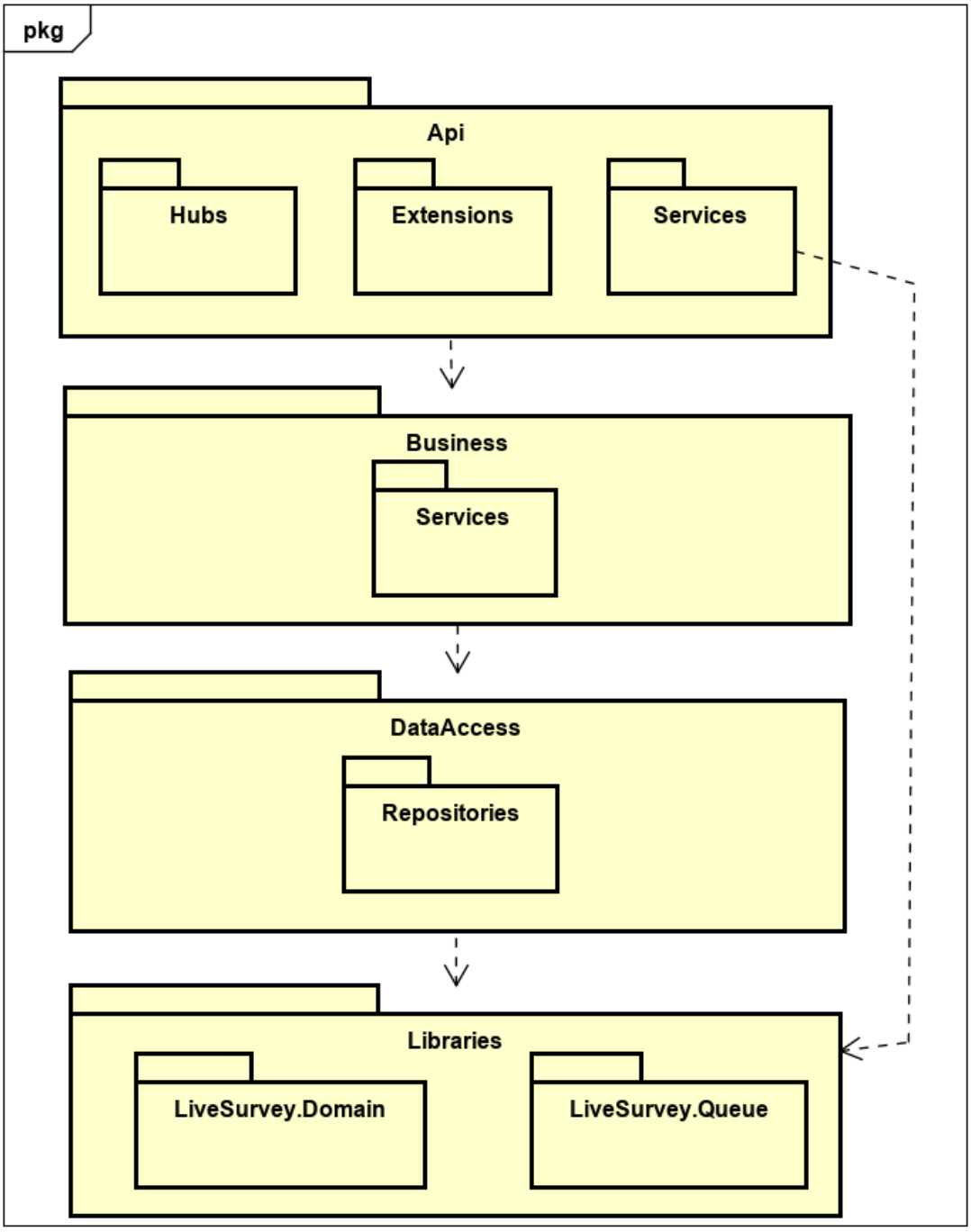


Abbildung 33.: Package-Diagramm eines SignalR Hubs
Quelle: Eigene Darstellung

Hosted Services In ASP.NET Core gibt es mit Hosted Services die Möglichkeit, Hintergrundjobs auszuführen. Ein solcher Service wird benötigt, um neue Nachrichten aus der Queue an die verbundenen Clients des Hubs zu senden. Dies kann nicht direkt im Hub implementiert werden, da die Instanz eines solchen direkt nach einem Request disposed wird und darum die asynchrone Nachricht aus der Queue nicht mehr verarbeiten kann.

Hosted Service und Queue Der Hosted Service geht nicht via Business-Layer auf die Queue, sondern direkt (siehe Abbildung 33, Abhängigkeit a). Dies wurde so umgesetzt, da ansonsten das Eintreffen von neuen Messages in der Queue durch alle Layers hätte durchgezogen werden müssen. Ebenso ist der Hosted Service eine sehr abgekapselte Komponente in der Applikation und so können Anpassungen bei Änderungen sehr spezifisch an einem Ort durchgeführt werden, weshalb auf ein konsequentes Layering verzichtet werden kann.

Kommunikation zwischen Queue und Hub Die im Konzept des Hosted Services beschriebene Problematik und Lösung beinhaltet einige Komponenten. Der Beispielablauf anhand des Voter Client und Voter Hub bei einer neuen Nachricht aus der Queue ist in der Abbildung 34 ersichtlich. Das Dispose im Schritt 4 ist der Grund, weshalb der Hosted Service verwendet werden muss. Über diesen wird im Schritt 8 via Hub eine Nachricht an den Voter Client gesendet.

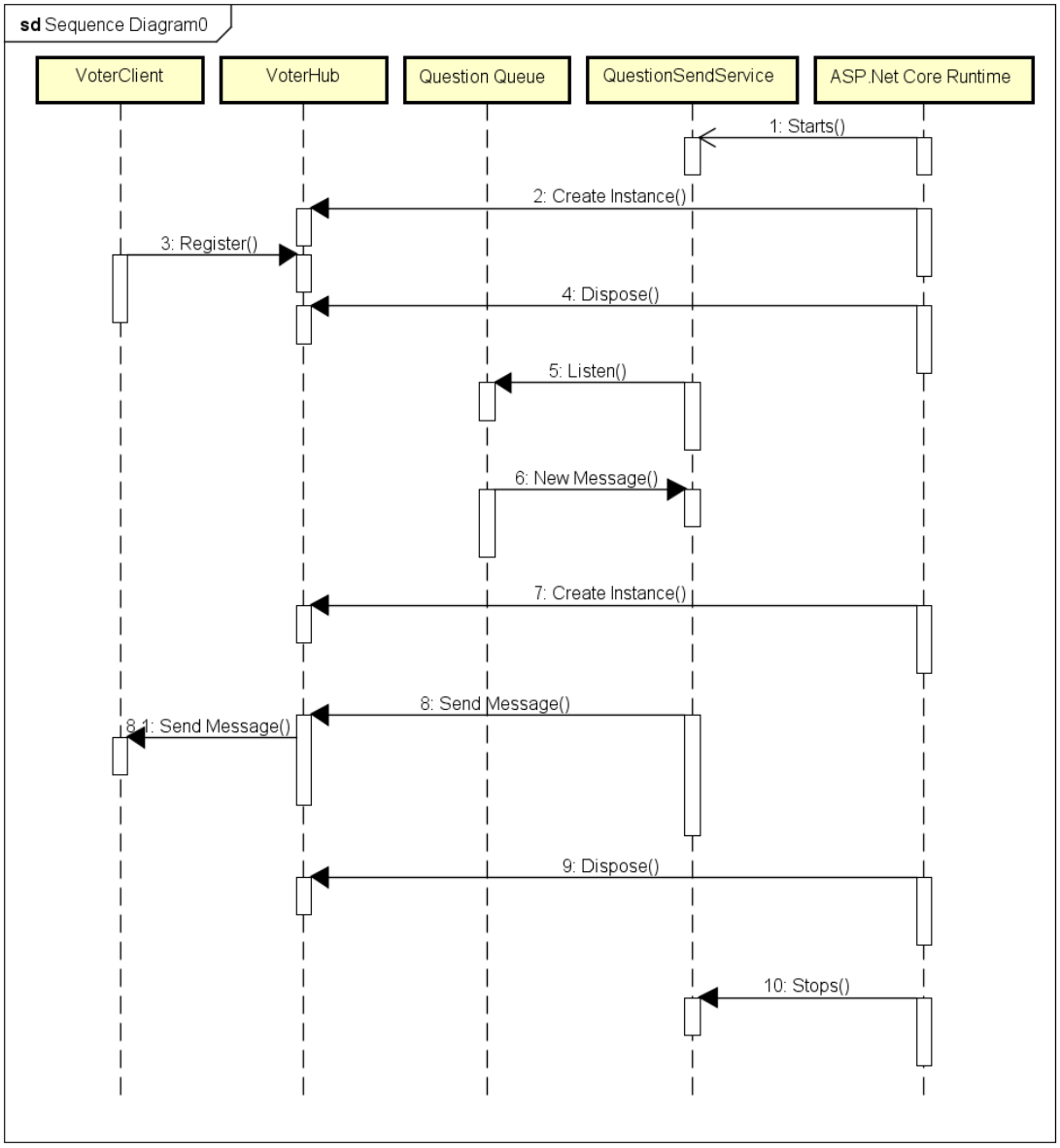


Abbildung 34.: Ablauf der Kommunikation zwischen Queue und Client via Hosted Service
Quelle: Eigene Darstellung

4.4.1.4. Backend Service

In der Abbildung 35 ist das Package Diagramm eines Backend Services ersichtlich. Nachfolgend werden einige wichtige Konzepte beschrieben.

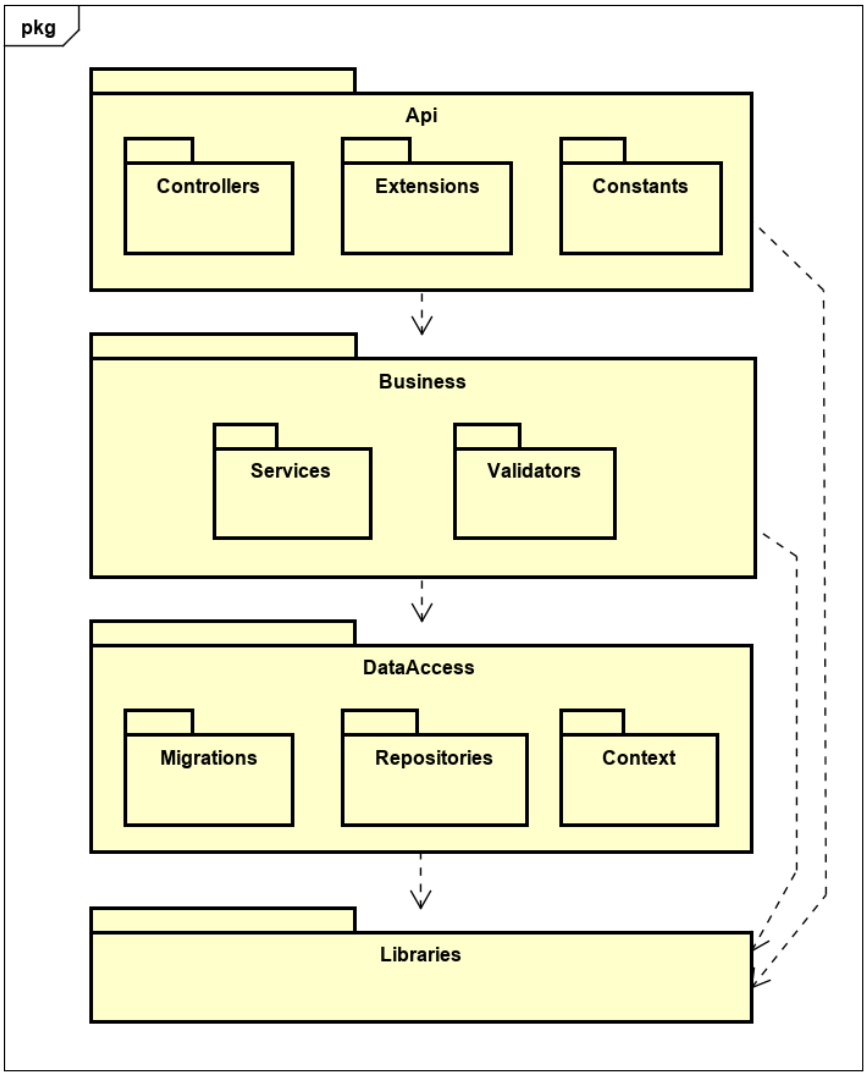


Abbildung 35.: Package-Diagramm eines Backend Service
Quelle: Eigene Darstellung

4.4.1.5. Wichtige Konzepte

Validators Im Package "Validators" sind die Validatoren für die Business-Objekte implementiert. Diese sind bewusst auf dem Business-Layer angesiedelt, da es sich bei der Validierung um Business-Regeln handelt und diese nicht vom Übertragungskanal, hier HTTP im Api-Layer, beeinflusst sind.

Repositories Um den Datenzugriff für den Business-Layer abkapseln zu können sind diese Zugriffe in Repositories implementiert. Nach aussen kann immer das Interface

verwendet werden, unabhängig davon auf welche Datenbasis Abfragen durchgeführt werden. Dies lässt eine einfache Änderung der Datenbasis zu, beispielsweise falls ein weiterer Microservice für den benötigten Teil hinzukommt.

Migrations Um das Schema der Datenbank ändern zu können, wird das Prinzip von Migrations vom Entity Framework verwendet. Diese Migrations sind als C#-Klassen implementiert, welche in diesem Package liegen.

Libraries Je nach Service werden unterschiedliche Libraries verwendet. Diese sind hier nicht explizit aufgelistet, in Kapitel 4.8 - Libraries und Frameworks existiert aber eine Matrix welche die Verwendung aufzeigt.

Extensions Für die Registrierung und Konfiguration von Services während des Startups wurden Extensions Methods erstellt. Diese abstrahieren die Konfiguration einzelner Teile und halten die Startup-Klasse schlank. Diese Extensions sind bei vielen Services gleich oder sehr ähnlich, trotzdem wurden diese in jedem Service implementiert und nicht ausgelagert. Dies stellt kein Problem der Code-Duplizierung dar, da diese Klassen nicht viel Logik beinhalten und die Konfiguration einem Service gehört und so punktuell angepasst werden kann.

4.4.2. Authentication Service

In der Abbildung 36 ist das vereinfachte Package-Diagramm des Authentication Service sichtbar. In grau hervorgehoben sind die Unterschiede zur allgemeinen Architektur. Gemeinsame Teile wurden zugunsten der Übersichtlichkeit weggelassen.

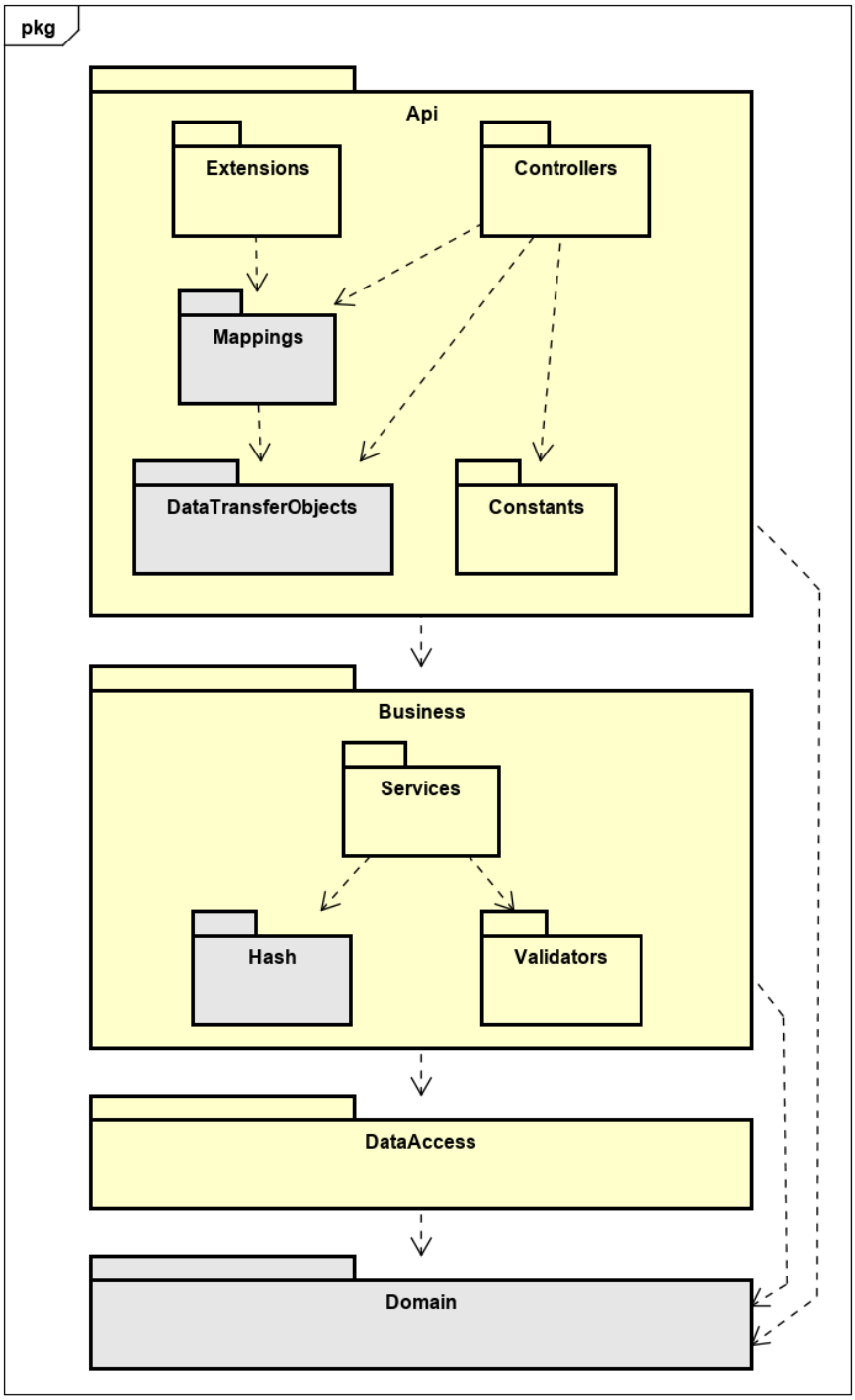


Abbildung 36.: Package-Diagramm AuthenticationService
Quelle: Eigene Darstellung

4.4.2.1. Wichtige Konzepte

Domain Der Authentication Service hat einen eigenen Domain-Layer, da die Klassen sehr spezifisch mit Autorisierung und Benutzerverwaltung zu tun haben und somit in anderen Services nicht benötigt werden.

DataTransferObjects Analog der eigenen Domain-Klassen existieren eigene DTOs.

Mappings In diesem Package sind die Mappings von den Domain Objects zu den Data Transfer Objects definiert.

Hash Um die Passwörter bei der Registrierung zu hashen und beim Login zu verifizieren, sind die entsprechenden Funktionen im Package "Hash" implementiert. Dabei wurde die Austauschbarkeit der Implementation beachtet.

4.4.3. Presentation Service

Der Presentation Service weicht nicht von der Standardarchitektur für Backend Services ab.

4.4.4. Answer Service

Der Answer Service weicht nicht von der Standardarchitektur für Backend Services ab.

4.4.5. Question Service

Der Question Service weicht nicht von der Standardarchitektur für Backend Services ab.

4.4.6. Voter Hub

Die Architektur des Voter Hub weicht nicht von der Standarcharchitektur für SignalR-Hubs ab.

4.4.7. Presenter Hub

Die Architektur des Presenter Hub weicht nicht von der Standarcharchitektur für SignalR-Hubs ab.

4.4.8. Presentation Designer

In der Abbildung 37 ist das Package Diagramm des Presentation Designers ersichtlich. Grau ersichtlich sind die von der allgemeinen Architektur abweichenden Komponenten.

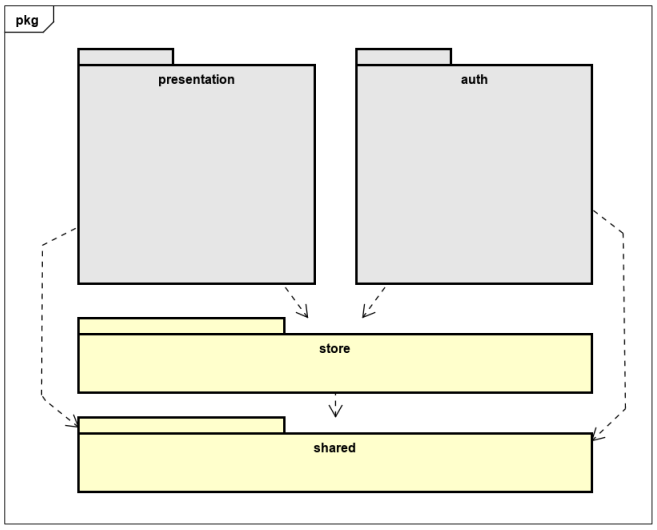


Abbildung 37.: Package-Diagramm PresentationDesigner
Quelle: Eigene Darstellung

Der Unterschied zur allgemeinen Architektur besteht darin, dass diese Applikation nicht nur aus einem Applikations-Teil besteht, sondern aus zwei. Diese beiden sind jedoch wie in der allgemeinen Architektur aufgebaut.

4.4.8.1. Wichtige Konzepte

presentation In diesem Package befinden sich alle relevanten Anzeigekomponenten und Container, welche mit der Präsentation in Zusammenhang stehen.

auth In diesem Package befinden sich alle relevanten Anzeigekomponenten und Container, welche mit der Autorisierung und der Benutzerverwaltung in Zusammenhang stehen.

4.4.9. Presenter Client

Der PresenterClient weicht nicht von der Standardarchitektur für Frontend Applikationen ab. Das Package *application* entspricht hier dem Package *presentation*.

4.4.10. Voter Client

Der VoterClient weicht nicht von der Standardarchitektur für Frontend Applikationen ab. Das Package *application* entspricht hier dem Package *participation*.

4.5. Deployment

Der folgende Abschnitt zeigt auf, wie die einzelnen Komponenten der Plattform HSR Live Survey gehostet werden.

4.5.1. Deployment Diagramm

In der Abbildung 38 ist das Deployment Diagramm ersichtlich. Die Abbildung 39 zeigt die Legende für das Deployment Diagramm.

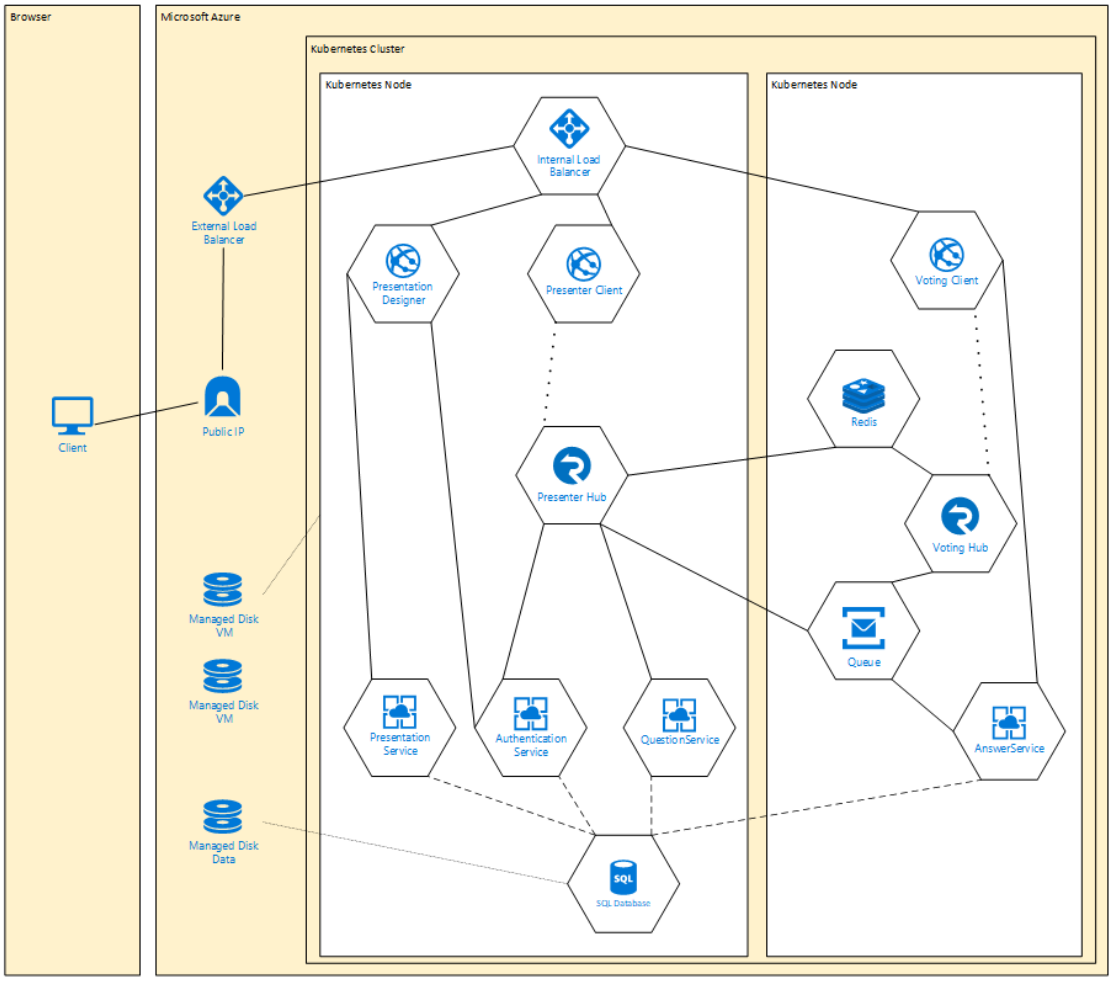


Abbildung 38.: Deployment Diagramm
Quelle: Eigene Darstellung

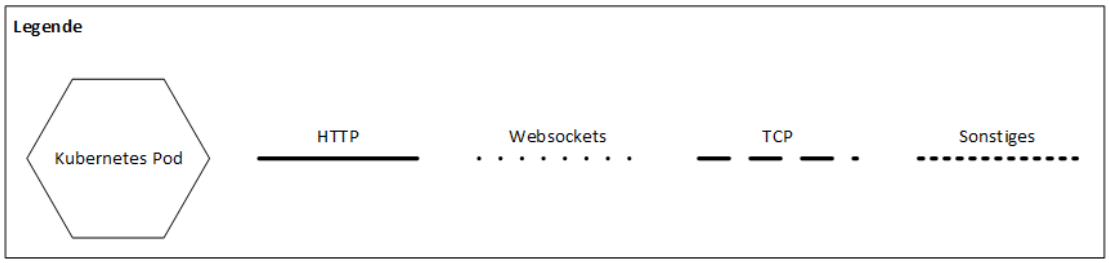


Abbildung 39.: Legende zu Deployment Diagramm
Quelle: Eigene Darstellung

4.5.1.1. Erklärungen

Client Der Client stellt den Benutzer der Applikation dar. Alle User Interfaces laufen dabei beim Client im Browser.

Public IP Die Public IP wird von Azure zur Verfügung gestellt und zeigt auf den External Load Balancer.

External Load Balancer Der externe Load Balancer von Azure leitet alle Anfragen an den Kubernetes Cluster weiter.

Internal Load Balancer Innerhalb des Kubernetes Cluster wird ein zweiter Load Balancer eingesetzt. Dieser kennt die interne Struktur des Clusters und leitet alle Anfragen nach gewissen Regeln an die internen Services und Applikationen weiter.

Kubernetes Cluster Der Kubernetes Cluster besteht aus mindestens einem Node. Aus Gründen der Verfügbarkeit und Skalierbarkeit macht es aber Sinn, mindestens zwei Nodes bereitzustellen.

Kubernetes Node Eine Kubernetes Node ist in Azure eine virtuelle Maschine. Zu jeder virtuellen Maschine gehört auch eine Managed Disk. Abgebildet sind zwei Nodes, dies kann zur Laufzeit aber dynamisch festgelegt werden und kann und soll sich je nach Last ändern.

Kubernetes Pub In einem Kubernetes Pod laufen Services und Applikationen. Diese laufen jedoch nicht direkt in dem Pod sondern in einem Docker Container, welcher vom Pod verwaltet wird.

Managed Disk Data Die SQL Datenbank läuft ebenfalls in einem Docker Container, welcher von Kubernetes gemanaged wird. Über den Lifecycle dieses Docker Containers hat von aussen niemand direkten Einfluss. Damit die Daten der Datenbank auch nach

der Neuerstellung dieses Docker Containers zur Verfügung stehen, werden diese in einer Managed Disk in Azure persistiert.

4.5.2. Anforderungen

Einzig der SQL Server hat konkrete Anforderungen an die Infrastruktur. Ein SQL Server benötigt im Minimum 2 GB an Arbeitsspeicher und 2 GB an Festplattenspeicher [33]. Diese müssen im Node, auf dem der SQL Server läuft, zur Verfügung stehen. Die Anforderungen werden automatisch beim Deployment via kubectl an den Node gestellt.

4.5.3. Verteilung der Pods in Nodes

Die Verteilung der Pods auf die zwei abgebildeten Nodes sind hier exemplarisch dargestellt. Diese Verteilung managed der Kubernetes Cluster, auf diese kann über gewisse Constraints in der Konfiguration indirekten Einfluss genommen werden [3]. Auf die explizite Konfiguration der Podverteilung wird verzichtet.

4.5.4. Struktur der Domains

Um die Erreichbarkeit der Komponenten sicherzustellen, sind diese vom Internet erreichbar. Für die Auflösung der Domain zu den Services ist Traefik verantwortlich. Es existiert eine Struktur in der Vergabe der Domains.

Frontends Jedes Frontend erhält eine eigene Subdomain. Diese ist mit einem passenden Verb der Aktivität, die an diesem Frontend durchgeführt wird, beschrieben. Die Tabelle 7 zeigt die aktuelle Struktur anhand der Beispieldomain *livesurvey.ch* auf.

Frontend	**Domain**
Presentation Designer	design.livesurvey.ch
Presenter Client	present.livesurvey.ch
Voter Client	vote.livesurvey.ch

Tabelle 7.: Struktur der Domains der Frontends

Services Alle Services werden auf dieselbe Subdomain mit dem Präfix *api* konfiguriert. Die Services werden anhand des Pfades nach der Domain identifiziert. Dieser Pfad beinhaltet den Service-Namen. Die Tabelle 8 zeigt die aktuelle Struktur anhand der Beispieldomain *livesurvey.ch* auf.

Service	**Domain**
Answer Service	api.livesurvey.ch/answerservice
Authentication Service	api.livesurvey.ch/authenticationservice
Presentation Service	api.livesurvey.ch/presentationservice
Presenter Hub	api.livesurvey.ch/presenterhub
Question Service	api.livesurvey.ch/questionservice
Voter Hub	api.livesurvey.ch/voterhub

Tabelle 8.: Struktur der Domains der Services

Infrastruktur Um die Infrastruktur zu überwachen, existieren einige Hilfsmittel wie beispielsweise das Dashboard von RabbitMQ oder Traefik. Diese erhalten eine eigene Subdomain, beschrieben mit der Technologie oder der Anwendung. Die Tabelle 9 zeigt die aktuelle Struktur anhand der Beispieldomain *livesurvey.ch* auf.

Infrastruktur	**Domain**
RabbitMQ Dashboard	rabbitmq.livesurvey.ch
Traefik Dashboard	traefik-ui.livesurvey.ch

Tabelle 9.: Struktur der Domains der Infrastruktur

4.6. Datenspeicherung

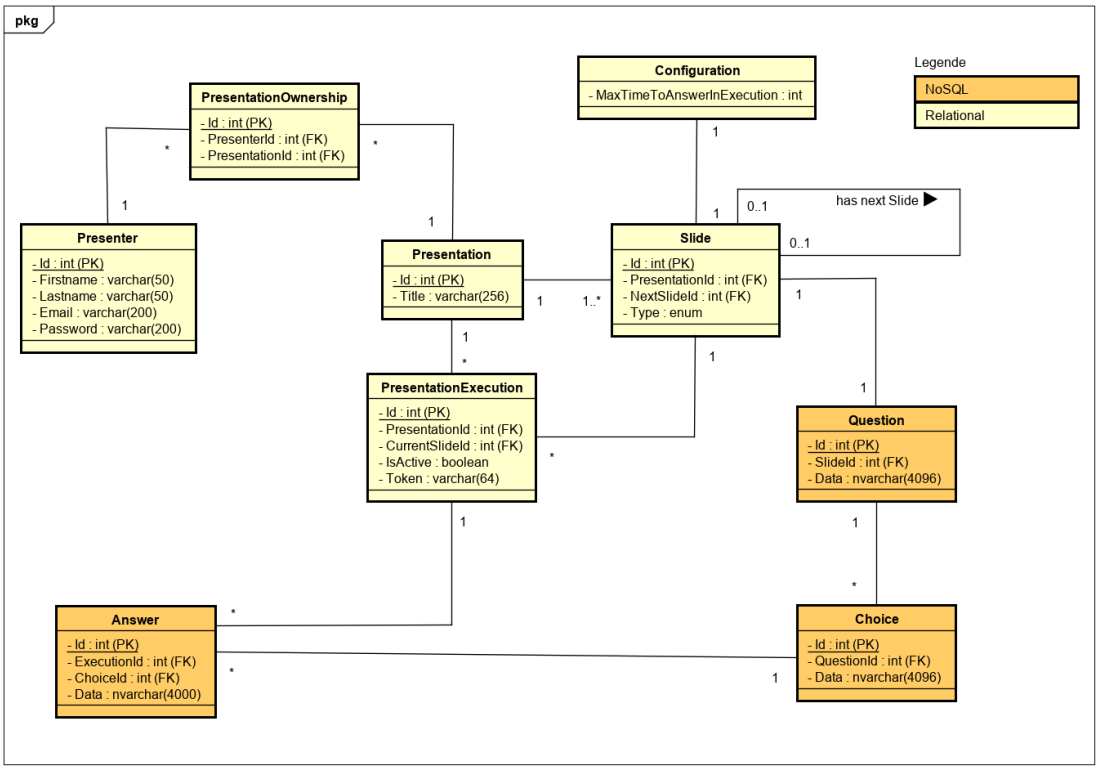


Abbildung 40.: Entity Relationship Diagramm
Quelle: Eigene Darstellung

4.6.1. Mix aus Relationalem und NoSQL Teil

In dieser Domain gibt es Komponenten, die statisch sind und solche, die sehr dynamisch sind. Die statischen Teile sind in einer relationalen Datenbank gehalten. Die Entitäten *Question*, *Choice* und *Answer* entsprechen dem dynamischen Teil, da diverse Ausprägungen zu Fragen, Auswahlmöglichkeiten und Antworten mit verschiedenen Eigenschaften existieren. Aus diesem Grund sind die spezifischen Eigenschaften jeweils als JSON serialisiert und abgespeichert (betrifft Eigenschaft *Data*). Damit die Beziehungen zu anderen Entitäten einfacher verwaltet werden können, sind diese nicht im JSON abgespeichert, sondern eigenständig als Spalte modelliert.

Im Rahmen der Evaluationsphase wurde geplant, dass die Konsistenz der Fremdschlüsselbeziehungen über Constraints sichergestellt werden. Durch die Einführung des Patterns *Database per service* erübrigt sich die Tatsache, dass solche Constraints serviceübergreifend erstellt werden. Fremdschlüsselbeziehungen werden demnach nur noch zwischen Tabellen gesetzt, welche zu einem einzigen Service gehören.

4.6.2. Reihenfolge der Slides

Für die Definition der Slide-Reihenfolge gibt es zwei Ansätze. Der erste ist, dass die fixe Reihenfolge pro Presentation gespeichert wird. Der andere Ansatz ist, nur jeweils den nächsten Slide im Stile einer Linked List abzuspeichern. Der Nachteil der ersten Variante ist, dass bei einer Verschiebung eines Slides im Vergleich zur zweiten Variante im Worst Case alle Slides aktualisiert werden müssen. Beim zweiten Ansatz sind dies im Worst Case nur drei Slides und deswegen die bessere Variante darstellt.

4.7. Wichtige Abläufe

Dieses Kapitel beschreibt einige wichtige Abläufe, die bei der Bedienung eintreten werden.

4.7.1. Präsentationsdurchführung

Bei einer Präsentationsdurchführung gibt es zwei Aktoren, zum einen den Teilnehmer, der das User Interface *Voter Client* bedient. Die Steuerung der Präsentation geschieht via *Presenter Client* und wird vom Präsentator durchgeführt.

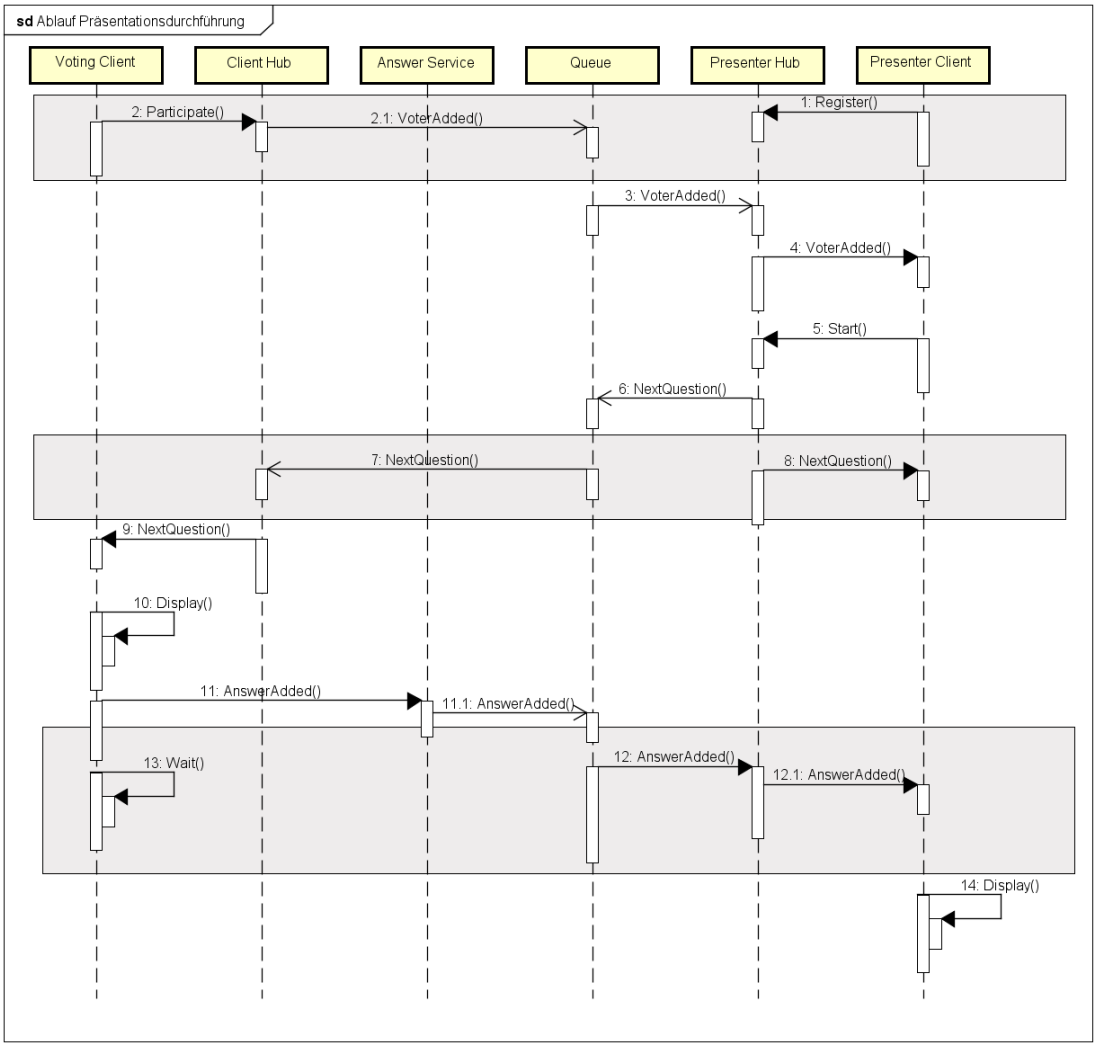


Abbildung 41.: Ablaufdiagramm einer Präsentationsdurchführung
Quelle: Eigene Darstellung

Queue Damit das Diagramm nicht unübersichtlich wird, ist der Einfachheit halber nur eine Queue dargestellt. In der Realität sind es verschiedene Queues oder zumindest verschiedene Topics innerhalb derselben Queue.

Undefinierte Abfolge Die grauen Boxen innerhalb des Diagrammes bedeuten, dass alle Schritte innerhalb der Box in einer nicht definierten Abfolge geschehen können. Dies ist aufgrund der Queue nicht anders möglich, stellt aber auch kein Problem dar, da die Schritte keinen kausalen Zusammenhang haben.

4.8. Libraries und Frameworks

Für die Umsetzung wurden einige Libraries und Frameworks eingesetzt. Dieses Kapitel listet diese auf und beschreibt kurz den Anwendungszweck der einzelnen Libraries und Frameworks.

4.8.1. UI Libraries und Frameworks

4.8.1.1. Übersicht

In der Abbildung 42 sind alle Abhängigkeiten zu den verwendeten Libraries ersichtlich, welche die Webapplikationen einbinden.

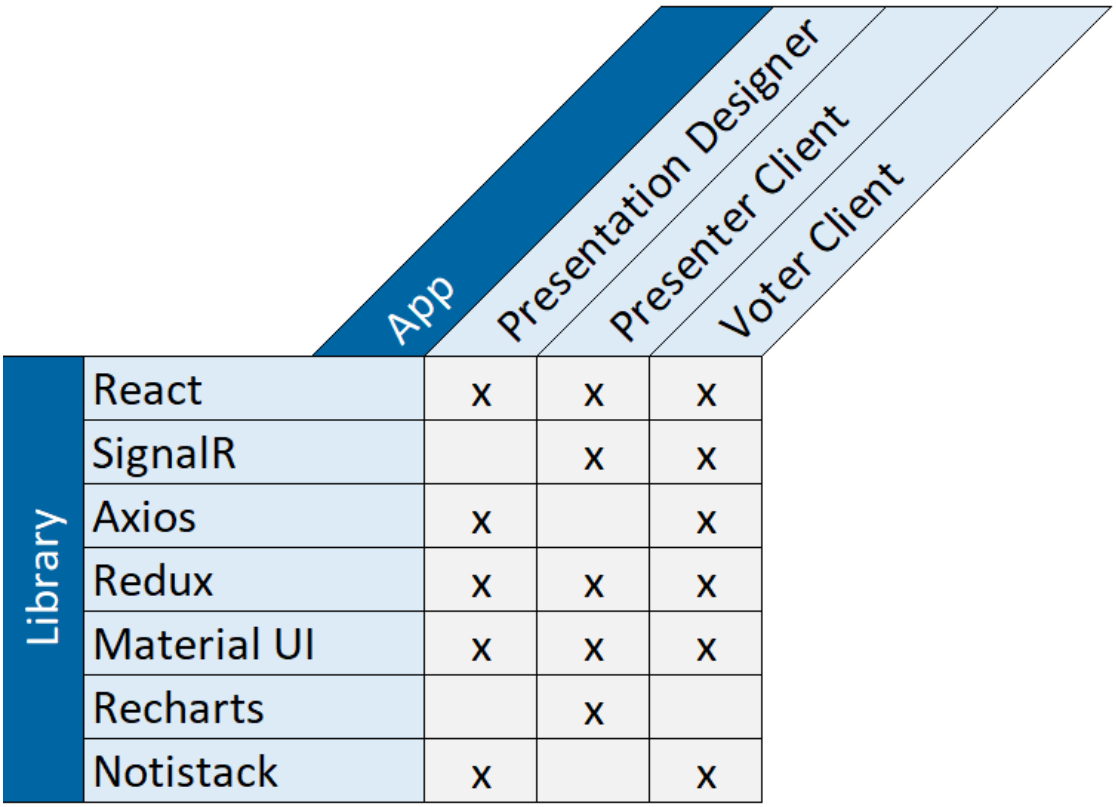


Abbildung 42.: Dependency-Matrix UI
Quelle: Eigene Darstellung

React Als UI-Library wird React verwendet.

SignalR Um mit SignalR Hubs kommunizieren zu können, wird die SignalR-Library benötigt. Mit dieser können Websocket-Verbindungen aufgebaut und entsprechende Remote Procedure Calls ausgeführt werden.

Axios Diese Library bietet mehr als die native Fetch-API an. Insbesondere ist die automatische Konvertierung von JSON zu einem Modelobjekt möglich, welches auch im Zusammenspiel mit TypeScript und dessen Typsicherheit ein Vorteil ist. `https://medium.com/@jeffrey.allen.lewis/http-requests-compared-why-axios-is-better-than-node-fetch-more-secure-can-handle-errors-better-39fde869a4a6`

Redux Im Client benötigen viele Komponenten immer wieder den State von anderen Komponenten. Damit der State nicht pro Komponente verwaltet werden muss, wird Redux als State-Management-Library in den React-Apps eingesetzt.

Material UI Um das Styling der Webapplikation zu vereinfachen, wird auf das bewährte Material Design von Google gesetzt. Dazu wurde für React extra die Library *Material UI* entwickelt.

Recharts Für die Darstellung von Charts wird Recharts verwendet.

Notistack Zuständig für die Darstellung von Notifications in Form von Snackbars, die man aus dem Material Design kennt.

4.8.2. C# Libraries und Frameworks

Nachfolgend aufgelistete Libraries und Frameworks wurden in den C#-Projekten verwendet.

4.8.2.1. Übersicht

In der Abbildung 43 sind alle Abhängigkeiten zu den verwendeten Libraries und Frameworks ersichtlich, welche die Komponenten einbinden.

Library/Framework	Komponente	Authentication Service	Presentation Service	Answer Service	Question Service	Presenter Hub	Voter Hub	Queue	Models	Error Handling	Web	Validation	Token
	ASP.NET Core	x	x	x	x	x	x			x			
	EntityFramework Core	x	x	x	x								
	AutoMapper								x				
	AutoMapper Dependency Injection	x	x	x	x	x							
	StyleCop	x	x	x	x	x	x	x	x	x	x	x	x
	Swashbuckle	x	x	x	x								
	Fluent Validation											x	
	RabbitMQ							x					
	Newtonsoft.Json							x	x	x	x		
	System.IdentityModel.Tokens.Jwt												x
	BCrypt.Net-Next	x											
	AspNet.Security.ApiKey.Providers		x	x	x								
	xunit		x						x				
	ApplicationInsights.AspNetCore	x	x	x	x	x	x						
	Queue			x		x	x						
	Models	x	x	x	x	x	x						
	Error Handling	x	x	x	x	x	x						
	Web		x	x		x							
	Validation	x	x	x	x								
	Token	x	x										

Abbildung 43.: Dependency-Matrix C#
Quelle: Eigene Darstellung

ASP.NET Core Als Frameworks für das Grundgerüst der Webapplikationen wird ASP.NET Core in der Version 2.2 eingesetzt. Mit ASP.NET Core kommen auch einige Ergänzungs-Libraries wie Logging, Configuration und Dependency Injection mit. Ebenso ist SignalR für die Real-Time Kommunikation mit den Clients Teil von ASP.NET Core.

EntityFramework Core Für den Datenbankzugriff und das Migrieren des Datenbankschemas wird EntityFramework Core verwendet.

AutoMapper AutoMapper bietet einge Grosse Hilfestellung für das Konvertieren von Domain-Objekten zu Data Transfer Objects und zurück.

AutoMapper Dependency Injection Dieses Package wird genutzt, um den eigentlichen Mapper zu injecten.

StyleCop Während dem Build wird mit StyleCop der Code nach gewissen Regeln überprüft. Bei einem Fehler erzeugt dies eine Build-Warnung, welche später zu einem Fehler wird.

Swashbuckle Für das automatische Generieren einer Swagger-Spezifikation und einem Swagger-UI wird Swashbuckle verwendet.

Fluent Validation Mit Fluent Validation lassen sich Objekte einfach nach gewissen Regeln validieren.

RabbitMQ Den Zugriff auf die RabbitMQ-Queue geschieht über die offizielle RabbitMQ-Library.

Newtonsoft.Json Für die Datenübertragung wird das Format JSON verwendet. Mit Newtonsoft.Json lassen sich JSON-Strukturen zu C#-Objekten serialisieren.

System.IdentityModel.Tokens.Jwt Mit dieser Library lassen sich Tokens für die Authentication erstellen.

BCrypt.Net-Next Für die sichere Speicherung der Passwörter müssen diese gehashed werden. Dafür wird die Library BCrypt benötigt.

AspNet.Security.ApiKey.Providers Dieser Provider wird benötigt, dass sich Services via Shared API Key authentisieren können.

xunit Mit xunit lassen sich einfach Unit Tests schreiben, ohne viel Boilerplate Code zu produzieren.

ApplicationInsights.AspNetCore Diese Library wird benötigt, um die Applikation in Application Insights zu integrieren.

Eigene Libraries Die restlichen Libraries stellen Projekte dar, die selber entwickelt wurden. Weitere Informationen sind unter Kapitel 4.8.3 zu finden.

4.8.3. Eigene Libraries

Da gewisser Code in mehreren Applikationen verwendet wird, wurde dieser in eigene Libraries ausgelagert. Diese Libraries werden als Nuget-Package deployed und können so in allen Applikationen einfach verwendet werden. Folgende Libraries wurden entwickelt.

Queue Beinhaltet die Implementation für den Zugriff auf die RabbitMQ-Queue.

Models Sämtliche Domain-Klassen, sowie Data Transfer Objects, Data Access Entities und Queue Messages. Ebenso sind hier alle Konverter für das Mapping zwischen Domain und DTO als auch zwischen Domain und Data Access Entities vorhanden. Es wurden alle Klassen in eine Library gepackt, da diese sehr stark voneinander abhängig sind. Beispielsweise hätte eine Änderung an einem Domain-Objekt auch eine Änderung am DTO, Data Access Entity und den entsprechenden Konvertern zur Folge hat.

Error Handling Fasst die gemeinsamen Exceptions und Code für das Error Handling zusammen, beispielweise die ASP.NET Core Middleware für das Abfangen und Bearbeiten von Exceptions.

Web Client Eine Abstraktion des HttpClients von .NET. Ebenso eine Factory, damit dieser via Dependency Injection verwendet werden kann.

Validation Basisklassen und Abstraktionen für die Validierung von Objekten.

Token Provider Services für das Handling von Tokens, aktuell bezogen auf JWT Tokens.

4.9. Security

Es gibt zwei verschiedene Arten von Zugriffen auf einzelne Services. Zum einen greifen Benutzer mit dem Browser auf diese zu, zum anderen benötigen Services Daten von anderen Services. Die letzteren Requests werden also nicht von einem Menschen gestartet. Aus diesem Grund werden zwei verschiedene Varianten benötigt, um einen Anfrager zu autorisieren.

4.9.1. Sicherheit zwischen Services

Zwischen den Services sollte ein Request schnell und einfach ablaufen. Deshalb sollte für die Autorisierung kein weiterer Roundtrip nötig sein. Deshalb funktioniert die Autorisierung zwischen zwei Services über ein Shared Secret, welche beide Services kennen. Dieses wird bei einem HTTP-Request im *Authorization*-Header gesendet.

4.9.2. Sicherheit zwischen UI und Services

Um einen Benutzer autorisieren zu können, werden JWT-Tokens verwendet. Diese werden vom Authentication Service nach Angabe von Benutzername und Passwort ausgestellt. Beim Erstellen dieser Tokens werden diese mit einem Shared Secret signiert. Das Token wird dann von den UI-Clients bei jedem Request im *Authorization*-Header gesendet und können von den einzelnen Services anhand der Signatur und des Shared Secrets validiert werden. In diesen Tokens werden Claims mitgesendet, mit welchem der Nutzer jederzeit in den Services eindeutig identifiziert werden kann. Dafür wird beispielsweise die PresenterId verwendet.

4.9.3. HTTPS

Sobald Daten im Web ausgetauscht werden, drängt sich der Einsatz für HTTPS auf. Chrome beispielsweise warnt den Benutzer bereits, sobald dieser Daten in einem Formular eingibt und die Webseite kein HTTPS verwendet. Der Einsatz von HTTPS ist beim Einsatz von JWT ein muss, da diese ansonsten von einem Angreifer abgefangen und wiederverwendet werden können. Gegen diesen Angriff besitzen die Tokens ein Ablaufdatum, dieses ist in der Regel aber grosszügig gewählt, dass sich der Benutzer nicht immer neu anmelden muss. HTTPS wurde in HSR Live Survey nicht eingesetzt. Es wurden Versuche mit HTTPS, Traefik und Let's Encrypt durchgeführt. Dieser Ansatz konnte aber nicht erfolgreich implementiert werden, weshalb HTTPS aus Zeit- und Prioritätsgründen nicht umgesetzt wurde.

4.9.4. Spezialfälle

4.9.4.1. Presenter Client

Ein Spezialfall ist der Presenter Client, da sich der Benutzer in diesem nicht explizit anmelden muss, es soll aber auch nicht für jeden möglich sein, die Applikation aufzurufen und zu bedienen. Aus diesem Grund wird beim Starten einer Durchführung ein JWT-Token durch den Presentation Service erstellt und vom Designer beim Öffnen des Presenter Client mitgegeben. Dieser verwendet das Token dann für die Verbindung mit dem Hub, welcher das Token wiederum anhand des Shared Secrets validieren kann. Dieser Ablauf ist in der Abbildung 44 zu sehen. So kann ein Benutzer eine Durchführung nur steuern, sofern er das Token kennt. Dieses Token ist nur ein Tag ab Ausstellung gültig und so gegen eine Replay-Attacke zumindest ein wenig geschützt ist. Im Falle einer erneuten Durchführung einer bestehenden Durchführung (beispielsweise zur Ansicht der Resultate) wird vom Designer ein neues Token angefordert und verwendet. Diese Variante hat den grossen Vorteil, dass sich der Benutzer nicht bei zwei Applikationen anmelden, ein solches Token im Backend nicht zwischengespeichert und zwischen mehreren Services synchronisiert werden muss. Einzig im Frontend wird das Token im Session Storage zwischengespeichert, damit es auch bei jedem Request mitgesendet werden kann.

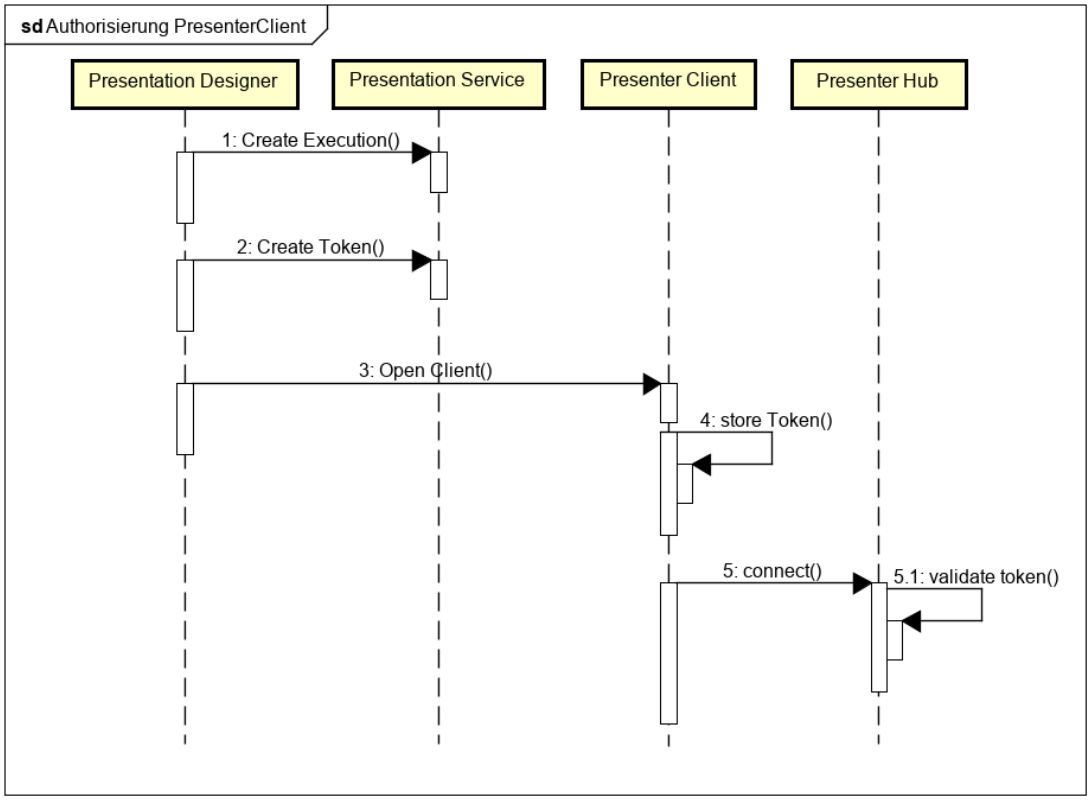


Abbildung 44.: Ablauf der Authorisierung des PresenterClients
Quelle: Eigene Darstellung

Damit das Token nicht in der URL für jeden sichtbar ist, wird nach dem Start des Presenter Clients das Token zwischengespeichert und ein Redirect gemacht. So ist das Token nur noch im Session Storage und nicht mehr für alle sichtbar. Dies ist von grosser Wichtigkeit, da der Presenter Client seinen Einsatz oft vor einer Gruppe via Beamer findet.

4.9.5. Sensible Daten

Besonders im Zusammenhang mit Benutzerdaten muss bei der Persistierung besonders acht gegeben werden.

4.9.5.1. Passwörter

Bei der Registrierung eines Benutzers muss dieser ein Passwort angeben. Um den Benutzer später authentisieren zu können, muss das Passwort abgespeichert werden. Dieses wird jedoch vor dem Abspeichern gehashed. Dafür wird die Hash-Methode BCrypt eingesetzt. Diese Methode speichert direkt im Hash in einem definierten Muster ein Salt, so ist

der Hash besser gegen Rainbow-Tables geschützt. Zudem beinhaltet das Muster eine Versionsnummer, so kann in Zukunft eine neue Version verwendet werden, bestehende Hashes sind aber trotzdem noch verwendbar. Für das Hashen der Passwörter in C# wird die Library *Bcrypt.net next* [12] verwendet. Diese Library verwendet per default das Hashverfahren SHA384, dies wird so auch in HSR Live Survey eingesezt.

4.10. Skalierung

Ein grosser Vorteil von Microservices ist die horizontale Skalierung. Damit diese funktionieren kann, muss bei der Entwicklung darauf geachtet werden, dass kein State verwendet wird. Denn sind mehrere Instanzen desselben Services vorhanden ist nicht sichergestellt, dass Anfragen immer von der gleichen Instanz verarbeitet werden. Bei einigen Komponenten von HSR Live Survey kann jedoch nicht auf ein State verzichtet werden, bei anderen wird der State besonders verwaltet. Diese Eigenheiten sind nachfolgend beschrieben.

4.10.1. SignalR

Für den Handshake von SignalR ist es wichtig, dass Requests während dem Handshake immer von der gleichen Instanz verarbeitet werden. Deshalb mussten beim Reverse Proxy Sticky Session konfiguriert werden [2]. Ein zweites Problem bei der Skalierung von SignalR ist, dass Clients derselben Durchführung einer Präsentation nicht auf der gleichen Instanz landen. Neue Fragen werden im Voter Hub über eine Queue gelesen, so kann irgendeine Instanz die neue Frage laden und versenden. Die Schwierigkeit ist es, dass nicht nur die instanzeigenen Clients diese Frage erhalten, sondern alle Clients der Durchführung. Für das wird eine Redis-Backplane verwendet. Diese Backlplane verwaltet die Clients und deren Connections und stellt sicher, dass Requests auch über mehrere Instanzen immer an alle Clients versendet werden. Diese Anwendung wird auch von Microsoft in ihrer Dokumentation beschrieben [2]. Diese Variante hat natürlich auch einige Nachteile, da beispielsweise die Sticky Session dazu führen kann, das eine Instanz viel Arbeit hat, andere jedoch keine. Durch die Gegebenheit, dass Verbindungen zu diesen Instanzen in der Regel nur wenige Minuten dauern (da die Durchführung einer Präsentation wohl eher kurz sein wird), kann dies aber vernachlässigt werden.

4.10.2. Autorisierung

Damit für die Autorisierung keine State-Daten gespeichert werden müssen, wird ein JWT für die Autorisierung verwendet. Damit fungiert das JWT als eine Art Speicher für relevante Daten, welche dann aber von irgendeiner Instanz ausgelesen und verwendet werden kann [40].

4.10.3. MS SQL Server

Eine sehr zentrale Komponente stellt die Datenbank dar. Diese dient als Datenspeicher und wird somit von allen Services verwendet. Eine solche Datenbank muss also hoch

verfügbar sein. Im Rahmen der Bachelorarbeit wurde auf eine Skalierung der Datenbank verzichtet, da diese die anzunehmende Last auch mit einer Instanz bewältigen kann. Es gäbe die Möglichkeit, eine Availability Group einzurichten [14], dies führt aber nicht generell zu einer schnelleren Datenbank, sondern eher zu einer besseren Verfügbarkeit. Eine ideale Lösung wäre, dass jeder Service seine eigene Datenbank besitzt. Somit würde die Last auf mehrere Datenbanken verteilt werden. Doch dieser Ansatz wurde aufgrund des Mehraufwandes in der Datensynchronisation und Zwecksmässigkeit nicht implementiert.

4.10.4. RabbitMQ und Redis

Bei Redis und RabbitMQ ist es möglich, sogenannte Cluster zu erstellen, damit eine höhere Verfügbarkeit und Performance erreicht werden kann. Dies wurde jedoch nicht eingerichtet, da bisher die Performance einer Instanz ausreichte.

4.10.5. Eigene Services

Alle eigenen Services sind horizontal, wie auch vertikal, skalierbar. Die Services wurden mit Ausnahme von SignalR stateless implementiert und werden von aussen nur via Traefik als Load Balancer angesprochen, welcher die Last mit dem Round Robin Prinzip verteilt. Bei Requests an Services innerhalb des Kubernetes Clusters werden die Services mit dem internen Kubernetes Service Proxy ausgewählt, ebenfalls mit dem Round Robin Prinzip [18].

4.10.6. Skalierung durchführen

Vertikale Skalierung Eine Vertikale Skalierung kann im Portal von Azure in der Kubernetes Resource unter dem Punkt Scale ausgeführt werden. Die vertikale Skalierung bedeutet, mehr Nodes und somit den Services mehr Ressourcen zur Verfügung zu stellen. Die Konfigurationsseite ist in Abbildung 45 zu sehen.

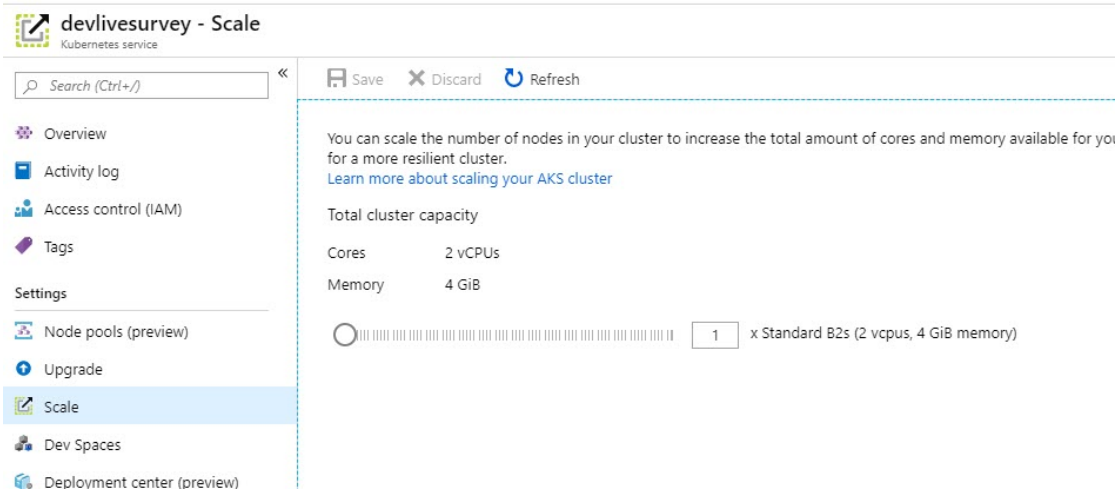


Abbildung 45.: Konfiguration der Skalierung in Azure
Quelle: Printscreen, portal.azure.com

Horizontale Skalierung Jeder Service kann über das Kommandozeilentool *kubectl* von Kubernetes skaliert werden. Dafür ist das Kommando *scale* [17] zu verwenden. Im Listing 4.1 wird ein Beispielbefehl für das Skalieren des Question Service gezeigt.

```
1  kubectl scale deployment questionservice-deployment --replicas=2
```

Listing 4.1: Skalierung via kubectl

4.11. Architektonische Entscheide

4.11.1. Aufteilung der Services

Die Aufteilung der Service in der Systemarchitektur ist eine grosse Herausforderung. Denn gemäss `microservices.io` existieren einige Anforderungen an einen Microservice [21]:

- Architektur muss stabil sein
- Ein Service sollte einen kleinen Satz an verwandten Funktionen implementieren
- Eine Änderung soll nur ein Servie betreffen
- Services müssen lose von anderen Service gekoppelt sein
- Ein Service muss testbar sein
- Ein Service muss von einem kleinen Team implementiert werden können

- Die Entwicklung eines Services soll autonom sein

Damit all diese Anforderungen erfüllt werden können müssen also viele Dinge in Betracht gezogen werden. Unterstützung dafür gibt es von `microservices.io` in Form von zwei Pattern.

Decompose by business capability Die Aufteilung der Services erfolgt anhand der Businesskonzepte. Ein Businesskonzept ist ein Ablauf, der einen Wert bietet. Beispiele dafür ist das Produktemanagement, das Ordermanagement oder das Bestellmanagement.

Decompose by subdomain Services werden anhand der Businessdomäne aufgeteilt. So werden Funktionalitäten innerhalb derselben Domäne zusammen in einem Service implementiert.
Die beiden beschriebenen Patterns sind sehr nahe beieinander und können in einer ähnlichen Aufteilung der Services enden. In HSR Live Survey wurde keines der beiden Patterns konsequent umgesetzt. Es wurde mehrheitlich der Ansatz der Unterteilung nach Businessdomäne verfolgt. Dies ist in der Architektur anhand der Services Answer Service, Authentication Service, Presentation Service oder Question Service ersichtlich. Diese Services sind für Funktionalität innerhalb ihrer Domäne verantwortlich. Zum anderen existieren die beiden SignalR-Hubs. Diese sind eher nach einem Businesskonzept, nämlich der Real-Time-Kommunikation mit dem Client, entworfen und umgesetzt worden.

4.11.2. Aufteilung der Datenbank

Bei der Entwicklung von Microservices in Zusammenhang mit Datenbanken existieren gemäss `microservices.io` zwei bekannte Patterns. Zum einen das Pattern *Shared Database* [23], dem gegenüber steht das Pattern *Database per Service* [20]. Diese zwei Patterns werden nachfolgend kurz erläutert.

Shared Database Eine einzige Datenbank, auf die jeder Service vollen Zugriff erhält. Jeder Service darf Daten lesen und bearbeiten. Auch das Schema darf von jedem Service angepasst werden. Dies führt zu einigen Nachteilen bei der Migration der Datenbank, da diese koordiniert werden müssen. Zudem kommt es zu einer hohen Kopplung der Services über die Datenbank, beispielsweise bei Transaktionslocks. Der Vorteil liegt in der Bekanntheit der Technologie und der Datenkonsistenz. Zudem ist das Betreiben einer einzigen Datenbank einfacher und mit weniger Aufwand verbunden.

Database per Service Jeder Service besitzt eine eigene Datenbank mit den für ihn relevanten Daten. Auf diese Daten können andere nur via diesen Microservice zugreifen. Dies führt dazu, dass die Entwicklung dieser Services weniger stark aneinander gekoppelt sind und die Datenbankmigration einfacher wird. Zudem kann so jeder Service die für ihn perfekte Datenbanktechnologie verwenden. Der grosse Nachteil dieses Patterns sind

Transaktionen über mehrere Services. Dafür können jedoch spezielle Patterns wie beispielsweise das Saga Pattern [22] implementiert werden, dies ist aber umständlicher. Nebst Transaktionen sind auch Abfragen über mehrere Datenbanken eine Herausforderung.
Im Rahmen der Plattform HSR Live Survey wurde das Pattern Database per Service verwendet und implementiert. Dabei wurde die Variante *Private-tables-per-service* umgesetzt. Diese beschreibt, dass jeder Service seine Tabellen besitzt, welche von keinem anderen Service verwendet oder angepasst werden dürfen. Da die Trennung rein konzeptionell und nicht technisch vorhanden ist, musste bei der Implementation besonders darauf geachtet werden, keine Daten anderer Tabellen direkt abzufragen. Eine Übersicht der Tabellen und der Zugehörigkeit ist in Tabelle 10 zu finden. Diese Variante hatte besonders in den Anforderungen an Infrastruktur und operativem Betrieb entscheidende Vorteile. Diese Bereiche fallen deutlich einfacher aus für Entwickler und sind so schnell umsetzbar. Zudem ist es mit Entity Framework Core möglich, nur einen Subset an Tabellen mit Migrations zu migrieren, was die technische Umsetzung ermöglichte. Ein weiterer grosser Vorteil ist, dass diese Tabellen mit vertretbarem Aufwand in eine eigene Datenbank verschoben werden können. So kann beispielsweise auf Performanceprobleme in relativ rascher Zeit reagiert werden. Die Tabellen sind aktuell in derselben Datenbank, da die Anforderungen an die Infrastruktur von einer Datenbank unterstützt werden. Denkbar wäre auch eine Migration von gewissen Tabellen auf eine andere Datenbanktechnologie. Um dieses Pattern zu verwenden mussten auch einige Nachteile in Kauf genommen werden. Das Abfragen von Daten dauert in der Regel länger, da diese über HTTP via dem API eines anderen Services laufen müssen. Zudem sind verteilte Transaktionen nicht einfach umsetzbar, siehe Kapitel 4.11.3 - Transaktionen. Ein weiteres Problem sind die Fremdschlüsselbeziehungen über Tabellen, die unterschiedlichen Services angehören. Diese wurden bei der Implementation explizit nicht gesetzt um dieses Pattern nicht zu verletzen. Diese Fremdschlüssel würden bei einer späteren Migration von Tabellen in eine eigene Datenbank so oder so wegfallen.

Tabelle	**Service**
Answer	Answer Service
Choice	Question Service
Presentation	Presentation Service
PresentationExecution	Presentation Service
Presenter	Authentication Service
Question	Question Service
Slide	Presentation Service

Tabelle 10.: Zuteilung der Tabellen zu Services

Zudem besitzt jeder Service seine eigene Migration-History Tabelle. Diese ist mit dem Namen des Service als Prefix gekennzeichnet.

4.11.3. Transaktionen

Ein Problem beim Einsatz einer Microservice-Architektur sind die Transaktionen über mehrere Services. Beim Monolith-Ansatz können Transaktionen über die Datenbank sichergestellt werden, Microservices haben aber im Grundsatz nicht dieselben Tabellen. Für die Lösung dieses Problems existiert das *Saga*-Pattern [22]. Diese besagt, dass ein Service nach dem Update von Daten ein Ereignis auslöst und sich andere Services auf dieses subscriben und ihre Daten entsprechend anpassen können. Ein Nachteil dieses Patterns ist die komplexe Umsetzung. Aus diesem Grund wurde auf die Umsetzung dieses Patterns verzichtet, denn im gesamten Kontext von HSR Live Survey würde nur in einem Fall eine verteilte Transaktion benötigt. Beim Kopieren einer Präsentation muss der Presentation Service über andere Services wie dem Question Service gehen, um diese ebenfalls kopieren zu können. Sollte es bei diesem Vorgang zu einem Fehler kommen, wird dieser dem Benutzer angezeigt. Der Benutzer hat dann im schlimmsten Fall einen inkonsistenten Zustand, den er aber von Hand beheben und den Vorgang zu einem späteren Zeitpunkt wiederholen kann.

4.11.4. Zugriffe von Frontends auf Services

Bei der Verwendung einer Microservice-Architektur im Zusammenhang mit Frontends muss definiert werden, mit welchen und wie vielen Services das Frontend kommuniziert. Gemäss `microservices.io` existiert ein Pattern in diesem Bereich, das *Api Gateway Pattern* [19]. Diese Pattern beschreibt, dass alle Frontends nur über diesen API Gateway mit den jeweiligen Services kommunizieren. Dieser Gateway kann zudem Dinge wie Authentisieren oder Zusammenziehen von Daten erledigen. Eine Ausprägung dieses Patterns wird mit dem Namen *Backend for Frontend* [19] bezeichnet. In diesem existiert pro Frontend ein Gateway, der genau auf die Bedürfnisse des Frontends zugeschnitten ist.

Implementiert wurde keines dieser beiden Patterns. Der Overhead eines solchen API Gateways wurde als zu gross empfunden. Als Ersatz wurde ein Reverse Proxy verwendet, welcher die Requests vom Frontend an den jeweiligen Service weiterleitet. So hat das Frontend trotzdem nur einen Einstiegspunkt. Dieser Ansatz hat in der Einfachheit der Umsetzung einen grossen Vorteil. Als grosser Nachteil ist die Kopplung der Frontends an die Services vorhanden. Denn so müssen die Frontends die Struktur der Services und deren APIs kennen. Ändern diese, müssen auch die Frontends angepasst werden. Zudem muss so jeder Service bei Bedarf die Clients authentifizieren oder autorisieren und es musste für die interne Service-Kommunikation ebenfalls einen Autorisierungsmechanismus implementiert werden, da alle Services von ausserhalb erreichbar sind. Trotz den vielen Nachteilen wurde kein API Gateway implementiert, da der Overhead für den Umfang der Applikation als zu gross eingestuft wurde. Bei einer Erweiterung des Umfangs würde sich der Einsatz eines Gateways aber wohl anbieten.

4.12. Patterns

In der Welt der Programmierung existieren unzählige Patterns. Patterns können in Architektur- und Code-Pattern unterschieden werden. In diesem Kapitel werden einige implementierte Patterns vorgestellt.

4.12.1. Architekturpattern

Eine Anforderung an die Applikation ist der Einsatz einer Microservice-Architektur. microservice.io ist eine Anlaufstelle für Patterns in einer Microservice-Architektur. Nachfolgend werden die implementierten Patterns aufgelistet und je nach Bedarf kurz beschrieben. Die Patterns sind in drei Kategorien unterteilt: Application Patterns, Application Infrastructure Patterns und Infrastructure Patterns.

4.12.1.1. Application Patterns

Database per Service Der Einsatz des Database per Service Patterns ist in Kapitel 4.11.2 - Aufteilung der Datenbank beschrieben.

Application Metrics Durch die Verwendung von Application Insights, beschrieben in Kapitel 5.5.1 - Application Insights, wird das Application Metrics Pattern eingesetzt.

4.12.1.2. Application Infrastructure Patterns

Externalized configuration Die Services werden via Umgebungsvariabeln konfiguriert. Diese Implementation ist die Umsetzung des Patterns Externalized configuration. Detailiert ist diese Umsetzung im Kapitel 5.4 - Konfiguration beschrieben.

Access Token Für die Autorisierung wird mit JWT das Access Token Pattern implementiert. Dies ist in Kapitel 4.9.2 - Sicherheit zwischen UI und Services beschrieben.

Messaging Für die asynchrone Kommunikation wird ein Message Broker verwendet. Der Einsatz ist in Kapitel 4.16 - Asynchrone Kommunikation beschrieben.

Exception tracking Exceptions werden durch Application Insights (Kapitel 5.5.1 - Application Insights) gesammelt und können im Application Insights Dashboard getrackt werden.

4.12.1.3. Infrastructure Patterns

Multiple Services per Host Kubernetes verteilt Pods automatisch auf die verfügbaren Nodes, wobei ein Node ein Host entspricht. Da aber weniger Nodes als Pods existieren wird das Pattern Multiple Services per Host eingesetzt. Details sind im Kapitel 4.5 - Deployment beschrieben.

Service-per-Container In jedem Docker-Container läuft nur ein Service.

Service deployment platform Kubernetes gilt als Service deployment platform und wird in HSR Live Survey verwendet.

4.12.2. Code Patterns

Options Pattern Der Einsatz des Options Pattern wird in Kapitel 5.4.2 - Backend Services beschrieben.

Factory Pattern Die eigene Library *LiveSurvey.Web* implementiert das Factory Pattern. Diese wird benötigt um einen eigenen HttpClient mit einer Konfiguration zu erstellen und diesen so via Dependency Injection verwenden zu können.

Template Method Pattern In den Basisklassen der Converter in der eigenen Library *LiveSurvey.Models* wird das Template Method Pattern angewendet um das JSON der fragespezifischen Felder zu erhalten. Die Abbildung 46 zeigt den Einsatz dieses Patterns anhand des Beispieles des Converters einer Question zum QuestionDto.

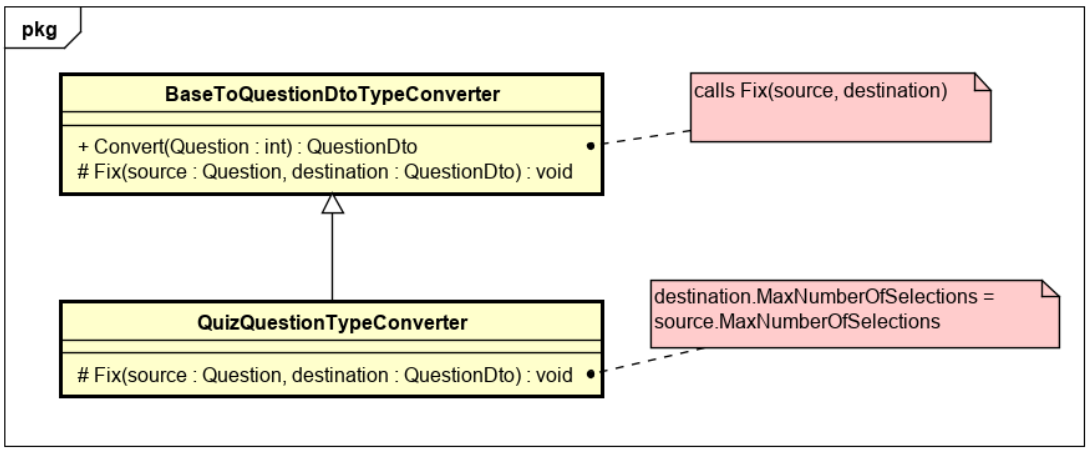


Abbildung 46.: Klassendiagram der Converter mit dem Template Method Pattern
Quelle: Eigene Darstellung

Dependency Injection Dependency Injection ist ein wichtiger Bestandteil von ASP.NET Core [?] und bereits in das Framework integriert. Das Pattern wird in jedem Service für das injecten von Services und Repositories eingesetzt.

Middleware Pattern Das Middleware-Pattern wird in den Backend-Services für das allgemeine Error-Handling eingesetzt. In den Frontend-Applikationen wird das Pattern

im Zusammenhang mit SignalR eingesetzt. Dort dient die Middleware für die Herstellung der Connection und das Senden und Empfangen von Daten.

Layers Pattern Besonders in den Backend-Services wurde das Layers Pattern eingesetzt. Dabei wurde aber auf ein striktes Layering verzichtet. Beispiele dieses Layerings sind in Kapitel 4.4.1 - Allgemeine Architektur beschrieben.

Redux Pattern Das Redux Pattern wird in den Frontend-Applikationen angewendet, um ein vernünftiges State Management über mehrere Komponenten sicherzustellen. Ansonsten müsste jede Komponente einen eigenen State verwalten und beim Rendern von weiteren React-Komponenten diesen an die zusätzlichen Komponenten als sogenannte Properties weitergeben. So kann die Komplexität stark wachsen, wenn Komponenten in einer Parent-Child- oder Sibling-Beziehung stehen. Der Einsatz von Redux schliesst aber nicht aus, dass Komponenten nach wie vor einen eigenen privaten State zusätzlich zum globalen State besitzen. Weitere Informationen sind unter Kapitel 4.14 - State Management zu finden.

Repository Pattern Das Repository-Pattern wird im Data Access Layer verwendet um den Zugriff auf die Daten zu abstrahieren und so in der Zukunft die Möglichkeit zu haben, diesen einfach auszutauschen.

DTO Pattern Für die Kommunikation der Backend Services mit Clients wurden Data Transfer Objects (DTO) eingesetzt. Diese abstrahieren die interne Struktur gegenüber den Clients damit diese bei Änderungen besser geschützt sind.

4.13. Error Handling

Falls Fehler auftreten, müssen diese abgefangen und wenn möglich von selbst korrigiert oder zumindest protokolliert und dem Benutzer gemeldet werden. Dieses Kapitel beschreibt das Error Handling und einige technische Details.

4.13.1. Ansatz

Die möglichen, bekannten Fehler werden so nah wie möglich am Ursprungsort abgefangen und protokolliert. Da die meisten Fehler aufgrund von falschen Anfragen oder Daten auftreten werden (fehlerhafter User-Input oder Abfrage einer nicht vorhanden Entität), können diese nicht von selbst behoben werden. Ebenso können Fehler bei nicht vorhandenen oder lauffähigen Abhängigkeiten wie Datenbank oder andere Services auftreten. Auch diese Fehler können nicht zur Laufzeit von der Applikation behoben werden. Deshalb werden die Fehler protokolliert und von den Services mit einem möglichst genauen Status Code und einer Fehlermeldung an den Anfrager zurückgesendet. Die verwendeten Status Codes sind im Unterkapitel 4.13.2.2 beschrieben.

4.13.2. Umsetzung

4.13.2.1. Error Model

Das Frontend wird von Menschen bedient, deshalb sollen die Benutzer eine für sie verständliche Fehlermeldung erhalten. Um die Fehler der Backend Services den Benutzern anzeigen zu können, werden diese in ein Error Model verpackt und so dem Frontend gesendet. Zudem können nicht direkt die Exceptions an den Client gesendet werden, da diese sensible Daten wie den Stack Trace oder Endpoints beinhalten könnten. Im Model werden aktuell noch die Exceptions mitgeschickt. Das Exception Handling kann aber so konfiguriert werden, dass diese nicht mehr mitgeschickt werden.

4.13.2.2. Status Codes

Nebst den in ASP.NET Core integrierten Status Codes (404 NotFound, 405 MethodNotAllowed) werden folgende HTTP Status Codes direkt verwendet:

- 200 OK
- 204 NoContent
- 400 BadRequest
- 401 Unauthorized
- 404 NotFound
- 500 Internal Server Error
- 503 BadGateway

Es wurden explizit nur eine kleine Untermenge der verfügbaren HTTP Status Codes verwendet um zum einen die Übersicht behalten zu können und zum anderen den Clients eine überschaubare Menge an Fehlern zur Behandlung zu bieten. Dieser Ansatz wurde von Dropbox beschrieben [11] und diente als Inspiration für die Umsetzung.

Code 204 wird zurückgegeben, falls ein Request erfolgreich durchgeführt werden konnte, aber kein Resultat zurückgegeben wird.

Code 400 wird zurückgegeben, falls die Input-Validation nicht erfolgreich war oder ein logischer Fehler in den Input-Daten existiert.

Code 404 wird zurückgegeben, falls ein angeforderter Datensatz nicht gefunden werden konnte. Zudem wird dieser Code vom ASP.NET Core Framework zurückgegeben wenn keine Route zu der angeforderten URL gefunden werden konnte.

Code 500 wird zurückgegeben, falls ein unbekannter Fehler aufgetreten ist oder ein Fehler erkannt wurde, der nicht zu einem anderen Code passt.

Code 503 wird zurückgegeben, falls Daten von einem anderen Service geladen werden sollten, dieser Request aber nicht erfolgreich verlief.

4.13.2.3. ASP.NET Core Middleware

Um alle aufgetretenen Exceptions in den entsprechenden HTTP Status Code und das Error Model umzuwandeln wird eine Middleware eingesetzt. Diese Middleware fängt alle Exceptions ab und wandelt diese in das entsprechende Model um. Damit diese Middleware in allen Services verwendet werden kann, wurde eine Library erstellt. Die Umsetzung dieser Middleware wurde von der eigen entwickelten Studienarbeit übernommen.

4.14. State Management

Damit die React-Applikationen sinnvoll eingesetzt werden können benötigen diese in den meisten Fällen einen State. In der Art werden die Komponenten zwischen Containerkomponente und Präsentationskomponente unterschieden. Containerkomponenten besitzen einen State, Präsentationskomponenten hingegen nicht [46]. In React werden Komponenten durch Komposition zusammengeführt [**?**], dies führt zu folgenden Beziehungen:

- Parent zu Child
- Child zu Parent
- Sibling zu Sibling

In gewissen Fällen muss ein States über mehrere Komponenten geteilt werden, beispielsweise der angemeldete Benutzer oder der aktuell zu bearbeitende Datensatz. Um diese Daten mit anderen Komponenten teilen zu können existieren verschiedene Möglichkeiten. Die Daten könnten in Form von Properties geteilt via Callback als Property geladen werden. Eine andere Möglichkeit ist der Einsatz einer State Management Library, die einen globalen State verwaltet und diesen den Komponenten zur Verfügung stellt.

4.14.1. Ansatz

Es wurde der Ansatz der State Management Library implementiert. Komponenten die geteilte Informationen in Form des States benötigen, werden als Connected Components entwickelt. Diese verbinden sich zu einem Store, in dem der globale State gehalten wird. Nebenbei existieren nach wie vor Komponenten, die nur zur Anzeige von Informationen dienen oder einen privaten State besitzen.

4.14.2. Umsetzung

Es wird die Library Redux verwendet, welche sich um das State Management der ganzen Applikation kümmert. Redux hat drei Hauptbestandteile, namentlich State, Action und Reducer.

State Definition des Application States.

Action Eine Aktion, allenfalls mit Parametern, welche einen State Change auslösen.

Reducer Nimmt eine Aktion entgegen und ist verantwortlich dafür, wie der State sich aufgrund der Aktion ändert.

4.14.2.1. Aufteilung

Der gemeinsame Application State, die entsprechenden Actions und Reducers wurden in verschiedene Teile aufgeteilt, so dass diese der logischen Aufteilung der Architektur entsprechen. Ohne diese Aufteilung würde der globale State sehr gross und unübersichtlich werden. Die Hauptkomponenten (Actions, AppState, RootReducer) importieren alle aufgesplitteten Teile und fügen diese wieder zusammen.

4.14.2.2. Eigener State

Trotz des Einsatzes einer globale State Management Library können Komponenten weiterhin einen privaten State besitzen. Dieser ist besonders sinnvoll, wenn Daten nur lokal von Wichtigkeit sind oder noch nicht komplett erfasst wurden. Zudem würde eine Verlagerung jedes privaten States in den globalen State ein deutlichen Mehraufwand bedeuten, da eine Vielzahl von Actions und Logik in den Reducer entstehen würde.

4.14.3. Begründung

Der Einsatz einer State Management Library bedeutet einen Mehraufwand im implementieren der Actions und der Reducer, die zum grossen Teil reiner Durchlauferhitzer sind. beispielsweise müssen für alle Requests immer zwei Actions definiert und implementiert werden, eine um das Laden der Daten anzustossen, eine um das Ende des Ladens anzuzeigen. Doch dieser Mehraufwand lohnt sich im Vergleich zu den Varianten mit der Datenteilung via Properties. Denn diese Variante wird sehr schnell sehr unübersichtlich, besonders bei der Verschachtelung von Komponenten. Zudem müssen einige Komponenten Daten nur als Property annehmen, um diese an die eigenen Kinder weitergeben zu können. Dies macht es schwieriger, in sich gekoppelte Komponenten zu erstellen.

4.15. Models und Converters

Pro Domänenobjekt bestehen je nach Layer verschiedene Anforderungen. So existieren Data Transfer Objects, Domain Objects und Data Access Entities. Zum Beispiel existieren für das Domänenobjekt Questions auf Business Layer Spezialisierung von Fragen, im Gegensatz dazu sind diese im Data Access Layer generisch gehalten. Hingegen ist der Typ Presentation auf allen Schichten gleich. Diese Trennung wurde eingeführt um die einzelnen Layers voneinander zu trennen und gegen Änderungen innerhalb eines Layers zu

schützen. Durch diese Trennung müssen die einzelnen Objekte jedoch von einem Format ins andere konvertiert werden.

4.15.1. Ansatz

Insgesamt gibt es zwei Schnittstellen, an denen es zu Konvertierungen kommt. Zum einen, wenn Daten im Web API entgegengenommen werden und zum anderen, wenn die Daten in den Data Access Layer gelangen. Grundsätzlich sollen alle Layers mit den Business-Objekten arbeiten und diese bei Bedarf in das eigene Format umwandeln.

4.15.2. Umsetzung

Um die Konvertierungen einfacher und weniger aufwändig zu gestalten, wurde AutoMapper[1] genutzt. Der Vorteil von AutoMapper ist, dass bei Entitäten mit gleicher Struktur über eine Zeile Code ein einfaches Mapping erstellt werden kann (siehe Listing 4.2). AutoMapper kümmert sich um das Konvertieren und mappt automatisch alle Eigenschaften mit gleichem Namen.

```
1  CreateMap<PresentationDto, Presentation>().ReverseMap();
```

Listing 4.2: Einfaches 1:1 Mapping

Komplexere Mappings wie bei Answers, Questions und Choices wurden über eigene Konverter gelöst. Diese Konverter sind besonders nötig, um die flache Struktur von Data Transfer Objects in die Klassenstruktur auf Business-Ebene umzuwandeln. Zuletzt müssen diese wieder in eine flache Struktur für die Datenspeicherung gebracht werden. Der Grund für den Einsatz einer Klassenhierarchie auf Business-Ebene war die saubere Struktur und den möglichen Einsatz von typspezifischen Methoden.

Pro Entitätstyp existieren jeweils zwei Base-Converters, in denen die gemeinsamen Properties gemappt werden. Spezialisierungen leiten von diesen Base-Converters ab und implementieren, wenn nötig die virtuelle Methode *Fix*, welche sich um die spezifischen Property-Konvertierungen kümmert. Diese virtuelle Methode wird in den Base-Converters nach der Konvertierung der Base-Properties aufgerufen. Das Klassendiagramm für den Question-Konverter ist in Abbildung 47 abgebildet.

[1] `https://automapper.org/`

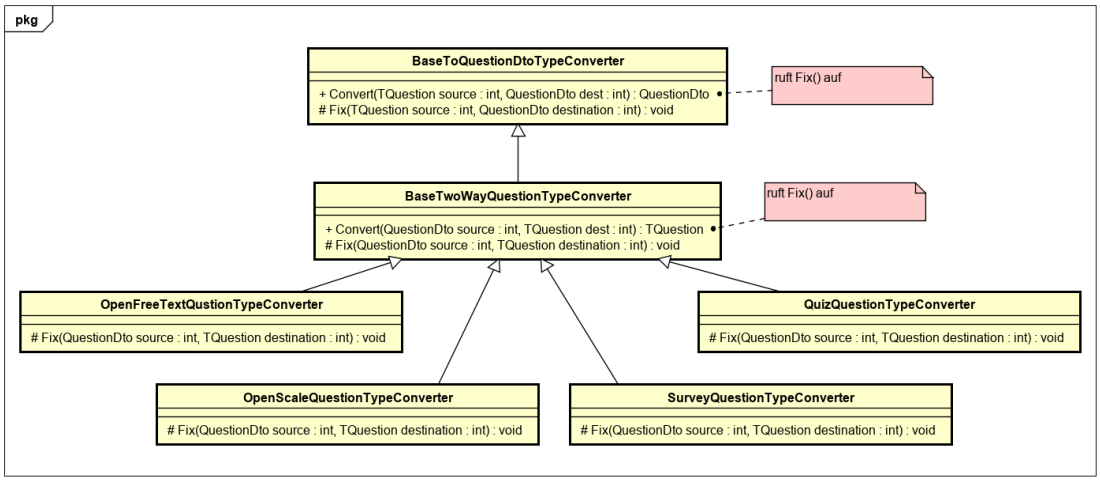


Abbildung 47.: Klassendiagramm der Konverterstruktur am Beispiel des Objektes Question

Quelle: Eigene Darstellung

Aufgrund der generischen Schnittstelle von AutoMapper und da der Basiskonverter in beide Wege konvertieren muss, mussten zwei Basisklassen erstellt werden. Der Compiler gibt an, das es je nach Typargument theoretisch möglich ist, dass die beiden Implementationen dieselbe sind (Fehlercode CS0695). Dies kann mit Typeinschränkungen nicht gelöst werden da in C# keine negativen Einschränkungen existierne. Diese Problematik ist als Beispiel in Listing 4.3 dargestellt.

```
1  // Interface von AutoMapper
2  class BaseConverter<TQuestion> : ITypeConverter<QuestionDto, TQuestion>,
       ITypeConverter<TQuestion, QuestionDto> {}
3
4  // folgende Typ-Variation wuerde zum Problem fuehren
5  var converter = new BaseConverter<QuestionDto>();
```

Listing 4.3: Problematik der Zweiweg-Konverter

4.15.2.1. Eigene Model-Library

Da mehrere Projekte immer wieder die gleichen Models benötigen, wurden diese in ein eigenes NuGet-Package ausgelagert. Zwingend nötig sind nebenbei die Converters, welche sich deswegen ebenfalls im Model-Package befinden.

4.16. Asynchrone Kommunikation

Der Aspekt der asynchronen Kommunikation ist insbesondere zwischen den beiden SignalR-Hubs relevant. Wenn beispielsweise Antworten vom Voter Hub direkt zum Presenter Hub gelangen, müsste der Presenter Hub immer wieder warten um neue Antworten abzuarbeiten bzw. der Voter Hub könnte keine weiteren Antworten absenden.

4.16.1. Ansatz

Durch die Platzierung einer Queue zwischen beiden Hubs entsteht eine vollkommen entkoppelte Kommunikation, so dass ein blockierendes Warten nicht mehr auftritt. Pro Nachrichtentyp wird eine eigene Queue definiert und verwendet, damit sich Consumer gezielt auf diese Typen subscriben können. Die Architektur inklusive Datenfluss der Queues ist in Abbildung 48 dargestellt.

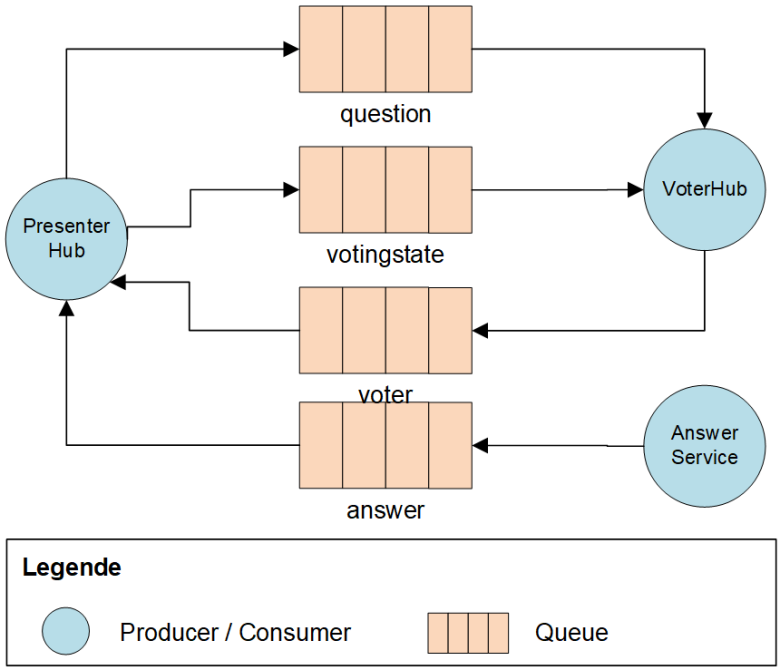


Abbildung 48.: Architektur der Queues
Quelle: Eigene Darstellung

4.16.2. Umsetzung

Als Message-Broker wird auf RabbitMQ[2] gesetzt. Wenn ein Hub Messages in eine Queue schreiben muss, geschieht dies jeweils abstrahiert in einem Repository basierend auf dem Repository Pattern. Zwecks Änderbarkeit enthält jedes Repository eine Konfiguration,

[2] `https://www.rabbitmq.com/`

in der Verbindungsdetails und Queue-Name enthalten sind. Pro Queue können mehrere Subscriber existieren, beispielsweise wenn der Voter Hub skaliert wurde und somit mehrere Instanzen laufen. Das Zustellen der Nachrichten verläuft im Round-Robin Prinzip [34]. Die Nachrichten werden nach dem Lesen automatisch quittiert, so werden diese nicht an anderen Consumer gesendet. Durch den Einsatz dieser Variante lassen sich einfach mehr Consumer hinzu- oder wegnehmen, was bei einer Skalierung der Hubs oder Services geschieht. Die Nachrichten werden dann via Redis-Backplane in den Hubs an anderen Instanzen der Hubs verteilt.

4.16.2.1. Eigene Queue-Abstraktion

Der RabbitMQ-Client wurde mithilfe einer selber entwickelte Library abstrahiert. Durch das eingeführte Interface *IQueueClient* kann in Zukunft einfach auf eine andere Queue-Technologie gewechselt werden, ohne grossen Einfluss auf die Projekte zu nehmen, welche die Queue-Library verwenden. Anpassungen sind in diesem Fall nur beim Dependency Injection nötig.

4.17. HTTP-Kommunikation

Einige Services dürfen aufgrund der Aufteilung der Zuständigkeit nicht direkt auf Daten anderer Services in der Datenbank zugreifen. Deswegen geschieht der Zugriff auf diese Daten via API mittels HTTP Requests.

4.17.1. Ansatz

Der Datenzugriff auf andere Service-APIs werden über Repositories abstrahiert. Die nötigen Informationen zu Base-URL, Endpoint und API-Key sind in der jeweiligen Repository-Konfiguration zu finden.

4.17.2. Umsetzung

Es wurde ein eigener Web-Client geschrieben, der die gängigsten HTTP-Methoden (GET, POST, PUT, DELETE) beherrscht und auf dem .NET-HTTP-Client basiert. Dieser Client beinhaltet zudem ein marginales Error-Handling und ist zuständig für das Mapping von JSON-Responses in DTOs. Durch das Bestehen des *IWebClient*-Interfaces ist es möglich, diese Implementation in Zukunft einfach auszutauschen. Um den Web-Client über Dependency Injection nutzen zu können, existiert die *HttpFactory* mit dem Interface *IHttpFactory*, welche dafür da ist einen entsprechenden Web-Client mit der richtigen Repository-Konfiguration zu erstellen. Ohne die Factory müsste jeder, der diesen Client verwendet, die Konfiguration des Clients vornehmen.

4.17.2.1. Eigene Web-Library

Mehrere Services nutzen andere Service-APIs für Datenabfragen. Der Web-Client wurde deswegen in ein eigenes NuGet-Package ausgelagert.

4.18. Caching

Aufgrund des Einsatzes von Microservices sind die einzelnen Services zum Teil abhängig von anderen Services und müssen diese Daten über das API des jeweiligen Services beziehen. Diese Requests dauern in der Regel nicht lange, trotzdem kann es aufgrund der hohen Anzahl an Requests zu Problemen kommen. Deshalb werden Daten punktuell und lokal gecached.

4.18.1. Ansatz

Caching kann zu einigen Problemen führen wenn es zu früh oder falsch eingesetzt wird. Ein Problem ist die Korrektheit der Daten oder der Zugriff auf den Cache in einer Multithreading-Applikation. Deshalb sollen Daten erst so spät wie möglich und so kurz wie nötig gecached werden.

4.18.2. Umsetzung

Zur Anwendung kommt der MemoryCache des .NET Core Frameworks. Dieser unterstützt das Setzen von Items in den Cache mit einer Ablaufzeit. Die gecachten Einträge werden so kurz wie möglich gechached.

4.18.3. Verwendung

Konkret wurde Caching nur im Answer Service eingesetzt, da dort sehr viele Requests zur gleichen Zeit ankommen und Daten von anderen Services für jeden ankommenden Request angefordert werden müssten. In anderen Services wurde keine Notwendigkeit für Caching gefunden.

4.18.3.1. Cache-Zeit

Im MemoryCache von .NET kann eine Lebensdauer der Einträge definiert werden. Diese wird bewusst sehr kurz gehalten. Somit kann eine Inkonsistenz der Daten im Cache mit einem vertretbarem Restrisiko ausgeschlossen werden. Als Beispiel ist die Lebensdauer der Antwortmöglichkeiten im Answer Service, die für die Validierung der Antworten benötigt werden auf eine Minute gesetzt. Diese Zeit wurde aus der Überlegung gesetzt, dass eine Antwort im Rahmen einer Durchführung einer Präsentation abgegeben wird und sich während einer Durchführung die Präsentation an sich, also die Antwortmöglichkeiten, nicht ändern werden.

4.18.3.2. Thread-Safety

Der MemoryCache des .NET Frameworks ist Thread-Safe [26]. Für .NET Core ist keine Angabe ersichtlich, es wird angenommen das diese Angabe für .NET Core ebenfalls zutrifft. Ein Problem mit einer Multithreading-Applikation ist allerdings vorhanden. Falls zwei Threads auf den Cache zugreifen, bevor dieser die benötigten Daten beinhaltet,

laden beide Threads die Daten und setzen diese im Cache. Dies ist kein Problem, da das Laden der Daten nur eine kurze Zeit dauert und die angeforderten Daten mit sehr hoher Wahrscheinlichkeit dieselben sind.

4.18.3.3. Distributed Caching

Auf ein Distributed Caching wird verzichtet. Die gecachten Einträge werden nur kurz zwischengespeichert und die Zeit, die für das Laden der Einträge verwendet wird ist relativ gering. Deshalb kann akzeptiert werden dass mehrere Instanzen diese Einträge selber laden und in ihrer Instanz cachen. Zudem werden nur Einträge gecached, die sich während der Cachezeit mit hoher Wahrscheinlichkeit nicht ändern, weshalb es möglich ist, diese lokal zu cachen.

4.19. Schnittstellen

Alle Services bieten ein öffentlich API an. Damit dieses für Entwickler ersichtlich und dokumentiert ist, wird Swagger verwendet. Swagger bietet zum einen ein JSON an, das die Schnittstelle beschreibt. Zum anderen existiert die Möglichkeit, ein benutzerfreundliches UI zu veröffentlichen. In diesem werden alle möglichen Operationen inklusive allen Models beschrieben.

4.19.1. Umsetzung

Damit die Swagger-Dokumentation immer auf einem aktuellen Stand ist, wird diese zum Start jedes Services automatisch generiert. Für diese Generierung und das UI wird Swashbuckle [3] eingesetzt. Der Einsatz von Swagger in der Konfiguration deaktiviert werden, so dass beispielsweise die Dokumentation auf der produktiven Umgebung nicht verfügbar ist und so keine Details über die Umsetzung preisgegeben werden. Im Rahmen der Bachelorarbeit wurde Swagger aber auf allen Umgebungen aktiviert. Die Abbildung 49 zeigt ein Ausschnitt aus der Swagger-Dokumentation des Presentation Service.

[3] `https://github.com/domaindrivendev/Swashbuckle.AspNetCore`

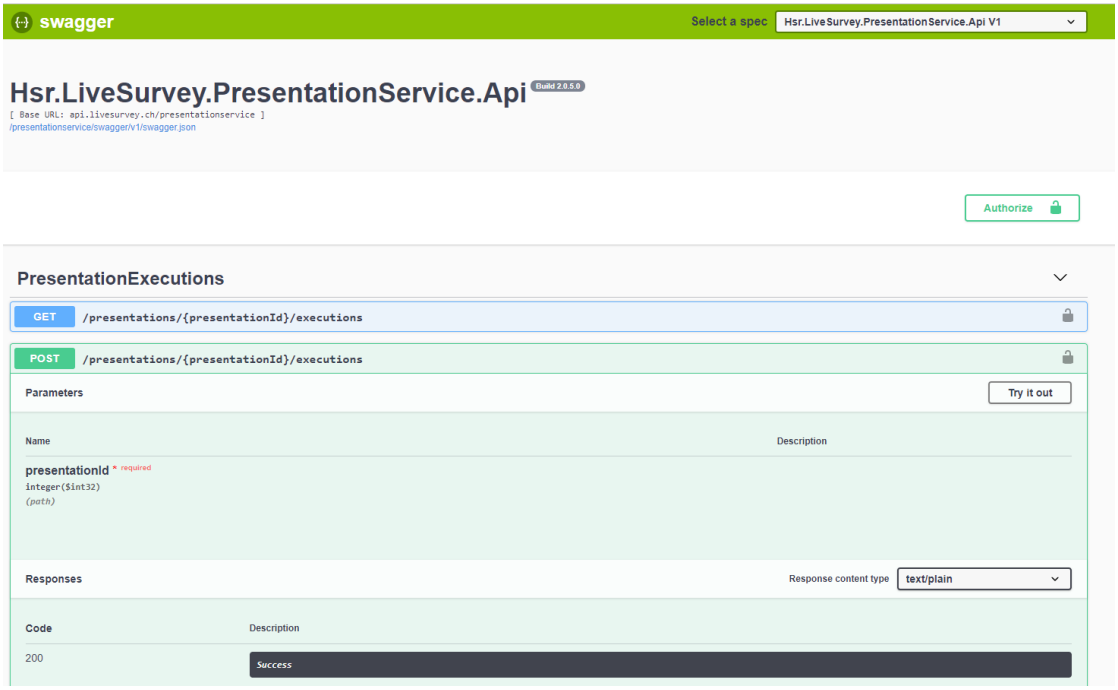


Abbildung 49.: Ausschnitt aus der Swagger-Dokumentation
Quelle: Printscreen, api.livesurvey.ch/presentationservice/

4.19.2. Verfügbarkeit

Bei allen Services, ausgeschlossen sind die SignalR-Hubs, da diese keine konkreten Operationen anbieten, existiert ein Swagger-UI. Dieses ist bei allen Services auf die Root-URL konfiguriert. Die Tabelle 11 listet die Links zu allen verfügbaren Schnittstellen-Beschreibungen in der produktiven Umgebung auf.

Service	**Swagger-Domain**
Answer Service	http://api.livesurvey.ch/answerservice/
Authentication Service	http://api.livesurvey.ch/authenticationservice/
Presentation Service	http://api.livesurvey.ch/presentationservice/
Question Service	http://api.livesurvey.ch/questionservice/

Tabelle 11.: Urls der Swagger-Dokumentationen der Services

4.19.3. Nachvollziehbarkeit

Alle Swagger-UIs können über die in der Tabelle 11 aufgerufen werden. Zudem können die Swagger-UIs auch lokal beim Starten der Services aufgerufen werden. Diese sind dort ebenfalls auf die Root-URL konfiguriert.

5. DevOps

5.1. Einführung

DevOps beschreibt ein Set von Entwicklungspraktiken, welche zum Entwickeln und Betreiben von Software gehören. Dabei geht es besonders um Aktivitäten wie Continuous Integration oder Continuous Deployment. Mit Azure DevOps Service bietet Microsoft eine Plattform an, die solche Dinge möglich macht. Dieses Kapitel beschreibt, wie DevOps zusammen mit Azure DevOps im Rahmen der Bachelorarbeit verwendet wird.

5.2. Sourcecode-Verwaltung

Der Sourcecode von Software, aber auch von Dokumenten, wird auf Azure DevOps in mehreren Git-Repositories verwaltet.

5.2.1. Dokumente

Die Dokumentation wird mit LaTeXerstellt. Die erstellten Files werden alle in einem eigenen Repository verwaltet. So entstehen keine Probleme und Durcheinander beim Builden von Software-Komponenten.

5.2.1.1. Branching-Strategie

Der Einfachheit halber werden Dokumente nur in einem Branch verwaltet. Für die Sicherstellung der Qualität werden diese jeweils vor dem Push einem Review unterzogen.

5.2.2. Software-Sourcefiles

In den anderen Repositories wird der Sourcecode der Software-Komponenten verwaltet. Es existiert pro Service ein Git-Repository.

5.2.2.1. Branching-Strategie

Für die Verwaltung von Sourcecode wird eine etwas komplexere Branching-Strategie gefahren. Diese baut auf zwei Hauptbranches und mehreren Arbeits- und Releasebranches auf und wurde von Vincent Driessen dokumentiert [52]. Nachfolgend werden die einzelnen Branches dieser Strategie kurz beschrieben. Eine Übersicht ist in der Abbildung 50 abgebildet.

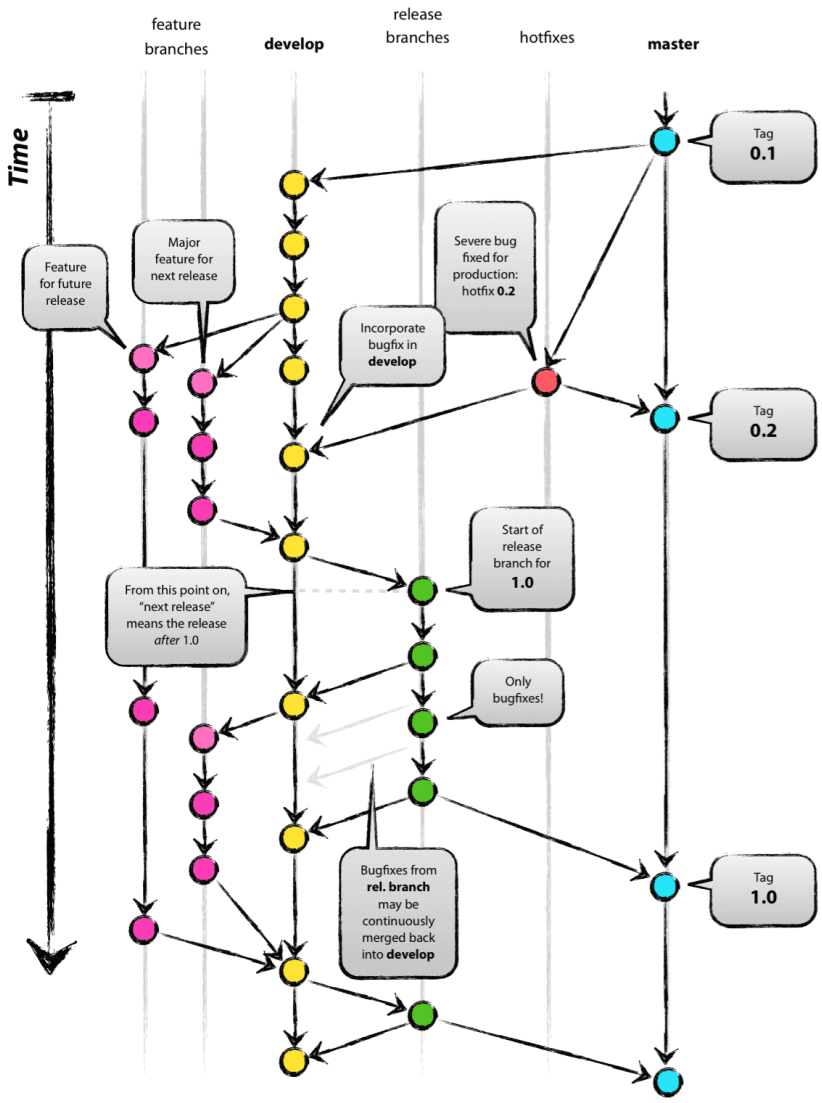


Abbildung 50.: Git-Branching Strategie
Quelle: `https://nvie.com/posts/a-successful-git-branching-model/`

Develop Der aktuellste, saubere Entwicklungsstand wird im Develop-Branch verwaltet. Auf diesen Branch darf kein Entwickler direkt neuen Code pushen, Änderungen daran werden nur via Pull-Request ausgeführt. Dies stellt sicher dass der neue Code einem Review unterzogen wurde und beim Merge keine Konflikte erzeugt. So steht jederzeit eine lauffähige Version zur Verfügung.

Feature/* Um Features ohne grossen Druck und ohne Verschmutzung des Develop-Branches implementieren zu können, wird pro Feature ein Branch erstellt. Dieser Branch

darf durchaus unfertigen Code enthalten und ist persönlich für den Entwickler zur Sicherung seiner Arbeit gedacht.

Release/* Vor einem Release wird vom Develop-Branch ein neuer Release-Branch mit der neuen Version erstellt. Auf diesem Release-Branch können allfällige Fehler, die beim Testen aufgetaucht sind, behoben werden. Ebenfalls dienen diese Branches der Nachvollziehbarkeit und ermöglichen es, auf einer alten Version einen Hotfix durchzuführen. Nach jedem Release werden allfällige Änderungen zurück in den Develop-Branch gemerged.

Master Nach einem Release wird der Release-Branch in den Master-Branch gemerged und getagt. So kann zu jederzeit nachvollzogen werden, welcher Code in welcher Version ausgerollt wurde.

5.3. Continuous Integration und Deployment

5.3.1. Infrastruktur

Für das Builden und Deployen von Applikationen werden Build- und Deploy-Agents benötigt. Diese werden von Microsoft zur Verfügung gestellt [1] und existieren für Windows, Linux und Mac OS X. Aufgrund der Anforderung von Docker an Windows wurde ein Linux Build Agent auf einem HSR-Server eingerichtet.

5.3.2. Continuous Integration

Jeder Push von neuem Code soll grundsätzlich ein Build auslösen welcher den neuen und bestehenden Code builded und testet. Es existieren verschiedene Arten von Builds, diese sind nachfolgend beschrieben.

Merge-Builds Bei einem Merge-Build werden die Sourcen gebuildet und Tests ausgeführt. Diese Builds stellen sicher, dass Code nach einem Merge noch immer funktionsfähig ist. Solche Arten von Builds werden bei Pull-Requests als Merge Policy eingerichtet.

Feature-Builds Feature-Builds builden Komponenten, führen Tests aus und erstellen Images für ein späteres Deployment. Länger dauernde Integrations-Tests können in einem solchen Build weggelassen werden. Ein solches Deployment wird aber nicht automatisch ausgeführt. Feature-Builds werden automatisch für Feature-Branches ausgeführt.

Release-Builds Analog den Feature-Builds führen auch Release-Builds das Kompilieren und Testen von Komponenten durch. Hier werden aber alle Tests durchgeführt, da solche Builds in der Regel nicht sehr häufig ausgeführt werden. Release-Builds triggern automatisch ein Deployment auf die entsprechende Umgebung.

[1] `https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents`

5.3.3. Continuous Deployment

Je nach Build wird automatisch ein Deployment auf die entsprechende Umgebung ausgeführt. Es existieren drei Umgebungen:

- Entwicklung
- Integration
- Produktion

Entwicklung Der Stand in der Entwicklung kann je nach Zeit variieren und auch einmal nicht-fertige Features beinhalten. Diese Umgebung ist für die Entwickler reserviert. Die Domains der Entwicklungsumgebung beinhalten alle vor dem Domain-Namen das Prefix *dev*, so könnte beispielsweise die Domain des Presentation Designers auf der Entwicklungsumgebung *design.dev.livesurvey.ch* lauten.

Integration Um aussagekräftige Tests durchführen zu können, muss eine Umgebung für solche Tests existieren. In der Integrations-Umgebung können Tests über mehrere Komponenten in einem definierten Zustand durchgeführt werden. Die Domains der Integrationsumgebung beinhalten alle vor dem Domain-Namen das Prefix *stage*, so könnte beispielsweise die Domain des Presentation Designers auf der Integrationsumgebung *design.stage.livesurvey.ch* lauten.

Produktion Endbenutzer kommunizieren während der Nutzung der Software nur mit der produktiven Umgebung. Die Domains der produktiven Umgebungen beinhalten kein Prefix. Dies würde den Benutzer allenfalls verwirren oder Aufschluss über die Struktur der Domains für Angreifer geben. So könnte beispielsweise die Domain des Presentation Designers auf der produktiven Umgebung *design.livesurvey.ch* lauten.

5.3.4. Technologien und Ablauf

Bei den Build und Release-Pipelines kommen einige Technologien zum Einsatz. Zusammen mit dem Ablauf eines Builds und Releases sind diese in der Abbildung 51 ersichtlich.

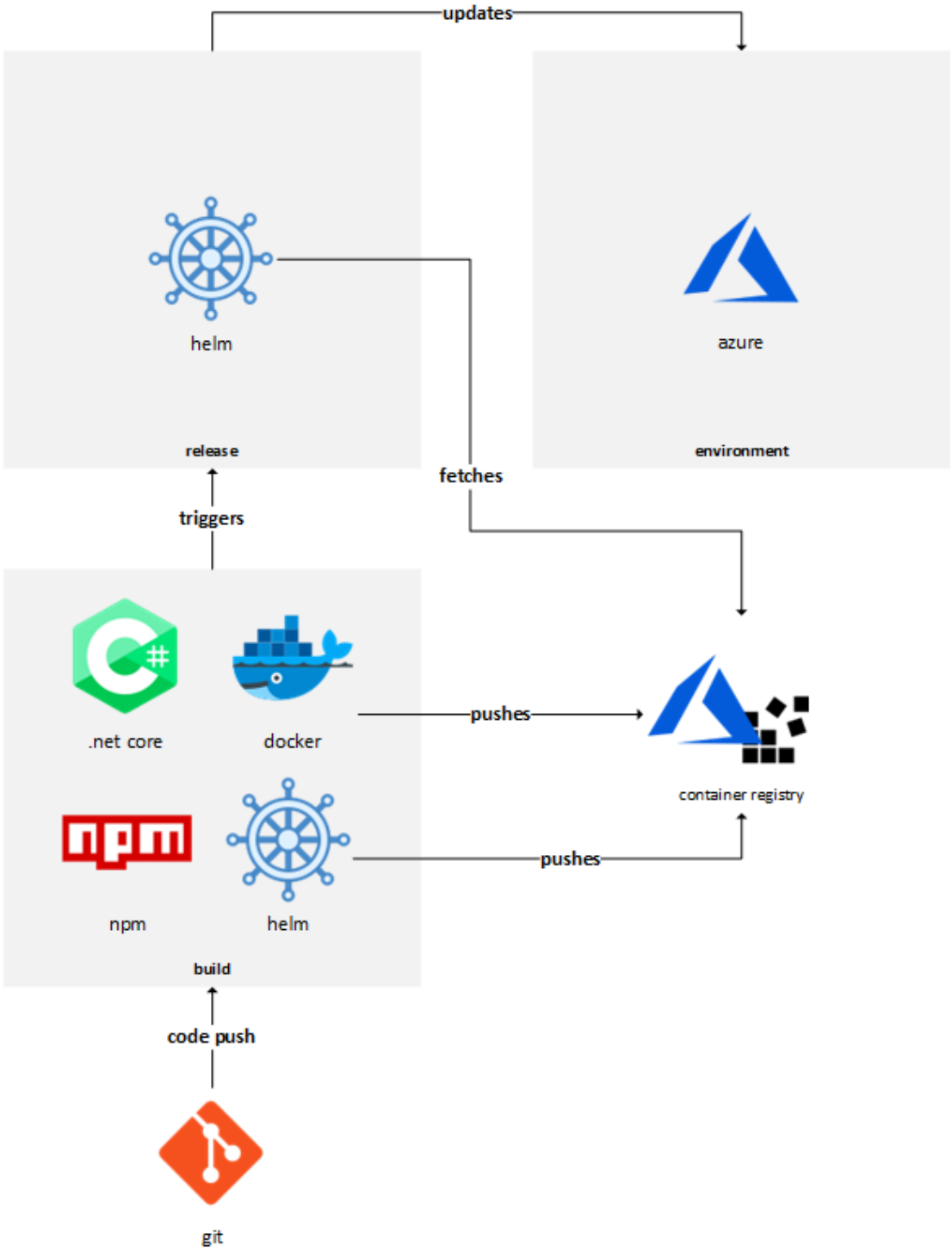


Abbildung 51.: Ablauf Build und Deployment
Quelle: Eigene Darstellung, Icons `https://icons8.com`

Container Registry In Azure steht eine private Container Registry zur Verfügung, in welcher unsere Docker-Images und Helm-Charts verwaltet werden.

Helm Helm ist ein Package Manager für Kubernetes, mit welchem sich mehrere Kubernetes-Ressourcen zu einem Package, genannt Chart, zusammenpacken lassen. Zudem ermöglicht es Helm, in diesen Ressourcen-Files Variablen zu verwenden, welche beim Einspielen eines Charts setzen lassen. So können dieselben Packages für mehrere Umgebungen verwendet werden. Mit Helm werden unsere eigenen Kubernetes-Charts beim Build erstellt und später mit dem Release eingespielt.

Docker Mit Docker werden die Images erstellt.

.NET Core Mit C# als Sprache und .NET Core werden unsere Projekte gebuildet und Tests ausgeführt.

NPM Node Package Manager, kurz NPM, wird verwendet um die User Interfaces von React zu JavaScript umzuwandeln und den Code auf Styleguides zu überprüfen.

5.3.5. Versionierung

Um jederzeit einen Überblick über die im Einsatz stehenden Versionen zu erhalten, werden alle Komponenten beim Kompilieren versioniert. Die Versionierung findet in der Build-Puipeline in Azure DevOps statt. Es existiert kein Automatismus, diese Build-Versionen automatisch hochzuzählen, deshalb müssen diese von Hand gesetzt werden. Damit ein zentraler Ort mit diesen Versionen existiert, werden diese in einer *Variable Group* [44] verwaltet.

5.4. Konfiguration

Die Konfiguration der Komponenten erfolgt in zwei verschiedenen Varianten. Beide sind ähnlich und basieren auf Umgebungsvariabeln. Da bei React diese aber zur Laufzeit nicht zur Verfügung stehen ist bei den Frontends eine etwas angepasst Version im Einsatz.
Die Grundanforderung des Teams an die Konfiguration war, dass diese nicht zur Kompilationszeit geschieht sondern erst zur Laufzeit. Dies wird auch von der Twelve-Factor App so als Best Practice beschrieben [41].

5.4.1. Konfiguration in Charts

Mit Helm gibt es die Möglichkeit, dem Chart Variabeln bei der Installation mitzugeben. Diese Variablen werden zum einen verwendet um die korrekte Version des Images zu installieren, zum anderen werden diese als Schleuse für die Umgebungsvariabeln für den auszuführenden Docker-Container verwendet.

5.4.2. Backend Services

ASP.NET Core bietet verschiedene Möglichkeiten an um Konfigurationen vorzunehemen. Diese reichen von verschiedenen Konfigurationsfiles über Umgebungsvariabeln bis hin zu Kommandozeilenargumenten oder Keys in Azure. Implementiert wurde die Variante der Umgebungsvariabeln, da diese einfach bei der Installation eines Services via Helm dem auszuführenden Docker-Container weitergegeben werden können. Dank dem ASP.NET Core Framework stehen dem Service alle Umgebungsvariabeln zur Verfügung, die mit dem Prefix ASPNETCORE beginnen. Diese werden dann während dem Start Klassen zugeordnet und so den Services und Repositories übergeben. Dafür wird das zur Verfügung gestellte *Options* Pattern [30] verwendet. Das Listing 5.1 zeigt, wie diese Options im Startup konfiguriert und denn in einem Service verwendet werden können.

```
1  // Startup.cs
2  services.Configure<TokenProviderConfiguration>(configuration.GetSection("TokenProvider
       "));
3
4  // Konstruktor in TokenProvider.cs
5  public JwtTokenProvider(IOptions<TokenProviderConfiguration> configuration)
6  {
7  }
```

Listing 5.1: Konfiguration und Verwendung der Optionen

5.4.3. Frontends

Die Frontends wurden in React implementiert, welche als Browser-Applikation auf dem Client läuft. Durch diese Gegebenheit stehen keine Umgebungsvariabeln zur Verfügung. React bietet ein Handling mit Umgebungsvariabeln an, diese werden aber zur Buildzeit eingebettet [35]. Um die Umgebungsvariabeln zur Laufzeit angeben zu können, wird im Docker-Container beim Start ein Script ausgeführt, das ein JavaScript-File mit den Umgebungsvariabeln als Inhalt erzeugt. Dieses JavaScript-File setzt Variablen auf dem *window*-Objekt und wird in der Index-Seite der React-App eingebunden. Diese Variabeln können so von React gelesen werden. Damit die Umgebungsvariabeln verwendet werden können müssen diese im *.env*-File deklariert und mit einem Default-Wert angegeben werden. Das Listing **??** zeigt ein Beispiel eines solchen generierten JavaScript-File.

```
1  window._env_ = {
2    ENVIRONMENT: "develop",
3    VERSION: "1.0.0"
4  }
```

Listing 5.2: Konfiguration und Verwendung der Optionen

Dieser Ansatz wurde von Krunoslav Banovac in einem Artikel [50] beschrieben und in einer leicht angepassten Version übernommen.

5.4.4. Azure DevOps Release Pipeline

Die Umgebungsvariabeln werden beim Release eines Helm Charts in der Azure DevOps Release Pipeline im Task *helm upgrade* gesetzt. Die Abbildung 52 zeigt die Konfiguration dieses Schrittes am Beispiel des Presentation Designers.

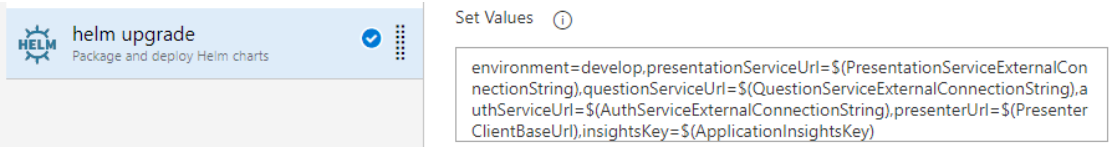


Abbildung 52.: Setzen der Variablen in Helm Upgrade
Quelle: Printscreen, dev.azure.com

Damit die Konfiguration in Azure DevOps an einem zentralen Ort verwaltet werden kann sind die Werte in so genannten *Variable Groups* [44] abgespeichert. Diese können den Pipelines auf Stage-Ebene zugewiesen werden. So können die Tasks der Pipeline einheitlich aufgebaut werden und beziehen die Werte aus den Gruppen.

5.5. Monitoring und Logging

Um einen schnellen Überblick über den Zustand der Services zu erhalten müssen diese überwacht werden. Da ein Einblick in ein Docker-Container zur Laufzeit schwer ist müssen zusätzlich Logs geschrieben und gesammelt werden. Da als Cloud-Anbieter Azure verwendet werden muss bietet sich der Einsatz von Application Insights [27] an. Ein Überblick über Application Insights und dessen Einsatz in HSR Live Survey wird in den nachfolgenden Unterkapiteln beschrieben.

5.5.1. Application Insights

Application Insights ist ein Application Performance Management Service von Azure. Dieser bietet das Überwachen Applikationen, vorzugsweise Web Applikationen, an. Application Insights sammelt nach der Konfiguration automatisch alle einkommenden Requests und Calls an abhängige Services und Infrastruktur. Alle Daten werden im Azure Portal in einem übersichtlichen Dashboard angezeigt und können jeweils bis in eine tiefen Detailgrad analysiert werden. Mit Application Insights lassen sich auch Availability Tests erstellen um so die Verfügbarkeit einer Applikation zu messen.

5.5.1.1. Backend Services

In allen Services wird Application Insights verwendet. Somit werden automatisch alle Requests inklusive Detail wie Dauer und Statuscode gesammelt und an Azure gesendet. So können Fehler besser analysiert und behoben werden. Zudem kann die Performance einer Web-Applikation so gemessen und auf Probleme reagiert werden. Zudem werden

alle Log-Messages ab einem konfigurierbarem Level in Application Insights gesammelt [24].

Rollen Azure Application Insights arbeitet mit einem sogenannten *Instrumentation Key*. Dieser wird für die Verbindung benötigt und ist bei allen Services derselbe. Damit im Dashboard die Services auseinandergehalten werden können werden den Services Rollen zugewiesen. Jeder Service besitzt sein Start-Assembly-Name als Rolle. Somit kann dieser im Dashboard identifiert werden.

Version Bei den Logs werden von Application Insights einige Daten automatisch gesammelt. Es können aber weitere Daten manuell hinzugefügt werden. Eine wichtige Angabe ist die Version des Service, der für den Eintrag verantwortlich ist. Dafür wird die Assembly-Version verwendet. Das Listing 5.3 zeigt die Konfiguration der Version in Application Insights. Abbildung 53 zeigt, wie diese in den Details eines Eintrags im Dashboard angezeigt wird.

```
1  services.AddApplicationInsightsTelemetry(o => o.ApplicationVersion = System.Reflection.
       Assembly.GetExecutingAssembly().GetName().Version.ToString(3));
```

Listing 5.3: Konfiguration der Version von Application Insights

Abbildung 53.: Anzeige der Application Version im Dashboard
Quelle: Printscreen, portal.azure.com

5.5.1.2. Frontends

In den Frontends wird auf den Einsatz von Application Insights verzichtet. Fehler können dort schnell von Benutzern gemeldet werden. Zudem befindet sich die wichtige und komplexe Logik in den Services und müssen dort überwacht werden.

5.5.1.3. Availability Tests

In Application Insights wurden Availability Checks für alle Backend Services und Frontends eingerichtet. Diese überprüfen alle 15 Minuten von verschiedenen Standorten die Verfügbarkeit der Komponenten über einen HTTP Request. Die Abbildung 54 zeigt das Resultat eines solchen Tests. Nebst der Verfügbarkeit ist auch die durchschnittliche Request-Dauer ersichtlich. Da der Benutzerkreis von HSR Live Survey sich in Europa befindet wurden alle Azure Locations in Europa als Test Locations konfiguriert.

AVAILABILITY TEST	20 MIN	AVAILABILITY	DURATION (AVG)
Overall	**100.00%**	**100.00%**	**113 ms**
Answer Service	100.00%	100.00%	112 ms
France Central	100.00%	100.00%	50.9 ms
France South	100.00%	100.00%	108 ms
North Europe	100.00%	100.00%	119 ms
West Europe	100.00%	100.00%	167 ms

Abbildung 54.: Resultat eines Availability Tests in Azure
Quelle: Printscreen, portal.azure.com

5.5.1.4. Performance

Um eine Übersicht über die Performance aller Services zu erhalten gibt es in Azure das Performance Dashboard. Die Daten dafür werden von Application Insights automatisch gesammelt. Die Abbildung **??** zeigt dieses Dashboard.

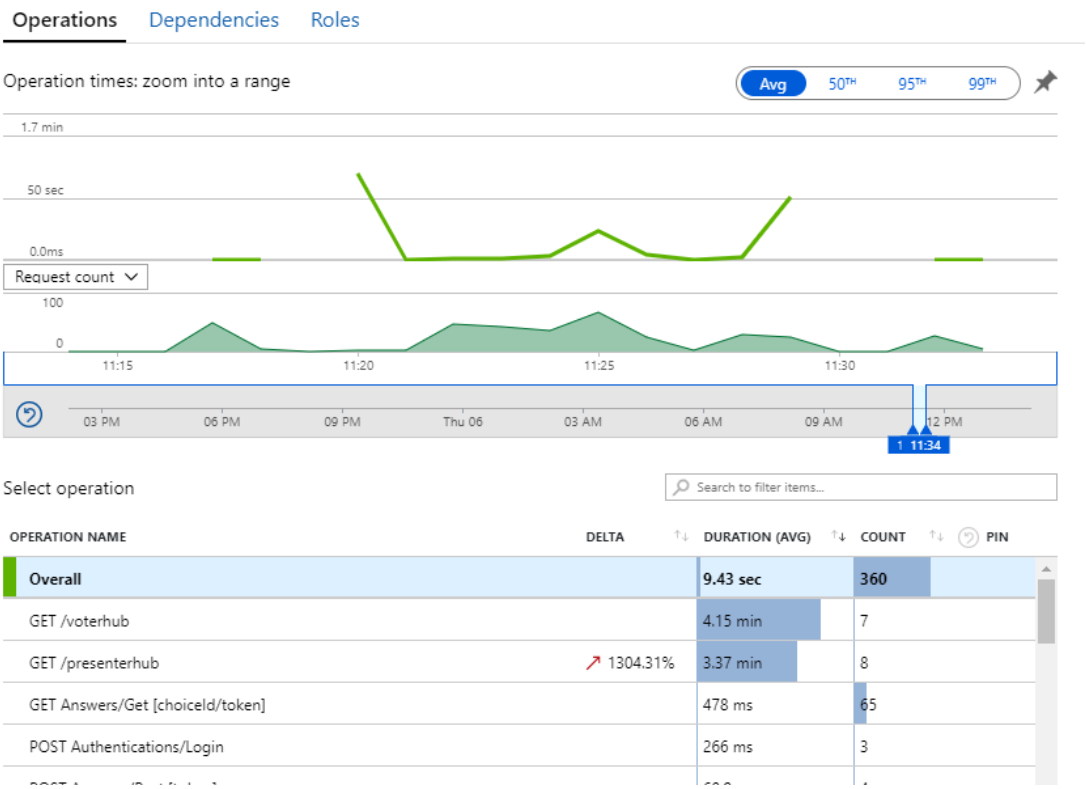


Abbildung 55.: Performance-Dashboard in Azure
Quelle: Printscreen, portal.azure.com

Ein Problem in dieser Übersicht stellen die beiden SignalR-Hubs dar. Denn diese öffnen eine Verbindung und halten diese via WebSockets offen. Dies wird von Application Insights nicht korrekt erkannt, deshalb erscheinen bei diesen Operationen (GET voterhub und GET presenterhub) immer sehr hohe Zahlen. Diese beiden Endpoints können aber mit einem Filter deaktiviert werden.

5.5.2. Logging

In den Backend Services werden auf allen Layers Logs geschrieben. Diese Logs werden zum einen auf die Konsole geschrieben und können über das Kubernetes Dashboard oder kubectl betrachtet werden. Zum anderen werden gewisse Logs in Application Insights geschrieben und können über das Dashboard in Azure angesehen werden. Es existieren verschiedene Log-Kategorien und für jede Kategorie kann die minimale Stufe definiert werden. Dafür existieren die Umgebungsvariabeln *ASPNETCORE__Logging___LogLevel___Default* in den jeweligen Services.

6. Testing

6.1. Load Tests

In diesem Kapitel werden die Ergebnisse der Load Tests zusammengefasst.

6.1.1. Ausgangslage

Getestet wird die Situation von der Vorbereitung bis zur Präsentationsdurchführung (Antwortabgabe) einer Vorlesung.

6.1.1.1. Gleichzeitige Benutzer

Gestartet wird jeweils mit einem Benutzer. Die maximale Anzahl Benutzer soll in 10 Sekunden erreicht werden.

Vorbereitung Die Vorbereitung einer Präsentation wird mit 10 Benutzern getestet, welches die Anzahl von Dozenten repräsentieren soll.

Durchführung 50 Benutzer entsprechen im Durchschnitt etwa der Anzahl von Studierenden während einer Vorlesung.

6.1.1.2. Explizit involvierte Systeme

Die Anfragen während dem Load Tests werden explizit an folgende Systeme gestellt:

- Answer Service
- Presentation Service
- Question Service

Folgende Systeme werden beim zweiten Durchgang auf zwei Instanzen skaliert:

- Answer Service
- Presentation Service
- Presenter Hub
- Question Service
- Voter Hub

6.1.1.3. Aufteilung

Zuerst werden alle Tests durchgeführt, welche die Erstellung von Präsentationen, Slides und Fragen simuliert. Anschliessend stehen die Tests bezüglich Präsentationsdurchführung an, wobei zu beachten ist, dass nicht alle Endpoints angesprochen werden können aufgrund des Einsatzes von Websockets.

6.1.1.4. Ablauf

Jeder Teil wird mit der definierten Anzahl gleichzeitiger Benutzer zwei Mal gestartet. Ein Durchgang dauert 5 Minuten. Dabei sind folgende Aspekte zu beachten:

- Im ersten Durchlauf existiert pro Service nur eine Instanz.
- Im zweiten Durchgang werden die Services auf zwei Instanzen hochskaliert.

6.1.1.5. Infrastruktur

Eingesetzt wird Locust, welches lokal in einem Docker Container betrieben wird. Die Tests befinden sich im *Infrastructure*-Repository unter dem Ordner *locust*.

6.1.2. Ergebnisse

6.1.2.1. Teil Vorbereitung: Durchgang 1

In der Abbildung 56 ist ersichtlich, dass während dem Test keine Fehler aufgetreten sind und alle Anfragen problemlos beantwortet werden konnten.

Type	Name	# requests	# fails	Median (ms)	Average (ms)	Min (ms)	Max (ms)	Content Size (bytes)	# reqs/sec
GET	/presentationservice/presentations	132	0	40	46	33.2338809967041	540.9848690032959	7710	0.5
POST	/presentationservice/presentations	121	0	55	70	46.89908027648926	620.4555034637451	113	0.6
POST	/presentationservice/presentations/6644/slides	129	0	57	64	48.8741397857666	542.5455570220947	144	0.5
POST	/questionservice/questions	716	0	58	129	47.82605171203613	4003.6685466766357	123	2.2
GET	/questionservice/questions/slides/6608	776	0	40	60	34.2409610748291	3018.6123847961426	441	2.6
	Total	**1874**	**0**	**52**	**86**	**33.2338809967041**	**4003.6685466766357**	**790**	**6.4**

Abbildung 56.: Teil Vorbereitung 1. Durchgang Übersicht
Quelle: Eigene Darstellung

Im ersten Durchgang wurden 90 % aller Requests in maximal 86 ms abgearbeitet. In den anderen Fällen dauerte die Abfertigung der Anfragen mehr als 500 ms (siehe 100. Perzentil). Die erhöhte Responsetime hängt mit der Datenbank zusammen, welche von mehrere Services gleichzeitige Lese- und Schreiboperationen abhandeln muss.

Name	# requests	50%	66%	75%	80%	90%	95%	98%	99%	100%
GET /presentationservice/presentations	132	40	44	46	48	50	54	82	98	540
POST /presentationservice/presentations	121	55	57	66	70	79	91	280	490	620
POST /presentationservice/presentations/6644/slides	129	56	61	64	66	75	80	120	120	540
POST /questionservice/questions	716	57	63	67	72	86	310	1100	2500	4000
GET /questionservice/questions/slides/6608	776	40	42	45	47	55	66	270	420	3000
Total	**1874**	**51**	**55**	**59**	**62**	**75**	**93**	**500**	**1000**	**4000**

Abbildung 57.: Teil Vorbereitung 1. Durchgang Statistik
Quelle: Eigene Darstellung

Die Anzahl Requests pro Sekunde und Benutzer blieben über die Testzeit ziemlich konstant.

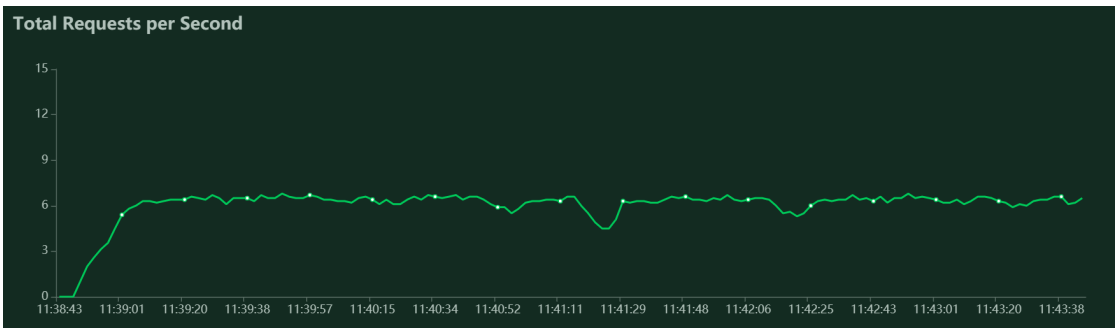


Abbildung 58.: Teil Vorbereitung 1. Durchgang Requests Per Seconds
Quelle: Eigene Darstellung

Die grüne Linie entspricht dem Median, die orange Linie mit gewissen Ausreissern bildet das 95. Perzentil ab. Die Ausreisser ergaben sich ebenfalls aus der Tatsache, dass die Datenbank sehr stark beansprucht wurde.

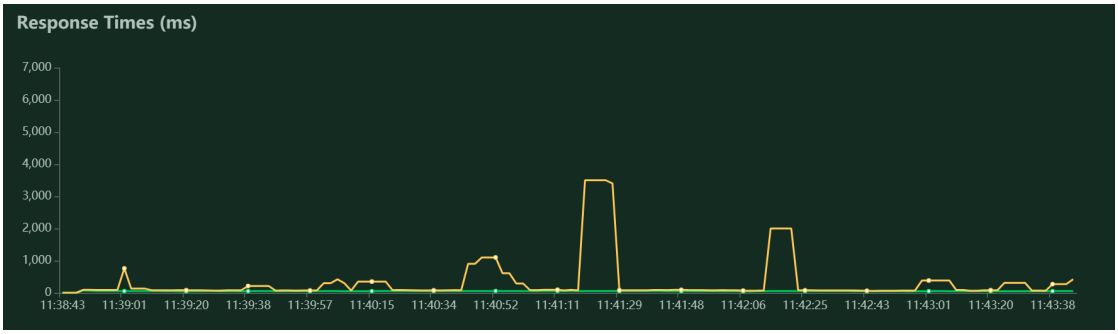


Abbildung 59.: Teil Vorbereitung 1. Durchgang Response Time
Quelle: Eigene Darstellung

6.1.2.2. Teil Vorbereitung: Durchgang 2

Beim zweiten Durchgang sind einige Fehler aufgetreten und weniger Requests abgearbeitet worden. Durch die Hochskalierung wurde die Datenbank mit noch mehr gleichzeitigen Anfragen belastet.

Type	Name	# requests	# fails	Median (ms)	Average (ms)	Min (ms)	Max (ms)	Content Size (bytes)	# reqs/sec
GET	/presentationservice/presentations	96	9	39	73	33.08582305908203	1403.381586074829	5576	2.3
POST	/presentationservice/presentations	81	7	71	123	47.258853912353516	1948.958396911621	113	0.7
POST	/presentationservice/presentations/7710/slides	80	3	66	95	49.68833923339844	578.3476829528809	144	1.3
POST	/questionservice/questions	39	4	70	107	48.30789566040039	689.5368099212646	123	0.4
GET	/questionservice/questions/slides/7676	66	4	40	49	34.6684455871582	279.09207344055176	441	1.6
	Total	**362**	**27**	**54**	**88**	**33.08582305908203**	**1948.958396911621**	**1630**	**6.3**

Abbildung 60.: Teil Vorbereitung 2. Durchgang Übersicht
Quelle: Eigene Darstellung

Maximal 100 ms Response Time in 90 % der Fälle wurden nur noch bei den Leseoperationen und bei der Slideerstellung erreicht.

Name	# requests	50%	66%	75%	80%	90%	95%	98%	99%	100%
GET /presentationservice/presentations	96	39	41	45	49	69	270	560	1400	1400
POST /presentationservice/presentations	81	71	81	91	110	140	250	1200	1900	1900
POST /presentationservice/presentations/7710/slides	80	66	79	92	100	160	300	350	580	580
POST /questionservice/questions	39	69	76	94	120	190	420	690	690	690
GET /questionservice/questions/slides/7676	66	39	41	45	46	52	66	260	280	280
Total	**362**	**54**	**66**	**75**	**81**	**140**	**250**	**400**	**690**	**1900**

Abbildung 61.: Teil Vorbereitung 2. Durchgang Statistik
Quelle: Eigene Darstellung

Die Anzahl Requests pro Sekunde war ziemlich konstant bis zu einem kurzen Einbruch, welche sich danach wieder stabilisiert hat.

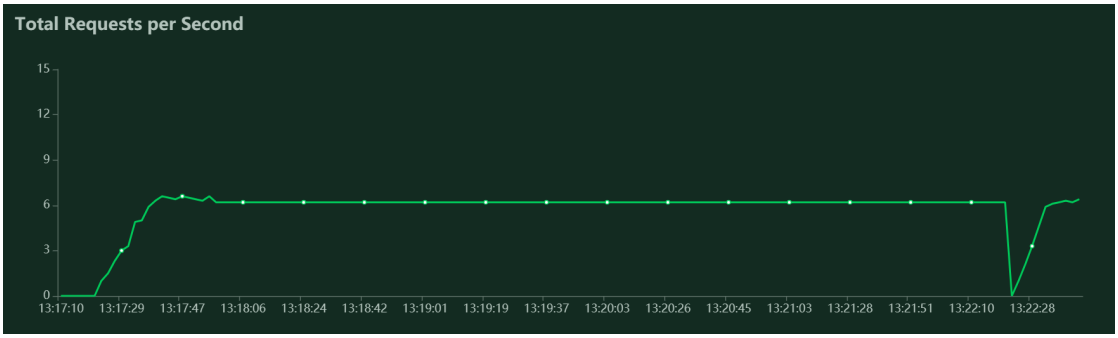


Abbildung 62.: Teil Vorbereitung 2. Durchgang Requests Per Seconds
Quelle: Eigene Darstellung

Es bestand eine lange Zeit, in der gar keine Requests mehr abgearbeitet wurden. Dies zeigt sich unteranderem in der Abbildung 63 und bei der Übersicht in Abbildung 60. Der lange Unterbruch ist auf die Datenbank zurückzuführen.

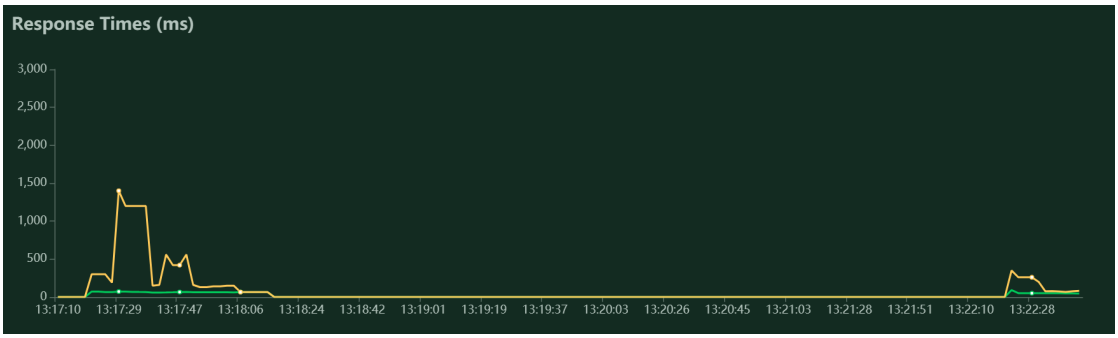


Abbildung 63.: Teil Vorbereitung 2. Durchgang Response Time
Quelle: Eigene Darstellung

6.1.2.3. Teil Durchführung: Durchgang 1

Bei der Antwortabgabe im ersten Durchgang traten einige Fehler auf, welches einer Rate von 3.5 % entspricht. Die Maximalwerte befinden sich bereits im zweistelligen Sekundenbereich. Hier ist die Datenbank mit mehreren Schreiboperationen die Ursache für die langen Antwortzeiten.

Type	Name	# requests	# fails	Median (ms)	Average (ms)	Min (ms)	Max (ms)	Content Size (bytes)	# reqs/sec
POST	/answerservice/answers/executions/221B8B	903	32	500	10572	127.2726058959961	55527.55045890808	272	0
	Total	**903**	**32**	**500**	**10572**	**127.2726058959961**	**55527.55045890808**	**272**	**0**

Abbildung 64.: Teil Durchführung 1. Durchgang Übersicht
Quelle: Eigene Darstellung

Die hälfte der Anfragen konnten nur mit einer minimalen Antwortzeit von 500 ms beantwortet werden.

Name	# requests	50%	66%	75%	80%	90%	95%	98%	99%	100%
POST /answerservice/answers/executions/221B8B	903	500	790	13000	20000	51000	53000	54000	54000	56000
Total	**903**	**500**	**790**	**13000**	**20000**	**51000**	**53000**	**54000**	**54000**	**56000**

Abbildung 65.: Teil Durchführung 1. Durchgang Statistik
Quelle: Eigene Darstellung

Die Anfragerate pro Sekunde blieb über eine lange Zeit auf 0. Ausserdem sind einige grosse Ausreisser bis zu 30 RPS zu sehen. Die Testbenutzer konnten keine Requests mehr absetzen, da bereits laufende Anfragen noch nicht beantwortet wurden.

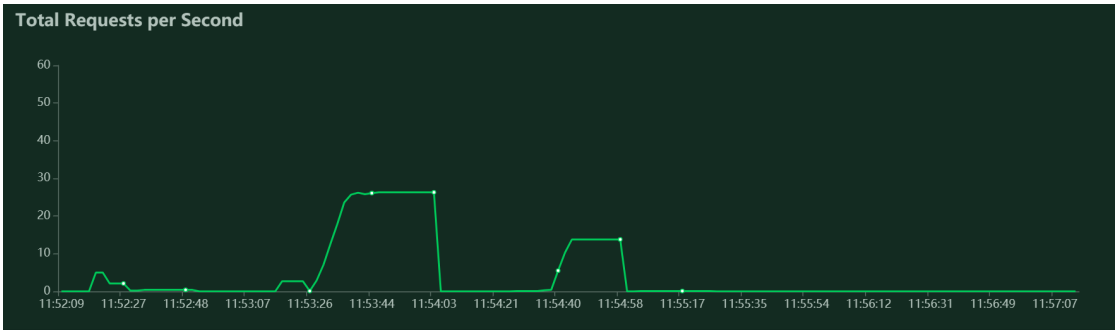


Abbildung 66.: Teil Durchführung 1. Durchgang Requests Per Seconds
Quelle: Eigene Darstellung

Die Median- und 95%-Linie verlaufen nahezu gleich.

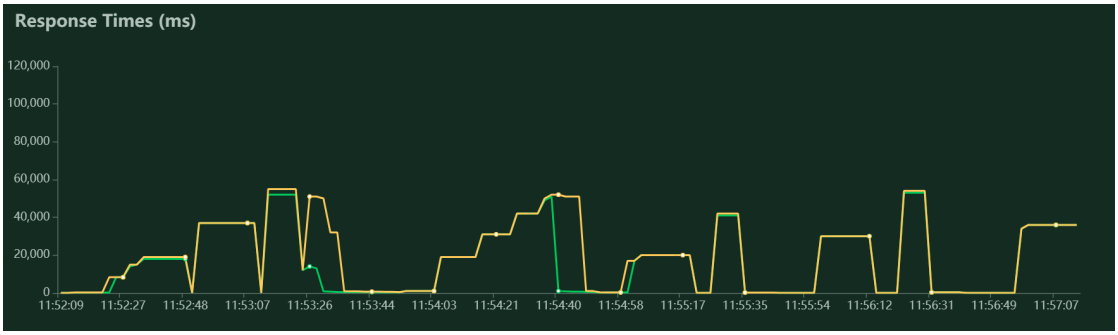


Abbildung 67.: Teil Durchführung 1. Durchgang Response Time
Quelle: Eigene Darstellung

6.1.2.4. Teil Durchführung: Durchgang 2

Im zweiten Durchgang wurden im Vergleich zum Durchgang 1 etwa nur halb so viele Requests verarbeitet. Dies entspricht einer Fehlerrate von ca. 67 %.

Type	Name	# requests	# fails	Median (ms)	Average (ms)	Min (ms)	Max (ms)	Content Size (bytes)	# reqs/sec
POST	/answerservice/answers/executions/4FA466	172	116	180	16613	87.75091171264648	67459.93852615356	272	0.6
	Total	**172**	**116**	**180**	**16613**	**87.75091171264648**	**67459.93852615356**	**272**	**0.6**

Abbildung 68.: Teil Durchführung 2. Durchgang Übersicht
Quelle: Eigene Darstellung

Es haben sich alle Werte ausser der Median im Vergleich zur vorherigen Durchführung verschlechtert.

Name	# requests	50%	66%	75%	80%	90%	95%	98%	99%	100%
POST /answerservice/answers/executions/4FA466	172	190	16000	42000	45000	55000	60000	62000	64000	67000
Total	**172**	**190**	**16000**	**42000**	**45000**	**55000**	**60000**	**62000**	**64000**	**67000**

Abbildung 69.: Teil Durchführung 2. Durchgang Statistik
Quelle: Eigene Darstellung

Die Requests pro Sekunde befinden sich etwa bei 6 RPS, dabei sind Anfangs einige Ausreisser vorhanden.

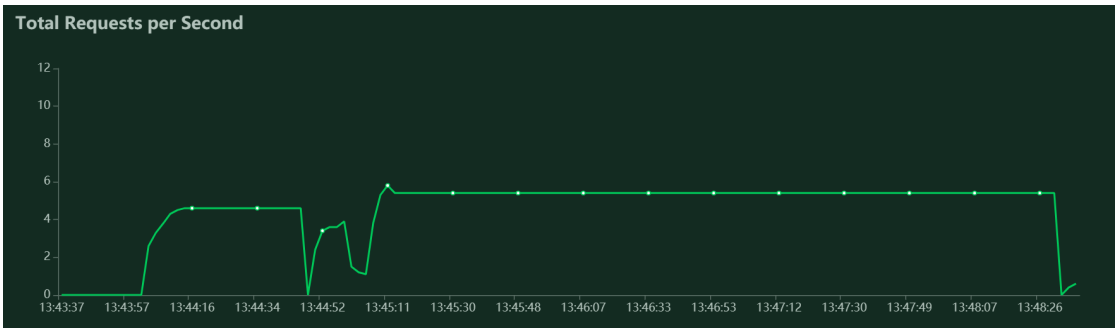


Abbildung 70.: Teil Durchführung 2. Durchgang Requests Per Seconds
Quelle: Eigene Darstellung

Die meiste Zeit wurden aufgrund der Überlastung der Datenbank keine Anfragen beantwortet.

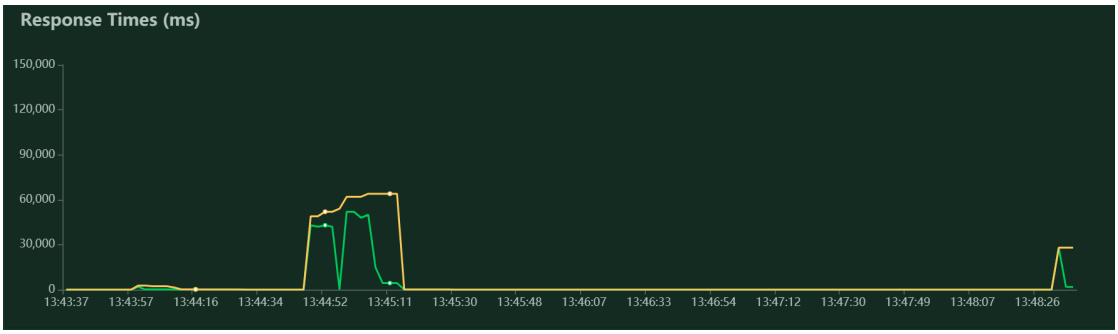


Abbildung 71.: Teil Durchführung 2. Durchgang Response Time
Quelle: Eigene Darstellung

6.1.2.5. Teil Durchführung: Durchgang 3

Der dritte Durchgang stellt hier eine Ausnahme dar um einen Vergleichswert zum zweiten Versuch zu produzieren. Der Grund ist, dass nach dem Versuch einige Verbesserungen im Code durchgeführt wurden. Im Answer Service wurden die Resultate vom Question Service und Presentation Service gecached.

Nach der Verbesserungen wurden mehr als 10 Mal so viele Anfragen beantwortet wie vorher. Es traten im Verhältnis so gut wie keine Fehler auf (4 Fehler, 0.2 %).

Type	Name	# requests	# fails	Median (ms)	Average (ms)	Min (ms)	Max (ms)	Content Size (bytes)	# reqs/sec
POST	/answerservice/answers/executions/4FA466	1818	4	97	6235	66.89977645874023	66075.45399665833	274	9.6
	Total	**1818**	**4**	**97**	**6235**	**66.89977645874023**	**66075.45399665833**	**274**	**9.6**

Abbildung 72.: Teil Durchführung 3. Durchgang Übersicht
Quelle: Eigene Darstellung

Die Antwortzeiten verbesserten sich bis zum 80. Perzentil auf 160 ms (vorher 45 s). In den oberen Bereichen werden nur leichte Verbesserungen verzeichnet.

Name	# requests	50%	66%	75%	80%	90%	95%	98%	99%	100%
POST /answerservice/answers/executions/4FA466	1818	97	110	130	160	31000	47000	49000	62000	66000
Total	**1818**	**97**	**110**	**130**	**160**	**31000**	**47000**	**49000**	**62000**	**66000**

Abbildung 73.: Teil Durchführung 3. Durchgang Statistik
Quelle: Eigene Darstellung

Die RPS-Rate stieg und zeigte auch Maximalwerte von 30 RPS. Im Durchschnitt waren es etwa 10 RPS.

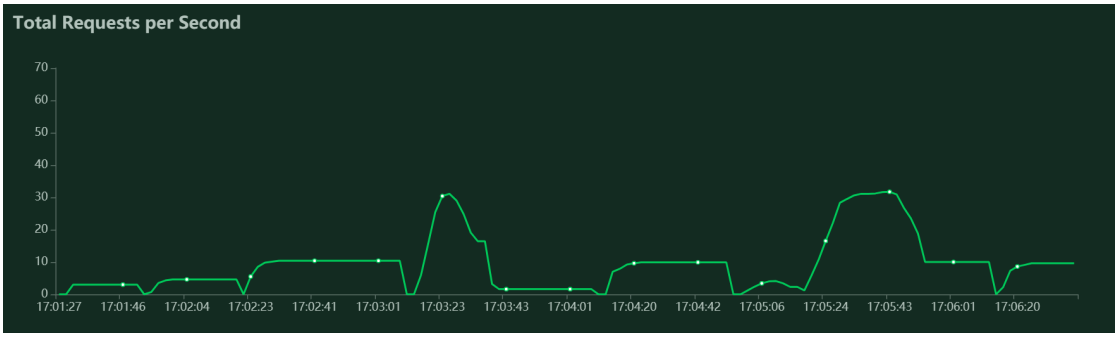


Abbildung 74.: Teil Durchführung 3. Durchgang Requests Per Seconds
Quelle: Eigene Darstellung

Anfragen wurden im Vergleich zum Durchgang 2 viel regelmässiger beantwortet. Zeiträume, wo Anfragen nicht beantwortet werden konnten, blieben leider nicht aus.

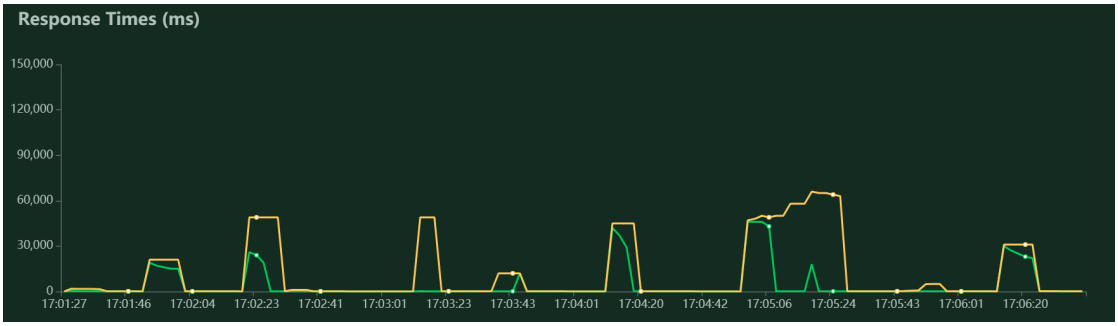


Abbildung 75.: Teil Durchführung 3. Durchgang Response Time
Quelle: Eigene Darstellung

6.1.3. Fazit

Die Services sind an sich gut horizontal skalierbar. Der Flaschenhals im ganzen System stellt die Datenbank dar, da nur ein Datenbankserver existiert und dieser von allen Services verwendet wird. Je mehr Services gleichzeitig Anfragen an die Datenbank senden, desto überlasteter wird diese. Ein Problem dabei sind die Anzahl Verbindungen, die geöffnet werden. Aus diesen Gründen müssen die Services bei vielen Anfragen auf eine Datenbankverbindung warten oder erhalten beim Verbinden einen Fehler. Dieses Problem wurde bei der Konzeption der Architektur angesprochen, wurde aber im Rahmen der Bachelorarbeit für gut empfunden und deshalb so umgesetzt.

6.1.3.1. Verbesserungsmöglichkeiten

Die Verbesserungen hinsichtlich Datenbank können aus dem Kapitel 4.10.3 entnommen werden. Eine Alternative wäre, dass jeder Service seine eigene Datenbank inklusive Datenbankserver erhält. So könnten die Anzahl Verbindungen pro Datenbank auf mehrere Server aufgeteilt werden.

7. Installationsanleitung

7.1. Tools

Damit die Installationsanleitungen korrekt funktionieren werden einige Tools benötigt. Diese müssen auf dem Rechner installiert werden.

7.1.1. Azure CLI

Um Kommandos und Abfragen in Azure ausführen zu können wird das Azure CLI Tool benötigt. Dieses kann anhand der Installationsanleitung [1] installiert werden.

7.1.2. Kubernetes CLI kubectl

Für die Kommunikation mit dem Kubernetes Service wird das CLI kubectl von Kubernetes benötigt. Dieses kann entweder via Azure CLI (Befehl in Listing 7.1) oder von Hand [2] installiert werden.

```
1  az aks install-cli
```

Listing 7.1: Installation von kubectl via Azure CLI

7.1.3. Docker

Damit die Docker-Container lokal erstellt und ausgeführt werden können wird Docker Community Edition [3] benötigt. Für eine reibungslose Installation unter Windows wird eine Mindestversion von Windows Pro, Enterprise oder Education mit einer 64bit-Architektur benötigt. Die von Docker zur Verfügung gestellte Docker Toolbox bietet nicht den benötigten Umfang. Unter Linux oder Mac OS X existieren keine bekannten Einschränkungen. Zusammen mit Docker wird automatisch Docker Compose installiert.

7.1.4. Helm

Helm, der Package Manager für Kubernetes, wird für das Installieren von eigenen Services und Infrastruktur benötigt. Helm bietet eine gute Installationsanleitung[4] für die Installation unter Windows, Linux oder Mac OS X. Die Installation via *Chocolatey* funktionierte

[1] `https://docs.microsoft.com/en-us/cli/azure/install-azure-cli`
[2] `https://kubernetes.io/docs/tasks/tools/install-kubectl/`
[3] `https://docs.docker.com/install/`
[4] `https://helm.sh/docs/install/`

bei allen Teammitgliedern einwandfrei. Der Befehl für die Installation mit *Chocolatey* ist in Listing 7.2 abgebildet.

```
1  choco install kubernetes-helm
```

Listing 7.2: Installation von Helm via Chocolatey

7.2. Kubernetes

7.2.1. Azure Kubernetes Service

Um ein Kubernetes Service verwenden zu können, muss dieser im Azure Portal erstellt werden.

7.2.2. Konfiguration Kubernetes und Azure CLI

Damit Kubernetes mit dem Azure CLI funktioniert müssen zwei Kommandos abgesetzt werden.
Listing 7.3 zeigt, wie die Kubernetes Resource als lokaler Kontext in kubectl verwendet wird.

```
1  az aks get-credentials --resource-group "gruppenname" --name "servicename" --overwrite-
       existing
```

Listing 7.3: Kontext für kubectl konfigurieren

Listing 7.4 zeigt, wie die Rolle für das Dashboard erstellt und dieses geöffnet werden kann.

```
1  kubectl create clusterrolebinding kubernetes-dashboard --clusterrole=cluster-admin --
       serviceaccount=kube-system:kubernetes-dashboard
```

Listing 7.4: Rolle für Dashboard erstellen

Weil die eigenen Docker-Images in der eigenen Azure Container Registry liegen, muss ein Secret innerhalb von Kubernetes erstellt werden, damit Kubernetes die Images herunterladen kann. Der Befehl ist in Listing 7.5 aufgeführt. Dabei müssen die Werte in Anführungszeichen angepasst werden. Die Email kann selber gewählt werden, Username und Passwort sind im Azure Portal in der Resource *Azure Container Registry* in der Lasche *Access keys* zu finden (siehe Abbildung 76, die Daten sind in der Abbildung Zwecks Sicherheit ausgegraut).

```
1  kubectl create secret docker-registry livesurvey-azurecr-login --docker-server
       livesurvey.azurecr.io --docker-email "email" --docker-username "username" --docker
       -password "password"
```

Listing 7.5: Secret für Container Registry konfigurieren

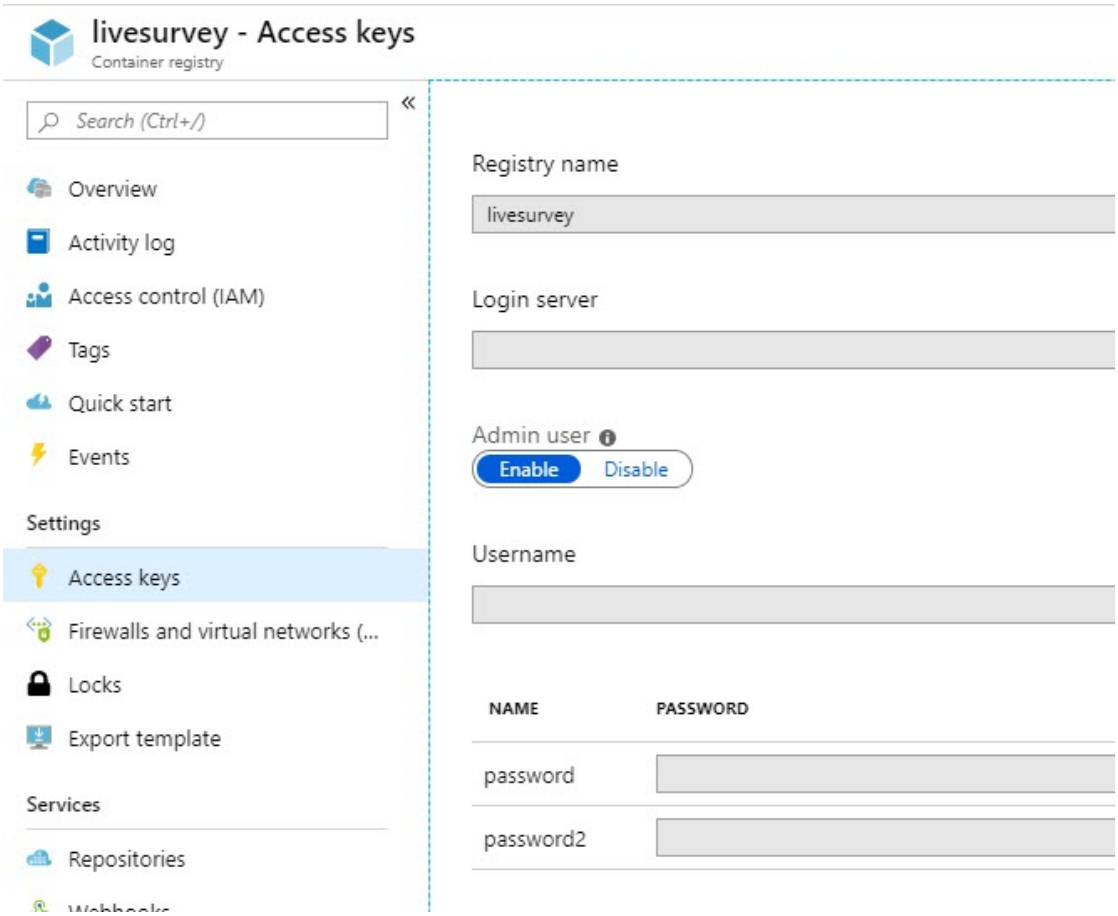


Abbildung 76.: Username und Passwort für die Container Registry
Quelle: Printscreen, portal.azure.com

7.2.3. Tiller und Helm

Für das deployen von Kubernetes Resources wird helm verwendet. Helm benötigt einige Konfigurationen. Die Schritte dafür sind in Listing 7.6 beschrieben.

```
1  kubectl create serviceaccount --namespace kube-system tiller
2  kubectl create clusterrolebinding tiller-cluster-rule --clusterrole=cluster-admin --
       serviceaccount=kube-system:tiller
3  helm init
4  // nachfolgenden Befehl eventuell in Bash (Git Bash) ausfuehren, da PowerShell
       Probleme mit dem Escaping hat
```

```
5   kubectl patch deploy --namespace kube-system tiller-deploy -p '{"spec":{"template":{"
        spec":{"serviceAccount":"tiller"}}}}'
6   helm install --namespace kube-system --set dashboard.enabled=true,dashboard.domain=
        traefik-ui.livesurvey.ch,rbac.enabled=true,dashboard.auth.basic.traefik='Password
        in base 64 encoded' stable/traefik
```

Listing 7.6: Installieren von Helm

Zum Schluss muss der DNS so konfiguriert werden, dass dieser auf die externe IP von Traefik zeigt. Traefik gibt beim Installieren den Befehl aus, der verwendet werden kann um diese IP herauszufinden. Ein Beispiel davon ist in Listing 7.7 zu finden.

```
1   kubectl get svc juiced-anteater-traefik --namespace kube-system -w
```

Listing 7.7: Auslesen der Public IP von Traefik

7.2.4. Backed Services

Einige Backed Services werden benötigt, damit die eigenen Services laufen. Diese können wie in Listing 7.8 gezeigt eingerichtet werden. Die Dateien befinden sich im git Repository *Infrastructure*.

```
1   // Redis
2   kubectl apply -f ./kubectl/redis/redis.yaml
3   // SQL Persistent Volume
4   kubectl apply -f ./kubectl/mssql/pvc.yaml
5   // MS SQL Server
6   kubectl apply -f ./kubectl/mssql/mssql.yaml
7   // rabbitmq
8   helm install ./charts/rabbitmq/ --set domain="dashboard-domain"
```

Listing 7.8: Installieren der Backed Services

7.2.5. HSR Live Survey Services

Die eigenen Services können entweder via Release Pipeline in Azure oder über die Helm Charts in den entsprechenden Rpeositories deployed werden. Bei dem Deployment über Charts muss aber darauf geachtet werden, dass alle Variabeln korrekt gesetzt werden. Die zu setzenden Variabeln sind in der Datei *values.yaml* ersichtlich. Ein Beispiel ist in Listing 7.9 ersichtlich, dabei sind aber einige Variabeln weggelassen worden.

```
1   // Auszufuehren in infrastructure/charts im PresentationDesigner-Repository
2   helm upgrade --install --recreate-pods --force --set environment=develop,
        presentationServiceUrl=http://api.dev.livesurvey.ch/presentationservice,--wait--
        set imageTag=2.0.1 ./presentationdesigner
```

Listing 7.9: Deployment eines Services mit Helm

7.3. Entwicklungsumgebung

Um lokal an den Services zu entwickeln existieren drei Möglichkeiten. Eine erste ist das Starten aller Services in der Entwicklungsumgebung. Die zweite ist das Starten der Services via Docker Compose, dabei nur die zu entwickelnden Services in der IDE starten. Die letzte Möglichkeit ist das Starten der benötigten Services in der IDE, für alle anderen werden die auf Azure deployten Services verwendet. Gewisse Varianten sind nicht immer in allen Konstellationen möglich.

7.3.1. Starten in der IDE

Entwickelt wurden alle Backend Services in Visual Studio 2019, alle Frontends in Visual Studio Code. Die Konfiguration der Ports und Urls sollten bereits korrekt konfiguriert sein. In Visual Studio 2019 lassen sich die Services mit der Taste F5 oder dem Kommando *Debug - Start Debugging*. Die Frontends lassen sich mit dem Befehl in Listing 7.10 starten. Dieser muss im Ordner *src* ausgeführt werden.

```
1  npm install
2  npm run start:with-config
```

Listing 7.10: Starten der Frontends

Nebst den eigenen Services wird einige Infrastruktur benötigt. Diese Dienste lassen sich alle via Docker Compose starten und müssen vor den eigenen Services gestartet werden. Der Befehl dafür ist in Listing 7.11 gezeigt. Dieser Befehl muss im Infrastruktur-Repository im Ordner *compose/livesurvey/* ausgeführt werden.

```
1  docker-compose -f docker-compose.infra.yml up -d
```

Listing 7.11: Starten der benötigten Dienste

7.3.2. Docker Compose

Für ein einfaches Starten aller Services inklusive Infrastruktur existieren Docker Compose Files im Infrastruktur-Repository unter *compose/livesurvey/*. Zu beachtem beim Start ist, dass die Infrastruktur vor den Services gestartet werden muss. Das Starten der Infrastruktur ist in Listing 7.12 abgebildet. Der Befehl für das Starten der Services ist in Listing 7.13 ersichtlich. Beide Befehle müssen im Infrastruktur-Repository im Ordner *compose/livesurvey/* ausgeführt werden.

```
1  docker-compose -f docker-compose.infra.yml up -d
```

Listing 7.12: Starten der Infrastruktur mit Docker Compose

```
1  docker-compose -f docker-compose.services.yml up -d
```

Listing 7.13: Starten der Services mit Docker Compose

Für das Starten und Stoppen aller Komponenten existieren die beiden Powershell-Skripts *start-all.ps1* und *stop-all.ps1*.
Die Konfiguration der Versionen und Umgebungsvariabeln kann dabei in der Datei *.env* angepasst werden.

7.3.3. IDE und Azure

Die gewünschten Services können analog Kapitel 7.3.1 gestartet werden. Dabei müssen die Konfigurationen der Services in der Datei *appsettings.Development.json* angepasst werden. Zu beachten dabei ist, dass nicht alle Dienste in Azure von aussen verfügbar sind. So ist beispielsweise die Datenbank, RabittMQ oder Redis nicht erreichbar. Falls diese Dienste benötigt werden müssen diese entweder lokal gestart werden, falls dies nicht reicht muss eine andere Variante verwendet werden.

7.3.4. Pitfalls

Authentication Service Sollte der Authentication Service lokal betrieben, alle anderen Dienste aber in Azure verwendet werden so funktioniert die Autorisierung nicht, da die Secrets für das Signieren der JWT-Tokens nicht dieselben sind.

RabbitMQ Der AnswerService und die beiden Hubs sind von der Queue abhängig. Damit alle Komponenten reibungslos zusammen funktionieren müssen somit alle drei Dienste zusammen entweder Lokal oder in Azure betrieben werden.

Presentation Service Der Presentation Service greift in gewissen Fällen auf den Question Service zu. Damit dieser Zugriff funktioniert müssen beide denselben ApiKey besitzen. Somit muss entweder der ApiKey lokal und in Azure derselbe sein oder beide Dienste müssen auf dieselbe Art betrieben werden.
Empfohlen wird die Variante mit Docker Compose. Diese Variante benötigt am wenigsten Arbeitsspeicher und funktioniert in allen Fällen. Zu beachten ist dabei nur, dass die korrekten Versionen der Docker Images verwendet werden.

7.4. Azure DevOps

Für Continuous Integration und Continuous Deplyoment wird Azure DevOps eingesetzt. Die einzelnen Build- und Release-Pipelines sind im Repository *Infrastructure* unter devops abgelegt.

7.4.1. Import der Pipelines

Die Pipelines können in Azure DevOps importiert werden. Diese können unter Pipelines im Menüpunkt “Builds” oder “Releases” mit einem Klick auf “New” und “Import a pipeline” (siehe Abbildung 77 importiert werden. Danach das gewünschte JSON-File auswählen und importieren (siehe Abbildung 78).

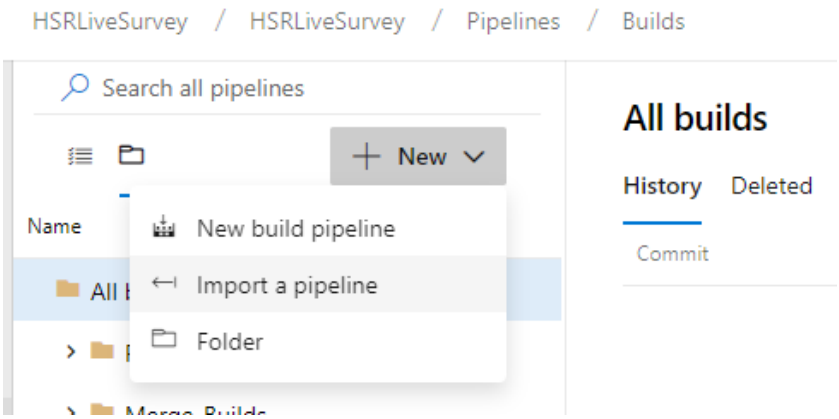


Abbildung 77.: Pipeline in Azure DevOps importieren, SChritt 1
Quelle: Printscreen, dev.azure.com

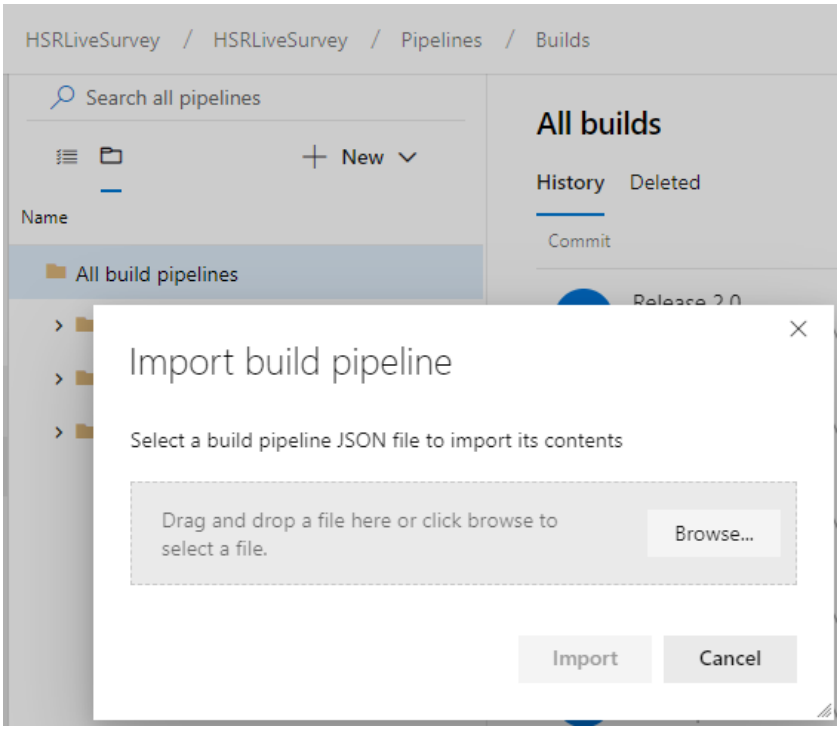


Abbildung 78.: Pipeline in Azure DevOps importieren, Schritt 2
Quelle: Printscreen, dev.azure.com

Nach dem Import müssen einige Punkte angepasst werden. Diese sind in der Pipeline rot hinterlegt.

7.4.2. Konfigurieren der Variabeln

Jede Pipeline ist mit einigen Variabeln-Gruppen verlinkt. Diese Verlinkung funktioniert nach dem Import nicht, sofern diese Gruppen nicht existieren. Die benötigten Gruppen sind in der Pipeline unter “Variables” und “Variabel groups” ersichtlich. Diese können unter dem Menüpunkt “Library” erstellt und danach mit der Pipeline verknüpft werden.

7.4.2.1. Benötigte Variabeln-Gruppen

Nachfolgend sind alle benötigten Variabeln-Gruppen abgebildet. Diese können nicht exportiert oder importiert werden und müssen daher von Hand erstellt werden.

Azure Container Registry Für das Pushen und Laden von Docker-Container werden die Zugangsdaten für die Azure Container Registry benötigt.

Variables

Name ↑	Value
RegistryLogin	7411954f-c765-4ae6-9e26-59f259fc7a0d
RegistryName	livesurvey
RegistryPassword	********
RegistryTenant	371626fa-799f-47c6-a97c-8fd9494cae88

Abbildung 79.: Variabeln-Gruppe für die Azure Container Registry
Quelle: Printscreen, dev.azure.com

Versionen Um die Versionen an einem Ort verwalten zu können existiert eine Variabeln-Gruppe. Diese Versionen werden für die Build-Versionierung benötigt.

Variables

Name ↑	Value
version.answerservice	2.0
version.authenticationservice	2.0
version.presentationdesigner	2.0
version.presentationservice	2.0
version.presenterclient	2.0
version.presenterhub	2.0
version.questionservice	2.0
version.voterclient	2.0
version.voterhub	2.0

Abbildung 80.: Variabeln-Gruppe für die Versionen
Quelle: Printscreen, dev.azure.com

Queue Um sich aus den Applikationen auf die Queue zu verbinden werden gewisse Daten wie Host, User und Passwort benötigt.

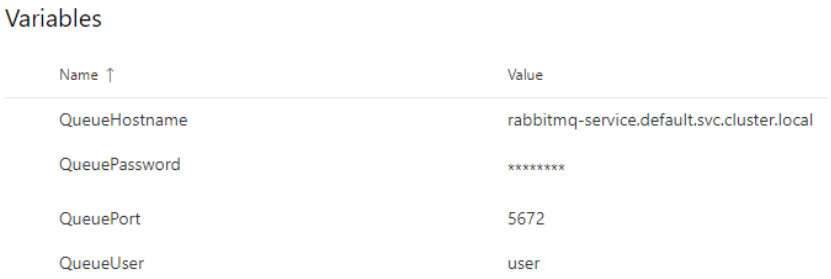


Abbildung 81.: Variabeln-Gruppe für die Queue
Quelle: Printscreen, dev.azure.com

Secrets In dieser Variabeln-Grupp sind die Secrets für die Autorisierung und das Ausstellen der JWT-Tokens hinterlegt.

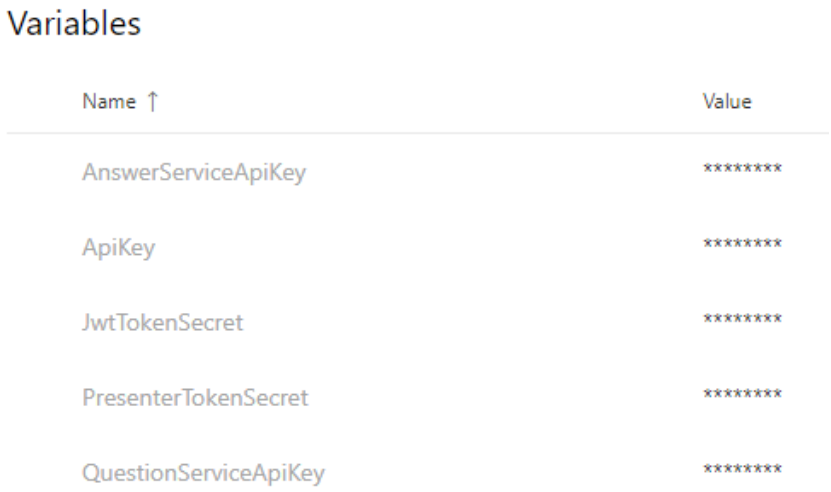
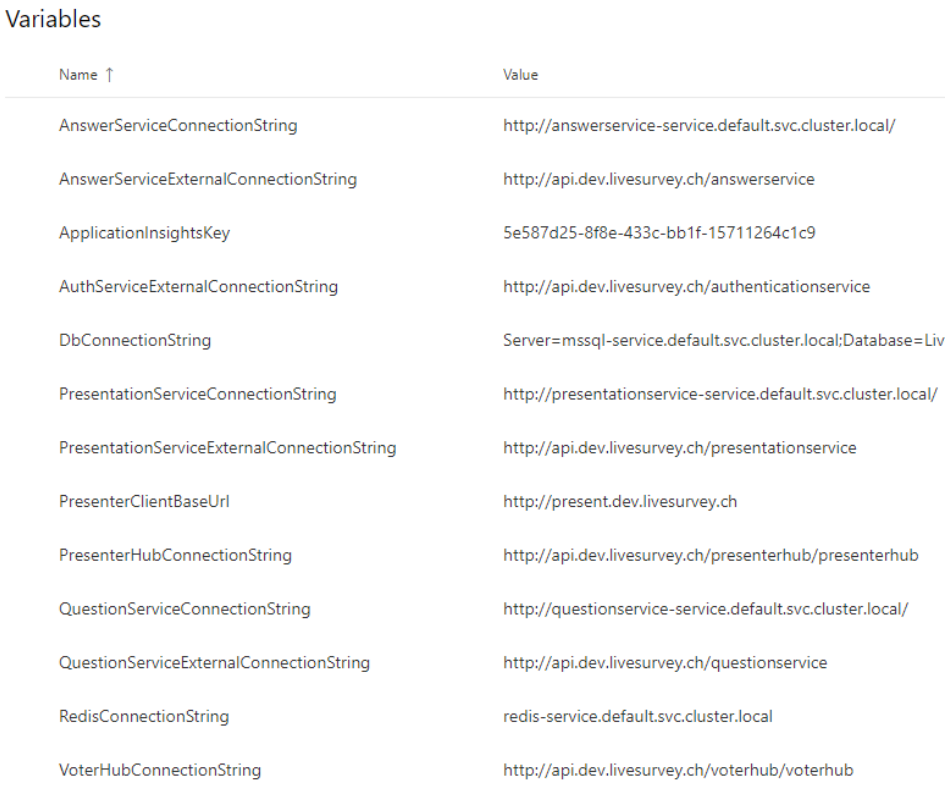


Abbildung 82.: Variabeln-Gruppe für die Secrets
Quelle: Printscreen, dev.azure.com

Connection Strings Für das Abfragen von Daten aus Services oder die Verbindung zur Datenbank werden die Connectionstrings benötigt. Interne Connectionstrings werden für den Zugriff auf Services innerhalb des Kubernetes-Clusters verwendet. Externe für den Zugriff von Frontend auf Service.



Variables

Name ↑	Value
AnswerServiceConnectionString	http://answerservice-service.default.svc.cluster.local/
AnswerServiceExternalConnectionString	http://api.dev.livesurvey.ch/answerservice
ApplicationInsightsKey	5e587d25-8f8e-433c-bb1f-15711264c1c9
AuthServiceExternalConnectionString	http://api.dev.livesurvey.ch/authenticationservice
DbConnectionString	Server=mssql-service.default.svc.cluster.local;Database=Liv
PresentationServiceConnectionString	http://presentationservice-service.default.svc.cluster.local/
PresentationServiceExternalConnectionString	http://api.dev.livesurvey.ch/presentationservice
PresenterClientBaseUrl	http://present.dev.livesurvey.ch
PresenterHubConnectionString	http://api.dev.livesurvey.ch/presenterhub/presenterhub
QuestionServiceConnectionString	http://questionservice-service.default.svc.cluster.local/
QuestionServiceExternalConnectionString	http://api.dev.livesurvey.ch/questionservice
RedisConnectionString	redis-service.default.svc.cluster.local
VoterHubConnectionString	http://api.dev.livesurvey.ch/voterhub/voterhub

Abbildung 83.: Variabeln-Gruppe für die Connection Strings
Quelle: Printscreen, dev.azure.com

7.5. Azure

Für die Erstellung der benötigten Ressourcen in Azure befinden sich Templates im Repository *Infrastructure* unter *azure*. Diese sind für den Kubernetes-Service, Application Insights und die Container Registry vorhanden.

Literaturverzeichnis

[1] About MariaDB . https://mariadb.org/about/. Zugegriffen am 05.03.2019.

[2] ASP.NET Core SignalR hosting and scaling . https://docs.microsoft.com/en-us/aspnet/core/signalr/scale?view=aspnetcore-2.2. Zugegriffen am 03.06.2019.

[3] Assigning Pods to Nodes . https://kubernetes.io/docs/concepts/configuration/assign-pod-node/. Zugegriffen am 13.06.2019.

[4] Azure Cosmos DB . https://azure.microsoft.com/de-de/services/cosmos-db/. Zugegriffen am 05.03.2019.

[5] Azure Database for MariaDB . https://azure.microsoft.com/de-de/services/mariadb/. Zugegriffen am 05.03.2019.

[6] Azure Database for MySQL . https://azure.microsoft.com/de-de/services/mysql/. Zugegriffen am 05.03.2019.

[7] Azure Database for PostgreSQL . https://azure.microsoft.com/de-de/services/postgresql/. Zugegriffen am 05.03.2019.

[8] Azure kubernetes Service (AKS) . https://azure.microsoft.com/de-de/services/kubernetes-service/. Zugegriffen am 05.03.2019.

[9] Azure Service Fabric . https://azure.microsoft.com/de-de/services/service-fabric/. Zugegriffen am 05.03.2019.

[10] Disk Storage . https://azure.microsoft.com/de-de/services/storage/disks/. Zugegriffen am 05.03.2019.

[11] Dropbox: How many HTTP status codes should your API use? . https://blogs.dropbox.com/developers/2015/04/how-many-http-status-codes-should-your-api-use/. Zugegriffen am 07.06.2019.

[12] Github: BCrypt.Net . https://github.com/BcryptNet/bcrypt.net. Zugegriffen am 04.06.2019.

[13] Hardware and Software Requirements for Installing SQL Server . https://docs.microsoft.com/en-us/sql/sql-server/install/hardware-and-software-requirements-for-installing-sql-server?view=sql-server-2017. Zugegriffen am 05.03.2019.

[14] High availability for SQL Server containers . https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-container-ha-overview?view=sql-server-2017. Zugegriffen am 04.06.2019.

[15] JSON data in SQL Server . https://docs.microsoft.com/en-us/sql/relational-databases/json/json-data-sql-server?view=sql-server-2017. Zugegriffen am 05.03.2019.

[16] Kubernetes - Concepts . https://kubernetes.io/docs/concepts/. Zugegriffen am 05.03.2019.

[17] Kubernetes: Scaling a Deployment . https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#scaling-a-deployment. Zugegriffen am 04.06.2019.

[18] Kubernetes: Virtual IPs and service proxies . https://kubernetes.io/docs/concepts/services-networking/service/#virtual-ips-and-service-proxies. Zugegriffen am 04.06.2019.

[19] Microservice Pattern: API Gateway / Backends for Frontends . https://microservices.io/patterns/apigateway.html. Zugegriffen am 06.06.2019.

[20] Microservice Pattern: Database per Service . https://microservices.io/patterns/data/database-per-service.html. Zugegriffen am 06.06.2019.

[21] Microservice Pattern: Decompose by subdomain . https://microservices.io/patterns/decomposition/decompose-by-subdomain.html. Zugegriffen am 06.06.2019.

[22] Microservice Pattern: Saga . https://microservices.io/patterns/data/saga.html. Zugegriffen am 06.06.2019.

[23] Microservice Pattern: Shared database . https://microservices.io/patterns/data/shared-database.html. Zugegriffen am 06.06.2019.

[24] Microsoft: ApplicationInsightsLoggerProvider for .NET Core ILogger logs . https://docs.microsoft.com/en-us/azure/azure-monitor/app/ilogger. Zugegriffen am 06.06.2019.

[25] Microsoft Azure SQL Database . https://azure.microsoft.com/en-us/services/sql-database/. Zugegriffen am 26.02.2019.

[26] Microsoft: MemoryCache Class . https://docs.microsoft.com/en-us/dotnet/api/system.runtime.caching.memorycache?redirectedfrom=MSDN&view=netframework-4.8. Zugegriffen am 11.06.2019.

[27] Microsoft: What is Application Insights? . https://docs.microsoft.com/en-us/azure/azure-monitor/app/app-insights-overview. Zugegriffen am 06.06.2019.

[28] MongoDB - Website . https://www.mongodb.com/. Zugegriffen am 06.03.2019.

[29] Nuget Microsoft.Azure.DocumentDB . https://www.nuget.org/packages/Microsoft.Azure.DocumentDB/. Zugegriffen am 05.03.2019.

[30] Options pattern in ASP.NET Core . https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/options?view=aspnetcore-2.2. Zugegriffen am 06.06.2019.

[31] Overview of Azure Service Fabric . https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-overview. Zugegriffen am 05.03.2019.

[32] Production-Grade Container Orchestration . https://kubernetes.io/. Zugegriffen am 05.03.2019.

[33] Quickstart: Run SQL Server container images with Docker . https://docs.microsoft.com/en-us/sql/linux/quickstart-install-connect-docker?view=sql-server-2017&pivots=cs1-powershell. Zugegriffen am 05.03.2019.

[34] RabbitMQ: Work Queues . https://www.rabbitmq.com/tutorials/tutorial-two-dotnet.html. Zugegriffen am 10.06.2019.

[35] React: Adding Custom Environment Variables . https://facebook.github.io/create-react-app/docs/adding-custom-environment-variables. Zugegriffen am 06.06.2019.

[36] React Architektur: Best Practices und Erfahrungen. Persönliches Gespräch mit Mirko Stocker. Gespräch am 05.04.2019.

[37] Running Kubernetes Locally via Minikube . https://kubernetes.io/docs/setup/minikube/. Zugegriffen am 05.03.2019.

[38] So you want to learn about Service Fabric? . https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-content-roadmap. Zugegriffen am 05.03.2019.

[39] SQL Server pricing . https://www.microsoft.com/en-us/sql-server/sql-server-2017-pricing#CP_StickyNav_1. Zugegriffen am 06.03.2019.

[40] The Purpose of JWT: Stateless Authentication . https://www.jbspeakr.cc/purpose-jwt-stateless-authentication/. Zugegriffen am 03.06.2019.

[41] The Twelve-Factor App: Config . https://12factor.net/config. Zugegriffen am 06.06.2019.

[42] Top 20 NuGet NOSQL Packages . https://nugetmusthaves.com/Category/NOSQL. Zugegriffen am 05.03.2019.

[43] Towards Robust Distributed Systems . http://awoc.wolski.fi/dlib/big-data/Brewer_podc_keynote_2000.pdf. Zugegriffen am 06.03.2019.

[44] Variable groups for Azure Pipelines . https://docs.microsoft.com/en-us/azure/devops/pipelines/library/variable-groups?view=azure-devops&tabs=yaml. Zugegriffen am 03.06.2019.

[45] Verwaltete Datenträger - Preise . https://azure.microsoft.com/de-de/pricing/details/managed-disks/. Zugegriffen am 05.03.2019.

[46] Web Engineering + Design 3 Modul, Vorlesung React von Mirko Stocker, Folie 70. https://skripte.hsr.ch/Informatik/Fachbereich/Web_Engineering_+_Design_3/WE3/01-Vorlesung/WE3-FS19-W2-4-React%20Teil%201,%202%20und%203.pdf. Zugegriffen am 07.06.2019.

[47] Windows container images of SQL Server 2017 . https://blogs.msdn.microsoft.com/sqlserverstorageengine/2018/02/11/new-windows-container-images-of-sql-server-2017-on-windows-server-1709/. Zugegriffen am 06.03.2019.

[48] Service Bus Pricing. https://azure.microsoft.com/en-us/pricing/details/service-bus/. Zugegriffen am 13.03.2019.

[49] What is Azure Service Bus? https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-messaging-overview. Zugegriffen am 13.03.2019.

[50] K. Banovac. How to implement runtime environment variables with create-react-app, Docker, and Nginx . https://www.freecodecamp.org/news/how-to-implement-runtime-environment-variables-with-create-react-app-docker-and-nginx-7f9d42a91d70/. Zugegriffen am 06.06.2019.

[51] V. Driessen. 10 Best JavaScript Frameworks to Use in 2019 . https://nvie.com/posts/a-successful-git-branching-model/. Zugegriffen am 13.06.2019.

[52] A. Goel. A successful Git branching model . https://hackr.io/blog/10-best-javascript-frameworks-2019. Zugegriffen am 13.06.2019.

Glossar

.NET Core .NET Core ist ein kostenloses, quelloffenes und plattformübergreifendes Framework von Microsoft. 45, 50, 54, 108

API Application Programming Interface. 10, 50, 66, 86, 97, 104, 107–109

ASP.NET Core ASP.NET Core ist ein kostenloses, quelloffenes und plattformübergreifendes Web-Framework von Microsoft. 50, 89, 99, 101

CAP Consistency, Availability, Partition Tolerance. 52

CLI Command Line Interface beschreibt eine Programgruppe, welche nur über die Kommandozeile bedient werden können. viii, 131, 132

DevOps DevOps beschreibt eine Software-Entwicklungs-Methode die Teile der Entwicklung und Operations kombiniert. 14, 111, 116

DNS Domain Name System, löst Domain zu IP auf. 134

DTO Data Transfer Objects. 76, 89, 100, 107

HSR HSR Hochschule für Technik Rapperswil. 2, 15, 17, 19, 29, 51, 63, 66, 78, 90, 92

HTTP HyperText Transfer Protocol. 10, 89, 101, 102, 107, 120

HTTPS HyperText Transfer Protocol Secure. 17, 90

IDE Integrated development environment, Tool für das Entwickeln von Software. ix, 135

JSON JavaScript Object Notation. 10, 50, 52, 59, 82, 86, 88, 99, 107, 109

JWT JSON Web Token. 89, 90, 92, 98, 140

NoSQL NoSQL steht für *Not only Structured Query Language* und ist ein Datenbanksystem ohne relationalen Ansatz. 50–52, 59

NPM Node Package Manager. 116

REST Representational State Transfer stellt ein Software-Architekturstil dar. 10

RPS Requests Per Second. 126, 128, 129

SaaS SaaS resp. Software as a Service ist ein Modell des Cloud Computings, wo eine Software und deren Infrastruktur von einem externen Anbieter betrieben wird. 17

SQL Die Structured Query Language ist eine Sprache für das Abfragen von Daten innerhalb einer relationalen Datenbank. 50–52, 59, 63, 79, 80

UI Das User Interface stellt die Schnittstelle zwischen Benutzer und Software dar. Hier handelt es sich um eine grafische Benutzeroberfläche. 69, 89, 109, 110

URL Uniform Resource Location, Begriff für eine Webadresse. 91, 101, 107, 110

VM Virtuelle Maschine. 51, 52, 56

Abbildungsverzeichnis

Tabellenverzeichnis

Listings