

```
const name = 'Individuelle praktische Arbeit'  
const topic = 'Webapplikation für gRPC Testing Tool'  
const date = new Date('2020-04-02')  
  
const author = 'Nils Benz'  
const candidateNumber = 'ASG010'  
const company = 'Adcubum AG'  
  
const supervisor = 'Stefan Loosli'  
const experts = [  
    'Thomas Kehl',  
    'Ueli Niederer'  
]
```

Abstract

gRPC ist eine von Google entwickelte Schnittstellentechnologie. Da diese Technologie relativ neu ist, gibt es noch wenige Testing Tools. Aus diesem Hintergrund ergab sich die Aufgabenstellung für diese IPA: es soll eine Webapplikation entwickelt werden, welche das Testing von gRPC Schnittstellen erleichtert. Dieses Tool kann gRPC Schnittstellen per Server Reflection auslesen und danach Requests anhand des erwarteten Schemas absetzen. Die Applikation kann auf einer Openshift Umgebung bereitgestellt werden.

Ein Server, welcher die REST Calls der Webapp in die gRPC Aufrufe umwandelt, wurde bereits vor der Arbeit realisiert und bereitgestellt. Ziel der Arbeit war es, eine grafische Schnittstelle zum Benutzer zu erstellen. Diese soll ihm ermöglicht, möglichst leicht den Request Body anzupassen und die Requests abzusetzen. Dafür standen zehn Arbeitstage zur Verfügung.

Vor der Umsetzung wurden die Anforderungen anhand von Use Cases analysiert und die Anwendung mit einem Sequenzdiagramm und Mockups geplant. Die Applikation wurde abschliessend mittels Unit Tests und einem Testprotokoll ausführlich getestet und enthält eine Anwender-, Betreiber- und Entwicklerdokumentation.

Für die Umsetzung wurde React mit Redux verwendet, es kam ausserdem die UI Library MaterialUI zum Einsatz. Durch die Verwendung dieser Tools gelang es, den zu Beginn der Arbeit erstellten Zeitplan einzuhalten und das Produkt fristgerecht und den Anforderungen entsprechend abzuliefern.

Inhaltsverzeichnis

1 Einleitung	3
1.1 Aufgabenstellung	3
1.2 Projektumfeld	3
1.2.1 Umfang der Arbeit	4
1.2.2 Backend Server	5
1.2.3 Schnittstellen	5
2 Planung	7
2.1 Wissensbeschaffung	7
2.2 Zeitplan	7
2.3 Projektmanagementmethode	10
3 Umsetzung	12
3.1 Analyse	12
3.1.1 Use Cases	12
3.1.2 Sequenzdiagramm	17
3.2 Applikation	18
3.2.1 Versionierung	18
3.2.2 Redux Store	19
3.2.3 Mockup	20
3.2.4 Implementierung	21
3.2.5 Dokumentation	29
3.3 Testing	31
3.3.1 Testkonzept	31
3.3.2 Testumfeld	31
3.3.3 Unit Testing	38
3.4 Release	39
4 Reflexion	40
5 Fazit	42
6 Glossar	43
7 Literaturverzeichnis	46
7.1 Online-Quellen	46
8 Abbildungsverzeichnis	47

9 Tabellenverzeichnis	49
10 Danksagungen	50
11 Anhang	51
11.1 Projektmanagement	51
11.2 Schnittstellen	54
11.3 Redux	59
11.4 Testing	61
11.5 Arbeitsjournal	62
12 Eigenständigkeitserklärung	82

1 Einleitung

In diesem Kapitel wird die Ausgangslage zur IPA dargelegt. Unter anderem werden die Themen Aufgabenstellung und Projektumfeld genauer behandelt.

1.1 Aufgabenstellung

Die Aufgabenstellung wurde von meinen beiden vorgesetzten Fachkräften Stefan Loosli und Javier Diez definiert. Sie umfasst die Erstellung von einem Webclient, welcher auf einen Server aufbaut, der bereits vorhanden und funktionsfähig bereitgestellt ist.

Die Adcubum setzt in neuen Modulen auf die Schnittstellentechnologie gRPC. Diese weist einige Vorteile gegenüber der herkömmlichen REST-Methodik auf, unter anderem dass sie effizienter ist. gRPC wurde von Google entwickelt und ist relativ neu, daher gibt es bisher wenige Testing Tools (wie z.B. Postman für REST-Schnittstellen). Es gibt einige Open-Source Anwendungen wie BloomRPC¹ und gRPCox². Diese Tools haben jedoch einige schwerwiegende Nachteile, zum einen muss der Benutzer bei BloomRPC ein Protobuf File angeben, zum anderen kann der gRPCox nicht in die Adcubum Security integriert werden.

Aus diesen Umständen hat sich die Aufgabenstellung für diese Projektarbeit ergeben. Das Produkt soll automatisch das Schema einer gRPC Methode anzeigen, welches dann vom Benutzer bearbeitet werden kann. Die Requests können anschliessend abgesetzt werden, und die Response wird wiederum im Webclient dargestellt. Der Umfang der Arbeit und das Projektumfeld wird im nächsten Abschnitt detailliert erklärt.

1.2 Projektumfeld

Alle Applikationen werden auf Openshift als Docker Images bereitgestellt. Zu diesen Applikationen gehören der Webclient, der Backend Server, welcher in Abschnitt 1.2.2 genauer erklärt wird, und mindestens ein Server, welcher eine gRPC Schnittstelle implementiert. Untereinander kommunizieren die Container über gRPC oder REST.

¹<https://github.com/uw-labs/bloomrpc> [17.03.2020]

²<https://github.com/gusaul/grpcox> [17.03.2020]

1.2.1 Umfang der Arbeit

Der Auftrag für diese IPA umfasst die Erstellung eines Webclients. Dabei sind folgende Voraussetzungen bereits erfüllt:

- Der Webclient ist mit den nachfolgend definierten Technologien aufgesetzt und bereits auf der Openshift Umgebung bereitgestellt. Dafür wurde eine Build Pipeline eingerichtet
- Der Client kann eine Verbindung zum Server herstellen und bereits die Endpoints auslesen
- Die Applikation ist in die Adcubum Security integriert
- Der Webserver und die Schnittstelle entsprechen dem Versionierungskonzept der Adcubum

Hier eine Liste der bereits erwähnten vordefinierten Technologien:

- React³: Frontend Framework, von Facebook entwickelt
- Redux⁴: Clientside State Management, welches auf dem Flux Pattern aufbaut
- Redux-Saga⁵: Library für die Verwaltung von Operationen mit Nebenwirkungen (Side Effects)
- Typescript⁶: Typisiertes Superset für JavaScript
- MaterialUI⁷: Library für React UI Komponenten nach den Material Style Guidelines von Google
- Jest⁸: Frontend Testing Library

Ausserdem wurden bereits einige Gespräche bezüglich Designideen geführt und erste Mockups erstellt. Diese müssen jedoch noch überarbeitet und fertig gestellt werden.

Es kann also auf einem bereits lauffähigen System aufgebaut werden. Ziel der Arbeit ist, ein Produkt zu erstellen, mit welchem ein gRPC Service getestet werden kann. Dies umfasst folgende Punkte:

- Das Aufbauen einer Verbindung zu einem gRPC Server
- Das Auslesen seiner Services und Methoden
- Das Bearbeiten des Schemas, welches vom Server erwartet wird, durch den Benutzer

³<https://reactjs.org/> [17.03.2020]

⁴<https://redux.js.org/> [17.03.2020]

⁵<https://redux-saga.js.org/> [17.03.2020]

⁶<https://www.typescriptlang.org/> [17.03.2020]

⁷<https://material-ui.com/> [17.03.2020]

⁸<https://jestjs.io/> [17.03.2020]

- Das Absetzen von Requests an einen gRPC Server
- Das Darstellen der Response und von eventuellen Fehlern

Diese Anforderungen sind im Abschnitt 3.1 anhand von einem Anwendungsfall- und einem Sequenzdiagramm genauer beschrieben.

1.2.2 Backend Server

Wie bereits erwähnt wurde vor der IPA ein Server geschrieben, welcher per Server Reflection eine gRPC Schnittstelle auslesen und gegen diese anschliessend Requests absetzen kann. Der Server ist an ein Adcubum Template angelehnt, da so die Build Pipeline und die Security mit minimalem Aufwand eingerichtet werden konnten. Der Server implementiert eine gRPC Schnittstelle, welche von einem generierten Gateway in eine REST-Schnittstelle umgewandelt wird. Der Webclient kann also mit HTTP Requests auf den Server zugreifen.

Der Server wird, wie alle anderen Applikationen, ebenfalls als Kubernetes Container auf Openshift bereitgestellt. Er kann auch alleinstehend, also ohne den Webclient, für automatisierte Tests installiert werden.

Der Client, welcher während dieser Arbeit zu erstellen ist, spricht nur mit diesem Server. Die REST-Schnittstelle wird im nächsten Abschnitt genauer erklärt.

1.2.3 Schnittstellen

Die Schnittstelle zum Backend Server wurde in einem separaten Projekt spezifiziert. Aus dieser Spezifikation werden json Files generiert, welche wiederum in den Swagger Editor⁹ importiert werden können. Dort wird anschliessend die Dokumentation in Abbildung 1.1 generiert.

Nachfolgend ist eine kurze Beschreibung zu den einzelnen Requests aufgeführt. Die von Swagger generierte ausführliche Dokumentation zu jedem Request ist im Anhang in Kapitel 11 zu finden.

- Die Abbildung 11.4 beschreibt den Request, um die Endpoints aus dem Kubernetes Cluster zu lesen. Dieser Request hat keine Parameter und gibt eine Liste von Strings zurück.
- Die Dokumentation in Abbildung 11.5 beschreibt den Request, um die Applikation mit einem gRPC Endpoint zu verbinden. Dies ist nicht unbedingt nötig, jedoch good practice, da die Verbindung ansonsten während dem Request abgesetzt aufgebaut wird. Der Name des Endpoints (z.B. `backend-service:7070`) wird im Request Body mitgesendet.
- Das Swagger Schema in Abbildung 11.6 beschreibt den Request, um die Verbindung zu einem gRPC Endpoint zu trennen. Dem gRPC Server wird so mitgeteilt, dass über

⁹<https://editor.swagger.io/> [17.03.2020]

EndpointService**GET** /api/endpoints List all available gRPC Endpoints in the Kubernetes Cluster**POST** /api/endpoints Connects to a new Endpoint**DELETE** /api/endpoints/{endpoint} Disconnect from an Endpoint**GrpcServiceService****GET** /api/endpoints/{endpoint}/services List all gRPC Services of a given endpoint**GrpcRequestService****POST** /api/endpoints/{endpoint}/services/{service}/methods/{method}/requests Creates a new GrpcRequest

Abbildung 1.1: Die Schnittstellen zum Backend Server

diesen Channel keine Requests mehr abgesetzt werden. Der Name des Endpoints wird als Request Parameter mitgegeben.

- Die Abbildung 11.7 beschreibt den Request, um die Services per Server Reflection von einem gRPC Server auszulesen. Der Name des Endpoints wird hier ebenfalls als Request Parameter angegeben.
- Die Dokumentation in Abbildung 11.8 beschreibt den Request, um einen gRPC Request abzusetzen. Dazu wird das Schema für den gRPC Request im Body als String mitgegeben. Außerdem müssen in der Request-URL der Endpoint-, Service- und Methodename angegeben werden.

2 Planung

In diesem Kapitel wird die Projektplanung genauer beschrieben. Dies geschieht unter anderem mittels Zeitplan und Projektmanagementmethodik.

2.1 Wissensbeschaffung

Die Webapplikation wird, wie bereits erwähnt, mit React-Redux und Redux-Saga realisiert. Diese Technologien waren mir bereits bekannt, und ich musste keine Grundlagen mehr erlernen. Facebook bietet eine ausführliche Dokumentation zu React¹, welche die verschiedenen Komponenten des Frameworks verständlich erklärt. Auch für Redux² und Saga³ gibt es entsprechende Dokumentationen.

Vieles über diese Technologien habe ich ausserdem von meinem ehemaligen Ausbildner Marius King gelernt. Dies, als wir gemeinsam an einem Beispielprojekt mit denselben Technologien gearbeitet haben.

Wenn Probleme während der Arbeit aufgetreten sind, habe ich zuerst versucht, im Internet nach einer Lösung zu suchen. Falls dies keinen Erfolg mit sich brachte, wandte ich mich an meine vorgesetzte Fachkraft oder an ein anderes Teammitglied.

2.2 Zeitplan

In der aktuellen Lage war es schwierig, einen zuverlässigen Zeitplan zu erstellen. Aufgrund der Massnahmen zur Verlangsamung der Verbreitung des COVID-19 wäre es denkbar gewesen, dass die IPA unterbrochen und zu einem späteren Zeitpunkt vollendet wird. Dieser Fall ist jedoch nicht eingetroffen und die IPA konnte wie geplant durchgeführt werden. Im Zeitplan, welcher nachfolgend in der Abbildung 2.1 aufgeführt ist, wurden die Daten so festgelegt, wie sie zu Beginn der Arbeit zu erwarten waren.

Für die Erstellung des Zeitplans wurde ein Gantt-Diagramm gewählt, welches sich eignet, um zeitliche Abfolgen innerhalb eines Projekts zu planen. Die gesamte Projektdauer wurde in Blöcke

¹<https://reactjs.org/docs/getting-started.html> [17.03.2020]

²<https://redux.js.org/introduction/getting-started/> [17.03.2020]

³<https://github.com/redux-saga/redux-saga> [17.03.2020]

von halben Tagen, welche ungefähr vier Stunden entsprechen, unterteilt. Der Aufwand für die einzelnen Aufgaben wird durch die Zahl im jeweiligen Feld angegeben. Ein Punkt soll dabei ungefähr einer Stunde Aufwand entsprechen, jedoch wird hier keine abschliessende Zeitangabe getroffen. Dieses Vorgehen wurde gewählt, da so der Sprint Backlog, welcher nachfolgend näher beschrieben wird, analog gestaltet und geschätzt werden konnte.

Die Aufgaben wurden in die fünf Kategorien Projektplanung, Applikationsplanung, Umsetzung, Testing und Dokumentation unterteilt. Jede der Kategorien wird durch eine separate Farbe gekennzeichnet. So wurden zu Beginn der Arbeit eher Aufgaben in der Planung aufgeführt, welche dann zur Umsetzung überleiten. Gegen das Ende des Projekts wurden zwei Tage für die Fertigstellung und Überarbeitung der Dokumentation reserviert. Pro Tag wurden jeweils Arbeiten von ca. acht Stunden geplant. Zusätzlich war ein täglicher Aufwand von etwa einer halben Stunde für das Daily und das Verfassen des Arbeitsjournals einkalkuliert, jedoch nicht im Plan aufgeführt.

Die blassen Farben im Plan entsprechen der Planung, die tatsächlichen Zeiten wurden mit satten Farben nachträglich eingefügt. Es ist ersichtlich, dass der erste Teil der Planung relativ genau mit der Realität übereingestimmt hat. Es gab kleine Abweichungen der verschiedenen Aufgaben, pro Tag wurde jedoch ziemlich genau das Soll erreicht. Diese Abweichungen sind im Arbeitsjournal ersichtlich. Ab dem Zeitpunkt der Implementierung wurden viele Aufgaben zu aufwändig geschätzt, da sich der Zeitplan bis zum Ende der Implementierung um fast einen Tag nach vorne verschoben hat. Eine Schwierigkeit der Planung war, dass eine UI-Library zur Entwicklung eingesetzt wurde und der genaue Aufwand für die Implementierung etwas ungewiss war.

Im Diagramm sind ebenfalls die verschiedenen Meilensteine als dunkelgraue, durchgängige Zeilen ersichtlich. Diese beschreiben den angestrebten Zustand des Projekts zu einem fixen Zeitpunkt. Bis zu diesem Zeitpunkt sollten alle zuvor aufgeführten Aufgaben erledigt sein. Diese Meilensteine wurden alle zum fixierten Zeitpunkt oder sogar davor erreicht.

	17. März Morgen	17. März Nachmittag	19. März Morgen	19. März Nachmittag	20. März Morgen	20. März Nachmittag	23. März Morgen	23. März Nachmittag	24. März Morgen	24. März Nachmittag	26. März Morgen	26. März Nachmittag	27. März Morgen	27. März Nachmittag	30. März Morgen	30. März Nachmittag	31. März Morgen	31. März Nachmittag	2. April Morgen	2. April Nachmittag
1 - Zeitplan erstellen	3																			
2 - Projektmanagementmethode wählen und beschreiben		1.5																		
3 - Projektabgrenzung und -umfeld beschreiben		2																		
4 - Wissensbeschaffung beschreiben		1																		
5 - Usecasesdiagramm erstellen			2																	
6 - Testfälle definieren			2																	
7 - Sequenzdiagramm erstellen				2																
8 - Redux Store planen				1.5																
9 - Redux Store implementieren (serviceState)					2															
10 - Redux Store implementieren (tabState)						3														
11 - Redux Store implementieren (messageState)							1.5													
12 - Store testen (Unit Tests)							2.5													
23. März: Der Store ist vollständig implementiert, d.h. die gesamte Funktionalität der Webapplikation ist vorhanden																				
13 - Testing dokumentieren							1.5													
14 - Mockups fertigstellen und dokumentieren							1.5	2												
15 - Endpoint angeben in der GUI							2	1												
16 - gRPC Services laden und anzeigen							2	3												
17 - Tabmanagement umsetzen								3.5	3											
18 - Schema anzeigen								1.5	2											
19 - RequestBody anzeigen und bearbeiten								3.5	4											
20 - Request abschicken und Response anzeigen									3	3										
21 - Error handling									2	4										
22 - Anwenderdokumentation erstellen										2	2									
27. März: Die Webapplikation ist vollständig implementiert und ready for testing																				
23 - Entwicklerdokumentation erstellen											1		1							
24 - Betreiberdokumentation erstellen											1	1								
25 - Testfälle prüfen										2		2								
26 - Bugs fixen										4	6									
30. März: Die Webapplikation ist erfolgreich getestet und allfällige Fehler verbessert																				
27 - Soll - Ist Vergleich Zeitplan dokumentieren											2		2							
28 - Reflexion und Fazit												3	4							
29 - Kurzzusammenfassung der Dokumentation												1	2							
30 - Fertigstellung der Dokumentation													10							
2. April: Die Dokumentation ist fertig gestellt und um spätestens 18:00 Uhr abgegeben																				
Legende: Projektplanung Planung der Applikation Umsetzung Testing Dokumentation																				

Abbildung 2.1: Der Zeitplan mit Soll-Ist Vergleich. Die blassen Farben entsprechen den geplanten Aufgaben, die tatsächlich aufgewendete Zeit wurde mit satten Farben nachgetragen. Die Zahlen entsprechen dem Aufwand der einzelnen Tätigkeiten

2.3 Projektmanagementmethode

Als Projektmanagementmethode wurde Scrum gewählt. Dieser Entscheid wurde anhand folgender Kriterien getroffen:

- In der Adcubum wird nach Scrum gearbeitet, daher sind die verschiedenen Tools vorhanden und bekannt
- Die verschiedene Aufgaben sind zu Beginn der Arbeit bekannt und können entsprechend geplant werden
- Das Projekt ist in sich abgeschlossen und hat einen Start- und einen Endtermin
- Sollten sich Änderungen in der Aufgabenstellung bzw. Projektplanung ergeben, können diese aufgrund der agilen Arbeitsmethodik nachgeführt werden

Für das gesamte Projekt wird ein Sprint aufgesetzt. Dies geschah analog zum Zeitplan, jedoch wurden einige Aufgaben zusammengefasst. Der geschätzte Aufwand ist als Zahl auf den verschiedenen Aufgaben im Zeitplan ersichtlich.

Im Anhang im Abschnitt 11.1 befinden sich zwei Screenshots vom Sprint Backlog und dem initialen Scrum Board. Für die ersten drei Tickets wurden bereits Unteraufgaben erstellt. Dieses Board wurde während der Arbeit fortlaufend aktualisiert und am Daily besprochen.

Bei Scrum gibt es verschiedene Ereignisse. Nicht alle waren für dieses Projekt geeignet. Nachfolgend werden die verschiedenen Ereignisse beschrieben und dokumentiert, ob und wann diese stattgefunden haben bzw. werden.

Am ersten Tag des Projekts wurde ein Grossteil der Planung erledigt. Es wurde ein Zeitplan definiert, User Stories geschrieben und die entsprechenden Tickets erstellt. Dieser Tag kann als Planning angesehen werden, da sowohl die Frage, was in diesem Sprint erledigt werden, als auch wie dies vonstatten gehen soll, beantwortet wurde.

Jeden Tag des Projekts hat ausserdem ein Austausch zwischen mir und meiner vorgesetzten Fachkraft, Stefan Loosli, stattgefunden. Dort wurden folgende Fragen beantwortet:

- Was habe ich gestern erledigt?
- Was will ich heute erreichen?
- Habe ich Probleme? Wenn ja, wie kann ich diese lösen?

Ausserdem wurde das Arbeitsjournal des Vortags besprochen und zu allfälligen Problemen mögliche Lösungswege besprochen.

Das Sprint Review findet im Rahmen der IPA Präsentation statt. Dort wird den Stakeholdern (in diesem Fall den Experten und Stefan Loosli) vom Team (mir) der Fortschritt des vergangenen Sprints demonstriert.

Eine Retrospektive wird nicht abgehalten, da dieses Projekt in sich abgeschlossen ist und keine Optimierungsmöglichkeiten innerhalb des Teams benötigt. Dies wurde so mit dem Chef-experten und Stefan Loosli besprochen. Das Burn-Down Diagramm wurde ebenfalls im Anhang in Abbildung 11.3 angefügt. Es zeigt den Fortschritt im Sprint.

3 Umsetzung

In diesem Kapitel wird die Umsetzung der Arbeit beschrieben. Zuerst wurden die Anforderungen analysiert, anhand dieser Analyse wurde die Applikation dann implementiert und abschliessend getestet.

3.1 Analyse

In den nachfolgenden Abschnitten werden die Anforderungen an die Applikation anhand von einem Anwendungsfall- und einem Sequenzdiagramm genauer beschrieben. Die verschiedenen Use Cases sind jeweils in einer Tabelle noch genauer spezifiziert.

3.1.1 Use Cases

Für die Applikation wurde als erstes ein Anwendungsfalldiagramm erstellt. Dieses ist nachfolgend in der Abbildung 3.1 aufgeführt.

Anschliessend wurden die im Diagramm aufgeführten Use Cases genauer beschrieben. Dies geschieht anhand der folgenden Kriterien:

- Pre-Condition: Welchen Zustand hat die Applikation vor diesem Anwendungsfall?
- Detaillierte Beschreibung: Beschreibung der Aktionen dieses Use Cases
- Post-Condition: Welchen Zustand hat die Applikation nachdem der Anwendungsfall erfolgreich durchgeführt wurde?
- Exceptions: Welche Fehler sind während der Ausführung des Use Cases zu erwarten?

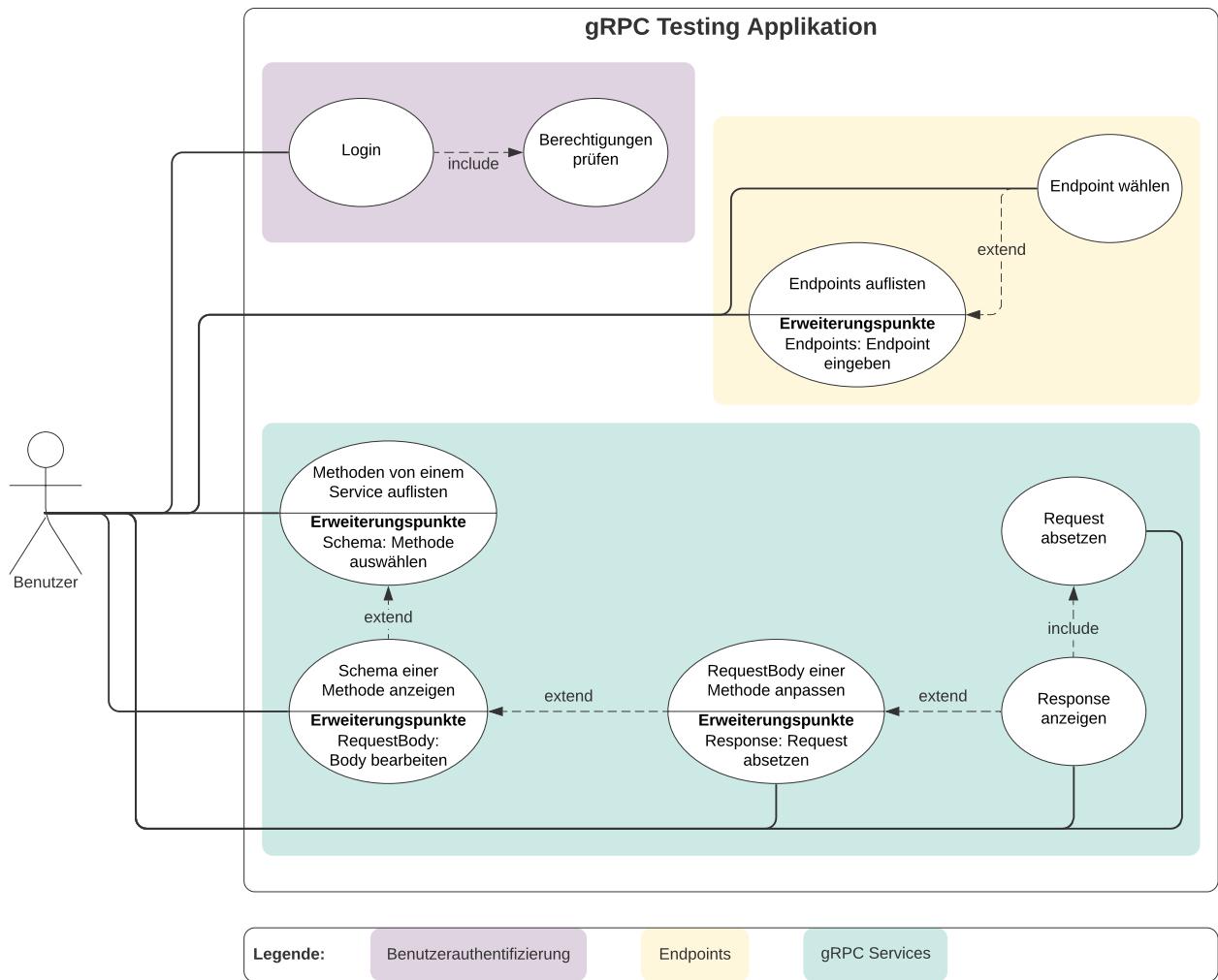


Abbildung 3.1: Das Anwendungsfalldiagramm

Tabelle 3.1: Anwendungsfall Login

Use Case	1 - Login
Pre-Condition	Ein Benutzer wurde auf der Umgebung erfasst und hat die Rechte erhalten, sich einzuloggen. Er ist jedoch noch nicht eingeloggt, hat also keinen validen Token.
Detaillierte Beschreibung	Dem Benutzer wird ein Formular präsentiert, wo er Benutzername und Passwort eingeben kann. Nachdem er valide Daten eingegeben und das Formular abgeschickt hat, werden die Credentials überprüft. Ist diese Überprüfung erfolgreich erhält er einen Bearer Token vom Webserver und wird auf die Homeseite weitergeleitet. Schlägt sie fehl, wird dies dem Benutzer mitgeteilt und er kann einen neuen Versuch starten.
Post-Condition	Der Benutzer hat einen Token erhalten, welcher jedem Request im Header beigefügt wird. Er ist nun eingeloggt und kann weitere Aktionen tätigen.
Exceptions	Der Benutzer existiert nicht oder das Passwort wurde falsch eingegeben.

Tabelle 3.2: Anwendungsfall Endpoints auflisten

Use Case	2 - Endpoints auflisten
Pre-Condition	Der Benutzer ist eingeloggt.
Detaillierte Beschreibung	Wenn der Benutzer auf das Feld klickt, in dem er einen Endpoint angeben kann, wird eine Liste mit den verfügbaren Services präsentiert.
Post-Condition	Gleicher Zustand wie vor dem Anwendungsfall.
Exceptions	Es wurden keine Endpoints vom Server gefunden.

Tabelle 3.3: Anwendungsfall Endpoint wählen

Use Case	3 - Endpoint wählen
Pre-Condition	Der Benutzer ist eingeloggt und die verfügbaren Endpoints werden aufgelistet.
Detaillierte Beschreibung	Der Benutzer kann entweder einen Endpoint aus der Liste wählen oder selber eine Adresse angeben. Wenn er die Eingabe bestätigt, versucht der Servier eine Verbindung zu diesem Endpoint aufzubauen. Ist dies nicht möglich, wird der Benutzer darauf hingewiesen.
Post-Condition	Die Applikation ist mit einem gRPC Service verbunden.
Exceptions	Der Endpoint existiert nicht oder ist aktuell nicht verfügbar.

Tabelle 3.4: Anwendungsfall Services und Methoden auflisten

Use Case	4 - Services und Methoden auflisten
Pre-Condition	Der Benutzer ist eingeloggt und die Applikation mit einem gRPC Service verbunden.
Detaillierte Beschreibung	In der Applikation wird eine Liste von gRPC Services mit den jeweiligen Methoden angezeigt. Der Benutzer kann nun eine Methode wählen, um anschliessend einen Request absetzen zu können.
Post-Condition	Eine Methode wurde angewählt und die Details zu dieser Methode werden auf dem Webclient dargestellt.
Exceptions	Keine.

Tabelle 3.5: Anwendungsfall Schema anzeigen und bearbeiten

Use Case	5 - Schema anzeigen und bearbeiten
Pre-Condition	Der Benutzer ist eingeloggt und die Applikation mit einem gRPC Service verbunden. Ausserdem hat der Benutzer eine Methode des Services angewählt.
Detaillierte Beschreibung	Dem Benutzer wird das Schema und ein Beispiel eines Requestbody der Methode angezeigt. Er kann nun den Requestbody anpassen, wobei die Eingabe auf seine Validität als JSON überprüft wird. Fehler werden entsprechend markiert.
Post-Condition	Der Requestbody wurde vom Benutzer angepasst und der Request ist bereit um abgesetzt zu werden.
Exceptions	Das JSON ist nicht valide.

Tabelle 3.6: Anwendungsfall Response anzeigen

Use Case	6 - Response anzeigen
Pre-Condition	Der Benutzer ist eingeloggt und die Applikation mit einem gRPC Service verbunden. Der Requestbody einer Methode wurde vom Benutzer angepasst und der Request ist bereit um abgesetzt zu werden.
Detaillierte Beschreibung	Wenn der Benutzer den Request absetzt, wird dieser vom Backend an den entsprechenden gRPC Service weitergeleitet. Dieser verarbeitet den Request und liefert eine Response, welche wiederum an den Client weitergeleitet wird. Dort wird sie dem Benutzer angezeigt, sodass dieser mit den erhaltenen Daten weiterarbeiten kann. Sowohl fachliche als auch technische Fehler werden abgefangen und dem Benutzer gemeldet. Der Requestbody kann in diesem Fall erneut abgeändert werden.
Post-Condition	Die Response wird in einem Fenster angezeigt.
Exceptions	Das JSON ist nicht valide oder der gRPC Service ist nicht verfügbar.

3.1.2 Sequenzdiagramm

Für die Applikation wurde zusätzlich ein Sequenzdiagramm erstellt (siehe Abbildung 3.2). Es wurden alle Use Cases in ein Diagramm zusammengefasst, da diese sequenziell ablaufen und aufeinander aufbauen.

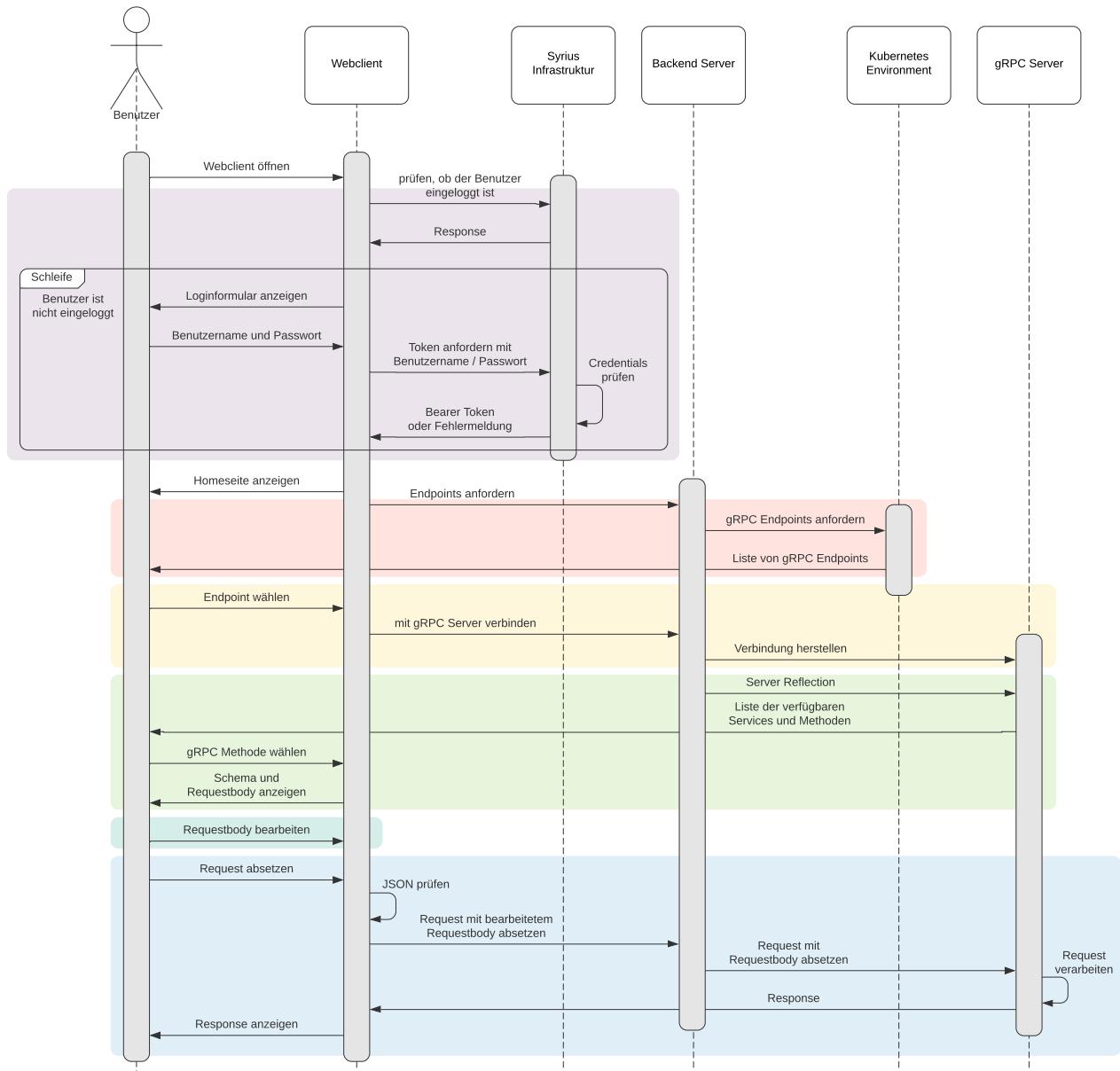


Abbildung 3.2: Das Sequenzdiagramm. Die verschiedenen Use Cases wurden durch Farben gekennzeichnet: 1 violett, 2 rot, 3 orange, 4 hellgrün, 5 dunkelgrün und 6 blau.

3.2 Applikation

Im nachfolgenden Abschnitt wird die Planung und Implementierung der Webapplikation genauer beschrieben.

3.2.1 Versionierung

Als Versionierungstool wurde Git verwendet. In der Adcubum wird dies mit Bitbucket von Atlassian umgesetzt. Außerdem steht die Verwendung von Nextcloud zur Verfügung, was für die Dokumentation und das Arbeitsjournal eingesetzt wurde.

Diese Art der Datensicherung und Versionierung hat sich sehr gut bewährt, da jederzeit ein beliebiger Stand der Arbeit wiederhergestellt hätte werden können. Nachfolgend sind die Einzelheiten zur Versionierung von Code, Dokumentation und Arbeitsjournal aufgeführt.

Code

Der Code wurde mehrmals täglich auf Bitbucket gesichert. Dies geschah immer dann, wenn ein Arbeitsschritt vollendet und der Code gemerget werden sollte. Dies war in der Regel nach zwei bis vier Stunden der Fall. Der neue Code wurde in einen Feature Branch gepusht und ein Pull Request erstellt. Florian Tanner und Stefan Loosli haben diese jeweils visiert und Rückmeldung gegeben. Sobald der Pull Request in Ordnung war, wurde er wieder in den Master Branch gemerget.

Dokumentation

Die Dokumentation wurde in LaTeX gesetzt. Daher war es sinnvoll, auch diesen Code mit GIT zu versionieren, da so der Stand jedes Tages abgerufen werden kann. Es wurde jeweils der Stand am Ende des Tages gesichert.

Der Ordner mit der Dokumentation befand sich auf Nextcloud, sodass die Dokumentation zusätzlich noch in der Cloud gesichert war. Dies brachte den weiteren Vorteil mit sich, dass die vorgesetzten Fachpersonen jederzeit Zugriff auf den aktuellen Stand der Dokumentation hatten.

Arbeitsjournal

Die täglichen Arbeiten im Arbeitsjournal wurde in Google Docs erfasst. So konnte ich von überall darauf zugreifen. War der Bericht fertig, wurde er jeweils am Ende des Tages für die Einsicht der vorgesetzten Fachperson ebenfalls auf Nextcloud abgelegt.

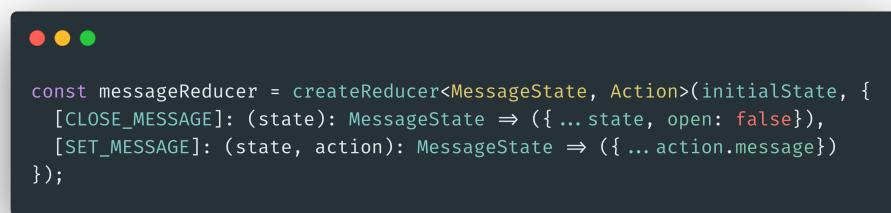
3.2.2 Redux Store

Dieser Abschnitt verschafft einen kurzen Überblick über das Flux Pattern und die Planung des Redux Stores.

Das Flux Pattern

Redux baut auf dem Flux Pattern¹ von Facebook auf. Dieses Pattern beschreibt das angestrebte Verhalten einer Webapplikation. Das Herzstück der Applikation ist der Store, die 'Single Source of Truth'. Dieser State ist readonly, kann also nicht verändert werden. Die einzige Möglichkeit, den State der Applikation zu ändern, besteht darin, sogenannte Actions abzusetzen. Diese Actions haben einen Typ, z.B. 'SET_USERNAME', und möglicherweise einen Body. In diesem Beispiel wäre das wahrscheinlich der Benutzername, welcher gesetzt werden soll. Die Funktionen, welche die Actions entgegennehmen und einen neuen State zurückgeben, werden Reducers genannt. Diese sind sogenannte 'Pure Functions', das heisst, dass genau vorhergesagt werden kann, welchen Rückgabewert sie bei definierten Parametern liefern. Sie verändern den State nicht, sondern erstellen ein neues und mutiertes State Objekt, welches dann zurückgegeben wird.

Die Abbildung 3.3 enthält ein Beispiel eines Reducers. Hier kann entweder eine Message geschlossen oder neu gesetzt werden. Der Code wurde aus dem Webclient entnommen, dort werden so die Fehlermeldungen gespeichert. Die Methode `createReducer()` nimmt ein Objekt als Parameter, welches den Typ der verschiedenen Actions als Keys und die jeweiligen Funktionen als Values hat.



```
const messageReducer = createReducer<MessageState, Action>(initialState, {
  [CLOSE_MESSAGE]: (state): MessageState => ({ ...state, open: false}),
  [SET_MESSAGE]: (state, action): MessageState => ({ ...state, message: action.message})
});
```

Abbildung 3.3: Ein Beispiel eines Reducers

Redux wird oft zusammen mit React verwendet. Redux kann aber auch mit anderen Frameworks verwendet oder in eine Javascript Webapplikation eingebunden werden. Für die React

¹<https://facebook.github.io/flux/> [31.03.2020]

Bindings gibt es eine sehr ausführliche Dokumentation². Für asynchrone Operationen und Operationen mit 'Side Effects' wurde eine weitere Library hinzugezogen: Redux-Saga. Mit dessen Hilfe können die Side Effects der Applikation gemanaged werden.

Planung

Sowohl der Aufbau des Stores als auch die verschiedenen Actions wurden vor der Implementierung geplant. Diese Planung ist im Anhang in Abschnitt 11.3 angefügt.

3.2.3 Mockup

Bei der Entwicklung einer GUI ist es wichtig, dass diese zuvor geplant wurde. So muss sich der Entwickler keine Gedanken über das Design und User Experience machen und kann sich auf die Implementierung konzentrieren. Eine Möglichkeit für diese Planung sind Mockups.

Vor der Projektarbeit wurden bereits Gespräche mit einigen Testern und Entwicklern geführt, um ihre Bedürfnisse abzuholen. Anhand dieser Anregungen wurden dann auch schon erste Entwürfe der Applikation festgehalten, welche aber noch überarbeitet werden mussten. In der Abbildung 3.4 ist das finale Mockup aufgeführt. Dabei geht es nicht darum, dass das Design endgültig bestimmt wurde, sondern mehr um die allgemeine Positionierung und Hervorhebung der einzelnen Elemente.

Bei der Implementation wird die UI-Library MaterialUI³ verwendet, welche an die Material Design Guidelines von Google angelehnt ist. Deshalb wurde auch im Mockup dieser Style gewählt.

²<https://react-redux.js.org/> [31.03.2020]

³<https://material-ui.com/> [31.03.2020]

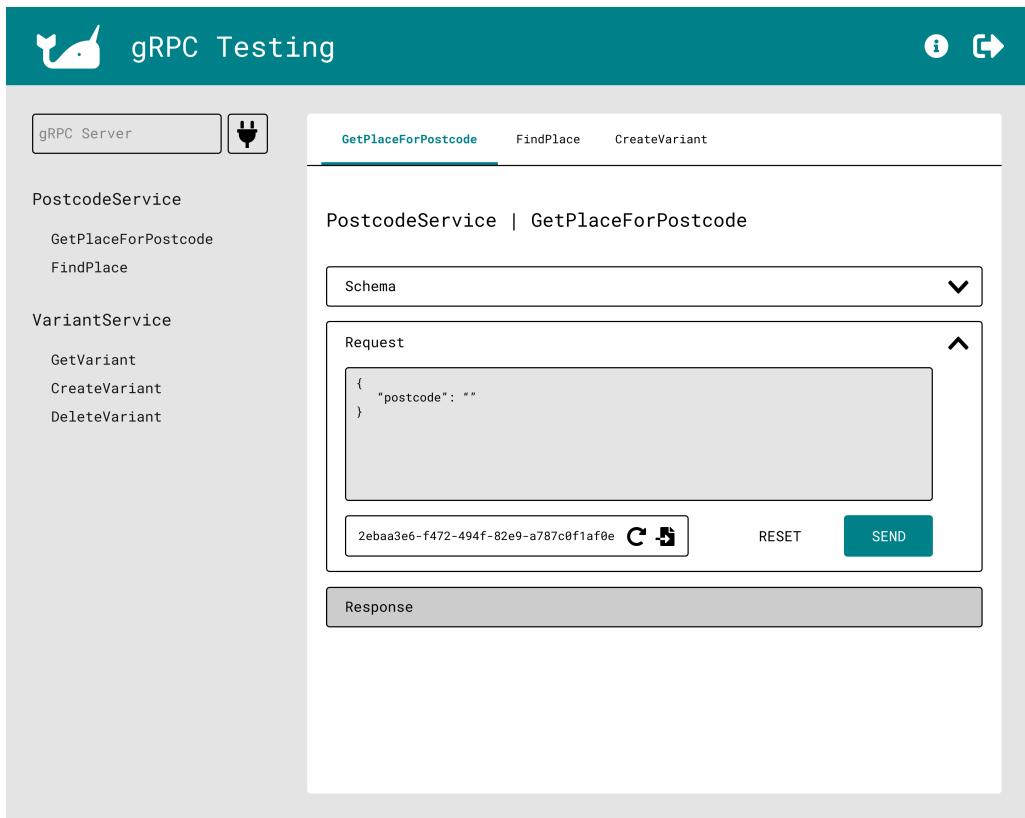


Abbildung 3.4: Das Mockup des Web-Clients

3.2.4 Implementierung

Dieser Abschnitt verschafft einen Überblick über die Implementierung der Webapp und zeigt die Resultate auf.

Redux Store

Der Store wurde anhand der Planung umgesetzt. Es mussten einige kleine Anpassungen vorgenommen werden, jedoch konnte der grösste Teil der Planung übernommen werden.

Dieser Teil der Applikation wurde als erstes implementiert, da sich hier der grösste Teil der Logik befindet. Bei der Umsetzung gibt es ein nützliches Hilfsmittel: die Redux DevTools⁴. Dieses Tool loggt alle Actions, welche ausgeführt wurden, und der State kann für jeden Zeitpunkt in der Vergangenheit rekonstruiert werden. Dies vereinfacht das Debugging sehr stark. Mit diesem Tool kann ausserdem eine visuelle Darstellung des States der App generiert werden. Ein Beispiel ist in Abbildung 3.5 aufgeführt.

Der State der Applikation in der Abbildung 3.5 verrät, dass die Applikation bereits in Ge-

⁴<https://github.com/zalmoxisus/redux-devtools-extension> [31.03.2020]

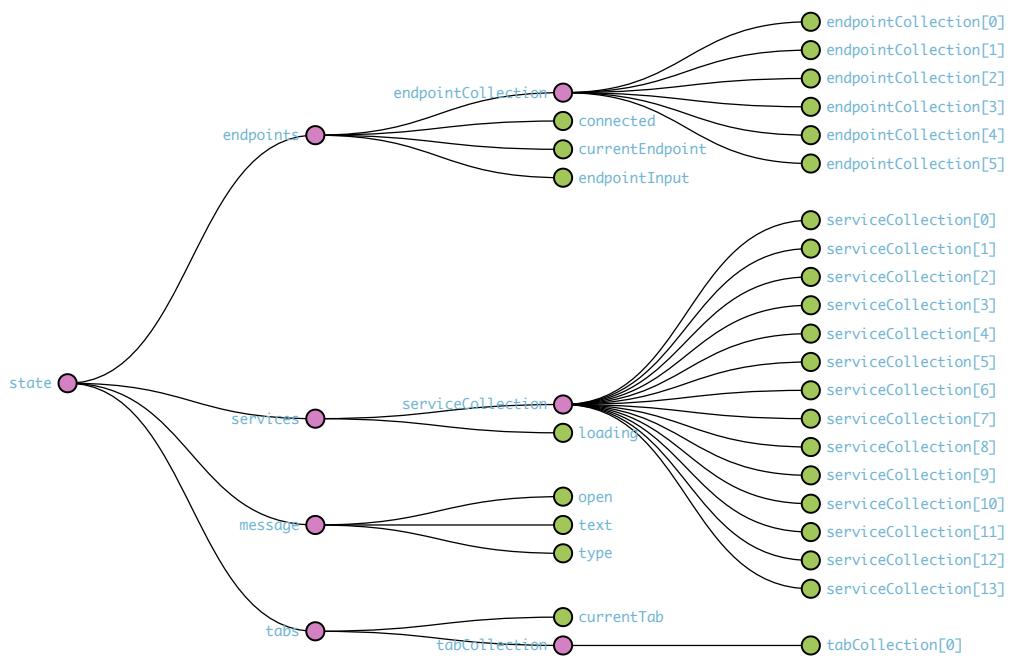


Abbildung 3.5: Ein Beispiel des States grafisch dargestellt

brauch war. Die Liste der Endpoints enthält sechs Elemente, und auch die ServiceCollection ist bereit abgefüllt. Ausserdem ist aktuell ein Tab geöffnet.

GUI

Die GUI wurde mit MaterialUI analog zum Mockup umgesetzt. Dabei wurde dieselbe Reihenfolge eingehalten, wie auch ein Benutzer die App schlussendlich benutzt. Das heisst, dass zuerst der Endpoint, dann eine Methode ausgewählt und schlussendlich der Body angepasst und abgeschickt wurde.

Wenn der Benutzer auf die Website kommt, wird als erstes überprüft, ob er authentifiziert ist. Falls nein, wird er auf die Loginseite (siehe Abbildung 3.6) weitergeleitet.

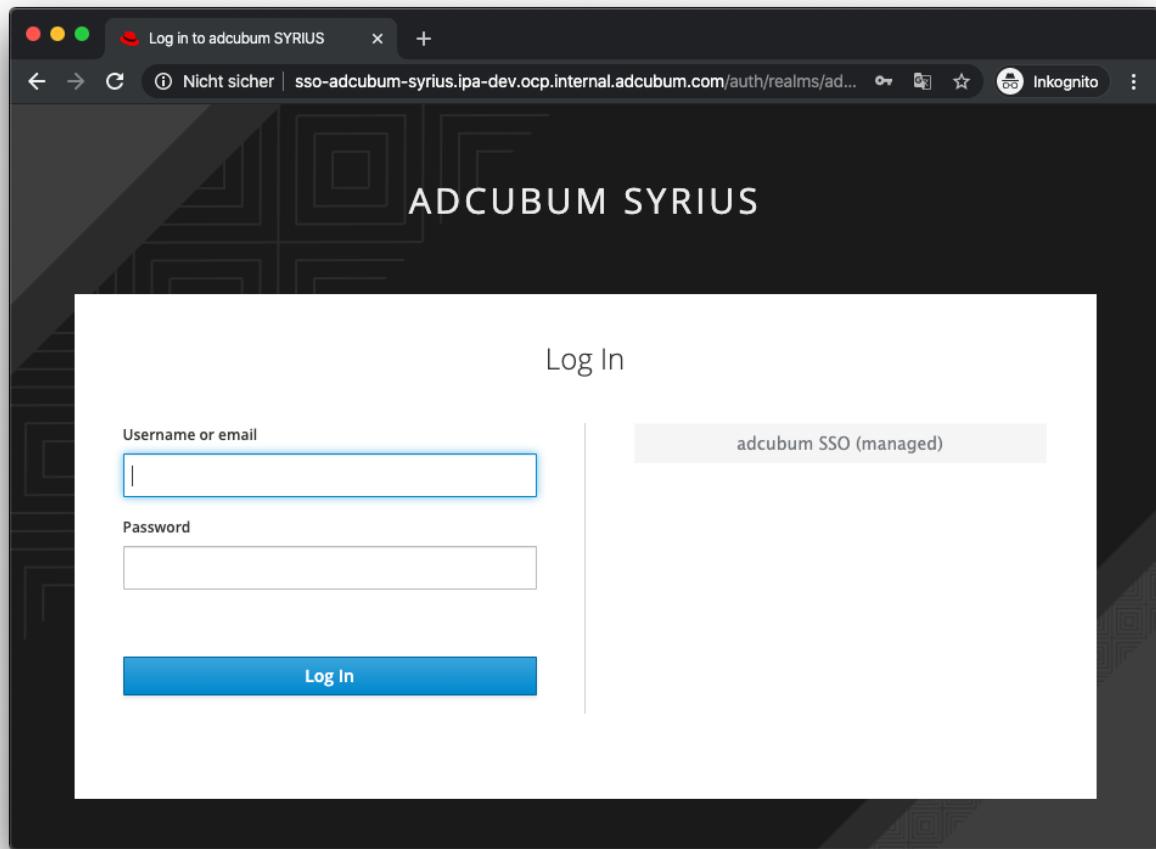


Abbildung 3.6: Die Loginseite

Hat sich der Benutzer authentifiziert, sieht er die Hauptseite der Webapp. Diese hat links ein Inputfeld, in welchem ein gRPC Endpoint ausgewählt werden kann. In einem Dropdown werden dann die verfügbaren Endpoints aufgelistet (siehe Abbildung 3.7).

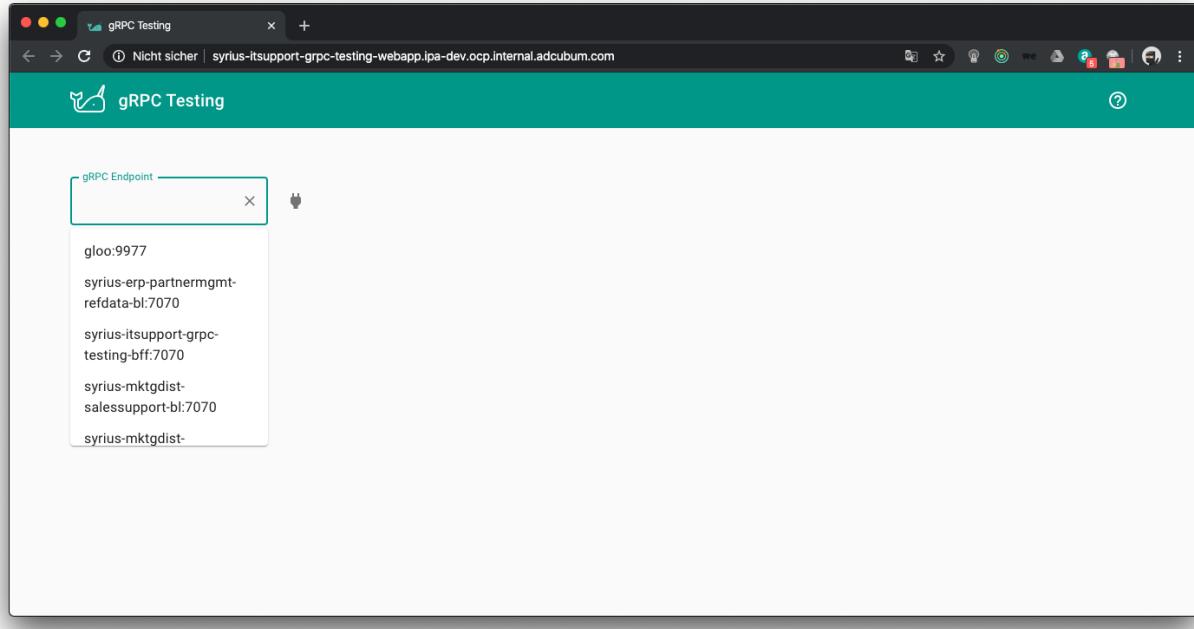


Abbildung 3.7: Endpoint wählen

Wenn der Benutzer einen Endpoint ausgewählt und bestätigt hat, werden die verfügbaren Services und Methoden aufgelistet. Dies geschieht in einer Baumstruktur, wobei alle Methoden initial ausgeblendet sind. Bei einem Klick auf einen Service werden die entsprechenden Methoden angezeigt (siehe Abbildung 3.8).

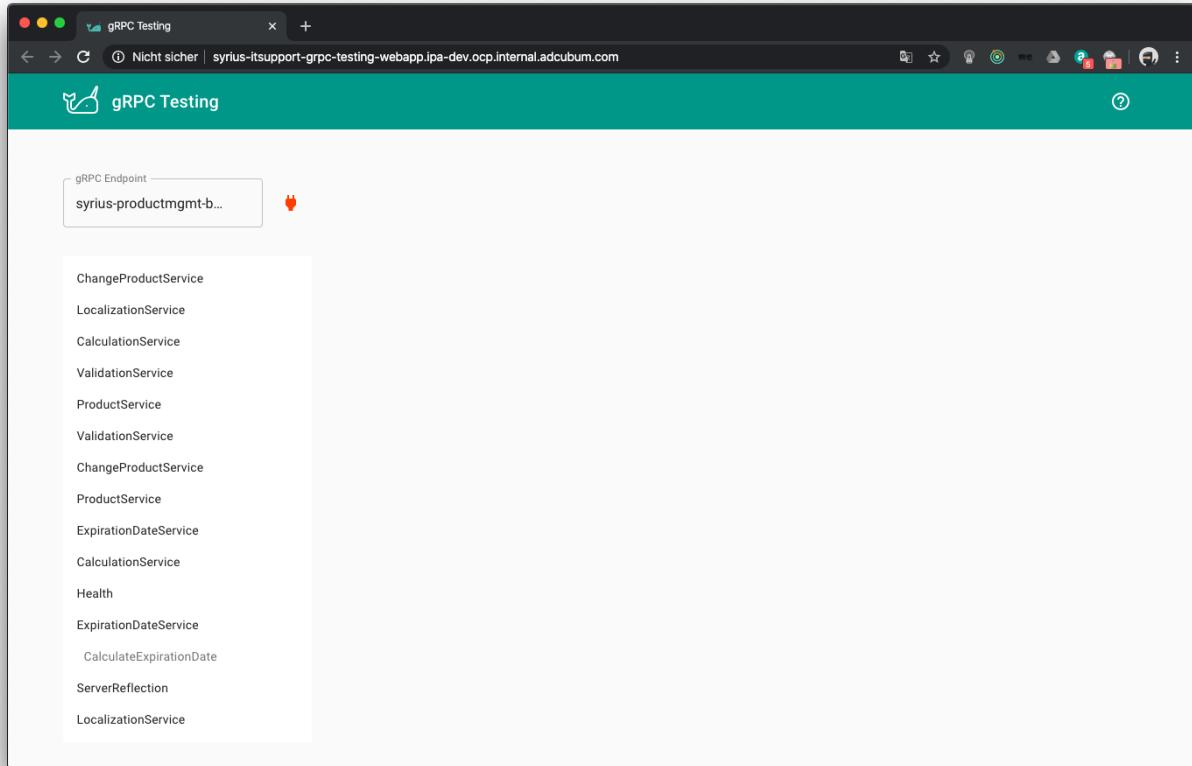


Abbildung 3.8: Liste der Services

Der Benutzer kann nun eine Methode wählen, indem er auf den entsprechenden Listeneintrag klickt. Dann wird ein neuer Tab geöffnet, welcher das Schema und den Requestbody der Methode enthält. Dies ist in Abbildung 3.9 dargestellt.

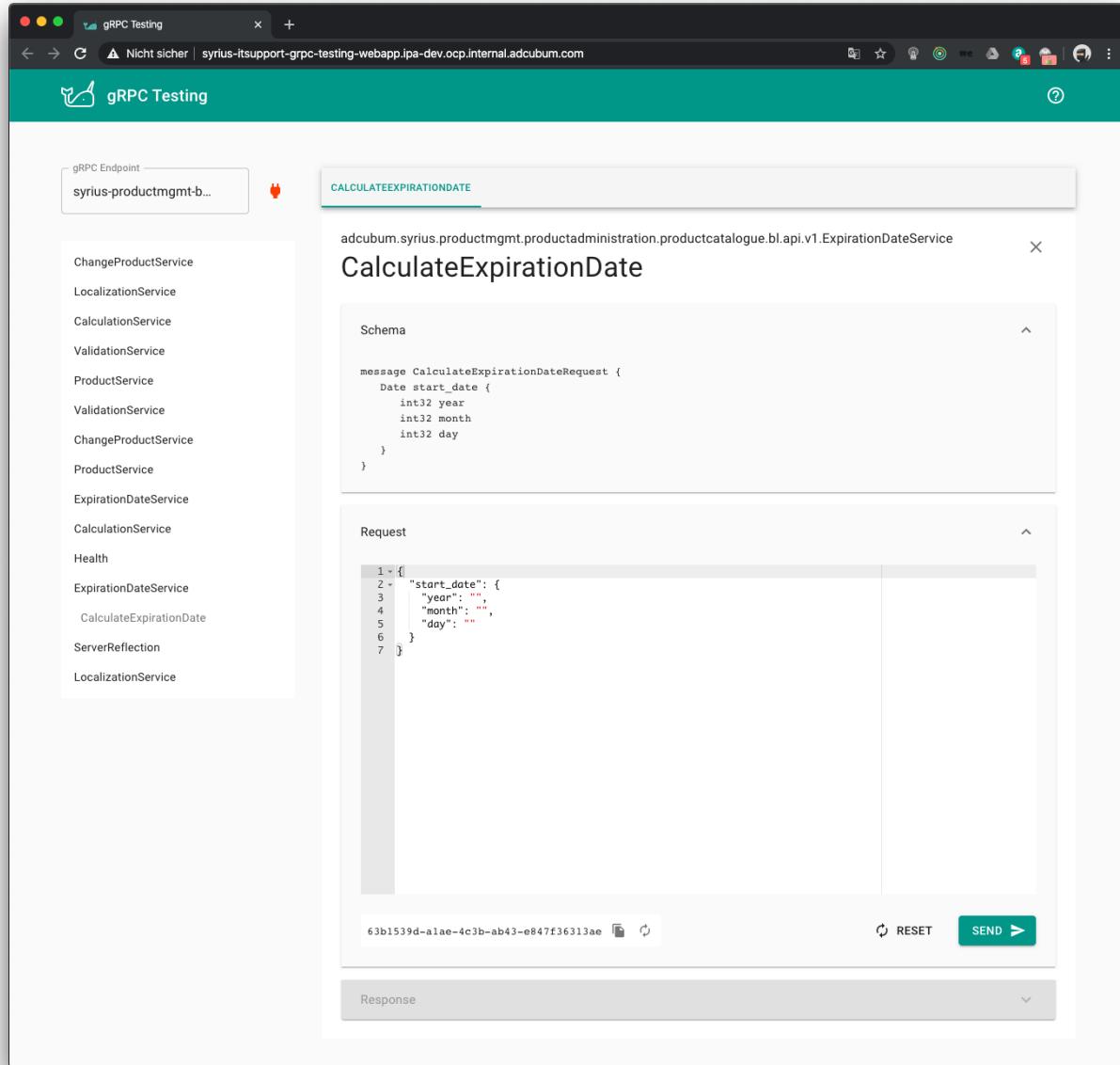


Abbildung 3.9: Schema und Requestbody einer Methode

Der Requestbody kann im Editor angepasst werden. Die Eingabe wird auf seine Validität als JSON überprüft und allfällige Fehler werden angezeigt. Ist der Benutzer mit seiner Eingabe zufrieden, kann er seinen Request absetzen. Die Response des Servers wird dann im Panel 'Response' angezeigt (siehe Abbildung 3.10).

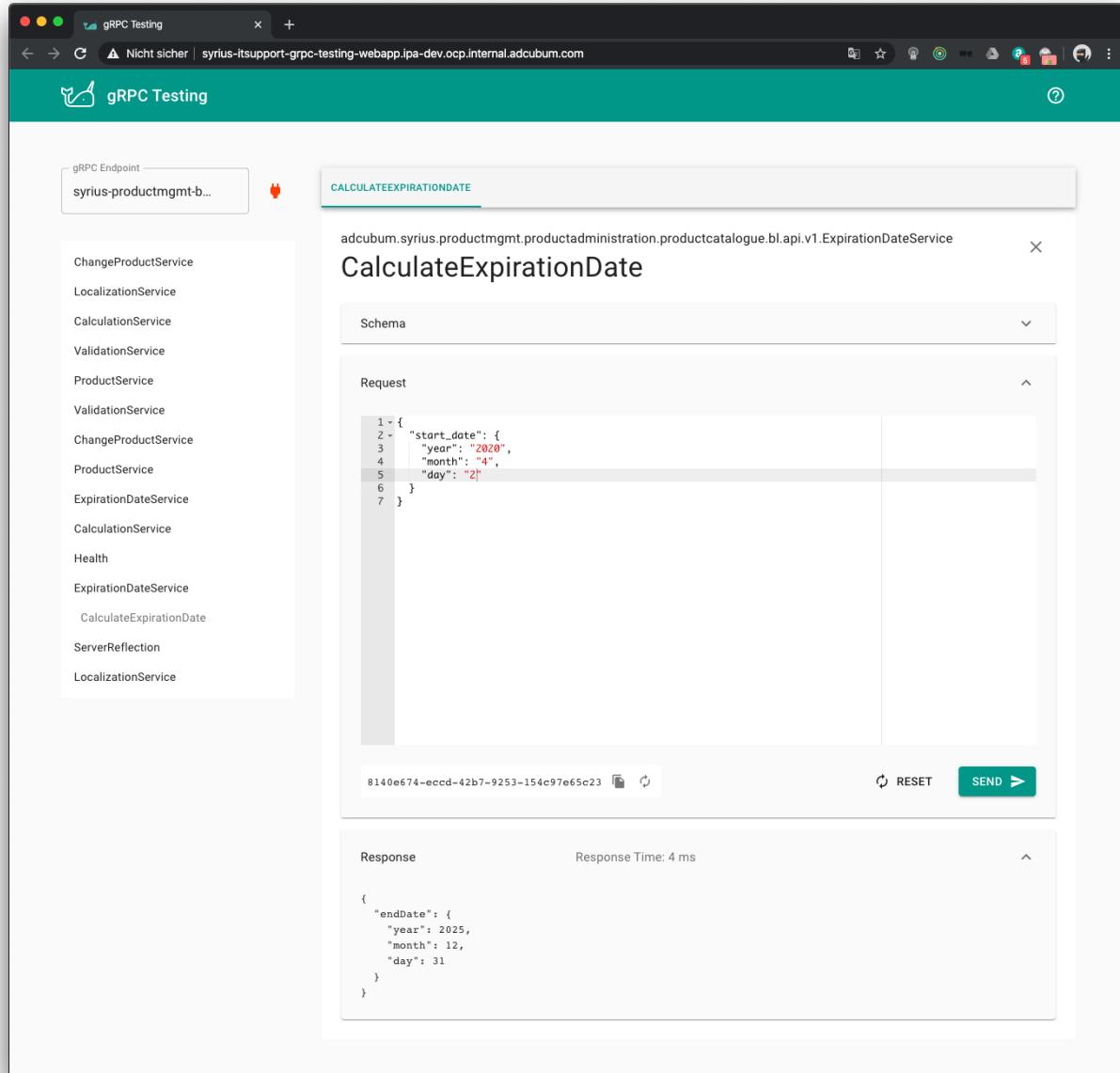


Abbildung 3.10: Die Response eines abgesetzten Requests

Wenn das JSON nicht valide ist, wenn der Benutzer den Request absetzt, wird dieser darauf hingewiesen und hat die Möglichkeit, seine Eingabe zu verbessern (siehe Abbildung 3.11). Passt im Server ein Fehler, beispielsweise wenn ein Feld nicht gefunden werden kann, so wird der Benutzer auch auf diesen hingewiesen, wobei die Response des Servers trotzdem angezeigt wird (siehe Abbildung 3.12).

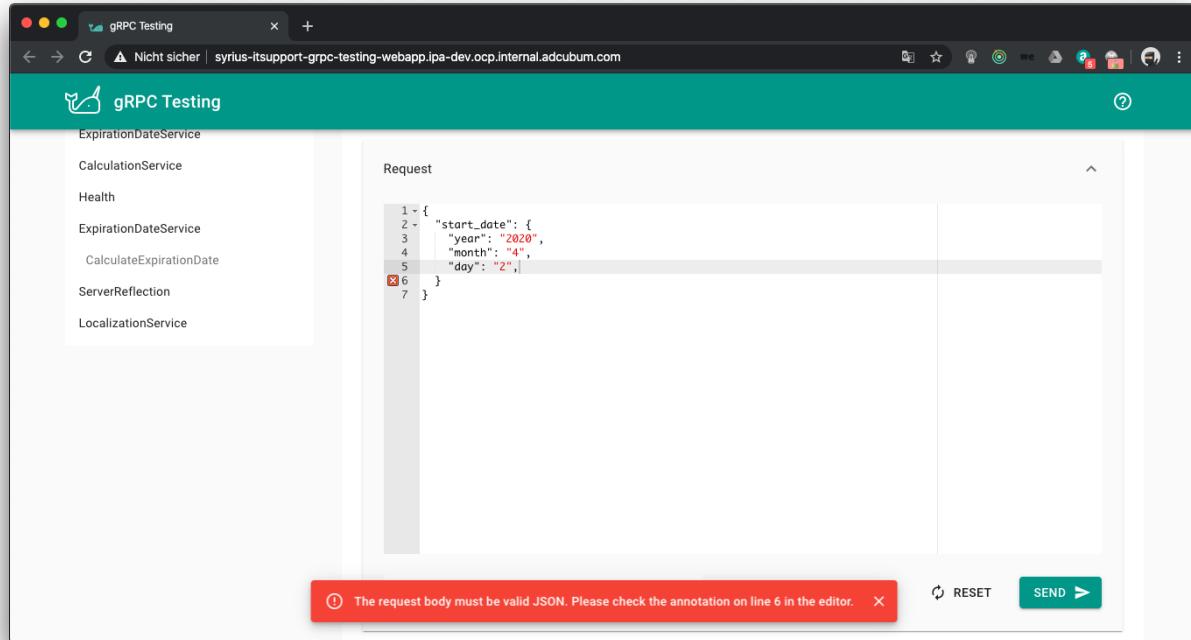


Abbildung 3.11: Der Fehler, wenn ein invalides JSON bestätigt wird

Die Applikation kann, anstatt mit einer Maus, auch ausschliesslich mit der Tastatur bedient werden. Dabei wird, wie in einem Browser üblich, mit der Tabulator-Taste zwischen den Elementen gewechselt und mit Enter eine Eingabe bestätigt.

Weitere Funktionalitäten der Applikation sind, dass der Request Body auf den Anfangswert zurückgesetzt oder das JSON formatiert werden kann. Ausserdem gibt es die Möglichkeit, mit einem Klick eine UUID zu generieren und zu kopieren. Eine Eingabe in den Editor kann mit **COMMAND + Z** rückgängig und mit **COMMAND + SHIFT + Z** wiederhergestellt werden.

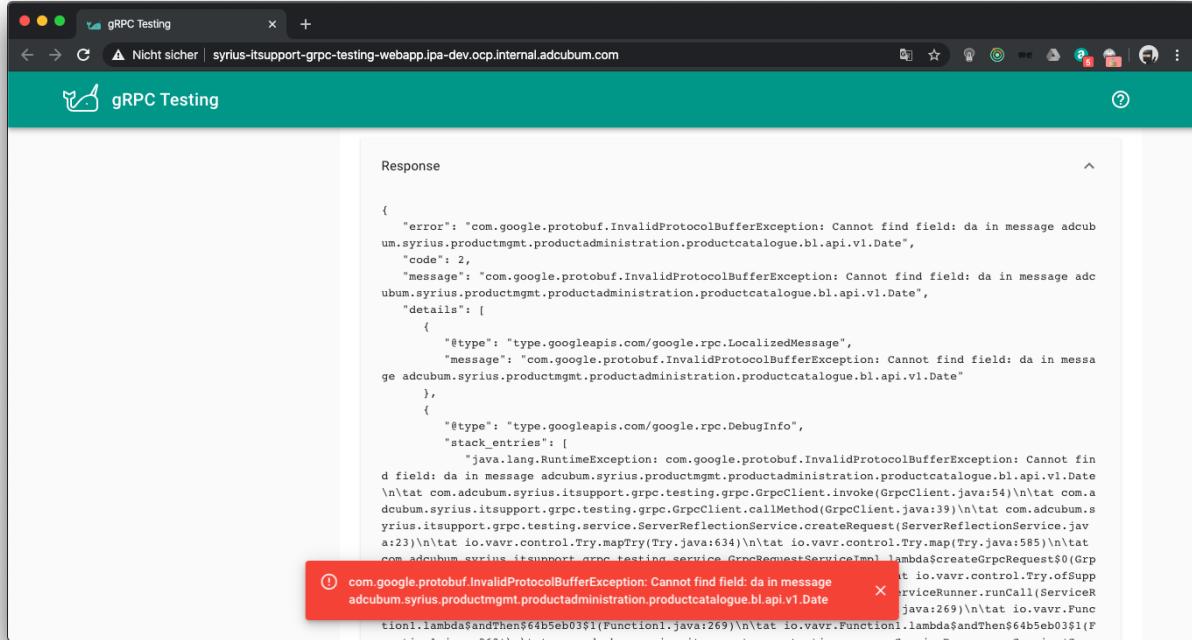


Abbildung 3.12: Der Fehler, wenn der gRPC Server eine Exception wirft

3.2.5 Dokumentation

Für die Applikation wurden drei Dokumentationen erstellt: eine Anwenderdokumentation, eine Betreiber- und eine Entwicklerdokumentation. Diese sind in den nachfolgenden Abschnitten genauer beschrieben.

Anwenderdokumentation

Die Anwenderdokumentation wurde direkt in die Applikation eingebunden. Dies wurde so gewählt, dass der Anwender keinen Zugriff zum Code benötigt und die Anleitung dort gefunden werden kann, wo sie gebraucht wird. Klickt der Benutzer auf das Hilfe Icon in der rechten oberen Ecke, so öffnet sich ein Overlay mit einer Anleitung (siehe Abbildung 3.13). Hier wird der Zweck der Applikation und auch deren Bedienung erklärt.

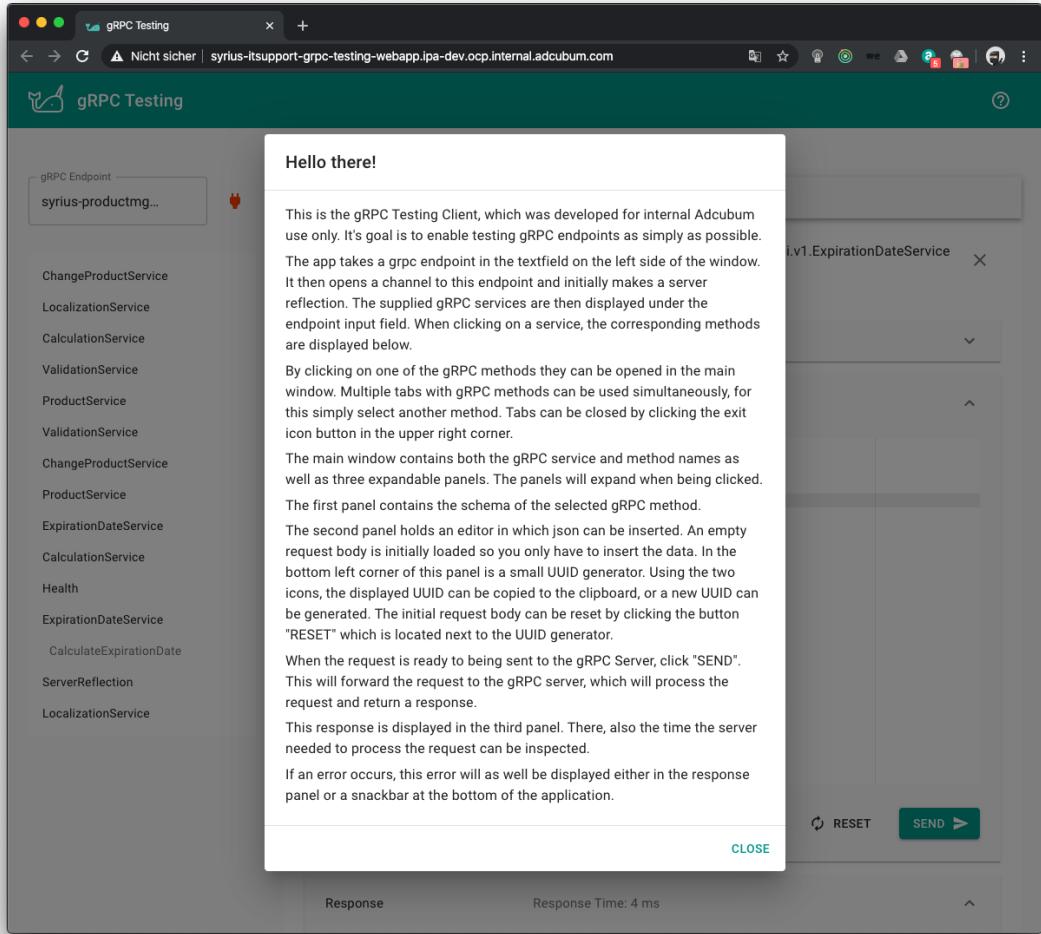


Abbildung 3.13: Die Anwenderdokumentation

Betreiber- und Entwicklerdokumentation

Diese zwei Dokumentationen wurden in ein Markdown File (`README.md`) zusammengefasst. Das File enthält alle notwendigen Schritte, um die Webapplikation auf einer internen Openshift Umgebung bereitzustellen. Außerdem wird beschrieben, welche Tools ein Entwickler installieren muss und wie er den Client lokal für die Weiterentwicklung starten kann. Es ist auch eine kurze Beschreibung der Ordnerstruktur und des Testings vorhanden.

Die Installationsanleitung wurde von Javier Diez getestet, und bis auf einen kleinen Fehler war die Anleitung realitätsgerecht und richtig. Dieser Fehler wurde anschliessend behoben.

3.3 Testing

In diesem Abschnitt wird das Testing der Applikation beschrieben und die Resultate dokumentiert und diskutiert. Das Testkonzept bestimmt Abgrenzung, Vorgehensweise, Mittel und Ablaufplan der Testaktivitäten. Das Testumfeld beschreibt die Umgebung, in welcher die Tests durchgeführt werden. Das Testprotokoll gibt Aufschluss über die durchgeführten Tests gemäss Testkonzept und dokumentiert die Testresultate und die daraus abgeleiteten Massnahmen.

3.3.1 Testkonzept

Die Webapplikation wird auf drei Arten getestet: Einerseits wird der Code automatisch mit ESLint auf Fehler und Unschönheiten überprüft. Andererseits werden bei jedem Build mit Jenkins die Unit Tests durchgeführt. Schlägt einer dieser Tests fehl wird der Build abgebrochen und der Code muss angepasst werden, bevor in den Master Branch gemerged werden kann.

Ein automatisiertes Unit Testing der grafischen Oberfläche ist in den meisten Fällen nicht sehr sinnvoll. Aus diesem Grund wurde für diese Arbeit definiert, dass nur der Code vom Redux Store getestet werden soll, da sich dort die Mehrheit der Funktionalität der Applikation befindet. Auch innerhalb des Stores gibt es einige Tests, welche angesichts der limitierten Zeitvorgabe bewusst weggelassen wurden. Beispielsweise, wenn ganze Module oder auch grosse Teile des Stores gemockt werden mussten.

Die GUI wird manuell mittels Testfällen auf Fehler geprüft. Diese Testfälle wurden vor der Implementation erstellt und sind im Abschnitt 3.3.2 aufgeführt. Die Resultate sind ebenfalls dokumentiert und diskutiert.

3.3.2 Testumfeld

Die Adcubum verwendet, wie bereits erwähnt, Git mit Bitbucket von Atlassian als Versionierungstool. Für dieses Projekt wurde eine Build Pipeline eingerichtet, sodass immer ein Jenkins Build angestoßen wird, wenn Änderungen auf einen Remote Branch gepusht werden. Während diesem Build wird der Code von ESLint überprüft und die Unit Tests durchgeführt.

Das Linting erfolgt nach dem Standard von ESLint⁵. Dieser Linter kann Probleme im Code finden und manchmal auch gleich selber beheben. Die Unit Tests werden im nachfolgenden Abschnitt 3.3.3 näher beschrieben.

Ein erfolgreicher Build erstellt ein Docker Image, welches automatisch in das interne Registry gepusht wird. Dieses kann dann in einem Kubernetes Environment bereitgestellt werden. In der Adcubum ist dies eine Openshift Umgebung. Dort können dann die Testfälle durchgeführt

⁵<https://eslint.org/> [31.03.2020]

werden. In einem solchen Testing Environment ist sowohl der gRPC Testing BFF, das gRPC Testing Webapp als auch einige Server vorhanden, welche eine gRPC Schnittstelle implementieren. Die Testfälle werden im nachfolgenden Abschnitt erläutert.

Testprotokoll

Die nachfolgenden Tabellen enthalten die während der Planung definierten Testfälle. Sie wurden in die Kategorien Authentifizierung, Endpoints und gRPC Services unterteilt. Die Testfälle wurden nach den folgenden Kriterien beschrieben:

- Nr. / Beschreibung: Im Testprotokoll ist dieser Testfall unter derselben Nummer aufzufinden. Die Beschreibung enthält das zu testende Themengebiet
- Art: ist der Testfall positiv oder negativ?
- Input / Aktion: Welche Aktionen werden vom Tester ausgeübt?
- Erwartetes Resultat: Der erwartete Zustand der Applikation, nachdem der Testfall durchgeführt wurde
- Tatsächliches Resultat: Der tatsächliche Zustand der Applikation, nachdem der Testfall durchgeführt wurde
- Status: Ist der Testfall OK oder nicht OK (NOK)?

Die Testfälle wurden nach der Implementation der Applikation getestet. Dies fand wie bereits erwähnt auf der Openshift Umgebung statt. Die getestete Version ist 0.1.0.

Tabelle 3.7: Testfälle Authentifizierung

Nr. / Beschreibung	Art	Input / Aktion	Erwartetes Resultat	Tatsächliches Resultat	Status
1 - Guard	negativ	Der Benutzer ist nicht eingeloggt und öffnet den Webclient.	Die Homeseite wird nicht angezeigt, stattdessen wird der Benutzer auf eine Loginseite weitergeleitet.	Die Homeseite wird nicht angezeigt, stattdessen wird der Benutzer auf eine Loginseite weitergeleitet.	OK

Nr. / Beschreibung	Art	Input / Aktion	Erwartetes Resultat	Tatsächliches Resultat	Status
2 - Guard	positiv	Der Benutzer ist eingeloggt und öffnet den WebClient.	Die Homeseite wird wie erwartet angezeigt.	Die Homeseite wird wie erwartet angezeigt.	OK
3 - Login	positiv	Der Benutzer füllt valide Credentials in das Loginformular ein und schickt dieses ab.	Der Benutzer wird eingeloggt, erhält ein Token und wird auf die Homeseite weitergeleitet.	Der Benutzer wird eingeloggt, erhält ein Token und wird auf die Homeseite weitergeleitet.	OK
4 - Login	negativ	Der Benutzer füllt invalide Credentials in das Loginformular ein und schickt dieses ab.	Der Benutzer wird nicht eingeloggt und wird auf die invaliden Credentials hingewiesen.	Der Benutzer wird nicht eingeloggt und wird auf die invaliden Credentials hingewiesen.	OK

Tabelle 3.8: Testfälle Authentifizierung

Nr. / Beschreibung	Art	Input / Aktion	Erwartetes Resultat	Tatsächliches Resultat	Status
5 - Endpoints auflisten	positiv	Der Benutzer klickt in das Inputfeld, wo ein Endpoint angegeben werden kann.	Es erscheint eine Liste von verfügbaren Endpoints.	Es erscheint eine Liste von verfügbaren Endpoints.	OK

Nr. / Beschreibung	Art	Input / Aktion	Erwartetes Resultat	Tatsächliches Resultat	Status
6 - Endpoints auflisten, jedoch sind keine Endpoints verfügbar	negativ	Der Benutzer klickt in das Inputfeld, wo ein Endpoint angegeben werden kann.	Es erscheint keine Liste.	Es erscheint keine Liste.	OK
7 - Endpoint aus der Liste wählen	positiv	Der Benutzer wählt einen Endpoint aus der Liste.	Der ausgewählte Wert wird in das Textfeld übertragen.	Der ausgewählte Wert wird in das Textfeld übertragen.	OK
8 - Endpoint eingeben	positiv	Der Benutzer ändert den Wert des Textfelds.	Der Wert des Textfelds wird entsprechend angepasst.	Der Wert des Textfelds wird entsprechend angepasst.	OK
9 - Endpoints filtern	positiv	Der Benutzer schreibt einige Zeichen in das Textfeld.	In der Liste mit Vorschlägen werden nur noch Ergebnisse angezeigt, welche mit den eingegebenen Zeichen beginnen.	In der Liste mit Vorschlägen werden nur noch Ergebnisse angezeigt, welche mit den eingegebenen Zeichen beginnen.	OK
10 - Mit Endpoint verbinden	positiv	Der Benutzer wählt einen Endpoint und bestätigt die Eingabe.	Die Applikation baut eine Verbindung mit dem entsprechenden gRPC Service auf.	Die Applikation baut eine Verbindung mit dem entsprechenden gRPC Service auf.	OK

Nr. / Beschreibung	Art	Input / Aktion	Erwartetes Resultat	Tatsächliches Resultat	Status
11 - Mit Endpoint verbinden	negativ	Der Benutzer gibt einen validen Hostnamen von einem nicht vorhandenen Endpoint an und bestätigt die Eingabe.	Der Benutzer wird auf seine fehlerhafte Eingabe hingewiesen und kann die Endpoint Adresse anpassen.	Der Benutzer wird auf seine fehlerhafte Eingabe hingewiesen und kann die Endpoint Adresse anpassen.	OK
12 - Mit Endpoint verbinden	negativ	Der Benutzer gibt einen ungültigen Hostnamen ein und bestätigt die Eingabe.	Der Benutzer wird auf seine fehlerhafte Eingabe hingewiesen und kann die Endpoint Adresse anpassen.	Der Benutzer wird auf seine fehlerhafte Eingabe hingewiesen und kann die Endpoint Adresse anpassen.	OK

Tabelle 3.9: Testfälle Authentifizierung

Nr. / Beschreibung	Art	Input / Aktion	Erwartetes Resultat	Tatsächliches Resultat	Status
13 - Services und Methoden anzeigen	positiv	Der Benutzer stellt eine Verbindung zu einem gRPC Service her.	Die per Server Reflection ermittelten Services und Methoden werden dargestellt.	Die per Server Reflection ermittelten Services und Methoden werden dargestellt.	OK
14 - Neue Methode auswählen	positiv	Der Benutzer wählt eine Methode indem er auf den Listeneintrag klickt.	Ein neuer Tab wird erstellt und geöffnet. Die Seite enthält den Namen des Services und der Methode.	Ein neuer Tab wird erstellt und geöffnet. Die Seite enthält den Namen des Services und der Methode.	OK

Nr. / Beschreibung	Art	Input / Aktion	Erwartetes Resultat	Tatsächliches Resultat	Status
15 - Bereits geöffnete Methode auswählen	positiv	Der Benutzer wählt eine bereits geöffnete Methode, indem er auf den Listeneintrag klickt.	Die Applikation navigiert zum Tab, in welchem die angewählte Methode geöffnet ist.	Die Applikation navigiert zum Tab, in welchem die angewählte Methode geöffnet ist.	OK
16 - Schema anzeigen	positiv	Der Benutzer öffnet einen Tab mit einer Methode.	Das Schema, welches der gRPC Service erwartet, wird auf der Seite dargestellt.	Das Schema, welches der gRPC Service erwartet, wird auf der Seite dargestellt.	OK
17 - Requestbody anzeigen	positiv	Der Benutzer öffnet einen Tab mit einer Methode.	Der "leere" Requestbody wird auf der Seite dargestellt.	Der "leere" Requestbody wird auf der Seite dargestellt.	OK
18 - Requestbody bearbeiten	positiv	Der Benutzer bearbeitet den Requestbody.	Der Wert des Requestbodys wird entsprechend angepasst.	Der Wert des Requestbodys wird entsprechend angepasst.	OK
19 - Requestbody validieren	negativ	Der Benutzer bearbeitet den Requestbody so, dass ein invalides JSON entsteht.	Der Fehler im JSON wird an der entsprechenden Stelle markiert.	Der Fehler im JSON wird an der entsprechenden Stelle markiert.	OK

Nr. / Beschreibung	Art	Input / Aktion	Erwartetes Resultat	Tatsächliches Resultat	Status
20 - Request absetzen	positiv	Der Benutzer setzt einen Request mit dem von ihm bearbeiteten validen Requestbody ab.	Der Request wird an den gRPC Service weitergeleitet. Anschliessend wird die Response des gRPC Services wird im Tab angezeigt.	Der Request wird an den gRPC Service weitergeleitet. Anschliessend wird die Response des gRPC Services wird im Tab angezeigt.	OK
21 - Fehler-handling	negativ	Der Benutzer setzt einen Request mit dem von ihm bearbeiteten validen Requestbody ab. Beim BFF oder dem gRPC Server entsteht ein Fehler.	Der Benutzer wird mit einer Fehlermeldung auf den Fehler hingewiesen. Der Responsebody wird, sofern vorhanden, im Responsepanel angezeigt.	Der Benutzer wird mit einer Fehlermeldung auf den Fehler hingewiesen. Der Responsebody wird im Response-panel angezeigt.	OK

Es wurden alle Anforderungen des Testprotokolls an die Applikation erfüllt. Die Authentifizierung wurde zu einem grossen Teil vom Template und der Sirius Infrastruktur übernommen. Die verfügbaren Endpoints werden dem Benutzer aufgelistet, welcher dann einen Endpoint wählen kann. Die Applikation kann anschliessend mit dem gewählten Endpoint verbinden (sofern dieser vorhanden ist) und die per Server Reflection erhaltenen Services und Methoden auflisten. Wenn der Benutzer dann einen Endpoint auswählt, werden in drei Panels das Schema, der Requestbody und die Response angezeigt. Der Requestbody kann vom Benutzer angepasst und an den Server geschickt werden. Die Response wird anschliessend im Panel 'Response' angezeigt. Treten während der Verarbeitung Fehler auf, werden diese dem Benutzer mitgeteilt, und er kann die fehlerhafte Eingabe anpassen.

Da die Applikation alle Anforderungen erfüllt, müssen keine Massnahmen eingeleitet werden. Einer der nächsten Schritte wäre die Implementierung von Tastaturbefehlen, mit deren Hilfe beispielsweise eine UUID eingefügt werden kann.

3.3.3 Unit Testing

Die Unit Tests werden, wie bereits beschrieben, bei jedem Build ausgeführt. Diese testen den Redux Store, wo das Unit Testing am sinnvollsten ist, da sich dort der grösste Teil der Logik des Clients befindet. Im Anhang ist die Abbildung 11.12 aufgeführt, welche die gesamte Test-coverage enthält. Mit Stefan Loosli wurde abgesprochen, dass nicht der gesamte Code getestet werden soll, sondern nur die sinnvollen Stellen. Insgesamt wurden 42 Unit Tests mit Jest, einer Testing Library für JavaScript, verfasst. Die Struktur ist ähnlich zu JUnit Tests in Java. In Abbildung 3.14 ist ein Test von einem Mapper als Beispiel aufgeführt. Die Tests wurden in die drei Teile 'given', 'when' und 'then' unterteilt, wie dies auch in der Adcubum üblich ist.

```

describe('service mappers', () => {
  it('should map from service collection', () => {
    // given
    const serviceCollection: ServiceCollection = [
      entities: [
        SERVICE_ENTITY_1,
        SERVICE_ENTITY_2
      ]
    ];
    const expected: Service[] = [
      SERVICE_1,
      SERVICE_2
    ];

    // when
    const services = mapFromServiceCollection(serviceCollection);

    // then
    expect(services).toEqual(expected);
  });
});

```

Abbildung 3.14: Testing mit Jest

Die Mappers und Sagas wurden nach diesem Muster getestet. Eine Test-Suite wird mit `describe(name, funktion())` beschrieben, wo ein Name und eine Funktion angegeben werden kann, welche ausgeführt werden soll. In dieser Funktion werden dann die einzelnen Tests mit `it(name, funktion())` definiert.

Die Sagas werden etwas anders getestet, da sie asynchrone Prozesse darstellen. Es wird lediglich überprüft, ob die erwarteten `yield` Aufrufe gemacht werden, wobei danach ein Mock-Resultat zurückgegeben werden kann (siehe Abbildung 3.15).

```
it('should send request', () => {
  // given
  const action: SendRequestAction = actions.sendRequest(UUID_1, GRPC_REQUEST)

  // when
  const generator = sagas.sendRequest(action)

  // then
  expect(generator.next().value).toEqual(call(api.sendRequest, GRPC_REQUEST))
  expect(generator.next(API_GRPC_RESPONSE).value).toEqual(put(setResponse(UUID_1, GRPC_RESPONSE)))
  expect(generator.next().done).toBeTruthy()
})
```

Abbildung 3.15: Asynchrones Testing mit Jest

3.4 Release

Die finale Version wurde am Ende der Arbeit als Minor Release 0.1.0 veröffentlicht.

4 Reflexion

Als erstes wurde Scrum als Projektmanagementmethode gewählt. Diese Wahl stellte sich als richtig heraus, da diese Methode sehr gut zum Projekt passt. Die Arbeit kann als in sich abgeschlossene Einheit, ein Sprint, gesehen werden. Zu Beginn können die verschiedenen Anforderungen und Aufgaben definiert werden, das System lässt jedoch trotzdem noch Spielraum für kurzfristige Änderungen. Außerdem war mir diese Methode bereits bekannt, und es konnten die Tools der Adcubum eingesetzt werden. Alternativ hätte auch ein Kanban Board eingesetzt werden können. Ich entschloss mich schlussendlich aber dagegen, da ich damit noch keine Erfahrungen gemacht habe.

Anschliessend wurde der Zeitplan aufgestellt. Zu Beginn des Projekts waren die Arbeiten recht präzise geschätzt worden, mit der Zeit stellte sich aber heraus, dass für einige Aufgaben ein höherer Aufwand geplant als nötig war. Ein Zeitplan entspricht selten der Realität, in diesem Fall war es aber ein sehr nützliches Tool zur Koordination, welche Aufgaben wann gelöst werden sollen und wie viel Zeit für die restlichen Aufgaben bleiben. Die Entscheidung, ein Gantt Diagramm zu verwenden, war sehr gut, denn so konnte unter anderem mit wenig Aufwand ein Soll-Ist Vergleich dargestellt werden.

Die Implementierung konnte sehr schnell abgeschlossen werden. Dies gelang unter anderem dadurch, dass einem Framework und einer UI Library ein grosser Teil der Arbeit abgegeben werden konnte. Ich brauchte mir beispielsweise nur wenige Gedanken über das Aussehen der Applikation machen, und auch die Funktionalität der Tabs und Panels wurde weitestgehend vom Framework bereitgestellt.

Redux brachte einen Mehraufwand mit sich, jedoch hat diese Methode die Eigenschaft, dass sie leicht erweitert werden kann. Für Aussenstehende ist der Code ausserdem schneller zu verstehen, da die Logik und das grafische Design getrennt werden konnten.

Material UI ist nur eine von vielen UI Libraries, jedoch die bekannteste für React. Da keine speziellen Anforderungen an das Aussehen der Applikation gestellt wurden, lag es nahe, diese Library zu verwenden: unter anderem wurde Material UI ausgiebig getestet und hat eine grosse Community hinter sich.

Eine bekannte Alternative zu diesen Technologien wäre zum Beispiel vue.js¹ gewesen. Dieses

¹<https://vuejs.org/> [31.03.2020]

Framework hat ein State Management (ähnlich zu Redux) 'out of the box', und wird oft als performanter als React-Redux beschrieben. Eine andere Möglichkeit wäre gewesen, ganz auf Frameworks zu verzichten und den gesamten Client in plain HTML, CSS und JavaScript zu schreiben.

Die Wahl des Frameworks und der UI Library ist grundsätzlich nebensächlich. Viele der grossen und bekannten Frameworks können und machen fast dasselbe, es kommt hier grösstenteils auf die Präferenzen des Teams an. Je mehr ein Tool verwendet wird, desto unwahrscheinlicher ist es, auf Bugs zu stossen. Aus diesem Grund wurde React-Redux gewählt, da es seit einigen Jahren an der Spitze der beliebtesten Frameworks steht und Facebook dahinter steht. Den Client ohne ein Framework zu implementieren stand ausser Frage, da dies ein viel zu grosser Aufwand für diese kurze Zeit bedeutet hätte.

Die Applikation wurde sehr ausgiebig getestet, sowohl durch Unit Tests als auch durch vordefinierte Testfälle. JUnit ist hier wiederum ein weit verbreitetes Testing Tool, welches ebenfalls aus den bereits genannten Gründen gewählt wurde. Die Applikation ist nicht zu 100% mit Tests abgedeckt, jedoch wurden die wichtigen Stellen berücksichtigt. Dies wurde auch so mit Stefan Loosli und Herrn Kehl abgesprochen, da es wenig sinnvoll ist, mehrere Tage für das Testing aufzuwenden.

5 Fazit

Das Projekt ist sehr gut verlaufen. Der Zeitplan konnte eingehalten oder sogar schneller abgearbeitet werden. Während dem Projekt tauchten keine grossen Probleme auf, was vor allem einer guten Vorarbeit und Planung zu verdanken ist. Die Softwarequalität ist gut, da nach dem Clean Code Prinzip gearbeitet und ein Grossteil der Funktionalität mit Unit Tests abgedeckt wurde.

Das Produkt der Arbeit erfüllt meines Erachtens die Anforderungen, welche an die Applikation gestellt wurden. Nach der Arbeit müssen noch einige kleine Aufgaben, wie zum Beispiel eine höhere Testabdeckung, noch nachgeführt werden. Trotzdem kann die Applikation bereits mit dem jetzigen Stand 'produktiv' eingesetzt werden.

Ich bin sehr zufrieden mit dem Verlauf und dem Resultat der Arbeit. Ich konnte sämtliche Ziele umsetzen und habe den realistisch aufgesetzten Zeitplan einhalten können. Die Applikation wurde zu einem vernünftigen Grad getestet und kann ohne grossen Aufwand erweitert werden.

6 Glossar

Begriff	Beschreibung
Adcubum Template	In der Adcubum stehen Vorlagen für Applikationen zur Verfügung, welche bereits an die interne Infrastruktur angepasst sind. So können beispielsweise relativ einfach Build Pipelines eingerichtet werden, ausserdem ist die App bereits in die Adcubum Security integriert.
Dispatcher (Flux Pattern)	Über den Dispatcher werden Actions von den UI Komponenten abgesetzt. Die Stores reagieren dann auf diese Actions und aktualisieren den State.
Flux Pattern	Das Flux Pattern wurde von Facebook entwickelt und beschreibt eine Architektur für Webclients. Es wird auf einen unidirektionalen Datenfluss gesetzt, wobei jede Applikation einen Dispatcher, einen Store und UI Komponenten hat.
gRPC	Eine Schnittstellentechnologie, welche es ermöglicht, Services einfach und effizient miteinander kommunizieren zu lassen. gRPC ist auch geeignet, um Mobileapplikationen und Websites mit einem Backend Server zu verbinden. Im Vergleich zur REST Schnittstellentechnologie sind die Payloads kleiner und werden schneller übermittelt.
Kubernetes	Kubernetes ist ein System zur Automatisierung der Bereitstellung, Skalierung und Verwaltung von containerisierten Anwendungen. Es gruppiert Container, aus denen sich eine Anwendung zusammensetzt, in logische Einheiten. Dies vereinfacht die Verwaltung und das Deployment von Applikationen.

Begriff	Beschreibung
Linting	Beim Linting wird der Code auf Probleme überprüft. Dies können beispielsweise Formatierungsfehler, Verstöße gegen Code Standards oder mögliche logische Fehler sein. Ein bekannter JavaScript Linter ist ESLint ¹ .
Openshift	Openshift wurde von RedHat entwickelt und ist eine Kubernetes Platform. Hier können beispielsweise Docker Images bereitgestellt werden.
Protobuf / Protocol Buffers	Protocol Buffers sind eine von Google entwickelte Plattformneutrale Art, um Daten zu strukturieren. Dies geschieht ähnlich wie mit XML, aber simpler und schneller. Aus den erstellten Files kann Code in verschiedenen Programmiersprachen generiert werden, welcher für die Weiterarbeit verwendet werden kann.
Reducer (Flux Pattern)	Ein Reducer nimmt eine Action entgegen und gibt einen mutierten State zurück. Dabei wird der ursprüngliche State nicht verändert, sondern ein Klon erstellt, an welchem dann die Änderungen vorgenommen werden. Reducers sind Pure Functions.
REST	Eine Schnittstellentechnologie, welche HTTP-Anfragen verwendet, um per GET, POST, PUT und DELETE auf Daten zuzugreifen.
Server Reflection	Wenn auf einem Server mit einer gRPC Schnittstelle die Server Reflection aktiviert ist, können alle exponierten Services und Methoden ausgelesen werden. Der Client erhält dann beispielsweise Informationen über den Request Body, welcher von einer spezifischen Methode erwartet wird.
Side Effects (Functional Programming)	Im Zusammenhang mit funktionaler Programmierung sind Side Effects die Nebenwirkungen, welche eine Methode bei der Ausführung auslöst. Eine Pure Function sollte keine Nebenwirkungen haben, sondern soll bei gleichen Inputparametern immer dasselbe Resultat liefern.

¹<https://eslint.org/> [31.03.2020]

Begriff	Beschreibung
State (Flux Pattern)	Im Flux Pattern wird unter dem State ein readonly Objekt verstanden, welches den gesamten State der Applikation enthält. Den einzigen Weg, diesen State zu verändern, ist das Objekt zu klonen und die Änderungen anzufügen.
Store (Flux Pattern)	Der Store ist das Herzstück einer Webapplikation, welche mit Redux implementiert wurde. Der Store ist die 'Single Source of Truth', das heisst, dass der gesamte State der Applikation in diesem Objekt abgespeichert und nur von hier gelesen wird.
Syrius / Syrius Infrastruktur	Syrius ist die Versicherungssoftware, welche von der Adcubum entwickelt wird. Die Syrius Infrastruktur kann auf einer Openshift Umgebung installiert werden und stellt die Authentifizierung in der Applikation zur Verfügung.

7 Literaturverzeichnis

In diesem Abschnitt sind Verweise zu sämtlichen verwendeten Quellen aufgeführt.

7.1 Online-Quellen

- Anwendungsfalldiagramm
<https://www.sparxsystems.de/ressourcen/literatur/leseprobe-zu-projektabwicklung-mit-uml-und-enterprise-architect/anwendungsfalldiagramm-use-case-diagram/> [02.04.2020]
- Flux Pattern Dokumentation
<https://facebook.github.io/flux/> [31.03.2020]
- gRPC Dokumentation
<https://grpc.io/> [31.03.2020]
- Kubernetes Dokumentation
<https://kubernetes.io/> [31.03.2020]
- Scrum Dokumentation
<https://www.scrum.org/resources/what-is-scrum> [02.04.2020]
- Sequenzdiagramme
<https://www.lucidchart.com/pages/de/uml-sequenzdiagramme> [02.04.2020]
- Swagger Dokumentation
<https://swagger.io/> [02.04.2020]

8 Abbildungsverzeichnis

1.1	Die Schnittstellen zum Backend Server	6
2.1	Der Zeitplan mit Soll-Ist Vergleich. Die blassen Farben entsprechen den geplanten Aufgaben, die tatsächlich aufgewendete Zeit wurde mit satten Farben nachgetragen. Die Zahlen entsprechen dem Aufwand der einzelnen Tätigkeiten	9
3.1	Das Anwendungsfalldiagramm	13
3.2	Das Sequenzdiagramm. Die verschiedenen Use Cases wurden durch Farben gekennzeichnet: 1 violett, 2 rot, 3 orange, 4 hellgrün, 5 dunkelgrün und 6 blau.	17
3.3	Ein Beispiel eines Reducers	19
3.4	Das Mockup des Web-Clients	21
3.5	Ein Beispiel des States grafisch dargestellt	22
3.6	Die Loginseite	23
3.7	Endpoint wählen	24
3.8	Liste der Services	25
3.9	Schema und Requestbody einer Methode	26
3.10	Die Response eines abgesetzten Requests	27
3.11	Der Fehler, wenn ein invalides JSON bestätigt wird	28
3.12	Der Fehler, wenn der gRPC Server eine Exception wirft	29
3.13	Die Anwenderdokumentation	30
3.14	Testing mit Jest	38
3.15	Asynchrones Testing mit Jest	39
11.1	Der Jira Backlog	51
11.2	Das Jira Board zum gestarteten Sprint. Der Sprint dauert planmäßig bis am 2. April 2020	52
11.3	Das Burn-Down Diagramm	53
11.4	Swagger Dokumentation zum Request <code>GET endpoints</code>	54
11.5	Swagger Dokumentation zum Request <code>POST endpoint</code>	55
11.6	Swagger Dokumentation zum Request <code>DELETE endpoint</code>	56
11.7	Swagger Dokumentation zum Request <code>GET services</code>	57

11.8 Swagger Dokumentation zum Request POST requests	58
11.9 Die Planung der Redux Reducers	59
11.10 Die Planung der Redux Sagas	59
11.11 Die Planung des Redux Store	60
11.12 Die Unit Test Coverage des Stores mit Jest	61

9 Tabellenverzeichnis

3.1	Anwendungsfall Login	14
3.2	Anwendungsfall Endpoints auflisten	14
3.3	Anwendungsfall Endpoint wählen	15
3.4	Anwendungsfall Services und Methoden auflisten	15
3.5	Anwendungsfall Schema anzeigen und bearbeiten	16
3.6	Anwendungsfall Response anzeigen	16
3.7	Testfälle Authentifizierung	32
3.8	Testfälle Authentifizierung	33
3.9	Testfälle Authentifizierung	35

10 Danksagungen

Diese praktische Arbeit wäre nicht möglich gewesen ohne die grosszügige Hilfe von verschiedenen Personen.

Ich möchte mich als Erstes bei meinen Berufsbildnern Stefan Loosli und Javier Diez für die Betreuung vor und während der Arbeit bedanken. Ich konnte immer auf ihre Unterstützung zählen, was ich sehr geschätzt habe.

Des Weiteren bedanke ich mich bei meinen Arbeitskollegen Florian Tanner und Raphael Keller. Sie haben sich stets Zeit genommen, meine offenen Fragen zu klären und meine Pull Requests zu überprüfen und kommentieren. Diese Unterstützung war sehr wertvoll.

Ausserdem danke ich den beiden Experten Thomas Kehl und Ueli Niederer für die Betreuung in der aktuell etwas herausfordernden Lage.

11 Anhang

Nachfolgend sind die Dokumente aufgeführt, welche im Bericht referenziert wurden.

11.1 Projektmanagement

The screenshot shows a Jira backlog board for the project 'IPA Nils Benz S2'. The board has 16 items listed. Each item includes a plus sign icon, a task name, an estimate value, and a user icon. A 'Sprint starten' button and three dots are at the top right. A 'Verknüpfte Seiten anzeigen' button is below the title. A 'Vorgang erstellen' button is at the bottom left. A '...' button is at the bottom center. A '16 Schätzung 71' button is at the bottom right.

Vorgang	Schätzung
APPR-197 Diagramme erstellen	4
APPR-204 Testfälle definieren	2
APPR-203 Redux Store planen	2
APPR-202 Redux Store implementieren	7
APPR-208 Redux Store testen	3
APPR-209 Mockups	2
APPR-191 Endpoint angeben	4
APPR-192 Services und Methoden auflisten	3
APPR-206 Tabmanagement umsetzen	3
APPR-193 Schema und RequestBody	6
APPR-194 Response anzeigen	3
APPR-205 Error handling	4
APPR-195 Anwenderdokumentation	2
APPR-196 Entwickler- und Betreiberdokumentation	2
APPR-190 Testing	8
APPR-207 Dokumentation	16

Abbildung 11.1: Der Jira Backlog

IPA Nils Benz

IPA Nils Benz S2

SCHNELL-FILTER: Nur meine Vorgänge Zuletzt aktualisiert

AUFGABEN	WIRD AUSGEFÜHRT	FERTIG
<p>APPR-197 OFFEN 2 Unteraufgaben Diagramme erstellen</p> <p>APPR-210 Use Case Diagramm Use Case Diagramm </p> <p>APPR-211 Sequenzdiagramm Sequenzdiagramm </p>	 Sprint abschließen	
<p>APPR-204 OFFEN 1 Unteraufgabe Testfälle definieren</p> <p>APPR-212 Do it </p>		
<p>APPR-203 OFFEN 1 Unteraufgabe Redux Store planen</p> <p>APPR-213 Do it </p>		

Andere Vorgänge 13

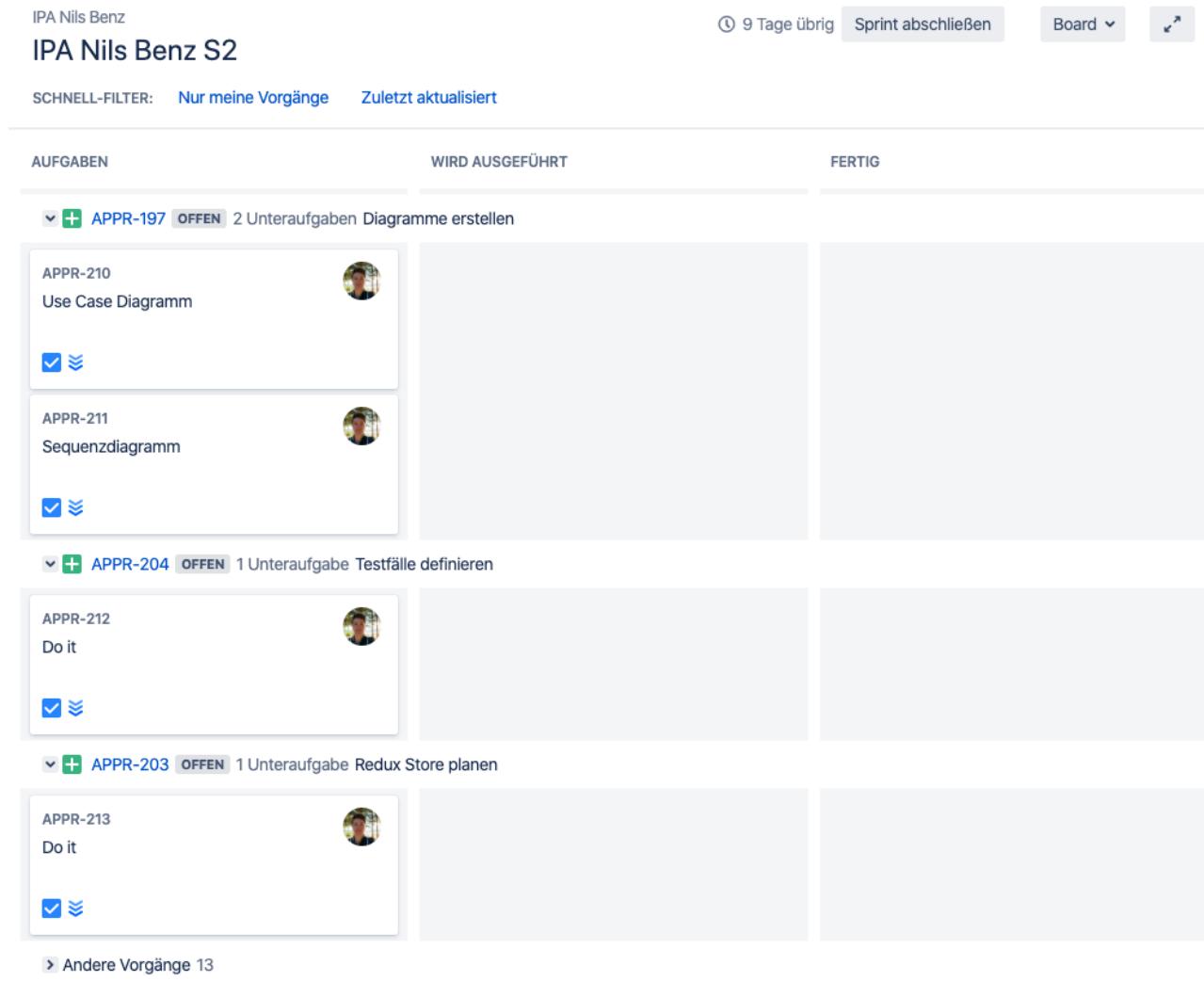


Abbildung 11.2: Das Jira Board zum gestarteten Sprint. Der Sprint dauert planmäßig bis am 2. April 2020

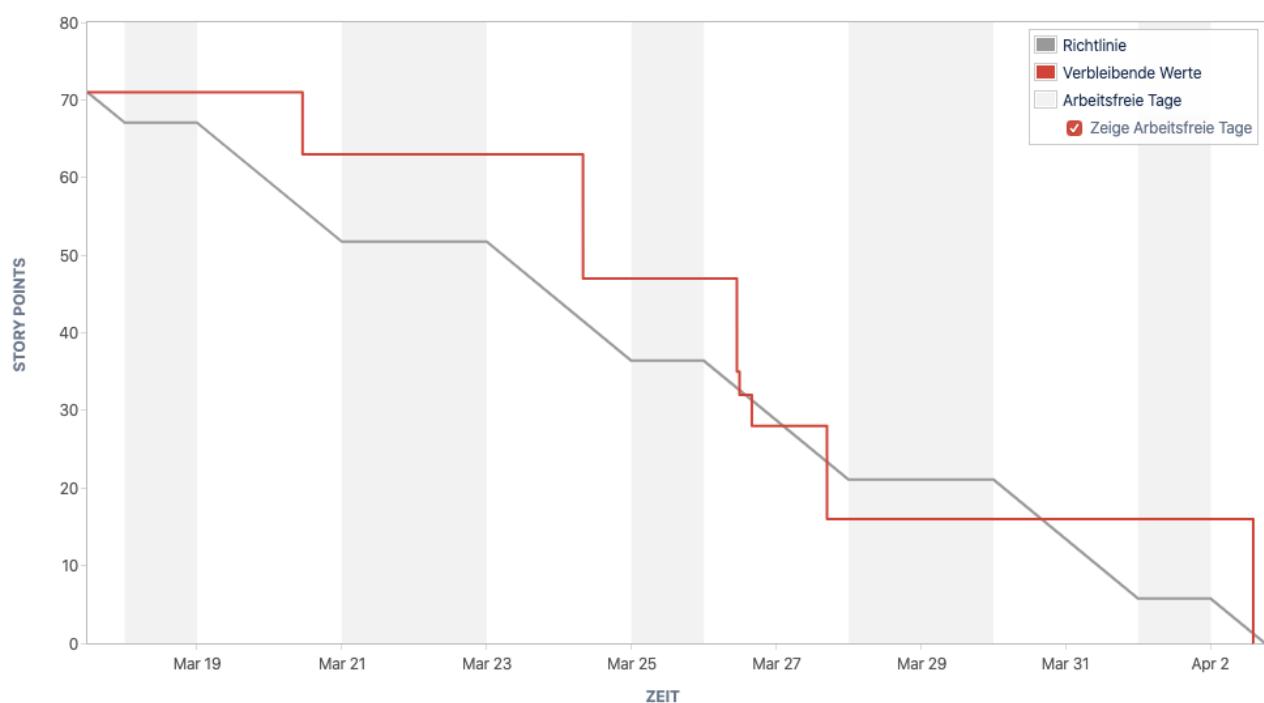


Abbildung 11.3: Das Burn-Down Diagramm

11.2 Schnittstellen

The screenshot shows a Swagger UI interface for a gRPC API. At the top, a blue header bar indicates the method **GET** and the endpoint **/api/endpoints**. The description states: "List all available gRPC Endpoints in the Kubernetes Cluster". Below this, a "Parameters" section notes "No parameters". On the right, a "Try it out" button is visible. In the middle section, a "Responses" table is shown with one row for status code 200. The "Code" column contains "200" and the "Description" column contains "A successful response.". Under the description, there are links for "Example", "Value", and "Model". The "Value" link is expanded, showing a JSON object:

```
{  
  "entities": [  
    {  
      "data": {  
        "display_name": "string"  
      }  
    }  
  ]  
}
```

Abbildung 11.4: Swagger Dokumentation zum Request `GET endpoints`

POST /api/endpoints Connects to a new Endpoint

Parameters

Name **Description**

body * required Example Value | Model
object
(body)

```
{  
    "display_name": "string"  
}
```

Parameter content type
application/json ▾

Responses

Code **Description**

200 A successful response.
Example Value | Model

```
{  
    "data": {  
        "display_name": "string"  
    }  
}
```

Response content type application/json ▾

Abbildung 11.5: Swagger Dokumentation zum Request POST endpoint

The screenshot shows a Swagger API documentation interface for a `DELETE /api/endpoints/{endpoint}` endpoint. The endpoint is described as "Disconnect from an Endpoint".

Parameters:

Name	Description
endpoint * required string (path)	Endpoint URL endpoint - Endpoint URL

Responses:

Code	Description
200	A successful response. Example Value Model {}

Response content type: application/json

Abbildung 11.6: Swagger Dokumentation zum Request `DELETE endpoint`

GET /api/endpoints/{endpoint}/services List all gRPC Services of a given endpoint

Parameters [Try it out](#)

Name	Description
endpoint * required string (path)	Endpoint URL endpoint - Endpoint URL

Responses Response content type application/json ▾

Code	Description
200	A successful response. Example Value Model

```
{  
  "entities": [  
    {  
      "data": {  
        "display_name": "string",  
        "grpcmethods": {  
          "entities": [  
            {  
              "data": {  
                "display_name": "string",  
                "request_body": "string",  
                "schema": "string"  
              }  
            }  
          ]  
        }  
      }  
    }  
  ]  
}
```

Abbildung 11.7: Swagger Dokumentation zum Request GET services

POST /api/endpoints/{endpoint}/services/{service}/methods/{method}/requests Creates a new GrpcRequest

Parameters

Name **Description**

endpoint * required
string
(path) endpoint - Endpoint URL

service * required
string
(path) service - Service name

method * required
string
(path) method - Method name

body * required
object
(body) Example Value | Model

```
{  
    "request_body": "string"  
}
```

Parameter content type
application/json

Responses

Code **Description**

200 A successful response.

Example Value | Model

```
{  
    "response_body": "string",  
    "response_time": 0  
}
```

Abbildung 11.8: Swagger Dokumentation zum Request POST requests

11.3 Redux

```
● ● ●

EndpointReducers
- setEndpoints()
- setCurrentEndpoint()
- setEndpointInput()
- setConnected()

ServiceReducers
- setServices()
- setLoadingServices()

TabReducers
- setCurrentTab()
- createTab()
- closeTab()
- setResponse()
- setOpenPanel()
- setLoadingTab()
- setRequestBody()
- setUuid()

MessageReducers
- setMessage()
- setOpen()
```

Abbildung 11.9: Die Planung der Redux Reducers

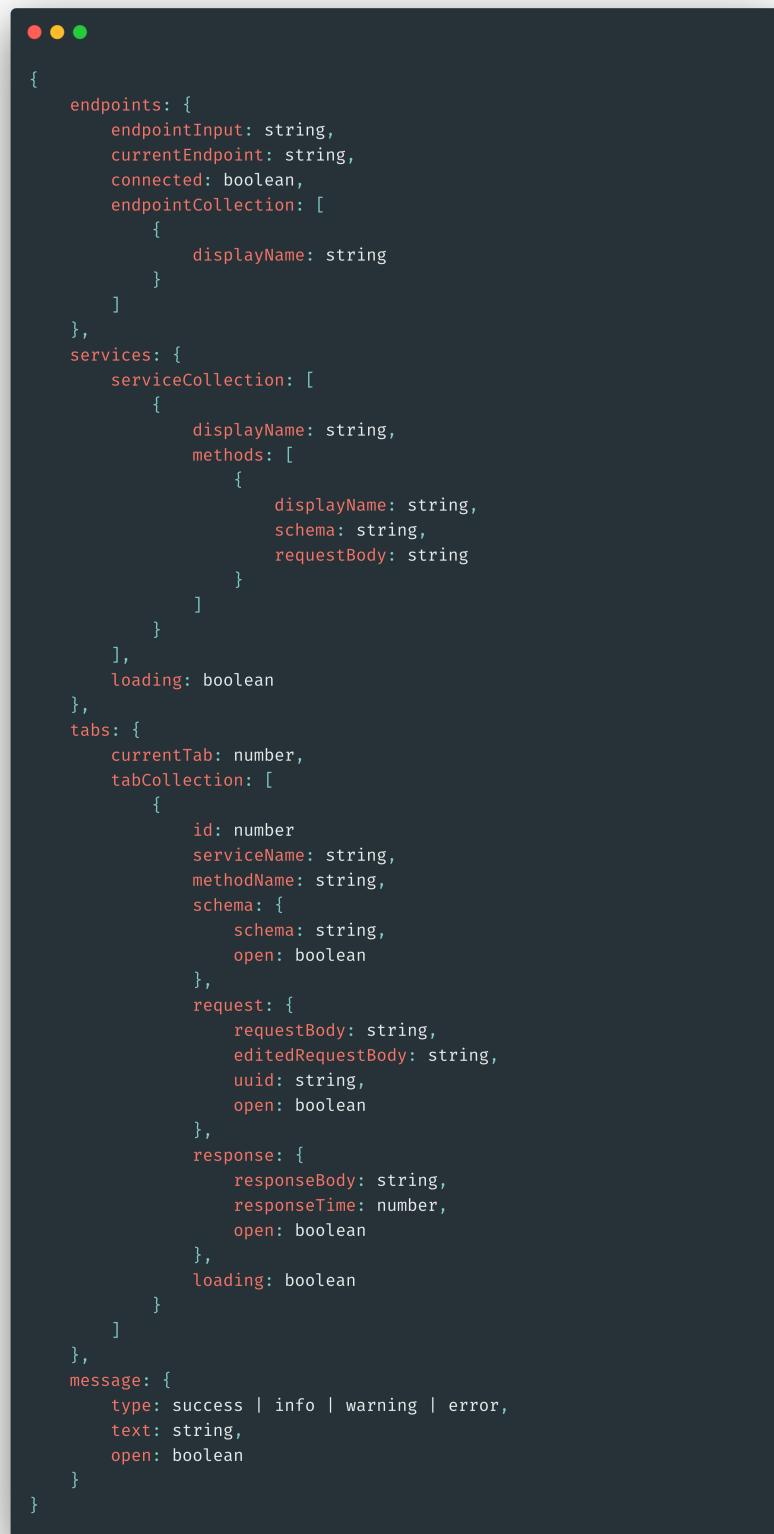
```
● ● ●

EndpointSagas
- listEndpoints()

ServiceSagas
- listServices()
- sendRequest()

TabSagas
- openTab()
- generateUuid()
```

Abbildung 11.10: Die Planung der Redux Sagas



```
{  
  endpoints: {  
    endpointInput: string,  
    currentEndpoint: string,  
    connected: boolean,  
    endpointCollection: [  
      {  
        displayName: string  
      }  
    ]  
  },  
  services: {  
    serviceCollection: [  
      {  
        displayName: string,  
        methods: [  
          {  
            displayName: string,  
            schema: string,  
            requestBody: string  
          }  
        ]  
      }  
    ],  
    loading: boolean  
  },  
  tabs: {  
    currentTab: number,  
    tabCollection: [  
      {  
        id: number  
        serviceName: string,  
        methodName: string,  
        schema: {  
          schema: string,  
          open: boolean  
        },  
        request: {  
          requestBody: string,  
          editedRequestBody: string,  
          uuid: string,  
          open: boolean  
        },  
        response: {  
          responseBody: string,  
          responseTime: number,  
          open: boolean  
        },  
        loading: boolean  
      }  
    ]  
  },  
  message: {  
    type: success | info | warning | error,  
    text: string,  
    open: boolean  
  }  
}
```

Abbildung 11.11: Die Planung des Redux Store

11.4 Testing

File	%Stmts	%Branch	%Funcs	%Lines	Uncovered Line #s
All files	71.13	18.03	64.08	70.5	
src	0	0	0	0	
index.tsx	0	100	100	0	13,15,22,24,34
serviceWorker.ts	0	0	0	0	... 40,141,143,146
src/api	33.33	0	0	33.33	
endpoints.ts	33.33	0	0	33.33	4,5
services.ts	33.33	0	0	33.33	7,8,12,18
src/components/app	0	100	0	0	
App.tsx	0	100	0	0	... 25,28,34,46,50
src/store	0	100	0	0	
index.ts	0	100	100	0	7
sagas.ts	0	100	0	0	8
src/store/endpoints	93.18	100	82.35	92.68	
actions.ts	80	100	80	80	16
mappers.ts	100	100	100	100	
reducers.ts	100	100	100	100	
sagas.ts	66.67	100	50	66.67	14
selectors.ts	90	100	66.67	87.5	14
testdata.ts	100	100	100	100	
types.ts	100	100	100	100	
src/store/grpcrequests	100	100	100	100	
mappers.ts	100	100	100	100	
testdata.ts	100	100	100	100	
types.ts	0	0	0	0	
src/store/grpcresponses	100	100	100	100	
mappers.ts	100	100	100	100	
testdata.ts	100	100	100	100	
types.ts	0	0	0	0	
src/store/messages	100	100	100	100	
actions.ts	100	100	100	100	
reducers.ts	100	100	100	100	
testdata.ts	100	100	100	100	
types.ts	100	100	100	100	
src/store/methods	100	100	100	100	
mappers.ts	100	100	100	100	
testdata.ts	100	100	100	100	
types.ts	0	0	0	0	
src/store/services	76.19	0	75	75.61	
actions.ts	100	100	100	100	
mappers.ts	100	100	100	100	
reducers.ts	100	100	100	100	
sagas.ts	66.67	100	66.67	66.67	28,29
selectors.ts	20	0	25	20	... ,9,10,11,12,15
testdata.ts	100	100	100	100	
types.ts	100	100	100	100	
src/store/tabs	89.58	44.44	90.63	89.89	
actions.ts	100	100	100	100	
reducers.ts	88.89	50	100	87.5	48,58,68,81,91
sagas.ts	81.25	75	80	80	60,61,62
selectors.ts	50	0	33.33	66.67	6
testdata.ts	100	100	100	100	
types.ts	100	100	100	100	
src/store/utility	100	100	100	100	
reducer.ts	100	100	100	100	

Abbildung 11.12: Die Unit Test Coverage des Stores mit Jest

11.5 Arbeitsjournal

IPA Nils Benz

17.03.2020

Arbeitsjournal 17.03.2020

Welche Tätigkeiten hatte ich für heute geplant?

#	Soll-Zeit	Beschreibung
1	3h	Zeitplan erstellen und dokumentieren
2	2h	Projektmanagementmethode wählen und beschreiben
3	2h	Projektabgrenzung und -umfeld beschreiben
4	1h	Wissensbeschaffung beschreiben

Was habe ich erreicht?

Arbeitszeit: 8 Stunden

#	Ist-Zeit	Beschreibung
1	2h	Zeitplan mit Lucidchart erstellt
1	1h	Zeitplan dokumentiert
		Daily
2	1h	Projektmanagementmethode Scrum gewählt und entsprechend beschrieben
2	0.5h	Scrum Backlog aufgesetzt und Sprint gestartet
3	2h	Projektabgrenzung und -umfeld dokumentiert. Die Schnittstellen habe ich mit Swagger dokumentiert.
4	1h	Wissensbeschaffung beschrieben
	0.5h	Einleitung schreiben begonnen

Legende

- grün: weniger Zeit benötigt als geplant
- gelb: zusätzliche Aufgaben, welche nicht im Zeitplan waren
- rot: mehr Zeit benötigt als geplant

Soll/Ist-Vergleich

Total geplante Zeit: 8 Stunden	Total gearbeitete Zeit: 8 Stunden
--------------------------------	-----------------------------------

IPA Nils Benz

17.03.2020

Für die meisten Aufgaben habe ich in etwa so viel Zeit beansprucht, wie ich dies geplant habe. Den Scrum Backlog erstellte ich sogar eine halbe Stunde schneller. Aus diesem Grund konnte ich bereits mit der Einleitung beginnen. Die Dokumentation ist nun auf dem Stand wie ich ihn mir vorgenommen habe. Er wird dann am letzten Tag nochmals überarbeitet werden. Ausserdem habe ich mit Stefan Loosli ein Daily durchgeführt, bei welchem wir meinen zu diesem Zeitpunkt bereits erstellten Zeitplan besprochen haben.

Wobei hatte ich Probleme? Habe ich Hilfestellungen beansprucht?

Die Schnittstellendokumentation wollte ich zuerst manuell erstellen, bis mir eingefallen ist, dass aus der Spezifikation ein Yaml generiert wird. Ich habe bei Veith Zäch nachgefragt, einem Entwickler der in diesem Bereich tätig ist.

Er hat mir gesagt, wo dass diese Yaml Files beim Build abgelegt werden und dass ich diese in den Swagger Editor (<https://editor.swagger.io/>) importieren kann. Anschliessend musste ich nur noch einige kleine Änderungen vornehmen.

Reflexion

Ich konnte heute alle Aufgaben vom Tagesplan erledigen. Ich hatte zusätzlich noch Zeit, um mit der Einleitung zu beginnen. Dies werde ich immer dann machen, wenn ich alle Aufgaben erledigt habe und es sich nicht mehr lohnt, eine neue Aufgabe vom nächsten Tag zu beginnen.

Ich bin zuversichtlich, dass der Zeitplan auch am zweiten Tag stimmt und werde mir dann wie geplant die Aufgaben "Diagramme erstellen", "Testfälle definieren" und "Redux Store planen" vornehmen.

Visum vorgesetzte Fachkraft



IPA Nils Benz

19.03.2020

Arbeitsjournal 19.03.2020

Welche Tätigkeiten hatte ich für heute geplant?

#	Soll-Zeit	Beschreibung
5	2h	Usecasediagramm erstellen und dokumentieren
6	2h	Testfälle definieren
7	2h	Sequenzdiagramm erstellen und dokumentieren
8	2h	Redux Store planen

Was habe ich erreicht?

Arbeitszeit: 8 Stunden

#	Ist-Zeit	Beschreibung
5	2h	Usecasediagramm erstellt und dokumentiert
6	2h	Testfälle definiert
		Daily
7	2h	Sequenzdiagramm erstellt und dokumentiert
8	1.5h	Redux Store geplant
	0.5h	Bisherige Dokumentation verbessern

Legende

- grün: weniger Zeit benötigt als geplant
- gelb: zusätzliche Aufgaben, welche nicht im Zeitplan waren
- rot: mehr Zeit benötigt als geplant

Soll/Ist-Vergleich

Total geplante Zeit: 16 Stunden	Total gearbeitete Zeit: 16 Stunden
---------------------------------	------------------------------------

Ich konnte wiederum alle Aufgaben vom Zeitplan erledigen. Mit der Definition der Testfälle war ich länger beschäftigt als ich ursprünglich geplant habe, jedoch konnte ich diese Zeit mit der Planung des Redux Store mehr als wieder gut machen. Da ich am Nachmittag noch Zeit

IPA Nils Benz

19.03.2020

hatte, konnte ich ausserdem Teile der Dokumentation verbessern, welche ich am ersten Tag geschrieben habe.

Wobei hatte ich Probleme? Habe ich Hilfestellungen beansprucht?

Heute bin ich nicht auf Probleme gestossen, da die Arbeiten zu einem grossen Teil Dokumentationsarbeiten waren. Ich musste im Internet nach Definitionen für Anwendungsdiagramme¹ und Sequenzdiagramme² suchen, da ich nicht mehr alle Konventionen im Kopf hatte und ich mich absichern wollte. Ich habe die Diagramme erneut mit Lucidchart erstellt. Ich bin heute auf die Limitationen dieses Tools gestossen, da in der Testversion nur drei Projekte pro Benutzer und eine gewisse Anzahl Elemente pro Projekt zugelassen sind. Diese Quotas bereiteten mir jedoch keine Probleme.

Reflexion

Der Zeitplan war erneut recht akkurat. Ich konnte alle Aufgaben, welche für heute geplant waren, erledigen, wenn auch mit leichten Zeitverschiebungen. Ich habe bei der Planung jedoch weniger Wert darauf gelegt, jede Minute richtig zu verplanen, sondern meinen Fokus auf die verschiedenen Tage gelenkt, sodass der Arbeitsaufwand jeden Tag machbar ist.

Morgen werden ich den Redux Store anhand der heutigen Planung implementieren und beginnen zu testen. Ich bin überzeugt, diese Aufgaben in einem Tag erledigen zu können.

Visum vorgesetzte Fachkraft



¹ <https://www.spektrum.de/sixcms/media.php/370/leseprobe.384438.pdf> [19.03.2020]

² <https://www.lucidchart.com/pages/de/uml-sequenzdiagramme> [19.03.2020]

IPA Nils Benz

20.03.2020

Arbeitsjournal 20.03.2020

Welche Tätigkeiten hatte ich für heute geplant?

#	Soll-Zeit	Beschreibung
9	2h	Service State implementieren
10	3h	Tab State implementieren
11	2h	Message State implementieren
12	1h	State testen

Was habe ich erreicht?

Arbeitszeit: 8 Stunden

#	Ist-Zeit	Beschreibung
9	2h	Service State implementieren
	(1h)	Meeting mit Herrn Kehl
		Daily
10	3h	Tab State implementiert
11	1.5h	Message State implementiert
12	1.5h	State getestet

Legende

- grün: weniger Zeit benötigt als geplant
- gelb: zusätzliche Aufgaben, welche nicht im Zeitplan waren
- rot: mehr Zeit benötigt als geplant

Soll/Ist-Vergleich

Total geplante Zeit: 24 Stunden	Total gearbeitete Zeit: 24 Stunden
---------------------------------	------------------------------------

Ich bin heute wieder gut voran gekommen. Ich habe die Planung insofern nicht eingehalten, dass ich die Tests nicht nach der Implementation geschrieben habe, sondern zeitgleich. Diese Änderung habe ich vorgenommen, da der zeitliche Aufwand so verringert wurde. Aus diesem Grund konnte ich die Implementation des Tab States noch nicht vollständig abschliessen. Ich denke aber, dass ich dies am Montag in 1-2 Stunden erledigen kann, was

IPA Nils Benz

20.03.2020

nach wie vor im Zeitplan liegt. Ich habe nämlich für den nächsten Tag nochmals zwei Stunden für das Testing eingeplant, welches ich heute bereits zu grossen Teilen erledigt habe.

Wobei hatte ich Probleme? Habe ich Hilfestellungen beansprucht?

Heute hat uns Herr Kehl einen Besuch abgestattet. Wir haben zusammen meine bisherige Arbeit angeschaut und er hat mir einige Tipps gegeben:

- Der Soll - Ist Zeitplan kann im selben Diagramm dargestellt werden. So kann einfach Schlüsse über das Zeitmanagement getroffen werden.
- Die Aufgaben im Zeitplan sollten eindeutig nummeriert werden, damit ich mich beispielsweise im Arbeitsjournal darauf beziehen kann.

Ansonsten konnte ich den gesamten Tag selbstständig und ohne grosse Probleme arbeiten. Einzig mit parametrierten Tests musste ich im Internet nachschauen, habe jedoch recht schnell eine gute Quelle¹ gefunden, welche mein Problem lösen konnte. Am Daily haben wir wieder meinen Fortschritt analysiert.

Reflexion

Ich konnte wieder alle geplanten Aufgaben erledigen bzw. dem Zeitmanagement entsprechend verschieben. Am nächsten Tag werden ich von Zuhause aus arbeiten können, da dies aufgrund der aktuellen Lage mit dem COVID-19 vom Kanton so bestimmt wurde. Dies wird für mich kein grosses Problem darstellen, da ich über das VPN auf die interne Infrastruktur zugreifen und meinen Bildschirm mit nach Hause nehmen kann.

Ich bin weiterhin zuversichtlich, dass ich den Zeitplan einhalten kann. Am Montag werde ich die GUI designen und beginnen, diese zu implementieren.

Visum vorgesetzte Fachkraft



¹ <https://itnext.io/reduce-unit-tests-boilerplate-with-jests-each-syntax-f5e48828437f>

IPA Nils Benz

23.03.2020

Arbeitsjournal 23.03.2020

Welche Tätigkeiten hatte ich für heute geplant?

#	Soll-Zeit	Beschreibung
12	2h	Store Testen
13	1h	Testing dokumentieren
14	2h	Mockups fertigstellen und dokumentieren
15	3h	Endpoint angeben in der GUI

Was habe ich erreicht?

Arbeitszeit: 8.5 Stunden

#	Ist-Zeit	Beschreibung
12	1h	Store Testen
13	1.5h	Testing dokumentieren
14	1.5h	Mockups fertigstellen und dokumentieren
		Daily
15	2h	Endpoint angeben in der GUI
16	2h	gRPC Services laden und anzeigen

Legende

- grün: weniger Zeit benötigt als geplant
- gelb: zusätzliche Aufgaben, welche nicht im Zeitplan waren
- rot: mehr Zeit benötigt als geplant

Soll/Ist-Vergleich

Total geplante Zeit: 32 Stunden	Total gearbeitete Zeit: 32.5 Stunden
---------------------------------	--------------------------------------

Heute bin ich schneller als geplant mit meinen Aufgaben fertig geworden. Da ich am Freitag bereits fast fertig war mit dem Testing des Stores habe ich dafür nur noch wenig Zeit benötigt. Ich nahm mir dafür etwas mehr Zeit für die Erweiterung der Dokumentation. Auch

IPA Nils Benz

23.03.2020

für die Mockups und das angeben der Endpoints in der GUI habe ich zu viel Zeit eingeplant.
Dies, weil ich den Aufwand dafür überschätzt habe.

Da ich anschliessend noch Zeit gehabt habe, machte ich bei der nächsten Aufgabe weiter,
dem laden und anzeigen der gRPC Services. Diese Aufgabe habe ich ebenfalls bereits
abschliessen können, warte momentan jedoch noch auf die Genehmigung des Pull
Requests.

Total habe ich zum heutigen Stand eine halbe Stunde mehr gearbeitet als geplant, dies weil
ich heute die Aufgabe 16 noch abschliessen wollte.

Wobei hatte ich Probleme? Habe ich Hilfestellungen beansprucht?

Heute stiess ich erneut auf wenige Probleme. Florian Tanner hat mir Feedback zum Testing
gegeben, welches ich dann Umgesetzt habe. Es ging um das Mocken von Sagas.

Ich war häufig in der Dokumentation von MaterialUI¹ unterwegs, da ich diese noch nicht so
gut kenne. Die Doku ist sehr verständlich und gut nachvollziehbar aufgebaut, weshalb ich
hiermit keine Probleme hatte.

Reflexion

Ich habe heute wieder alle Punkte, welche ich mir vorgenommen habe, abschliessen
können. Ich denke, dass mein Vorgehen richtig war und ich werde Morgen in diesem Stil
weiterarbeiten, wenn ich das Tabmanagement umsetze und das Schema anzeigen lasse.

Der Plan verschiebt sich um ca. einen halben Tag, ich denke jedoch dass wir die
Massnahmen (falls nötig) im Daily von Morgen besprechen werden. Ich denke, dass es
sinnvoll ist wenn ich am Plan festhalte und die Aufgabe in der geplanten Reihenfolge
ab arbeite. Dann habe ich am Schluss eventuell noch Zeit, einige zusätzliche Features zu
implementieren oder noch mehr ins Testing zu investieren.

Visum vorgesetzte Fachkraft



¹ <https://material-ui.com/>

IPA Nils Benz

24.03.2020

Arbeitsjournal 24.03.2020

Welche Tätigkeiten hatte ich für heute geplant?

#	Soll-Zeit	Beschreibung
17	3h	Tabmanagement umsetzen
18	2h	Schema anzeigen

Was habe ich erreicht?

Arbeitszeit: 8.5 Stunden

#	Ist-Zeit	Beschreibung
17	3.5h	Tabmanagement umgesetzt
		Daily
18	1.5h	Schema angezeigt
19	3.5h	Request Body angezeigt und bearbeitet

Legende

- grün: weniger Zeit benötigt als geplant
- gelb: zusätzliche Aufgaben, welche nicht im Zeitplan waren
- rot: mehr Zeit benötigt als geplant

Soll/Ist-Vergleich

Total geplante Zeit: 40 Stunden	Total gearbeitete Zeit: 41 Stunden
---------------------------------	------------------------------------

Heute habe ich für die Aufgaben in etwa so lange gebraucht wie geplant. Da ich gestern bereits ungefähr einen halben Tag vor dem Zeitplan war, habe ich auch heute bereits die Aufgaben vom nächsten Morgen erledigt. Ich habe wieder eine halbe Stunde länger gearbeitet als geplant, was daran gelegen hat, dass ich die Aufgabe 19 noch heute erledigen wollte.

Total bin ich nun eine halbe Stunde über dem geplanten Aufwand, jedoch bin ich wie bereits erwähnt einen halben Tag vor dem Zeitplan und ich habe 80 Stunden geplant, weshalb ich noch 10 "Überstunden" machen darf.

IPA Nils Benz

24.03.2020

Wobei hatte ich Probleme? Habe ich Hilfestellungen beansprucht?

Ich habe heute wenig Hilfe beansprucht. Ich war wie gestern oft auf der Dokumentation von MaterialUI unterwegs, und habe auch sonst einige Antworten auf StackOverflow gesucht. Insgesamt wurde ich nur von einem Problem mit dem Layout aufgehalten, welches ich nun für einen normalen Bildschirm beheben konnte. Soll die App auch in einem Fenster von weniger als 1200 Pixel Breite benutzt werden können, kann dies in einem späteren Schritt nach der IPA erfolgen.

Florian Tanner hat erneut meine Pull Requests kommentiert, und ich konnte noch einige Kleine Sachen verbessern, beispielsweise dass ich drei Conditions, welche ich zu Beginn in einer Condition hatte, aufgeteilt habe.

Reflexion

Ich bin meiner Meinung nach auch heute wieder richtig vorgegangen und werde am nächsten Tag vielleicht bereits die Implementierung abschliessen können. Die Aufgabe 19 werde ich am nächsten Morgen mergen und dann an der Aufgabe 20, Request absetzen und Response anzeigen, arbeiten.

Am Daily wurde besprochen, dass ich die Aufgaben gemäss Terminplan abarbeite, damit ich am Schluss mehr Zeit für die Dokumentation zur Verfügung habe.

Visum vorgesetzte Fachkraft



IPA Nils Benz

26.03.2020

Arbeitsjournal 26.03.2020

Welche Tätigkeiten hatte ich für heute geplant?

#	Soll-Zeit	Beschreibung
20	4h	Request abschicken und Response anzeigen
21	3h	Error handling
22	1h	Anwenderdokumentation erstellen

Was habe ich erreicht?

Arbeitszeit: 8 Stunden

#	Ist-Zeit	Beschreibung
20	3h	Request abschicken und Response anzeigen
21	2h	Error handling
		Daily
22	2h	Anwenderdokumentation erstellen
23	1h	Entwicklerdokumentation erstellen

Legende

- grün: weniger Zeit benötigt als geplant
- gelb: zusätzliche Aufgaben, welche nicht im Zeitplan waren
- rot: mehr Zeit benötigt als geplant

Soll/Ist-Vergleich

Total geplante Zeit: 48 Stunden	Total gearbeitete Zeit: 49 Stunden
---------------------------------	------------------------------------

Ich habe auch heute wieder mehr Zeit eingeplant, als dass ich schlussendlich gebraucht habe. Ich konnte alle Aufgaben, welche ich mir heute vorgenommen habe, erledigen, und sogar eine Stunde länger an der Anwenderdokumentation arbeiten. Ausserdem habe ich bereits die Aufgabe 23 erledigen können, welche eigentlich für morgen eingeplant war. Insgesamt bin ich wie gestern bereits beschrieben eine Stunde über der Planung.

Ich konnte auch mein Problem von gestern mit dem Layout lösen, eigentlich war es gar nicht so schwierig. Ich habe anscheinend nur ein bisschen Abstand und neue Gedanken benötigt.

IPA Nils Benz

26.03.2020

Wobei hatte ich Probleme? Habe ich Hilfestellungen beansprucht?

Ich hatte einige Probleme bei der Implementierung der Fehlermeldungsanzeige. An sich beschreibt die Dokumentation von MaterialUI diese Komponente sehr ausführlich, ich hatte jedoch eine leicht andere Datenstruktur und musste somit selber eine Änderungen vornehmen. Nach einigem Ausprobieren habe ich dieses Problem ohne weitere Hilfestellungen bewältigen können, sogar eine Stunde schneller als geplant.

Ansonsten bin ich im Laufe des Tages auf keine nennenswerten Probleme gestossen.

Reflexion

Ich war heute wieder schneller mit den Aufgaben fertig als geplant. Bei der Planung war es schwierig einzuschätzen, welchen Zeitaufwand ich für die UI Library einplanen musste. Auch der Aufwand für die Einbindung des Texteditors war schwierig zu schätzen.

Am Daily haben wir besprochen, dass ich weiterhin die Aufgaben abarbeite und am Schluss, falls ich dann noch Zeit habe, einige kleine Features wie etwa die Sortierung der Services und Methoden und Tastaturbefehle implementieren werde.

Den Meilenstein, dass die Implementierung der Applikation soweit abgeschlossen ist, habe ich erreicht. Morgen wird der Schwerpunkt meiner Arbeit auf dem Testing der Applikation und der Behebung der gefundenen Bugs liegen. Ausserdem werde ich wahrscheinlich Zeit haben, um in der Dokumentation weiterzuarbeiten.

Visum vorgesetzte Fachkraft



IPA Nils Benz

27.03.2020

Arbeitsjournal 27.03.2020

Welche Tätigkeiten hatte ich für heute geplant?

#	Soll-Zeit	Beschreibung
24	1h	Betreiberdokumentation erstellen
25	2h	Testfälle prüfen
26	5h	Bugs fixen

Was habe ich erreicht?

Arbeitszeit: 8 Stunden

#	Ist-Zeit	Beschreibung
24	1h	Betreiberdokumentation erstellen
	(0.5h)	Zweites Meeting mit Herrn Kehl
		Daily
25	2h	Testfälle prüfen
26	4h	Bugs fixen
	1h	An Dokumentation weiterarbeiten

Legende

- grün: weniger Zeit benötigt als geplant
- gelb: zusätzliche Aufgaben, welche nicht im Zeitplan waren
- rot: mehr Zeit benötigt als geplant

Soll/Ist-Vergleich

Total geplante Zeit: 56 Stunden	Total gearbeitete Zeit: 57 Stunden
---------------------------------	------------------------------------

Ich konnte auch heute wieder alle Aufgaben erledigen. Da ich für das Refactoring eine Stunde weniger Zeit gebraucht habe als geplant war, konnte ich noch eine Stunde für die Arbeit an der Dokumentation verwenden. Dort habe ich den Abschnitt Testing fertig gestellt.

Total bin ich wie die letzten beiden Tage bereits eine Stunde über der geplanten Zeit.

IPA Nils Benz

27.03.2020

Wobei hatte ich Probleme? Habe ich Hilfestellungen beansprucht?

Ich war mir nicht sicher, welche Hostnames für die gRPC Server valide sind. Aus diesem Grund habe ich Florian Tanner danach gefragt, ob er mir hier weiterhelfen kann. Tatsächlich konnte er mir sagen, was ich validieren musste, und nach kurzer Suche im Internet wurde ich fündig. Ich konnte somit die RegEx `^a-zA-Z0-9-]+:[0-9]{1,5}$` erstellen.

Ansonsten bin ich wieder auf keine nennenswerten Probleme gestossen.

Reflexion

Das Gespräch mit Herrn Kehl verlief sehr gut, und wir haben uns auf kein weiteres Meeting geeinigt. Falls Fragen auftauchen werde ich mich bei ihm melden.

Ich habe heute die meiste Zeit Refactoring betrieben. Dazu habe ich auch die Meinung von Javier Diez und Stefan Loosli eingeholt, und Florian Tanner hat meine Pull Requests visiert. Ich denke, dass ich richtig vorgegangen bin, indem ich weitgehend selbstständig gearbeitet und bei Fragen auf meine Arbeitskollegen zugegangen bin.

Am Montag liegt der Schwerpunkt auf der Dokumentation.

Visum vorgesetzte Fachkraft



IPA Nils Benz

30.03.2020

Arbeitsjournal 30.03.2020

Welche Tätigkeiten hatte ich für heute geplant?

#	Soll-Zeit	Beschreibung
27	2h	Soll - Ist Zeitplan erstellen und dokumentieren
	4h	Implementation dokumentieren
28	2h	Reflexion schreiben

Was habe ich erreicht?

Arbeitszeit: 8 Stunden

#	Ist-Zeit	Beschreibung
27	2h	Soll - Ist Zeitplan erstellen und dokumentieren
		Daily
	4h	Implementation dokumentieren
	0.5h	Bugs fixen
28	1.5h	Reflexion schreiben

Legende

- grün: weniger Zeit benötigt als geplant
- gelb: zusätzliche Aufgaben, welche nicht im Zeitplan waren
- rot: mehr Zeit benötigt als geplant

Soll/Ist-Vergleich

Total geplante Zeit: 64 Stunden	Total gearbeitete Zeit: 65 Stunden
---------------------------------	------------------------------------

Heute habe ich kurzfristig noch eine weitere Aufgabe in den Tagesplan genommen. Ich habe die Implementierung der App noch nicht dokumentiert, jedoch war es sinnvoll, dies vor der Reflexion und dem Fazit zu erledigen. Aus diesem Grund bin ich nun nur noch einen halben Tag vor dem Zeitplan, insgesamt jedoch nach wie vor eine Stunde über der gesamten geplanten Zeit. Ich konnte alle Aufgaben, welche ich mir für heute vorgenommen habe, erledigen und hatte sogar noch kurz Zeit, um einen kleinen Bug zu fixen.

IPA Nils Benz

30.03.2020

Wobei hatte ich Probleme? Habe ich Hilfestellungen beansprucht?

Javier Diez hat mich darauf hingewiesen, dass in der Installationsanleitung noch ein kleiner Fehler war. Diesen habe ich aber schnell gefunden und konnte ihn verbessern. Stefan Loosli hat mich beim Daily auf einen Bug im Client hingewiesen, nämlich dass jedes Mal die UUID kopiert wird, wenn ein neuer Tab geöffnet wird. Ich nahm mir auch für diesen fehler kurz Zeit und fand schnell eine Lösung, da ich bereits ahnte, wo das Problem sein könnte.

Ansonsten habe ich heute die meiste Zeit dokumentiert, wobei ich manchmal etwas über die verwendeten Technologien nachgelesen habe. Ich bin auf keine Probleme gestossen, welche ich nicht selber habe lösen können.

Reflexion

Heute habe ich vorwiegend dokumentiert und bin daher ein grosses Stück in der Dokumentation weiter gekommen. Nun fehlt nur noch ein Fazit, und dann muss ich die Doku nochmals überarbeiten und auf Rechtschreibfehler prüfen. Mein Ziel von gestern, die Doku bereits heute Abend abgeschlossen zu haben, habe ich nicht erreicht, da ich wie bereits beschrieben noch kurzfristig eine neue Aufgabe in den Tagesplan genommen habe.

Ich denke, dass ich bis morgen Abend die Dokumentation fertig geschrieben habe und bereits einen Teil korrekturlesen konnte.

Visum vorgesetzte Fachkraft



IPA Nils Benz

31.03.2020

Arbeitsjournal 31.03.2020

Welche Tätigkeiten hatte ich für heute geplant?

#	Soll-Zeit	Beschreibung
28	2h	Fazit schreiben
29	1h	Kurzzusammenfassung schreiben
30	6h	Dokumentation fertigstellen

Was habe ich erreicht?

Arbeitszeit: 8.5 Stunden

#	Ist-Zeit	Beschreibung
28	1.5h	Fazit schreiben
29	1h	Kurzzusammenfassung schreiben
		Daily
	3h	Bug fixing und JSON formatieren
30	3h	Dokumentation fertigstellen

Legende

- grün: weniger Zeit benötigt als geplant
- gelb: zusätzliche Aufgaben, welche nicht im Zeitplan waren
- rot: mehr Zeit benötigt als geplant

Soll/Ist-Vergleich

Total geplante Zeit: 72 Stunden	Total gearbeitete Zeit: 73.5 Stunden
---------------------------------	--------------------------------------

Heute war der Plan, einen ersten Stand der Dokumentation fertigzustellen und beginnen, diese zu überarbeiten. Kurzfristig habe ich, weil ich für den letzten Tag sowieso noch keinen fixen Plan habe, einige Bugs in der Applikation gefixt. Außerdem werden nun die Services und Methoden alphabetisch sortiert angezeigt, und der Request Body kann formatiert werden.

IPA Nils Benz

31.03.2020

Schlussendlich habe ich eine halbe Stunde länger gearbeitet als geplant war, da ich die Dokumentation noch fertig durchlesen wollte. Dies fällt nicht ins Gewicht, da ich am letzten Tag theoretisch noch über 16 Stunden arbeiten durfte.

Ich liege gut im Zeitplan, und auch die insgesamt gearbeitete Zeit stimmt mit den Vorgaben überein.

Wobei hatte ich Probleme? Habe ich Hilfestellungen beansprucht?

Heute habe ich wieder viel dokumentiert, wobei ich auf keine Probleme gestossen bin. Ich habe mich bei Stefan vergewissert, dass mein Glossar-Eintrag über die Sirius Infrastruktur stimmt. Ansonsten konnte ich wieder selbstständig arbeiten.

Reflexion

Nun ist der zweitletzte Tag der Arbeit vorbei, und ich habe ein sehr gutes Gefühl, die Arbeit positiv abschliessen zu können. Am letzten Tag werde ich an der Überarbeitung der Dokumentation weiterfahren und vielleicht noch Zeit für kleinere Features wie Tastaturbefehle haben.

Visum vorgesetzte Fachkraft



IPA Nils Benz

02.04.2020

Arbeitsjournal 02.04.2020

Welche Tätigkeiten hatte ich für heute geplant?

#	Soll-Zeit	Beschreibung
30	5h	Dokumentation fertigstellen
	3h	Puffer für Bugs fixen, zusätzliche Features und Dokumentation abgeben

Was habe ich erreicht?

Arbeitszeit: 8.5

#	Ist-Zeit	Beschreibung
30	7h	Dokumentation fertigstellen und abgeben
		Daily
	1.5h	Bug fixen (Endpoints werden gefiltert)

Legende

- grün: weniger Zeit benötigt als geplant
- gelb: zusätzliche Aufgaben, welche nicht im Zeitplan waren
- rot: mehr Zeit benötigt als geplant

Soll/Ist-Vergleich

Total geplante Zeit: 80 Stunden	Total gearbeitete Zeit: 82 Stunden
---------------------------------	------------------------------------

Die IPA konnte erfolgreich abgeschlossen werden. Die totale Arbeitszeit lag bei 82 Stunden, zwei Stunden über der geplanten Zeit. Der Zeitplan war also, über die gesamte Zeitspanne gesehen, relativ realitätsgerecht.

Heute stand vor allem der Abschluss der Dokumentation und das Reparieren von allfälligen Bugs auf dem Plan. Ich habe Feedback zur Dokumentation und auch zur Applikation von Stefan Loosli und Javier Diez erhalten. Nachdem ich die Dokumentation verbessert habe, machte ich mich an das Verbessern eines Bugs, auf welchen mich Javier aufmerksam gemacht hat: die Endpoints werden bei einer Eingabe nicht gefiltert. Ich habe den Fehler dazu relativ schnell gefunden und beheben können, da ich lediglich das Standardverhalten der MaterialUI Komponente überschrieben hatte und dies rückgängig machen musste. Anschliessend habe ich probiert, Tasturbefehle in den Editor einzubauen, sodass UUIDs an der Position des Cursors eingefügt werden können. Ich musste dies jedoch wieder

IPA Nils Benz

02.04.2020

abbrechen, da die Applikation noch veröffentlicht und die Dokumentation abgeschlossen werden musste.

Wobei hatte ich Probleme? Habe ich Hilfestellungen beansprucht?

Ich habe wie bereits erwähnt probiert, eine UUID per Tastenbefehl einzufügen. Dies hat zwar funktioniert, jedoch wurde immer dieselbe UUID eingefügt. Ich habe Florian Tanner nach Hilfe gefragt, bin jedoch schlussendlich selber zum Schluss gekommen, dass dies ein Bug in der Komponente des Editors sein dürfte. Ich habe anschliessend die Änderungen wieder rückgängig gemacht und werde dies im Anschluss an die IPA nochmals probieren, wenn ich mehr Zeit zur Verfügung habe.

Ansonsten bin ich auf keine nennenswerten Probleme gestossen.

Reflexion

Ich bin sehr zufrieden mit dem Abschluss meiner IPA. Wie bereits in der Doku erwähnt konnten sämtliche erwarteten Features fristgerecht implementiert und dokumentiert werden. Ich war nicht imstande, in dieser kurzen Zeit, welche ich heute noch zur Verfügung hatte, das Feature mit den Tastaturbefehlen einzubinden. Jedoch bin ich froh, dass ich es ausprobiert habe, da ich jetzt weiss, wo die Schwierigkeiten sind und auf was ich achten muss.

Visum vorgesetzte Fachkraft



12 Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle sinngemäß und wörtlich übernommenen Textstellen aus der Literatur bzw. dem Internet wurden unter Angabe der Quelle kenntlich gemacht.

St. Gallen, den 2. April 2020



Nils Benz