Cloud Native App Entwicklung im Finanzbereich

Bachelorarbeit

Studiengang Informatik
OST – Ostschweizer Fachhochschule
Campus Rapperswil-Jona

Frühlingssemester 2022

Autoren:	Damian Kalberer, Gian Flütsch
Betreuer:	Prof. Mirko Stocker
Experte:	Leo Büttiker
Gegenleser:	Prof. Stefan F. Keller
Projektpartner:	LGT Financial Services AG

Abstract

Seit der Corona Pandemie haben viele Mitarbeitenden das Homeoffice kennen und schätzen gelernt. Arbeitgebende überlassen es mittlerweile oft den Mitarbeitenden selbst, ob sie im Büro oder von zu Hause aus arbeiten möchten. Aufgrund dieser Möglichkeiten sind sehr selten alle Mitarbeitenden gleichzeitig im Büro und es werden somit nicht mehr alle Arbeitsplätze benötigt. Dadurch kommt in immer mehr Firmen das Prinzip „Desk-Sharing“ auf, bei welchem es keine fix zugeteilten Arbeitsplätze mehr gibt.

Im Rahmen dieser Arbeit soll anhand des Vorbilds der geteilten Arbeitsplätzen die Grundlage einer Applikation für das Teilen der Parkplätze bei der LGT Financial Services AG (LGT) entwickelt werden. Damit soll die Auslastung optimiert werden. Um der LGT diese Funktionalität zu ermöglichen, wurde im Verlauf dieser Arbeit eine Cloud Native Applikation, bestehend aus modular aufgebauten Microservices in einem Azure Kubernetes Cluster, entwickelt.

Weiter soll eine Cloud IDE evaluiert werden, über welche die gesamte Entwicklung umgesetzt werden kann. Damit erhofft sich die LGT eine bessere und einfachere Wartbarkeit der benötigten SDKs, da diese nun zentral und nicht mehr auf jedem Client gemanaged werden müssen.

Als Ergebnis dieser Arbeit entstand die „LGT Parkonomy“-Applikation über welche es möglich ist, Benutzer, Standorte sowie Parkplätze zu erfassen und diese für einen gewissen Zeitraum freizugeben. Freie Parkplätze können für einen ausgewählten Zeitraum über die Applikation gebucht und eingesehen werden.

Die „LGT Parkonomy“-Applikation kann aufgrund der Progressive Web App (PWA) Architektur über einen herkömmlichen Web-Browser, aber auch auf allen mobilen Plattformen als App gespeichert werden. Mit der PWA als App kann ohne zusätzlichen Aufwand ein nahezu „native-App feeling“ auf der jeweiligen Plattform gewährleistet werden.

Die Applikation besteht aus verschiedenen Backend-Microservices, geschrieben in ASP.NET sowie mehreren Frontend-Microservices, welche auf React basieren. Die Frontend Microservices wurden als Single-Page Application (SPA) entwickelt und werden schlussendlich über den Haupt-Frontend-Microservice den Benutzern als PWA angeboten. Die gesamte Applikation wird im eigenen Azure Tenant der LGT betrieben. Die Microservices laufen in einem Kubernetes Cluster und für die Persistenz wurde der Azure SQL-Datenbank Service ausgewählt. Die Microservices werden automatisch über die GitLab CI/CD Pipeline im entsprechenden Namespace des Kubernetes Clusters deployed, sobald ein Merge in den entsprechenden Branch stattfindet. Die gesamte Entwicklung wurde mit der evaluierten Cloud IDE Gitpod umgesetzt, welche ebenfalls auf Microservices basiert und in einem separaten Namespace auf dem gleichen Kubernetes Cluster gehosted wird.

Danksagung

Wir möchten uns bei unserem Betreuer Mirko Stocker herzlich bedanken. Er hat uns durch das gesamte Projekt unterstützt und war auch ausserhalb der vereinbarten Meetings schnell und unkompliziert erreichbar. Aufgrund seiner Erfahrungen konnte er uns mit wertvollen Inputs, welche auch die administrativen Punkte umfassten, stets weiterhelfen.

Weiter möchten wir uns bei Markus Weidmann und Christian Keel von der LGT bedanken. Wir konnten während dem ganzen Projekt auf eine sehr gute Zusammenarbeit zählen und uns wurde grösstmögliche Freiheit in Bezug auf die eingesetzten Technologien und Azure Services gewährt.

Autoren

Die beiden Autoren dieser Bachelorarbeit stellen sich in diesem Kapitel vor, sodass sicher der Leser ein Bild der Studenten machen kann.

Damian Kalberer

- Wohnt in Mels, SG
- Lehre als Informatiker EFZ Fachrichtung Systemtechnik
- Studienschwerpunkt Software Engineering
- In der Freizeit auf den Tourenskiern und dem Bike

Gian Flütsch

- Wohnt in Flims, GR
- Lehre als Informatiker EFZ Fachrichtung Systemtechnik
- Studienschwerpunkt Cyber Security
- In der Freizeit auf den Skiern und dem Bike

Inhaltsverzeichnis

1 Management Summary

In diesem Kapitel wird das Projekt mithilfe eines Management Summarys kurz erläutert und ein Überblick darüber gewährt.

1.1 Ausgangslage

Die LGT besitzt mehrere Parkplätze und Parkhäuser, welche für die Mitarbeitenden zur Verfügung stehen. Aufgrund des steigenden Bedürfnisses ans HomeOffice sowie auch dem Wachstum der Firma, soll eine Lösung entwickelt werden, mit welcher die Parkplätze besser ausgelastet werden können.

Weiter besitzt die LGT einen privaten Azure Cloud Tenant, welcher allerdings noch nicht stark ausgelastet ist. Im Rahmen dieser Bachelorarbeit soll eine Cloud Native Applikation entwickelt werden, welche in der Azure Cloud läuft und die Auslastung der Parkplätze verbessern soll. Dabei soll es den Parkplatzbesitzern[1] möglich sein, ihre Parkplätze für einen gewissen Zeitraum freizugeben und andere Mitarbeitende sollen die Möglichkeit haben, diese freigegebenen Parkplätze zu buchen.

1.2 Vorgehen

Im Vorfeld der Bachelorarbeit wurde eine Umfrage bei den Mitarbeitenden der LGT durchgeführt, um das Interesse und den Nutzen einer solchen Parkplatz-Sharing Applikation zu evaluieren. Aufgrund des positiven Feedbacks, soll zusammen mit der LGT eine Cloud Native Applikation entwickelt werden, welche diesen Bedarf abdeckt und zugleich wichtige Learnings für die Entwicklung und den Betrieb einer Cloud-Applikation aufzeigt.

Während der Implementation konnten wir uns am zuvor erstellten Releaseplan orientieren und haben iterativ die gewünschten Features umgesetzt.

Nach jeder Iteration wurde die Applikation in der Cloud neu deployed und dem Kunden vorgestellt. Anhand der neuen Feedbacks konnten weitere Optimierungen in die nächste Iteration aufgenommen und zur Zufriedenheit des Kunden umgesetzt werden.

[1] In dieser Bachelorarbeit wird aus Gründen der besseren Lesbarkeit das generische Maskulinum verwendet. Weibliche und anderweitige Geschlechter werden dabei implizit mitgemeint.

1.3 Ergebnisse

Mit dem Abschluss dieser Arbeit wurde eine Cloud Native Parkplatz-Sharing Applikation für die LGT entwickelt. Die Applikation besteht aus verschiedenen modular-aufgebauten Microservices im Backend sowie im Frontend. Die Backend Microservices wurden mit ASP.NET entwickelt und besitzen jeweils eine GraphQL-API. Die Frontend Microservices wurden mit React aufgebaut und werden schlussendlich über NGINX als SPA resp. PWA ausgeliefert. Die asynchrone Kommunikation von den verschiedenen Backends läuft über verschiedene RabbitMQ-Queues und über eine Consul-Instanz werden die Backend-Microservices automatisch erkannt und erfasst. Mit dieser Architektur kann von allen gängigen Plattformen darauf zugegriffen werden und aufgrund des PWA Frontends entsteht auf Smartphones ein nahezu „native-App feeling“.

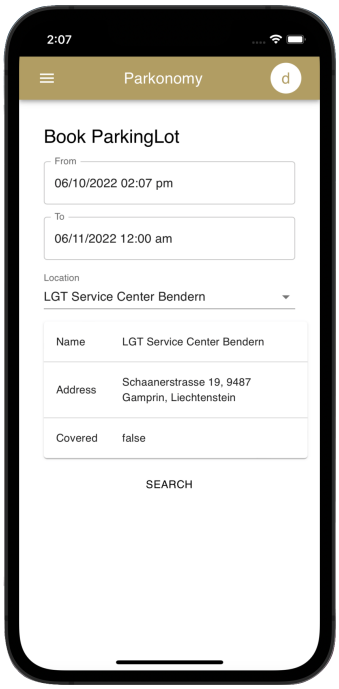


Abbildung 1.1: Parkplatz Buchen

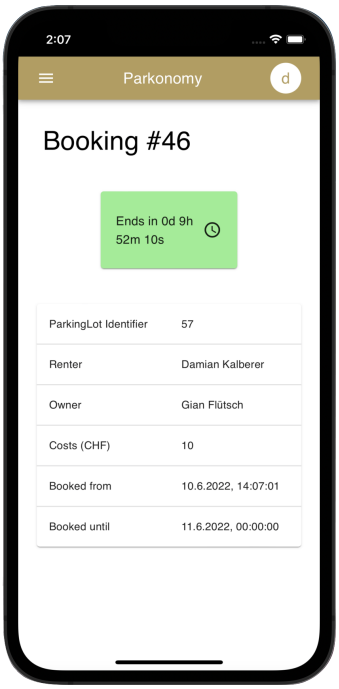


Abbildung 1.2: Buchungsbestätigung

Die LGT kann mit der Applikation verschiedene Parkplätze resp. Parkhäuser erfassen. Die jeweiligen Parkplatzbesitzer können ihren Parkplatz am entsprechenden Standort erfassen und freigeben. Ein Mitarbeiter hat die Möglichkeit, einen freigegebenen Parkplatz am gewünschten Standort zu buchen.

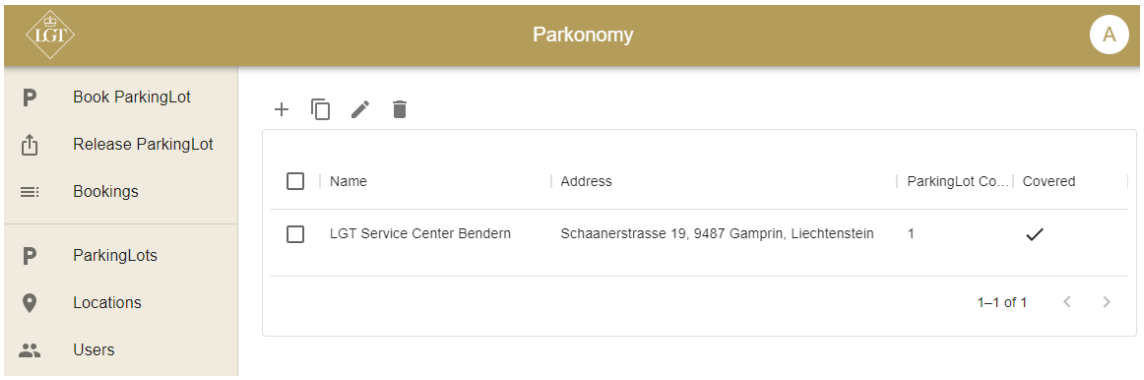


Abbildung 1.3: Parkplatzverwaltung im Browser

Die Applikation wird in einem Azure Kubernetes Cluster betrieben und automatisch über GitLab in den entsprechenden Namespace und mit der gewünschten Update-Strategie deployed. Mit Gitpod konnte zudem eine gute Cloud IDE evaluiert und selber im Kubernetes Cluster gehosted sowie betrieben werden. Eine solche Cloud IDE bringt einen grossen Vorteil für den Betrieb aber auch für die Entwickler selbst mit sich und aufgrund des Hostings bleibt dennoch volle Kontrolle über die Daten vorhanden.

Weiter konnten die gemachten Learnings mit der Entwicklung und dem Deployment in der Azure Cloud für die LGT festgehalten werden.

1.4 Ausblick

Die LGT-Parkonomy Applikation umfasst nun die für den täglichen Betrieb wichtigsten Funktionen, sodass mit einer Pilot-Phase bei den Mitarbeitern begonnen werden kann. Wir sind überzeugt, die Applikation der LGT in einem guten Zustand übergeben zu können und denken, dass durch die bessere Auslastung der Parkplätze ein Mehrwert für die Mitarbeitenden generiert wird.

In Zukunft wird die Cloud ein immer grösseres und wichtigeres Thema und wir hoffen, dass wir mit unseren gemachten Learnings einen Nutzen für die LGT erreichen konnten und dieses Wissen in den weiteren Ausbau und zukünftige Cloud-Projekte bei der LGT fliessen kann.

Teil I

Projektplan

2 Übersicht

Im Abschnitt „Projektplan“ wird die Aufgabenstellung erläutert, der Zeitplan definiert und das Projektvorgehen geplant. Ausserdem werden Projektprozesse sowie das Risiko und Qualitätsmanagement spezifiziert. Zusammengefasst geht es hier um organisatorische Aspekte, welche vor dem technischen Teil des Projektes definiert werden müssen.

2.1 Aufgabenstellung

Der Auftraggeber möchte Erfahrung in der Entwicklung von Cloud-Native Microservice-Anwendungen sammeln und dazu eine Proof-of-Concept Applikation erstellen lassen. Folgende Ziele sollen in der Bachelorarbeit erreicht werden:

- Generische Ressourcen-Applikation in einer Microservice-Architektur
- Parkplatz-Sharing Modul, mit Reservation und Freigabe von Parkplätzen
- GitLab CI/CD Pipeline Integration, mit build und deployment in der Azure Cloud als AppService
- Development in einer Cloud IDE

Zusätzlich sollen folgende Kann-Ziele bearbeitet werden:

- Orchestrierung via Kubernetes/ OpenShift
- High-Level Security Blueprint
- Zusätzliches Modul: FreeSeating Microservice, das dieselbe Ressourcen-Applikation nutzt.
- Zusätzliche Schnittstellen:
  - User-Onboarding über Azure-AD
  - Mitarbeiter mit fixem Parkplatz inkl. Autonummer
  - Verrechnung

Diese Aufgabenstellung wurde aus der Original-Aufgabenstellung im Anhang entnommen.

2.2 Erwartete Resultate

- Software in Form einer Cloud Native Applikation, welche die in der vorgegebenen Zeit möglichen Features in einem dem Kunden zufriedenstellenden Umfang beinhaltet
- Integration von diversen Azure Features sowie Automatisation des Deployment
- Dokumentation der Arbeit (inklusive Vorgehen und kritischer Bewertung der getroffenen Entscheide)

2.3 Einschränkungen

Das Modul „Bachelorarbeit“ ist zeitlich begrenzt. Das Team hat gemeinsam 720 Stunden Zeit, die Projektarbeit in 17 Wochen durchzuführen. Diese zeitliche Begrenzung setzt sich folgendermassen zusammen:

- 720h = 2 Teammitglieder * 12 ECTS Punkte * 30h pro ECTS
- 17 Semesterwochen

Die Studienarbeit beginnt offiziell am 21.02.2022 und endet mit der Abgabe am 17.06.2022 um 17:00 Uhr.

2.4 Persönliche Ziele

Neben dem Endprodukt umfasst das Projekt folgende persönliche Ziele der Teammitglieder:

- Erfolgreiche Umsetzung eines vorgegebenen Projekts in der zur Verfügung gestellten Zeit
- Kennenlernen der Entwicklung von Cloud Native Applikationen
- Auseinandersetzung mit der Entwicklung in der Cloud sowie der Integration von Cloud Features
- Erfahrungen sammeln für weitere schulische, private und berufliche Projekte
- Gute Voraussetzung für berufliche Tätigkeiten erarbeiten

3 Organisation

In diesem Kapitel wird die Organisation innerhalb des Projekts und die Verteilung der Aufgaben beschrieben. Auch werden Vorgehensweisen, wie die Projektmethodik und die Projektabläufe, definiert. Erwähnt werden ausserdem die Werkzeuge, welche für das Projektmanagement eingesetzt wurden.

3.1 Rollenverteilung

Das Projektteam besteht aus zwei Personen – Damian Kalberer und Gian Flütsch. Mirko Stocker übernimmt die Betreuung sowie die Bewertung, bei welcher er von Leo Büttiker als Experte sowie Stefan F. Keller als Gegenleser unterstützt wird. Der Kunde, die LGT Financial Services AG, wird durch Markus Weidmann sowie Christian Keel vertreten.

Während der Arbeit wird es keine fixe Aufteilung der Aufgaben geben.

- **OST – Ostschweizer Fachhochschule**
  - Mirko Stocker – Betreuung/ Bewertung
  - Leo Büttiker – Experte
  - Stefan F. Keller – Gegenleser
  - Damian Kalberer – Studierender
  - Gian Flütsch – Studierender
- **LGT Financial Services AG**
  - Markus Weidmann – Bereichsleiter Information System Architecture
  - Christian Keel – Teamleiter System Engineering

3.2 Besprechungen

Besprechungen mit dem Betreuer werden voraussichtlich wöchentlich abgehalten, jedoch wird dies nach Bedarf geregelt. Mit dem Kunden wird versucht, sich so oft wie möglich und nötig auszutauschen, um ein zufriedenstellendes Endresultat zu erzielen. In beiden Fällen wird jedoch auf fixe Termine verzichtet. Innerhalb des Projektteams gibt es am Ende jedes Sprints ein Review- und Planungsmeeting. Weitere Besprechungen werden dank des kleinen Teams, spontan und der Notwendigkeit angepasst, einberufen.

3.3 Arbeitspakete

Sowohl der Entwicklungs- als auch der Dokumentationsprozess basiert auf dem Feature Branch Workflow. Dabei werden wir uns an der Atlassian Feature Branch Workflow Dokumentation orientieren [1]. Es wird allerdings auf den Branch *develop* verzichtet. Der letzte stabile Release wird stattdessen durch einen Tag markiert.

Für den Workflow gelten folgende Regeln:

- Es existiert ein Branch *main*, welcher als Integrationsbranch aller Feature-Branches fungiert.
  - Pro Version wird auf dem Branch *main* ein Tag gesetzt und es findet ein neuer Release statt.
- Für jedes Feature wird ein eigener Feature-Branch erstellt.
  - Ein Feature-Branch wird ausgehend von einem Issue erstellt.
  - Ein Feature-Branch wird per Merge-Request in den Branch *main* integriert.

Ausserdem gelten spezifisch für Arbeitspakete folgende Regeln:

- Arbeitspakete werden als Issues erstellt.
- Die Zeitschätzung und Rapportierung erfolgt über Clockify.
- Nach Abschluss der zu erwartenden Arbeit im Issue wird ein Review durchgeführt.
- Es wird erst ein Tag gesetzt, wenn die Definition of Done für alle Arbeitspakete im Release gemäss Qualitätsmanagement erreicht wurde.

Der folgende Ablauf gilt für Issues jeglicher Art, unabhängig davon, ob es sich dabei um Dokumentation oder Implementation handelt. Der Arbeitsschritt „Tests schreiben“ fällt für die Dokumentation weg:

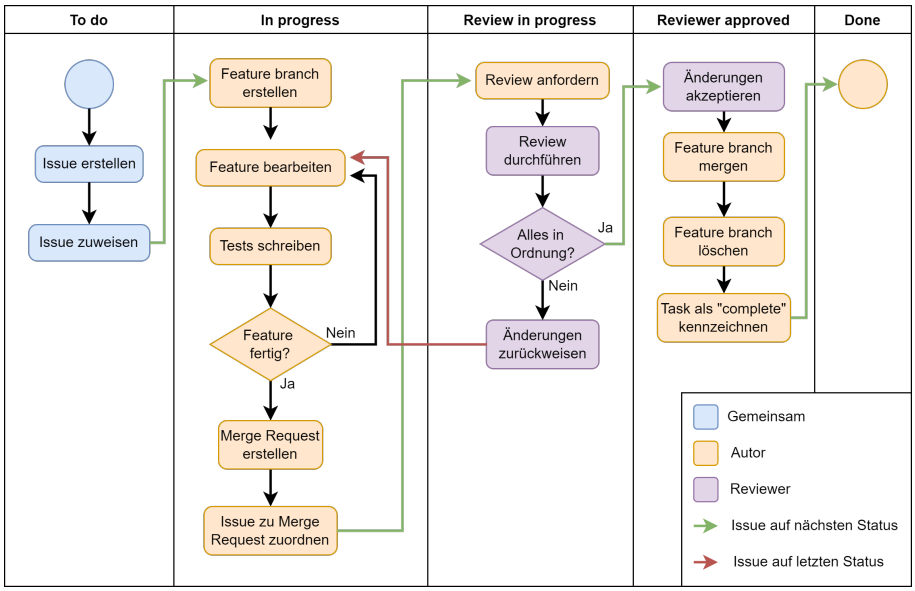


Abbildung 3.1: Lebenszyklus eines Arbeitspakets

3.4 Tools

Um das Projektmanagement zu erleichtern, werden folgende Tools eingesetzt:

- GitLab:
  - Zur sauberen Aufteilung wird mit verschiedenen Repositories für Code und Dokumentation gearbeitet.
  - GitLab-Projects ermöglicht eine Übersicht über alle Issues in Form eines Kanban-Boards, aufgeteilt nach Status.
- LaTeX:
  - LaTeX wird verwendet, um die Dokumentation zu verfassen.
  - Der textbasierte LaTeX-Quellcode ermöglicht es, auch die Dokumentation über Git zu verwalten.
- Clockify:
  - Clockify ermöglicht die Schätzung von Zeitaufwänden, deren Rapportierung und Auswertung.
  - Mithilfe einer Browsererweiterung lassen sich GitLab Issues direkt als Tasks in Clockify übertragen.
- Microsoft Teams:
  - Microsoft Teams wird als primärer Kommunikationskanal innerhalb, aber auch ausserhalb des Projektteams genutzt.
  - Es dient als Ersatz für persönliche Meetings während der Covid-19 Pandemie.

4 Zeitplan

In diesem Kapitel wird näher auf den zeitlichen Aspekt und die geplanten Schritte eingegangen. Ebenfalls werden Meilensteine definiert und beschrieben. Zusätzlich werden die verschiedenen Phasen nach Scrum+ näher erläutert.

4.1 Übersicht

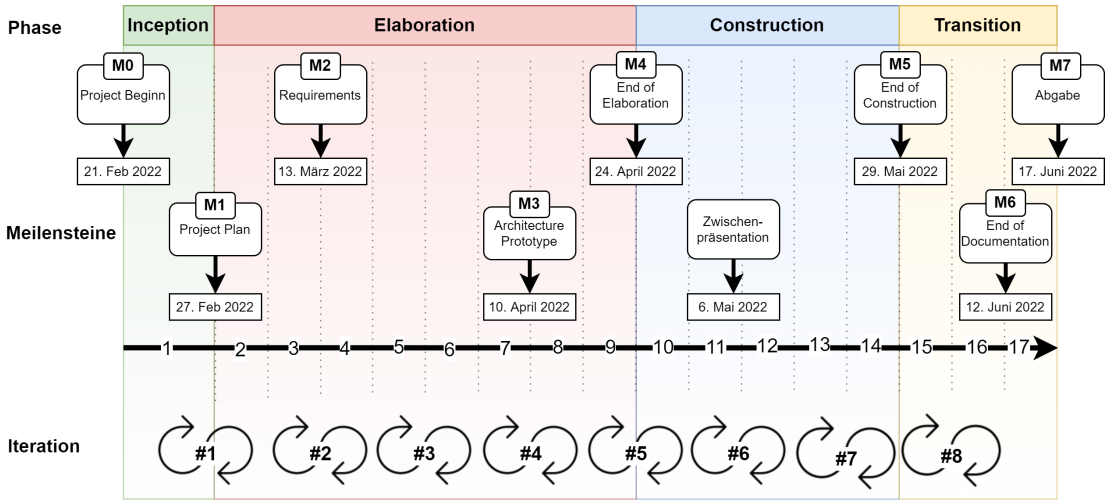


Abbildung 4.1: Übersicht Zeitplan

4.2 Projektmethodik

Es wurden diverse Projektmethoden evaluiert, genauer gesagt Rational Unified Process (RUP), Wasserfall, Scrum und Scrum+. Anschliessend wurden Faktoren eruiert, welche die Wahl der Methodik beeinflussen:

1. Das Projektteam hat noch keine Erfahrungen in der Entwicklung von Cloud Native Applikationen und kann sich dementsprechend nicht auf historische Daten beziehen.
2. Die Arbeit erfordert Exploration, was für einen agilen Ansatz spricht.
3. Das Projektteam konnte bereits im Engineeringprojekt sowie in der Studienarbeit Erfahrungen mit der Scrum+ Methodik sammeln und fühlt sich daher mit dieser vertraut.

Da die oben genannten Fakten einen agilen Ansatz nahelegen, ein End of Elaboration aber trotzdem als notwendig empfunden wird, fällt die Wahl auf Scrum+.

4.3 Projektabläufe

Während des ganzen Projekts finden zwei zentrale Abläufe statt. Zum Einen ist dies das Sprint-Review und -Planning nach jeder Iteration und zum Anderen das Abarbeiten der Arbeitspakete.

4.3.1 Sprint-Review und Planning

Am Ende jeder Iteration erfolgt ein Sprint Review Meeting, bei welchem die Arbeitspakete des letzten Sprints überprüft werden. Konnten gewisse Arbeitspakete nicht abgeschlossen werden, sind diese im nächsten Sprint einzuplanen.

Sind alle Arbeitspakete besprochen, werden die Neuen evaluiert, aus dem Backlog entnommen und für den nächsten Sprint eingeplant.

4.4 Phasen

Das Projekt wird nachfolgend, gemäss Projektmethodik aus Kapitel 4.2, in Phasen unterteilt. Das bedeutet konkret:

- Die Arbeit wird in vier Phasen geplant.
- In jeder Phase gibt es Iterationen.
- Meilensteine werden auf das Ende von Iterationen gelegt.
- Es wird bei jeder Iteration (in der „Construction“-Phase) ein potentially shippable product angestrebt.
- Nach jeder Iteration findet eine Sprint-Review-Sitzung statt, um den Backlog mit Arbeitspaketen zu pflegen und das weitere Vorgehen zu besprechen.
- Nach jeder Iteration findet ein Austausch über das agile Vorgehen statt (Sprint Retrospective), um Probleme im Projektmanagement zu vermeiden.
- Es wird Wert auf ein sauberes End of Elaboration gelegt. Dazu wird die End of Elaboration Checklist verwendet.
- Am Ende der „Construction-Phase“ gibt es einen Feature-Freeze.

4.4.1 Inception

- **Dauer:** 1 Woche
- **Zeitraum:** 21.02.22 - 27.02.22

Obwohl ein agiles Vorgehen für das Projekt angestrebt wird, wird dieses in der ersten Phase grob durchgeplant. Dies dient in erster Linie zur Aufteilung der begrenzten Projektzeit, um spätere Zeitengpässe zu vermeiden. Ausserdem ermöglicht der Projektplan für alle Beteiligten einen gemeinsamen Ausgangspunkt zu schaffen und die Rahmenbedingungen zu definieren.

Des Weiteren werden die Anforderungen des Kunden aufgenommen und erstmalig spezifiziert. Die Spezifikation wird in einem weiteren Schritt mit dem Kunden besprochen.

4.4.2 Elaboration

- **Dauer:** 8 Wochen
- **Zeitraum:** 28.02.22 - 24.04.22

In einem ersten Schritt werden in dieser Projektphase die Anforderungen ausgearbeitet. Durch die agile Natur des Projekts können sich diese im Laufe der Arbeit noch ändern. Es ist jedoch sinnvoll die Anforderungen möglichst genau zu erfassen, damit diese bei der anschliessenden Planung der Architektur auch berücksichtigt werden können.

Weiter wird die technische Machbarkeit des Projekts überprüft. Dies geschieht mit einem Prototypen, welcher die wichtigsten Aspekte der geplanten Architektur sowie Integration in die Cloud abbildet. So sind am Ende der „Elaboration-Phase“ alle Unklarheiten beseitigt, was einen gut vorbereiteten Start der Implementation ermöglicht.

4.4.3 Construction

- **Dauer:** 5 Wochen
- **Zeitraum:** 25.04.22 - 27.05.22

Während dieser dritten Phase wird hauptsächlich das geplante Produkt umgesetzt. Parallel dazu werden, zur Gewährleistung der Code-Qualität, Tests geschrieben und die Dokumentation nachgeführt.

Schlussendlich wird ein Usability-Test des entwickelten Produkts durchgeführt, damit der Kunde direkt Feedback dazu einbringen und dieses bestmöglichst umgesetzt werden kann.

Am Ende der „Construction-Phase“ findet ein Feature-Freeze statt. Dies bedeutet, dass keine neue Funktionalität mehr zum Produkt hinzugefügt wird.

4.4.4 Transition

- **Dauer:** 3 Wochen
- **Zeitraum:** 28.05.22 - 17.06.22

Diese letzte Phase dient zur Behebung verbleibender Fehler, zur Qualitätsprüfung und zur Übergabe des Produkts an dessen weiteren Betreiber. Ausserdem wird das Projekt ausgewertet und letzte Verbesserungen an der Dokumentation vorgenommen, damit diese zur Abgabe bereit ist.

4.5 Iterationen / Sprints

Ein Sprint dauert stets zwei Wochen. Die Sprints beginnen am Montag und werden am Sonntag abgeschlossen. Ausgewertet werden diese am zweiten Freitag mit anschliessender Planung des darauffolgenden Sprints. Die Auswertung am Freitag erlaubt es, noch nicht abgeschlossene Tätigkeiten bis Sonntag nachzuliefern.

Insgesamt werden in diesem Projekt acht Sprints durchgeführt.

4.6 Meilensteine

Nachstehend werden die einzelnen Meilensteine spezifiziert. Jeder dieser Meilensteine enthält Arbeitsprodukte, welche in die folgenden Kategorien eingeteilt werden:

- **Priorität 1:** Arbeitsprodukt, welches zwingend vor dem entsprechenden Meilenstein fertiggestellt werden muss.
- **Priorität 2:** Arbeitsprodukt, welches auch in der darauffolgenden Iteration beendet werden kann.

4.6.1 Projektplan

Ziel:	Projektplanung abgeschlossen
Zeitpunkt:	Ende Sprint 1 am 27.02.22
Arbeitsprodukte:	• Priorität 1 – Der Projektplan ist inklusive Projektorganisation, Zeitplan, Qualitätsmanagement und Risikoanalyse fertiggestellt. • Priorität 2 – keine

Tabelle 4.1: Meilenstein: Projektplan

4.6.2 Requirements

Ziel:	Anforderungsanalyse abgeschlossen
Zeitpunkt:	Halbzeit Sprint 2 am 13.03.22
Arbeitsprodukte:	• Priorität 1 – Use Cases sind im „Brief“-Stil verfasst. – Nicht-funktionale Anforderungen sind definiert. – Domainanalyse ist durchgeführt. • Priorität 2 – Schnittstellenbeschreibung ist erstellt.

Tabelle 4.2: Meilenstein: Requirements

4.6.3 Architecture Prototype

Ziel:	Technischer Architekturdurchstich
Zeitpunkt:	Halbzeit Sprint 4 am 10.04.22
Arbeitsprodukte:	• Priorität 1 – Funktionierender Prototyp über alle Layer – Minimierung aller Projektkritischen Risiken – Cloud IDE • Priorität 2 – Containerorchestrierung

Tabelle 4.3: Meilenstein: Architecture Prototype

4.6.4 End of Elaboration

Ziel:	Alle Unsicherheiten beseitigt
Zeitpunkt:	Halbzeit Sprint 5 am 24.04.22
Arbeitsprodukte:	• Priorität 1 – End of Elaboration Checklist ist überprüft. – Architektur ist überprüft. • Priorität 2 – Dokumentation ist auf dem aktuellsten Stand.

Tabelle 4.4: Meilenstein: End of Elaboration

4.6.5 End of Construction

Ziel:	Geplante Funktionalität implementiert
Zeitpunkt:	Ende Sprint 7 am 27.05.22
Arbeitsprodukte:	• Priorität 1 – Alle geplanten Funktionalitäten sind implementiert. • Priorität 2 – Einzelne nicht-geplante Funktionalitäten sind implementiert. – Dokumentation ist auf dem aktuellsten Stand.

Tabelle 4.5: Meilenstein: End of Construction

4.6.6 End of Documentation

Ziel:	Dokumentation fertiggestellt
Zeitpunkt:	Ende Sprint 8 am 10.06.22
Arbeitsprodukte:	• Priorität 1 – Dokumentation ist komplett nachgeführt. – Das Abstract ist dem Betreuer abgegeben worden. • Priorität 2 – Layout- und Grammatikfehler sind nochmals überprüft worden.

Tabelle 4.6: Meilenstein: End of Documentation

4.6.7 Abgabe

Ziel:	Projekt abgeschlossen
Zeitpunkt:	Abgabetermin der Bachelorarbeit am 17.06.22
Arbeitsprodukte:	• Priorität 1 – Die Dokumentation ist fertiggestellt. – Alle Dokumente sind auf https://archiv.i.ost.ch/ hochgeladen. • Priorität 2 – Das Projekt ist dem Kunden zum Betrieb übergeben.

Tabelle 4.7: Meilenstein: Abgabe

5 Qualitätsmanagement

In diesem Kapitel wird beschrieben, welche Massnahmen ergriffen werden um die Qualität der Dokumentation als auch des Endprodukts zu garantieren.

5.1 Qualitätsmassnahmen

Zeitraum	Massnahme	Ziel
Freitags	Sprint Review	Austausch und Review über den Stand der Arbeitspakete.
Freitags	Sprint Planning	Neuen Sprint mit Arbeitspaketen aus Backlog planen. Falls nötig Prioritäten anpassen.
Mittwochs	Meeting mit Projektbetreuung	Allfällige Unklarheiten klären, Projektstand besprechen.
Kontinuierlich	Continuous Integration	Durchführen von automatisierten Tests und statischer Code-Analyse.
Kontinuierlich	Code Review	Gegenseitige Codereviews bevor Features/ Bugs in den Main/ Development-Branch integriert werden.
Kontinuierlich	Dokumentationsreview	Gegenseitige Dokumentationsreviews bevor Änderungen in den Main/ Development-Branch integriert werden.

Tabelle 5.1: Eingeplante Qualitätsmassnahmen

5.1.1 Projektmanagement

Die Basis für das Projektmanagement bildet ein Projektboard, welches über GitLab realisiert wird. Arbeitspakete werden als Issues angelegt und dem entsprechenden Meilenstein im Repository zugeordnet. Dies ermöglicht uns eine Übersicht über sämtliche Arbeitspakete und deren aktuellen Status. Die Zeiterfassung über das gesamte Projekt erfolgt über Clockify. Weitere Tools werden für das Projektmanagement vorerst nicht eingesetzt. Die Projektbetreuung hat Zugriff auf die GitLab Organisation und somit Zugriff auf sämtliche Repositories und den Stand der Arbeiten.

5.1.2 Dokumentation

Die Dokumentation befindet sich in einem eigenen Repository auf GitLab. Die Qualität bezüglich Inhalt, Formatierung und Grammatik wird mittels Peer-Reviews vor dem Integrieren der Merge Requests gewährleistet.

5.1.3 Entwicklung

Der Source Code wird mit Git versioniert und befindet sich getrennt von der Dokumentation auf einem separaten Git-Repository auf GitLab.

Die Qualität des Source Codes wird, in einem ersten Schritt, automatisiert geprüft. Dies erfolgt im Rahmen einer Continuous Integration durch einen Linter, mit einem vordefinierten Stylesheet. Des Weiteren werden in dieser Continuous Integration auch Unit und Integration Tests durchgeführt. In einem letzten Schritt wird die Codequalität zusätzlich durch ein Peer-Review der Merge Requests auf GitLab geprüft.

5.2 Definition of Done

Um Unklarheiten bezüglich der Vollständigkeit von Arbeitspaketen, Sprints und Meilensteine zu beseitigen, wird für diese in den folgenden Abschnitten eine Definition of Done festgelegt.

5.2.1 Arbeitspaket

- Automatisierte Tests in der Continuous Integration zeigen keine Fehler an, allfällige Warnungen wurden überprüft und abgearbeitet.
- Die GitLab CI/CD Pipeline ist erfolgreich durchlaufen.
- Sämtliche Unit Tests wurden fehlerfrei ausgeführt.
- Die nicht-funktionalen Anforderungen sind/ bleiben erfüllt.
- Es wurde ein Review des Codes beziehungsweise der Dokumentationsänderungen durchgeführt.

5.2.2 Sprint

- Nicht abgeschlossene Arbeitspakete in den nächsten Sprint verschoben
- Definition of Done für alle verbleibenden Arbeitspakete im Sprint erreicht
- Sprint Review durchgeführt

5.2.3 Meilenstein

- Definition of Done für alle Sprints im Meilenstein erreicht
- Die geplanten Resultate des Meilensteins fertiggestellt

5.3 Testing

In diesem Kapitel wird beschrieben, welche Arten von Tests durchgeführt werden. Auch wird definiert, wann diese Tests erstellt und wie diese dokumentiert werden.

5.3.1 Unit und Integration Tests

Um eine gute Testabdeckung zu garantieren, wird zum Schreiben der Tests genügend Zeit eingeplant. Diese werden im Review angeschaut und auf ihre Tauglichkeit geprüft. Priorisiert wird der anwendungskritische Code, konkret die Businesslogik.

Trivialer Code, wie beispielsweise Getter und Setter, wird nicht getestet. Diese werden in weiteren Tests abgefangen. Ausserdem wird generell auf Tests für Code verzichtet, der lediglich als Bindeglied zwischen zwei Bibliotheken fungiert.

5.3.2 Systemtests

Anhand der definierten Use-Cases werden die Systemtests durchgeführt und ausgewertet. Dies wird regelmässig manuell von den Teammitgliedern erledigt.

5.3.3 Nicht-funktionale Tests

Basierend auf den nicht-funktionalen Anforderungen werden geeignete nicht-funktionale Tests durchgeführt. Geplant ist zudem ein Usability-Test gegen Ende der Construction-Phase.

5.3.4 Abnahmetests

Es ist ein Abnahmetest mit dem Endkunden vorgesehen. Dieser wird voraussichtlich an einer der letzten Sitzungen mit dem Kunden durchgeführt.

6 Risikomanagement

Arbeitspakete werden gemäss ihrem Risiko priorisiert. Je grösser das Risiko, an welchem ein Arbeitspaket scheitert, desto früher wird es erledigt. So soll möglichst früh ein „Durchstich“ durch alle technischen Schichten erfolgen, damit technische Probleme frühzeitig erkannt werden. Am Schluss eingeplant sind „Nice to have“ - Anforderungen, wie das Umsetzen der Lösung auf dem LGT Tenant.

Es werden nur wenige zeitliche Reserven gebildet – stattdessen ist das Projekt so geplant, dass möglichst früh ein funktionierender Prototyp vorhanden ist, der im weiteren Verlauf optimiert und ausgebaut wird. So kann auch dann ein lauffähiges Produkt ausgeliefert werden, wenn zeitlich bedingt nicht alle Arbeitspakete abgeschlossen werden konnten.

Die letzte Woche der Bachelorarbeit ist für abschliessende Arbeiten vorgesehen und kann somit als zusätzlicher Zeitpuffer dienen.

6.1 Risiken

In diesem Kapitel geht es um die evaluierten Risiken und die Abschätzung, welchen Einfluss diese auf den Verlauf der Arbeit haben könnten.

6.1.1 Risiko 1: Probleme bei der Umsetzung

In dieser Arbeit werden brandaktuelle Themen aufgegriffen und umgesetzt. Dadurch besteht die Gefahr, dass Funktionen nicht wie gewünscht entwickelt oder noch gar nicht vorhanden sind.

- **Auswirkungen:** Es muss mit sehr viel Mehraufwand gerechnet werden, um sich zu informieren. Im schlimmsten Fall muss die Lösung von selbst neu aufgebaut werden.
- **Maximaler Schaden:** 48h
- **Eintrittswahrscheinlichkeit:** 40%
- **Gewichteter Schaden:** 19.2h
- **Vorbeugung:** Es wird am Anfang sehr viel Zeit investiert um sich zu informieren und die neuen Technologien auszuprobieren. Somit können die schlimmsten Überaschungen verhindert oder frühzeitig erkannt werden.
- **Verhalten beim Eintreten:** Die Betreuungsperson wird informiert und es werden Massnahmen miteinander besprochen.

6.1.2 Risiko 2: Ausfall eines Teammitglieds

Da das Team nur aus zwei Personen besteht, hat der Ausfall eines Mitglieds einen starken Einfluss auf den Verlauf des Projektes.

- **Auswirkungen:** Der Zeitplan kann je nach Schweregrad des Ausfalls nicht mehr eingehalten werden.
- **Maximaler Schaden:** 72h
- **Eintrittswahrscheinlichkeit:** 20%
- **Gewichteter Schaden:** 14.4h
- **Vorbeugung:** Durch regelmässige Absprachen im Team wird sichergestellt, dass alle Teammitglieder auf dem selben Informationsstand sind. Dadurch sind bei einem Ausfall alle nötigen Informationen zur Fortsetzung der Arbeit vorhanden.
- **Verhalten beim Eintreten:** Die ausfallende Person arbeitet in dem Masse weiter, wie es ihr möglich ist. Die andere Person arbeitet normal weiter. Gegebenenfalls muss mit dem Kunden eine Reduktion des Umfangs angeschaut werden, um den Ausfall zu kompensieren.

6.1.3 Risiko 3: Unterschätzung des Aufwandes

Das Team hat noch wenig Erfahrung mit den neuen Themen. Das kann dazu führen, dass die Planung des Aufwandes einzelner Anforderungen unterschätzt wird.

- **Auswirkungen:** Die Anforderungen können nicht vollständig umgesetzt werden. Deshalb müssen einzelne Anforderungen gekürzt oder gestrichen werden.
- **Maximaler Schaden:** 48h
- **Eintrittswahrscheinlichkeit:** 30%
- **Gewichteter Schaden:** 14.4h
- **Vorbeugung:** Es werden für die Anforderungen, welche Probleme bereiten können, genügend Reserven eingebaut.
- **Verhalten beim Eintreten:** Die Betreuungsperson wird informiert und es werden Massnahmen miteinander besprochen.

6.1.4 Risiko 4: Späte Änderungen der Anforderungen

Da aus dem Projekt auch sehr viele Learnings gezogen werden sollen, kann es sein, dass zu den geplanten Anforderungen später noch zusätzliche definiert werden.

- **Auswirkungen:** Die Anforderungen müssen neu priorisiert werden. Dadurch können geplante Anforderungen nicht wie gewünscht umgesetzt werden.
- **Maximaler Schaden:** 24h
- **Eintrittswahrscheinlichkeit:** 40%
- **Gewichteter Schaden:** 9.6h
- **Vorbeugung:** Die Anforderungen werden so früh wie möglich mit Absprache aller Parteien definiert.
- **Verhalten beim Eintreten:** In Absprache mit der Betreuungsperson und dem Kunden werden die Anforderungen neu priorisiert. Möglicherweise müssen Arbeitspakete nach hinten verschoben oder weggelassen werden.

6.1.5 Risiko 5: Abhängigkeit der Produkte

Da das Projekt für die LGT durchgeführt wird, bestehen verschiedene Abhängigkeiten von Produkten. Das heisst, es müssen Produkte für die Lösung gewählt werden, welche bei der LGT bereits im Einsatz sind.

- **Auswirkungen:** Für die Umsetzung einiger Anforderungen eignen sich gewisse Produkte besser, welche aber nicht eingesetzt werden können. Es muss deshalb eine eigene Lösung für das Produkt entwickelt werden.
- **Maximaler Schaden:** 24h
- **Eintrittswahrscheinlichkeit:** 40%
- **Gewichteter Schaden:** 9.6h
- **Vorbeugung:** Bei der Definition der Anforderungen werden Produkte, welche die LGT bereits benutzt, einbezogen und somit die Anforderungen darauf abgestimmt.
- **Verhalten beim Eintreten:** In Absprache mit der Betreuungsperson und dem Kunden werden die Anforderungen, welche Probleme bereiten, neu definiert oder allenfalls andere Massnahmen zur Umsetzung besprochen.

6.1.6 Risiko 6: Missverständnisse in der Kommunikation

In diesem Projekt arbeiten drei Parteien zusammen. Aufgrund dessen kann es zu Missverständnissen und Fehlinterpretationen kommen, welche Zeit kosten.

- **Auswirkungen:** Je nach Missverständnis muss im schlimmsten Fall ein ganzer Anwendungsfall neu implementiert oder weggelassen werden.
- **Maximaler Schaden:** 48h
- **Eintrittswahrscheinlichkeit:** 10%
- **Gewichteter Schaden:** 4.8h
- **Vorbeugung:** Die Anwendungsfälle werden früh spezifiziert und mit dem Kunden abgesprochen. Während der Umsetzung werden regelmässige Feedbacks vom Kunden eingeholt.
- **Verhalten beim Eintreten:** Die Betreuungsperson sowie der Kunde werden über das Missverständnis informiert. Anschliessend wird besprochen, wie weiter vorgegangen werden soll.

6.1.7 Risiko 7: Azure Tenant

Während dem Projekt arbeiten wir auf einem Azure Tenant, welcher für die LGT als Test Tenant verwendet wird.

- **Auswirkungen:** Andere Personen, welche auf dem Tenant arbeiten, können verhindern, dass wir geplante Anpassungen rechtzeitig vornehmen. Diese müssen somit auf einen späteren Zeitpunkt verschoben werden.
- **Maximaler Schaden:** 8h
- **Eintrittswahrscheinlichkeit:** 20%
- **Gewichteter Schaden:** 1.6h
- **Vorbeugung:** Es wird versucht die grösseren Anpassungen nicht während den Bürozeiten durchzuführen.
- **Verhalten beim Eintreten:** In Absprache mit der Person, welche unsere Anpassung verhindert, wird ein geeignetes Zeitfenster gesucht, bei welchem wir uns nicht gegenseitig stören.

6.1.8 Risiko 8: Datenverlust

Ein Teammitglied löscht oder modifiziert versehentlich die Daten, welche auf dem GitLab Repository gespeichert sind.

- **Auswirkungen:** Die Daten müssen zurückgeholt werden.
- **Maximaler Schaden:** 8h
- **Eintrittswahrscheinlichkeit:** 20%
- **Gewichteter Schaden:** 1.6h
- **Vorbeugung:** Die Daten vom Repository werden bei beiden Teammitgliedern lokal gespeichert.
- **Verhalten beim Eintreten:** Die Daten werden neu auf das GitLab Repository gepusht oder der Commit wird rückgängig gemacht.

6.2 Risikomatrix

Wahrscheinlichkeit / Auswirkung	1 - Unmöglich	2 - Unwahrscheinlich	3 - Möglich	4 - Wahrscheinlich	5 - Sehr Wahrscheinlich
5 - Sehr Hoch					
4 - Hoch		R2			
3 - Mittel		R6	R3	R1	
2 - Gering				R4, R5	
1 - Sehr Gering		R7, R8			

Tabelle 6.1: Risikomatrix

6.3 Erkenntnisse

Der maximal gewichtete Schaden beträgt 75.2h. Aufgrund der frühen Erkennung von Problemen sollte es möglich sein, diesen Schaden zu minimieren. Wenn im schlimmsten Fall mehrere oder alle Risiken eintreffen, muss ein Treffen mit der Betreuungsperson und dem Kunden gehalten werden, um das Projekt neu zu evaluieren.

Teil II

Vorstudie

7 Übersicht

Im Abschnitt *Vorstudie* werden die für die Umsetzung der Arbeit notwendigen Informationen zusammengetragen. Ausserdem werden Anforderungen, welche an das Endergebnis gestellt werden, definiert. Zu guter Letzt werden die erkannten Herausforderungen spezifiziert, analysiert und die daraus entstandenen Entscheidungen dokumentiert.

7.1 Ausgangslage

Da wir die Bachelorarbeit mit der LGT durchführen, resultieren einige Abhängigkeiten, welche nicht mehr hinterfragt beziehungsweise evaluiert werden sollen.

Zu diesen Abhängigkeiten gehört zum Ersten der Cloud Provider. Die LGT besitzt eine Private Cloud von Microsoft Azure. Zusätzlich zum produktiven LGT Tenant, existiert ein Test Tenant, welcher zum evaluieren und ausprobieren von unbekannten Funktionalitäten dienen soll. Wir arbeiten während unserer Bachelorarbeit auf dem Test Tenant, mit dem Ziel die Applikation am Ende des Projektes auf den produktiven Tenant zu migrieren.

Die zweite Abhängigkeit ist das Code-Repository, für welches die LGT eine On Premise GitLab Instanz verwendet. Aus Simplizität arbeiten wir allerdings auf der öffentlichen GitLab Instanz. Aufgrund der Kompatibilität von GitLab können wir somit gewährleisten, dass unser Code ohne Probleme auf die LGT Instanz migriert werden kann.

Bei der LGT verwenden die meisten Mitarbeiter Visual Studio Code als IDE. Als weitere Abhängigkeit soll aus diesem Grund eine Visual Studio Code ähnliche IDE in der Cloud angeboten werden.

8 Anforderungsanalyse

In diesem Kapitel geht es darum, die Anforderungen an die Applikation aufzunehmen und diese anschliessend zu spezifizieren sowie zu dokumentieren. Hierbei ist eine gute Zusammenarbeit mit dem Kunden sehr wichtig, damit Missverständnisse und daraus resultierende Anpassungen vermieden werden können.

8.1 Lizenzierung

Aufgrund der Tatsache, dass die LGT keine Anforderungen an die Lizenzierung stellt, sind wir in deren Wahl frei. Die Entscheidung wurde dementsprechend innerhalb des Entwicklerteams besprochen und es wurde die MIT-Lizenz gewählt. Diese Lizenz ist sehr offen und entwicklerfreundlich. Dadurch wird verhindert, dass Lizenzbedingungen allfällig verwendeter Software-Bibliotheken verletzt werden. Es wurde ebenfalls entschieden, den Quellcode öffentlich einsehbar zu machen, damit andere Firmen oder Privatpersonen auch von diesem profitieren können. Auch können dadurch etwaige Fehler mithilfe der Community schneller erkannt werden und es stehen den Entwicklern mehr Tools ohne Zusatzkosten zur Verfügung.

8.2 Umfrage LGT

Bei der LGT wurde eine Umfrage über das Bedürfnis einer Parkplatz-Sharing Lösung bei den Mitarbeitern an den Standorten Vaduz und Bendern (FL) durchgeführt. Anhand dieser Umfrageergebnisse ist deutlich ersichtlich, dass das Bedürfnis für so eine Lösung, sowie auch die Bereitschaft etwas dafür zu bezahlen, vorhanden ist.

Weiter ist aus den Ergebnissen zu erkennen, dass die Häufigkeit der Freigabe sowie der Dauer einer Buchung sehr unterschiedlich sein können.

Um möglichst viele Anforderungen und Bedürfnisse der Mitarbeiter abdecken zu können, müssen folgende Punkte in den Anforderungen definiert und nach Möglichkeit umgesetzt werden.

- Freigabe von Parkplätzen
- Buchung von Parkplätzen
- Buchung über mehrere Tage möglich

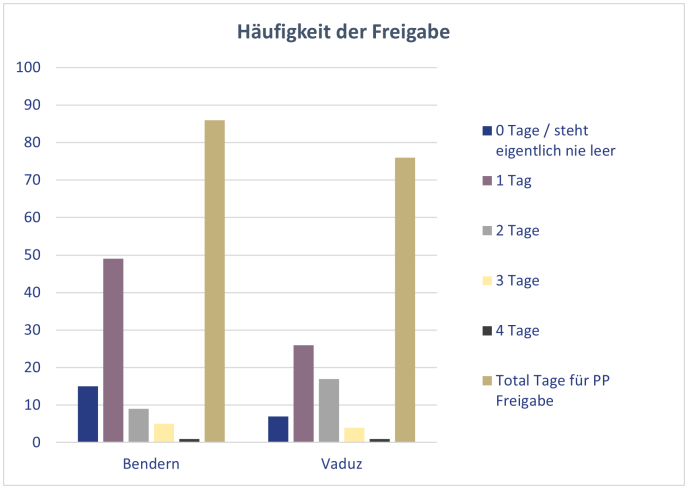


Abbildung 8.1: Dauer der Parkplatzbuchung gemäss Umfrage

Im nächsten Kapitel Funktionale Anforderungen (Use Cases) gehen wir auf alle genannten und noch weitere Anforderungen ein und definieren sie mit verschiedenen möglichen Ausbaustufen.

8.3 Funktionale Anforderungen (Use Cases)

In diesem Kapitel werden die funktionalen Anforderungen an die Applikation definiert. Diese wurden in einem ersten Schritt vom Entwicklerteam erarbeitet und werden in einem zweiten Schritt zusammen mit der LGT finalisiert. In der Abbildung 8.2 sind die Use Cases grafisch dargestellt. Die in der Grafik dargestellten Aktoren werden in Kapitel 8.3.1 noch genauer spezifiziert.

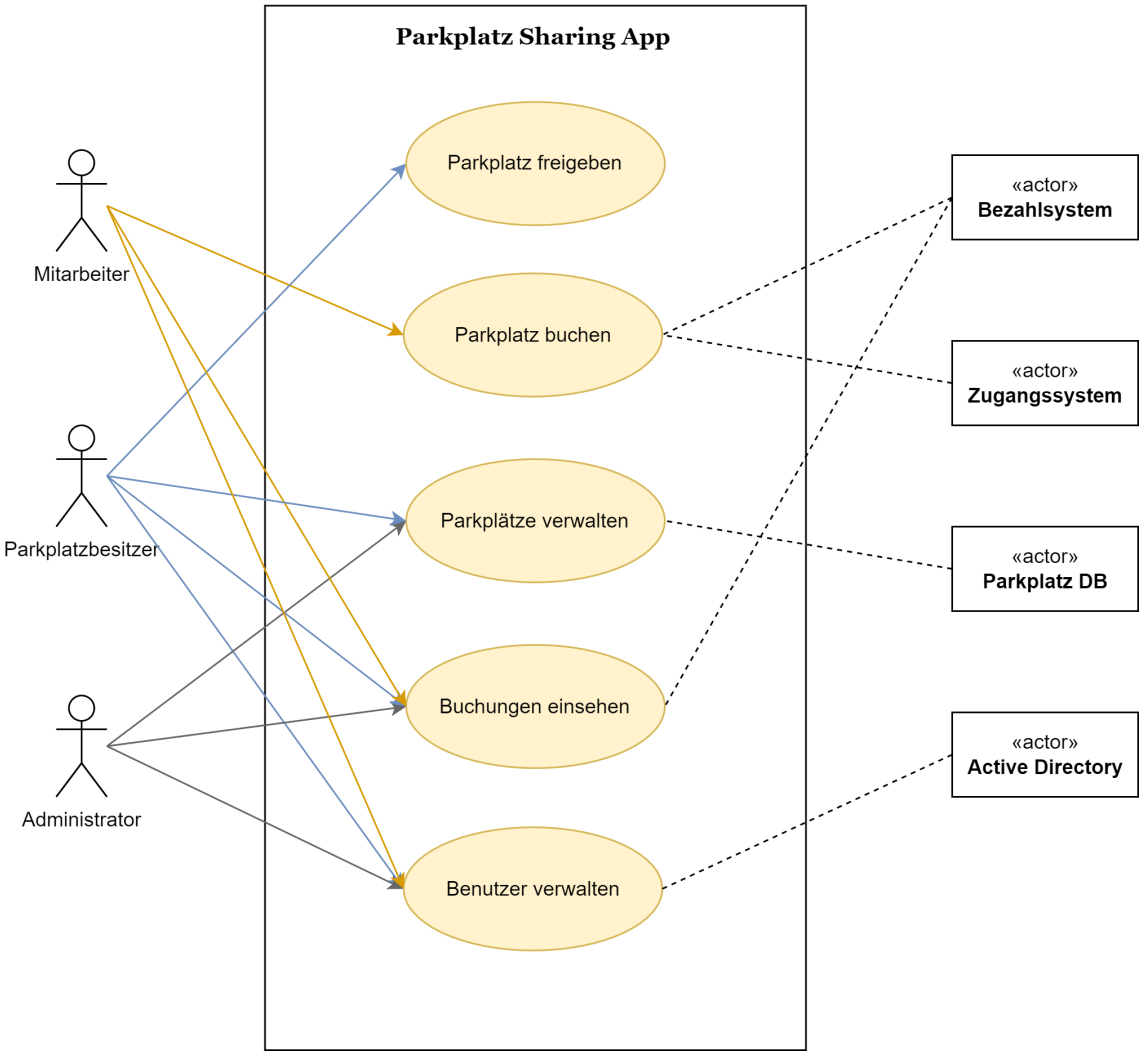


Abbildung 8.2: Übersicht Use Cases

8.3.1 Aktoren

Benutzer	Der Benutzer fungiert als Überbegriff für alle anderen genannten Aktoren.
Mitarbeiter	Der Mitarbeiter will einen freigegebenen Parkplatz buchen. Zu seinen applikationsrelevanten Tätigkeiten gehört die Buchung eines freigegebenen Parkplatzes.
Parkplatzbesitzer	Der Parkplatzbesitzer besitzt einen fixen Parkplatz, welcher er für andere freigeben will. Zu seinen applikationsrelevanten Tätigkeiten gehört die Freigabe von Parkplätzen.
Administrator	Der Administrationsmitarbeiter ist für die Verwaltung der Applikation zuständig. Zu den applikationsrelevanten Tätigkeiten des Administrationsmitarbeiters gehört die Verwaltung der Parkplätze.

Tabelle 8.1: Aktoren

8.3.2 UC01 – Parkplatz freigeben

Mithilfe der Applikation muss es möglich sein, einen fixen Parkplatz zur Buchung freizugeben.

8.3.2.1 US01.01 – Parkplatz freigeben

Als Parkplatzbesitzer möchte ich meinen Parkplatz für andere freigeben, damit diese ihn buchen können.

Ausbaustufe – Einfach
Ein Parkplatzbesitzer muss einen Parkplatz freigeben können.

Ausbaustufe – Begrenzt
Ein Parkplatzbesitzer soll nur seinen Parkplatz freigeben können.

Ausbaustufe – Mitarbeiterbasiert
Ein Parkplatzbesitzer kann seinen Parkplatz für ausgewählte Personen freigeben.

8.3.2.2 US01.02 – Parkplatzgebühr erhalten

Als Parkplatzbesitzer möchte ich für meinen freigegebenen Parkplatz eine Entschädigung erhalten.

Ausbaustufe – Einfach
Ein Parkplatzbesitzer soll einen fixen Betrag bei einer Buchung erhalten.

Ausbaustufe – Standortabhängig
Ein Parkplatzbesitzer kann einen, auf den Standort des Parkplatzes abhängigen Betrag, bei einer Buchung erhalten.

Ausbaustufe – Zeitbasiert
Ein Parkplatzbesitzer kann einen Stundentarif, für die Zeit bei welchem der Parkplatz gebucht wurde, erhalten.

8.3.3 UC02 – Parkplatz buchen

Mithilfe der Applikation muss es möglich sein, einen freigegebenen Parkplatz zu buchen.

8.3.3.1 US02.01 – Parkplatz buchen

Als Mitarbeiter möchte ich einen freigegebenen Parkplatz buchen.

Ausbaustufe – Einfach
Ein Mitarbeiter muss einen freigegebenen Parkplatz buchen können.

Ausbaustufe – Begrenzt
Ein Mitarbeiter soll einen ausgewählten, freigegebenen Parkplatz buchen können.

Ausbaustufe – Zugangssystem
Ein Mitarbeiter soll einen Parkplatz buchen können und sein Batch wird für diesen Tag über das Zugangssystem für den entsprechenden Parkplatz/ Parkgarage freigeschalten.

8.3.3.2 US02.02 – Parkplatzgebühr bezahlen

Als Mitarbeiter möchte ich meinen gebuchten Parkplatz bezahlen.

Ausbaustufe – Einfach
Um den gebuchten Parkplatz zu bezahlen, soll der Mitarbeiter eine Rechnung erhalten.

Ausbaustufe – Bezahlsystem
Ein Mitarbeiter kann direkt in der Applikation die Zahlung abwickeln.

Ausbaustufe – Plattform
Ein Mitarbeiter überweist den fälligen Betrag an eine Bezahlplattform und diese überweist den Betrag wiederum an den Parkplatzbesitzer. So läuft die Transaktion nicht Peer-to-Peer ab.

8.3.4 UC03 – Parkplätze verwalten

Mithilfe der Applikation muss es möglich sein, Parkplätze zu verwalten.

8.3.4.1 US03.01 – Parkplätze verwalten

Als Administrator möchte ich Parkplätze in der Applikation erfassen, bearbeiten und entfernen.

Ausbaustufe – Einfach
Ein Administrator muss einen Parkplatz manuell erfassen, bearbeiten und entfernen können.

Ausbaustufe – Automatisch
Ein Administrator kann die automatisch erfassten Parkplätze einsehen und verwalten.

8.3.4.2 US03.02 – Parkplätze zuweisen

Als Administrator möchte ich Parkplätze in der Applikation einem Parkplatzbesitzer zuweisen.

Ausbaustufe – Einfach
Ein Administrator muss einen Parkplatz manuell zuweisen können.

Ausbaustufe – Automatisch
Ein Administrator kann die automatisch zugewiesenen Parkplätze einsehen und verwalten.

8.3.4.3 US03.03 – Parkplatzgebühr festlegen

Als Administrator möchte ich die Gebühren für Parkplätze in der Applikation festlegen.

Ausbaustufe – Einfach
Ein Administrator soll eine Gebühr für einen Parkplatz manuell festlegen können.

Ausbaustufe – Automatisch
Ein Administrator kann die automatisch festgelegte Gebühr für einen Parkplatz einsehen und verwalten.

Ausbaustufe – Individuell
Ein Parkplatzbesitzer kann einen fixen Betrag wählen, welcher für seinen Parkplatz bezahlt werden muss.

8.3.5 UC04 – Buchungen einsehen

Mithilfe der Applikation soll es möglich sein, Buchungen einzusehen.

8.3.5.1 US04.01 – Buchungen einsehen

Als Benutzer möchte ich die Buchungen der Parkplätze einsehen.

Ausbaustufe – Einfach
Ein Benutzer soll alle Buchungen einsehen können.

Ausbaustufe – Mitarbeiter
Ein Mitarbeiter kann die von ihm getätigten Buchungen einsehen.

Ausbaustufe – Parkplatzbesitzer
Ein Parkplatzbesitzer kann die für seinen Parkplatz getätigten Buchungen einsehen.

Ausbaustufe – Administrator
Ein Administrator kann alle Buchungen einsehen.

Ausbaustufe – Nummernschild
Ein Benutzer soll für einen gebuchten Parkplatz das jeweilige Nummernschild auslesen können.

8.3.5.2 US04.02 – Bezahlstatus einsehen

Als Benutzer möchte ich die Bezahlstatus einsehen.

Ausbaustufe – Einfach
Ein Benutzer kann alle Bezahlstatus einsehen.

Ausbaustufe – Mitarbeiter
Ein Mitarbeiter kann die Bezahlstatus von ihm getätigten Buchungen einsehen.

Ausbaustufe – Parkplatzbesitzer
Ein Parkplatzbesitzer kann die Bezahlstatus für seinen Parkplatz getätigten Buchungen einsehen und bearbeiten.

Ausbaustufe – Administrator
Ein Administrator kann alle Bezahlstatus einsehen.

Ausbaustufe – Bezahlsystem
Ein Benutzer kann alle für ihn relevanten Bezahlstatus einsehen, welche direkt vom Bezahlsystem erfasst worden sind.

8.3.6 UC05 – Benutzer verwalten

Mithilfe der Applikation muss es möglich sein, Benutzer jeglicher Art zu verwalten.

8.3.6.1 US05.01 – Benutzer verwalten

Als Administrator möchte ich Benutzer in der Applikation erfassen, bearbeiten und entfernen.

Ausbaustufe – Einfach
Ein Administrator muss einen Benutzer manuell erfassen, bearbeiten und entfernen können.

Ausbaustufe – Automatisch
Ein Administrator kann die automatisch erfassten Benutzer einsehen und verwalten.

8.3.6.2 US05.02 – Profil bearbeiten

Als Benutzer möchte ich mein eigenes Profil bearbeiten, um meine Informationen auf dem neusten Stand zu halten.

Ausbaustufe – Einfach
Ein Benutzer soll sein Profil manuell bearbeiten können.

Ausbaustufe – Automatisch
Ein Benutzer kann die automatisch erfassten Informationen seines Profils einsehen und einige verwalten.

Ausbaustufe – Passwort
Ein Benutzer soll sein Passwort zurücksetzen oder anpassen können.

8.4 Nicht-funktionale Anforderungen

Die Anforderungen heissen „nicht funktional“, da diese nicht direkt mit der Erfüllung des Auftrages der Software zu tun haben, sondern mit deren Qualität. Die nicht funktionalen Anforderungen wurden mithilfe des FURPS-Model definiert. FURPS selbst ist ein Akronym aus dem Englischen und steht für die Softwarequalitätsmerkmale (Funktionalität, Benutzbarkeit, Zuverlässigkeit, Effizienz und Änderbarkeit). Es ist daher bei jedem NFR angegeben, in welche Kategorie des Akronyms es fällt. Die deutschen Begriffe der Kategorien wurden aus der ISO/IEC 9126-1:2001 Norm entnommen [2].

8.4.1 Cloud

Bei der Erhebung der nicht-funktionalen Anforderungen der Cloud IDE haben wir die für uns relevanten Punkte aus dem „Microsoft Azure Well-Architectured Framework“ [3] adaptiert und angepasst.

8.4.1.1 NFR01

- **Kategorie:** Availability
- **Beschreibung:** Der Azure Tenant soll zu 99.99% verfügbar sein ("four nines").

8.4.1.2 NFR02

- **Kategorie:** Resource Utilization
- **Beschreibung:** Für die GitLab CI/CD Pipeline stehen genügend shared/ dedizierte Runners mit der nötigen Kapazität zur Verfügung.

8.4.1.3 NFR03

- **Kategorie:** Resource Utilization/ Scalability
- **Beschreibung:** Der Azure Tenant hat genügend Ressourcen vorhanden, um Deployments starten zu können.

8.4.1.4 NFR04

- **Kategorie:** Capacity
- **Beschreibung:** Azure Ressourcen sollen ohne Einschränkung durch Budget der LGT benutzt werden können. Das heisst, wir können alle Dienste, welche wir als sinnvoll betrachten, auch nutzen.

8.4.1.5 NFR05

- **Kategorie:** Analyzability
- **Beschreibung:** Mit Measured Services sollen die verwendeten Azure Services überwacht werden können. Daraus können die Kosten abgeleitet sowie die aktuelle Auslastung der Ressourcen eingesehen werden.

8.4.2 Cloud Native Applikation

Für eine gute Cloud Applikation gibt es Empfehlungen von verschiedenen Quellen, welche sich grösstenteils überschneiden. Wir haben im Modul CldSol [4] die IDEAL Cloud Application Properties sowie die Twelve-Factor App kennengelernt. Aufgrund der generischen Auslegung der IDEAL Cloud Application Properties haben wir uns entschieden, bei den Requirements nur auf diese Empfehlungen einzugehen.

8.4.2.1 NFR06

- **Kategorie:** Functionality, Isolated-State [5]
- **Beschreibung:** Die Applikation wird *stateless* entwickelt, um die Skalierbarkeit mit Containern zu gewährleisten. Zudem werden dynamische Konfigurationen in Umgebungsvariablen und Konfigurationsfiles ausgelagert und im Sourcecode nicht hardcodiert.

8.4.2.2 NFR07

- **Kategorie:** Functionality
- **Beschreibung:** Daten können erst nach erfolgreicher Anmeldung ausgelesen werden.

8.4.2.3 NFR08

- **Kategorie:** Reliability
- **Beschreibung:** Tritt unerwartetes Verhalten auf, so wird dem Benutzer eine Fehlermeldung angezeigt.

8.4.2.4 NFR09

- **Kategorie:** Reliability, Loose Coupling [5]
- **Beschreibung:** Die Applikation soll möglichst wenige Abhängigkeiten zwischen den verschiedenen Komponenten haben. Damit wird gewährleistet, dass einzelne Komponenten ohne grosse Anpassung der anderen ausgetauscht werden können.

8.4.2.5 NFR10

- **Kategorie:** Performance Efficiency, Automated Management [5]
- **Beschreibung:** CI/CD-Aufgaben müssen schnell und automatisiert erledigt werden. Dies wird mit einer GitLab CI/CD Pipeline sichergestellt.

8.4.2.6 NFR11

- **Kategorie:** Capacity, Elasticity [5]
- **Beschreibung:** Bei erhöhter Ressourcenauslastung soll die Applikation automatisch skalieren können. Dabei werden zusätzliche Ressourcen zur Leistungssteigerung hinzugefügt (*scale out*).

8.4.2.7 NFR12

- **Kategorie:** Usability
- **Beschreibung:** Ein Benutzer soll alle Aufgaben, welche gemäss den Use Cases aus Kapitel 8.3 dem Aktor „Mitarbeiter“ oder „Parkplatzbesitzer“ zugeordnet sind, selbstständig ausführen können. Hierbei soll keine zusätzliche Hilfestellung von aussen notwendig sein.

8.4.2.8 NFR13

- **Kategorie:** Usability
- **Beschreibung:** Da bei der LGT die offizielle Firmensprache Englisch ist, werden alle textuellen Informationen in Englisch angegeben.

8.4.2.9 NFR14

- **Kategorie:** Modularity, Distribution [5]
- **Beschreibung:** Die Applikation soll aus kleinen Komponenten (Modulen) bestehen, welche unabhängig voneinander funktionieren und aufgrund der tieferen Komplexität einfacher ausgetauscht werden können.

8.4.2.10 NFR15

- **Kategorie:** Modifiability (Maintainability)
- **Beschreibung:** Klassen sind nicht länger als 300 Zeilen.

8.4.2.11 NFR16

- **Kategorie:** Modifiability (Maintainability)
- **Beschreibung:** Methoden sind nicht länger als 45 Zeilen.

8.4.2.12 NFR17

- **Kategorie:** Modifiability (Maintainability)
- **Beschreibung:** Zeilen sind nicht länger als 80 Zeichen.

8.4.2.13 NFR18

- **Kategorie:** Modifiability (Maintainability)
- **Beschreibung:** Der McCabe-Wert jeder Methode in der Codebase liegt unter 10.

8.4.2.14 NFR19

- **Kategorie:** Modifiability (Maintainability)
- **Beschreibung:** Keine Methode in der Codebase hat mehr als fünf Argumente.

8.4.2.15 NFR20

- **Kategorie:** Modifiability (Maintainability)
- **Beschreibung:** Die Testabdeckung des Codes beträgt mindestens 80%.

8.4.2.16 NFR21

- **Kategorie:** Modifiability (Maintainability)
- **Beschreibung:** Alle möglichen Aufrufszenarien der API werden dokumentiert.

8.4.2.17 NFR22

- **Kategorie:** Modifiability (Maintainability)
- **Beschreibung:** Es wird eine Installationsanleitung sowie eine Wartungsanleitung erstellt.

8.4.2.18 NFR23

- **Kategorie:** Portability
- **Beschreibung:** Zusätzliche Module können einfach ergänzt werden.

8.4.3 Cloud IDE

8.4.3.1 NFR24

- **Kategorie:** Availability
- **Beschreibung:** Die Cloud IDE soll jederzeit auf Abruf (On Demand) als Software-as-a-Service (SaaS) bereitstehen.

8.4.3.2 NFR25

- **Kategorie:** Availability
- **Beschreibung:** Bei hoher Auslastung soll die IDE automatisch skalieren.

8.4.3.3 NFR26

- **Kategorie:** Reliability
- **Beschreibung:** Die Cloud IDE soll verlässlich sein. Damit wird gewährleistet, dass das System sich nach einem Ausfall automatisch wieder in einen funktionsfähigen Zustand versetzen kann.

8.4.3.4 NFR27

- **Kategorie:** Adaptability
- **Beschreibung:** Die IDE Umgebung soll durch die Entwickler angepasst und um zusätzliche Plugins ergänzt werden können.

8.4.3.5 NFR28

- **Kategorie:** Analyzability
- **Beschreibung:** Die verwendeten Ressourcen sollen überwacht und gemessen werden, damit daraus die Kosten ersichtlich sind.

8.4.3.6 NFR29

- **Kategorie:** Usability
- **Beschreibung:** Die IDE soll den Entwickler mit einer IntelliSense sowie einem Debugger unterstützen.

8.4.3.7 NFR30

- **Kategorie:** Compatibility
- **Beschreibung:** Die Cloud IDE unterstützt verschiedene Programmiersprachen.

8.4.3.8 NFR31

- **Kategorie:** Compatibility
- **Beschreibung:** Die Dependencies und SDKs sollen einfach aktualisiert resp. hinzugefügt werden können.

8.4.3.9 NFR32

- **Kategorie:** Operability
- **Beschreibung:** Es soll ein Version Control System (VCS) integriert werden können.

8.5 Modellüberlegungen

In diesem Kapitel wird versucht ein erstes Modell zu erstellen. Dieses soll die benötigten Funktionalitäten, welche vom Kunden gewünscht wurden, abbilden. Hierzu wurde eine erste Skizze des Domain Models erstellt, welche alle benötigten Business-Objekte abbildet. Auch werden in diesem Kapitel erste Überlegungen zur Benutzeroberfläche gemacht und mit Wireframes grafisch abgebildet.

8.5.1 Domain Model

Im Domain Model der Problemdomain wurden alle Business-Objekte abgebildet, welche möglicherweise in der eigenen Datenbank benötigt werden. Dies ist abhängig von der jeweiligen Ausbaustufe 8.3. Es wurde ausserdem jedes Objekt textuell erläutert.

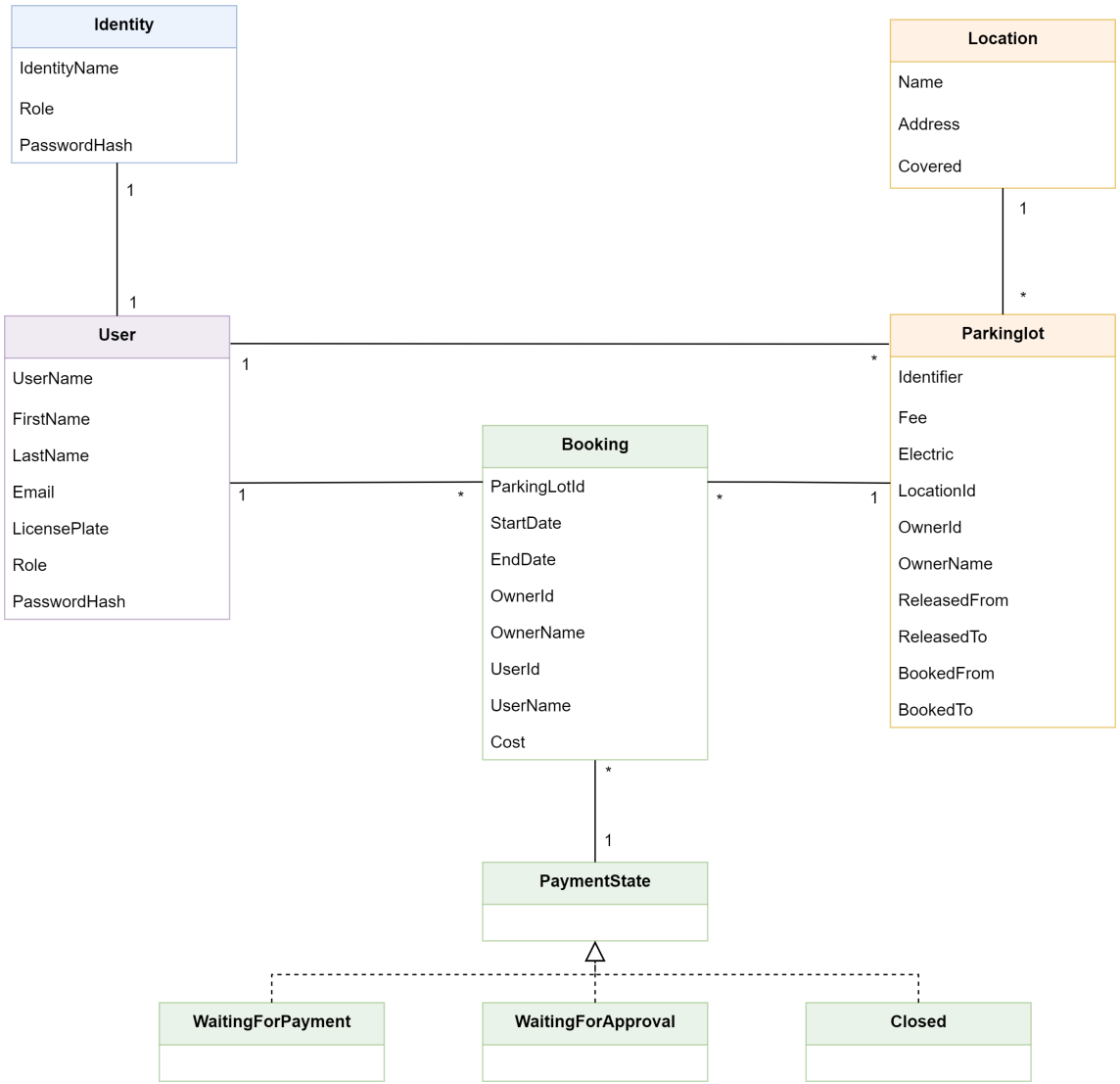


Abbildung 8.3: Übersicht geplantes Domain Model

Klasse	Beschreibung
Identity	Beschreibt eine Identity fürs Login des entsprechenden Users.
User	Beschreibt einen Benutzer, welcher die Applikation nutzt.
Location	Beschreibt einen Standort mit Parkplätzen und definiert Attribute für diese Parkplätze, z.B. ob diese überdacht sind.
Parkinglot	Beschreibt einen Parkplatz, welcher freigegeben, gebucht und verwaltet werden kann.
Booking	Beschreibt die Buchung eines Parkplatzes.
PaymentState	Beschreibt den aktuellen Bezahlstatus einer Buchung.

Tabelle 8.2: Beschreibung Domain Model

8.5.2 State Diagram

Im State Diagram wurde der State der Klasse Booking abgebildet. Mithilfe dessen ist ersichtlich, wie der Wert dieses Attributes zustandekommt.

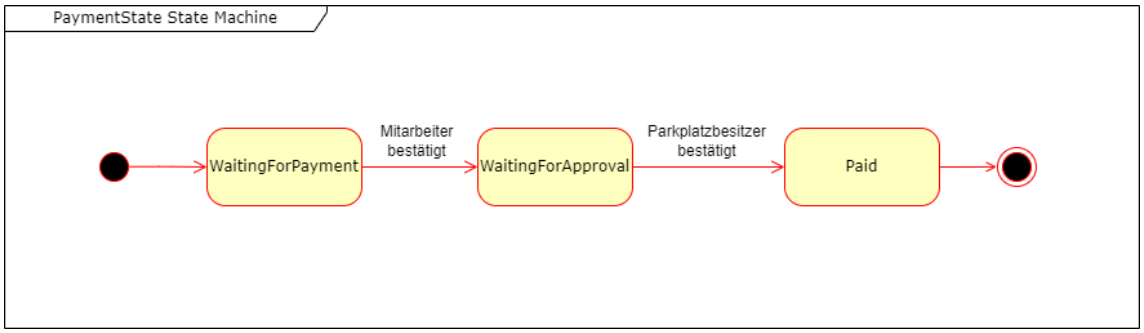


Abbildung 8.4: Übersicht State Diagram

8.5.3 Benutzeroberfläche

Die Benutzeroberfläche ist eines der wichtigsten Elemente für die Anwender, da eine gute Usability die Zufriedenheit um ein Vielfaches steigert. Aus diesem Grund wurde diese sorgfältig und durchdacht aufgebaut. Nachfolgend sind die Punkte dokumentiert, welche während der Planung relevant für die Benutzeroberfläche waren.

8.5.3.1 Design

Massgebend für das Design der Benutzeroberfläche sind die nicht-funktionalen Anforderungen in der Kategorie Usability, sowie das Corporate Design der LGT. Für das Corporate Design der LGT können wir uns an die zuständigen Mitarbeiter innerhalb der Firma wenden, welche uns bei der Einhaltung des Designs helfen werden. Ausserdem kann auf UX Experten zurückgegriffen werden, um die Usability so gut wie möglich zu programmieren. Sollten zur Hervorhebung oder Kategorisierung (z.B. Info- / Warn- und Fehlermeldungen) weitere Farben benötigt werden, wird auf die Farbpalette des verwendeten UI-Frameworks zurückgegriffen.

8.5.3.2 Wireframes

Zur verbesserten Veranschaulichung der funktionalen Anforderungen für den Kunden wurden Wireframes erstellt. Des Weiteren fungieren die Wireframes als Implementationsvorlage für die Benutzeroberfläche.

Die wichtigsten Funktionalitäten für Mitarbeiter und Parkplatzbesitzer sind im Format eines Smartphone Apps abgebildet. Für die administrativen Funktionalitäten, welche primär von Administratoren verwendet werden, wurden Wireframes im Desktop-Format erstellt.

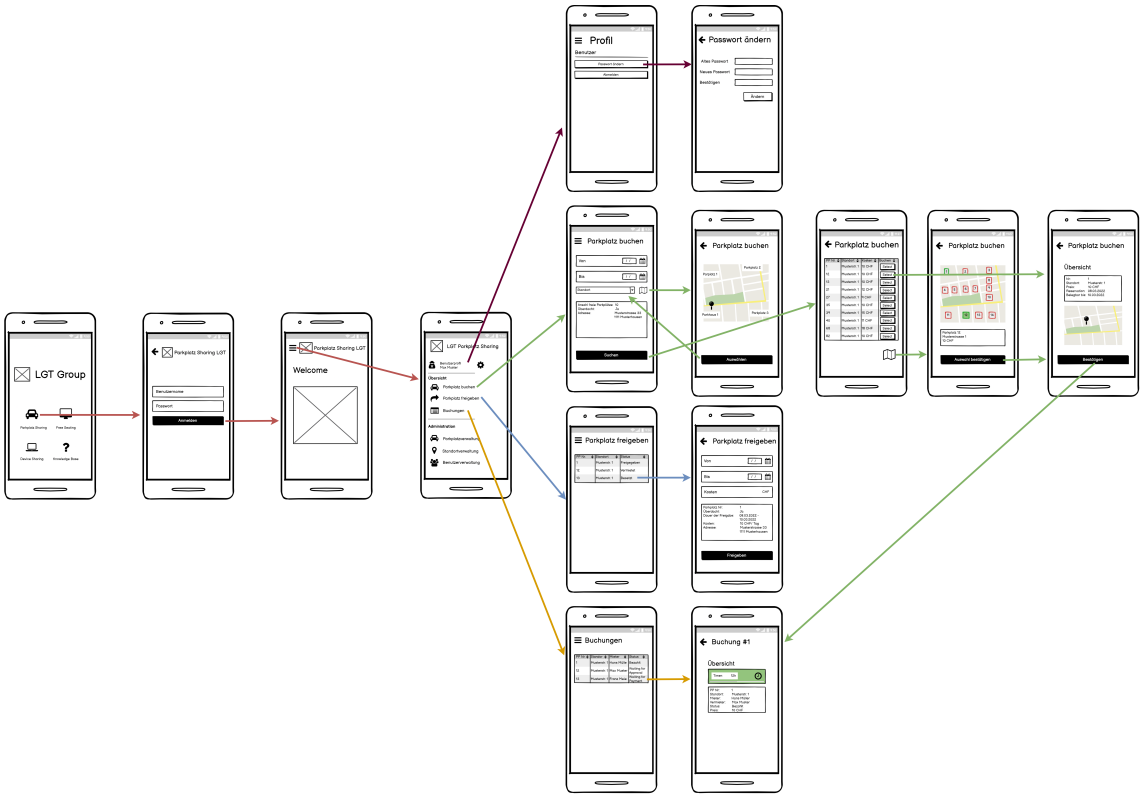


Abbildung 8.5: Übersicht Wireframes App

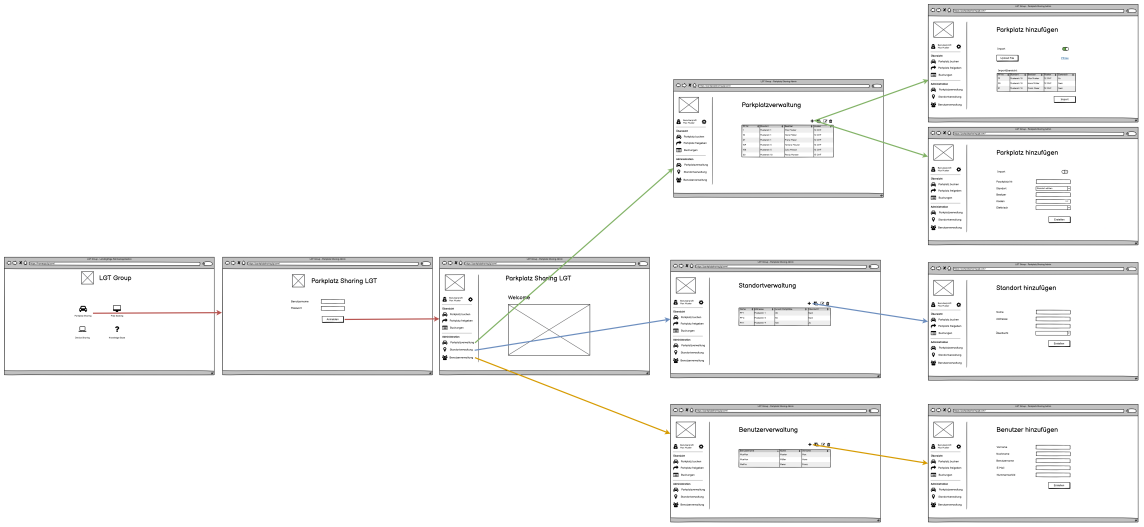


Abbildung 8.6: Übersicht Wireframes Administrations-Webseite

Eine Übersicht aller Wireframes ist unter Kapitel G.1.1 zu finden.

Teil III

Architektur

9 | Übersicht

Im Teil „Architektur“ werden zum Einen die Entwicklungstools evaluiert und dokumentiert, welche für die Umsetzung des Projekts benötigt werden. Zum Anderen wird in diesem Teil die gesamte Architektur des Projekts beschrieben.

Die zugrundeliegenden Entscheidungen der Wahl von Technologien und Architekturprinzipien, werden unter Berücksichtigung der Evaluationen aus der Vorstudie und in Abstimmung mit der Implementation des technischen Prototyps aus diesem Kapitel getroffen. Allen Entscheidungen liegt hierbei der Grundsatz des Single-Responsibility-Prinzip zugrunde, welches das Projekt stark beeinflusst hat. Sollten andere Faktoren bei Entscheidungen eine wichtige Rolle gespielt haben, so ist dies dokumentiert. Zusätzlich werden einige Entscheidungen im Abschnitt Herausforderungen weiter ausgeführt.

10 Evaluation

10.1 Client App Evaluation

Im Rahmen dieser Arbeit wird eine Cloud Native Applikation entwickelt, welche auf beiden im Moment gängigen Betriebssystemen für Mobiltelefone (Android und iOS) laufen muss und die Verwaltung über einen Webbrowser gemacht werden kann. In diesem Kapitel werden diverse solcher Client Apps angeschaut und miteinander verglichen. Das Ziel ist hierbei, die am besten geeignete Client App, für die im Rahmen dieser Arbeit entwickelte Applikation, zu finden.

10.1.1 Einschränkungen

Da sich unsere Arbeit auf eine Cloud Native Applikation Entwicklung richtet, fokussieren wir uns bei dieser Evaluation auf Web Apps. Dadurch muss beim Benutzer nichts installiert werden und die App kann vollständig in der Cloud betrieben werden. Die Anforderung *Unterstützung für iOS, Android und Windows als Betriebssystem* wird somit von allen vorgestellten Technologien erfüllt, da die Applikation über einen Browser läuft und dieser bekanntlich auf allen Betriebssystemen installiert werden kann.

10.1.2 Anforderungen

Zu Beginn der Evaluation werden die Anforderungen, welche an die zu verwendende Client App gestellt werden, definiert:

- State-of-the-art Technologie
- Funktionalität soll auf verschiedenen Microservices verteilt sein.
- Unterstützung für iOS, Android und Windows als Betriebssystem
- Microservices können auf einem anderen Microservice eingebunden werden.

10.1.3 Vergleich

Im Rahmen dieser Evaluation werden die folgenden Client Apps miteinander verglichen:

- Serverframeworks [6]
- HTML5 mit/ohne JS-Frameworks [7]
- SPA [8]
- PWA [9]

10.1.3.1 Serverframeworks

Ein Serverframework wird hauptsächlich für das Backend eingesetzt. Es kann allerdings auch Server-Side Rendering betreiben und deshalb ein Frontend zur Verfügung stellen. Dies kann mit verschiedenen Frameworks, wie zum Beispiel ASP.NET, Java EE Framework oder ExpressJS, umgesetzt werden.

Erfüllung der Anforderungen

- State-of-the-art Technologie.
  - Grundsätzlich sind Serverframeworks State-of-the-art, allerdings nur für das Backend. Für das Frontend gibt es viel einfachere und schönere Möglichkeiten eine Client App zu gestalten, welche auch mehr Funktionalitäten anbieten.
- Funktionalität soll auf verschiedenen Microservices verteilt sein.
  - Dies ist möglich, in dem verschiedene Webserver betrieben werden, auf welchem jeweils ein kleiner Teil der gesamten Funktionalität läuft.
- Microservices können auf einem anderen Microservice eingebunden werden.
  - Nur mit Mehraufwand möglich. Microservices können verlinkt werden, aber nicht eingebunden.

Fazit
Da diese Art Frontend Web Apps zu entwickeln nicht State-of-the-art ist, kommt es für uns nicht in Frage, unsere Applikation so aufzubauen.

10.1.3.2 HTML5 mit/ ohne JS-Frameworks

Eine HTML5 Web App besteht aus drei verschiedenen Technologien. Aus HTML5 für die Seitenstruktur, aus CSS für die Eigenschaften sowie das Design und aus JavaScript für die Programmierlogik. Um eine HTML5 Web App zu programmieren, können verschiedene Frameworks, wie zum Beispiel Webcomponents, JQuery oder Bootstrap, verwendet werden.

Erfüllung der Anforderungen

- State-of-the-art Technologie
  - Heutzutage gibt es viele Technologien, welche diese genannten Files erstellen können. Damit kann der ganze Prozess vereinfacht werden. Diese Methode gehört also nicht mehr zu State-of-the-art.
- Funktionalität soll auf verschiedenen Microservices verteilt sein.
  - Dies ist möglich, indem verschiedene Webserver betrieben werden, auf welchem jeweils ein kleiner Teil der gesamten Funktionalität läuft.
- Microservices können auf einem Microservice eingebunden werden.
  - Nur mit Mehraufwand möglich. Microservices können verlinkt werden, aber nicht eingebunden.

Fazit
Da diese Art Web Apps zu entwickeln nicht mehr State-of-the-art ist, kommt es für uns nicht in Frage, unsere Applikation so aufzubauen.

10.1.3.3 SPA

Eine Single Page Application besteht aus einem einzigen Web Dokument, welches beim initialen Laden der Webseite zum Client gesendet wird. Danach läuft die gesamte Logik auf dem Client und es werden jeweils nur Anfragen für Informationen ans Backend gesendet und dynamisch geladen. Dies hat zur Folge, dass die Serverlast reduziert wird. Um diese Art Client Apps zu programmieren, gibt es verschiedene Frameworks, wie zum Beispiel Angular, React oder Vue.

Erfüllung der Anforderungen

- State-of-the-art Technologie
  - Die Art Frontends zu bauen ist definitv State-of-the-art. Für Webseiten wird hauptsächlich SPA genutzt.
- Funktionalität soll auf verschiedenen Microservices verteilt sein.
  - Dies ist möglich, in dem verschiedene SPA erstellt werden, auf welchem jeweils ein kleiner Teil der gesamten Funktionalität läuft. Die Unterstützung für die modulare Entwicklung ist in den Frameworks gross. Diese haben bereits Patterns implementiert und die Entwickler müssen sich nicht mehr um solche Anforderungen selbst kümmern.
- Microservices können auf einem Microservice eingebunden werden.
  - Es wird jeweils eine JavaScript Datei generiert, welche einfach eingebunden werden kann.

Fazit
Die SPA Architektur ist sehr verbreitet und wird in der heutigen Zeit hauptsächlich eingesetzt. Es gibt sehr viele Frameworks, welche unterstützen Clean Code zu schreiben. Wir haben zudem bereits Erfahrungen mit verschiedenen Frameworks gemacht und benötigen keine lange Einarbeitungszeit mehr.

10.1.3.4 PWA

Eine Progressive Web App ist grundsätzlich gleich aufgebaut wie eine SPA. Die wesentlichen Unterschiede sind die Service Workers, welche für Offline Verfügbarkeit und Push Notifications verantwortlich sind sowie das Web App Manifest. Das Web App Manifest enthält alle nötigen Metadaten von der PWA in einer JSON Datei. Um diese Art Client Apps zu programmieren, gibt es die gleichen Frameworks wie beim SPA-Ansatz.

Erfüllung der Anforderungen

- State-of-the-art Technologie
  - Die Art Frontends zu bauen ist definitv State-of-the-art. Für Webseiten wird der PWA-Ansatz immer grösser.
- Funktionalität soll auf verschiedenen Microservices verteilt sein.
  - Dies ist möglich in dem verschiedene PWA erstellt werden, auf welchem jeweils ein kleiner Teil der gesamten Funktionalität läuft. Die Unterstützung für die modulare Entwicklung ist in den Frameworks gross. Diese haben bereits Patterns implementiert und die Entwickler müssen sich nicht mehr um solche Anforderungen selbst kümmern.
- Microservices können auf einem Microservice eingebunden werden.
  - Es wird jeweils eine JavaScript Datei generiert, welche einfach eingebunden werden kann.

Fazit
Die PWA Applikation ist hinter SPA die am zweitmeisten verbreitete Art. Sie gewinnt momentan sehr an Popularität, da viele Vorteile einer SPA wiederspiegelt werden. Zusätzlich kann mit einer PWA Applikation auch Offline gearbeitet werden und die Push-Benachrichtigungen funktionieren ohne Mehraufwand.

10.1.4 Ergebnis

Wir haben uns für die Entwicklung einer PWA Applikation entschieden. Die PWA Applikation erfüllt alle Anforderungen ohne zusätzliche Services oder Libraries. Weiter ist die Ladezeit einer PWA Applikation extrem gering und verfügt über einen besseren Sicherheitsaspekt, da es sich bei PWAs immer um HTTPS-Webseiten handelt. Das User Interface wiederspiegelt ein natives Applikationsgefühl, da eine PWA Applikation auf dem Desktop angeheftet und wie eine native App benutzt werden kann.

10.2 Frontend Framework Evaluation

In diesem Kapitel werden diverse Frameworks zur Entwicklung einer PWA Applikation angeschaut und miteinander verglichen. Das Ziel ist hierbei, das am besten geeignete Framework, für die im Rahmen dieser Arbeit entwickelte Applikation, zu finden.

10.2.1 Einschränkungen

Es gibt sehr viele Frameworks auf dem Markt, welche unsere Anforderungen potentiell erfüllen können. Aufgrund der begrenzten Zeit und des marginalen Nutzens wurden hier nicht alle uns bekannten Frameworks verglichen, sondern lediglich einige wenige. Bei der Auswahl der in diesem Kapitel aufgeführten Frameworks haben wir primär auf die Verbreitung und somit auf die Langlebigkeit und den Support geachtet. Je mehr ein Framework genutzt wird, desto wahrscheinlicher ist es, dass dieses noch lange bestehen wird und gute Antworten auf gängige Fragen vorhanden sind. Die ausgewählten Frameworks sind alles State-of-the-art Technologien, deshalb wird diese Anforderung bereits von allen erfüllt.

10.2.2 Anforderungen

Zu Beginn der Evaluation werden die Anforderungen, welche an das zu verwendende Framework gestellt werden, definiert:

- Unterstützung von Microservices
- Unterstützung eines Themes
- Dokumentation des Frameworks
- Grosse und hilfreiche Community

10.2.3 Vergleich

Im Rahmen dieser Evaluation werden die folgenden Technologien miteinander verglichen:

- Angular [10]
- React [11]
- Vue [12]

10.2.3.1 Angular

Angular ist ein TypeScript-basiertes Framework, welches es erlaubt, ganze Produktfamilien zu entwicklen. Egal ob eine Web, Mobile oder Desktop Applikation benötigt wird.

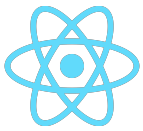
Erfüllung der Anforderungen

- Unterstützung von Microservices
  - Framework unterstützt Microservice Architektur mithilfe von Modulen.
- Unterstützung eines Themes
  - Theme wird mithilfe von zusätzlicher Bibliothek unterstützt.
- Dokumentation des Frameworks
  - Angular besitzt eine umfangreiche und hilfreiche Dokumentation.
- Grosse und hilfreiche Community
  - Die Community ist eher kleiner, da das Framework komplexer ist.

Fazit
Angular wird hauptsächlich für die Entwicklung von ganzen Produktfamilien benutzt. Es fordert einen initialen Mehraufwand um das Framework zu verstehen. Damian hat bereits Erfahrungen mit dem Framework, Gian allerdings noch nicht. Deshalb müsste Gian diesen initialen Mehraufwand auf sich nehmen, um sich in das Framework einzuarbeiten.

Disclaimer: Alle Logos, welche in der Evaluation aufgeführt sind, gehören den jeweiligen Herstellern und wurden von deren offiziellen Webseiten genommen.

10.2.3.2 React



React ist eine JavaScript Bibliothek, welche für das Bilden von interaktiven Benutzeroberflächen benutzt werden kann. React ist komponentenbasiert, das heisst mehrere Komponenten, welche hierarchisch aufgebaut sind, bilden ein gesamtes UI. Ausserdem ist React deklarativ, was dazu führt, dass nur Komponenten neu gerendert werden, welche sich verändert haben.

Erfüllung der Anforderungen

- Unterstützung von Microservices
  - Bei der Kompilation wird eine JavaScript Datei generiert, welche in einem anderen Microservice eingebettet werden kann.
- Unterstützung eines Themes
  - Theme wird mithilfe von zusätzlicher Bibliothek unterstützt.
- Dokumentation des Frameworks
  - Die Dokumenation ist etwas minimalistisch. Die Informationen muss man sich im Internet zusammensuchen.
- Grosse und hilfreiche Community
  - React besitzt die grösste Community der Frontend Framework. Deshalb findet man auch viele Hilfestellungen und Diskussionen im Internet.

Fazit
React ist für das Entwicklerteam ein bereits bekanntes Framework, da schon Projekte mit diesem Framework durchgeführt wurden. Ausserdem ist die Community von React sehr gross, was dabei hilft, ein Problem schnellstmöglich zu lösen.

10.2.3.3 Vue

Vue ist ein JavaScript-basiertes Framework, welches für Webanwendungen eingesetzt wird. Vue ist progressiv und inkrementell adaptierbar. Dies erlaubt es, die Applikation nach eigenen Ansprüchen zu entwickeln.

Erfüllung der Anforderungen

- Unterstützung von Microservices
  - Bei der Kompilation wird eine JavaScript Datei generiert, welche in einem anderen Microservice eingebettet werden kann.
- Unterstützung eines Themes
  - Theme wird mithilfe von zusätzlicher Bibliothek unterstützt.
- Dokumentation des Frameworks
  - Die Dokumenation ist etwas minimalistisch. Die Informationen muss man sich im Internet zusammensuchen.
- Grosse und hilfreiche Community
  - Das Framework besitzt eine etwas kleinere Community. Deshalb ist es schwieriger an Informationen zu gelangen.

Fazit
Vue ist für uns beide ein unbekanntes Framework. Allerdings haben wir von unseren Mitstudenten viel Positives gehört und die Einarbeitung in das Framework sei einfach.

10.2.4 Ergebnis

Wir haben uns aufgrund unserer Vorkenntisse und der Verbreitung des Frameworks für React entschieden.

10.3 Backend Framework Evaluation

In diesem Kapitel werden diverse Frameworks zur Entwicklung des Backends angeschaut und miteinander verglichen. Das Ziel ist hierbei, das am besten geeignete Framework, für die im Rahmen dieser Arbeit entwickelte Backend-Applikation, zu finden.

10.3.1 Einschränkungen

Es gibt sehr viele Frameworks auf dem Markt, welche unsere Anforderungen potentiell erfüllen könnten. Aufgrund der begrenzten Zeit und des marginalen Nutzens wurden hier nicht alle uns bekannten Frameworks verglichen, sondern lediglich einige wenige. Bei der Auswahl der in diesem Kapitel aufgeführten Frameworks haben wir auf die Verbreitung und somit auf die Langlebigkeit sowie den Support geachtet. Je mehr ein Framework genutzt wird, desto wahrscheinlicher ist es, dass dieses noch lange bestehen wird und gute Antworten auf gängige Fragen vorhanden sind. Ausserdem haben wir bei der Beschränkung der Frameworks darauf geachtet, welche wir bereits kennen und somit nicht von Beginn an das Wissen aufbauen müssen. Zusätzlich sind alles Frameworks State-of-the-art, weshalb diese Anforderung von allen erfüllt wird.

10.3.2 Anforderungen

Zu Beginn der Evaluation werden die Anforderungen, welche an die zu verwendende Technologie gestellt werden, definiert:

- Unterstützung von Microservices
- Sinnvolle Auswahl von Bibliotheken
- Dokumentation des Frameworks
- Grosse und hilfreiche Community

10.3.3 Vergleich

Im Rahmen dieser Evaluation werden die folgenden Technologien miteinander verglichen:

- ASP.NET [13]
- ExpressJS [14]
- Spring Boot [15]

10.3.3.1 ASP.NET

ASP.NET Core ist ein modulares Open-Source Framework, welches auf der .NET-Plattform aufgebaut ist. Es läuft allerdings auch auf dem .NET Framework, weshalb es die Ablösung von ASP.NET sein soll. Wir verwenden in dieser Arbeit den Begriff ASP.NET, um das Framework ASP.NET Core zu beschreiben. Eigentlich ist ASP.NET der Vorgänger von ASP.NET Core. Allerdings kann wie erwähnt ASP.NET Core auf beiden Plattformen ausgeführt werden.

Erfüllung der Anforderungen

- Unterstützung von Microservices
  - Mithilfe der Containertechnologie Docker kann der Service in einem Container laufen gelassen werden und deshalb ist die Unterstützung von Microservices gegeben. Ausserdem bietet die Projektstruktur zusammen mit dem Solution File von Visual Studio eine ausgezeichnete Grundlage für die Microservice Architektur. Es gibt auch noch zusätzliche Tools von Microsoft wie Tye, welche die Entwickler unterstützen.
- Sinnvolle Auswahl von Bibliotheken
  - Bei diesem Framework gibt es gute Bibliotheken, wie zum Beispiel das Entity Framework. Ausserdem werden viele Bibliotheken von Microsoft entwickelt und dadurch die Qualität sichergestellt.
- Dokumentation des Frameworks
  - Microsoft bietet eine sehr gute und hilfreiche Dokumenation zu ASP.NET, welche ständig überarbeitet wird.
- Grosse und hilfreiche Community
  - Bei diesem Framework besteht die Community grundsätzlich aus den .NET Entwicklern und Liebhabern. Diese gibt es mittlerweile bis zur Genüge, vorallem seit .NET Open-Source ist. Microsoft bietet auch eigene Foren an, um zu diskutieren bzw. Probleme miteinander zu lösen.

Fazit
ASP.NET kennen wir schon aus unserer Studienarbeit. Wir waren sehr zufrieden mit dem Framework. Es erfüllt alle unsere Anforderungen und ist auf der Cross Plattform .NET Core aufgebaut.

10.3.3.2 ExpressJS

ExpressJS ist ein Open-Source Framework, welches auf der NodeJS-Plattform aufbaut. Es erweitert NodeJS, um das Entwickeln einer Webanwendung zu vereinfachen. ExpressJS ist eines der populärsten und am weitesten verbreiteten Frameworks überhaupt, da es mit JavaScript entwickelt wird und diese Programmiersprache sehr weit verbreitet ist.

Erfüllung der Anforderungen

- Unterstützung von Microservices
  - Mithilfe der Containertechnologie Docker kann der Service in einem Container laufen gelassen werden und deshalb ist die Unterstützung von Microservices gegeben. Die Projekte sind im Vergleich zu den anderen Frameworks leichtgewichtig und eignen sich deshalb zur Microservice Entwicklung.
- Sinnvolle Auswahl von Bibliotheken
  - Es gibt sehr viele Bibliotheken, manche mehr und manche weniger nützlich. Da die Community gross ist, werden immer wieder neue Bibliothek erstellt. Allerdings gibt es hier keinen grossen Antreiber, um die Qualität sicherzustellen.
- Dokumentation des Frameworks
  - Es gibt eine fundamentale Dokumenation des Frameworks. Allerdings wird für die meisten Themen die Community benötigt.
- Grosse und hilfreiche Community
  - Die Community ist bei diesem Framework die grösste. Deshalb kann man auch viele Artikel oder Ähnliches im Internet finden.

Fazit
ExpressJS war das erste Backend Framework, welches man an der OST anschaut. Es ist ein sehr populäres Backend Framework und wird sehr häufig eingesetzt. Der Package Manager ist der gleiche wie bei React. ExpressJS erfüllt alle unsere Anforderungen.

10.3.3.3 Spring Boot

Spring Boot ist ein Open-Source Framework, welches auf der Java EE-Plattform aufgebaut ist. Das Framework wurde von VMware entwickelt und erfreut sich einer grossen Popularität.

Erfüllung der Anforderungen

- Unterstützung von Microservices
  - Mithilfe der Containertechnologie Docker kann der Service in einem Container laufen gelassen werden und deshalb ist die Unterstützung von Microservices gegeben.
- Sinnvolle Auswahl von Bibliotheken
  - Hier gibt es viele Bibliotheken, welche aus der Welt von Java genutzt werden können. Diese haben sich über die Jahre etabliert.
- Dokumentation des Frameworks
  - Die Dokumentation des Frameworks ist aus der Sicht des Entwicklerteams für Einsteiger etwas zu kompliziert. Diese könnte einfacher und übersichtlicher gemacht werden.
- Grosse und hilfreiche Community
  - Die Community von Spring Boot ist gross, da das Framework auf Java basiert und viele diese Programmiersprache beherrschen.

Fazit
Spring Boot ist wegen der Programmiersprache Java ein weit verbreitetes Framework. Viele können diese Sprache selbst, da es meistens die erste objekt-orientierte Sprache ist, welche man in der Schule lernt. Das Framework erfüllt alle unsere Anforderungen. Allerdings kennen wir Spring Boot noch fast gar nicht und wir müssten uns in dieses Framework einarbeiten.

10.3.4 Ergebnis

Wir haben uns für ASP.NET [13] entschieden, da wir damit schon sehr gute Erfahrungen in der Studienarbeit gemacht haben. Somit müssen wir uns da nicht mehr gross einarbeiten und können möglicherweise gewisse Funktionalitäten übernehmen. Ausserdem stammt das Framework von Microsoft und die LGT [16] verwendet sehr viele Technologien, welche von Microsoft stammen. Somit ist es auch einfacher, die Applikation in den Betrieb zu geben.

10.4 API Evaluation

Die im Rahmen dieser Arbeit entwickelte Applikation stellt einige Anforderungen bzgl. der zentralen Speicherung und Abrufbarkeit von Daten mittels API. In diesem Kapitel werden verschiedene Techniken, welche die Implementation einer solchen API ermöglichen, begutachtet und miteinander verglichen. Mithilfe dieser Analyse soll die für die Applikation am besten geeignete Technologie gefunden werden.

10.4.1 Einschränkungen

Auf dem Markt gibt es diverse Möglichkeiten wie APIs aufgebaut werden können. Um diese Evaluation in einem nützlichen Rahmen zu halten, wurden bereits im Vorfeld drei mögliche Technologien ausgewählt, welche hier genauer analysiert werden sollen. Bei der Auswahl dieser Kandidaten wurde primär auf deren Verbreitung im Markt geachtet.

10.4.2 Anforderungen

Zu Beginn der Evaluation werden die Anforderungen, welche an die zu verwendende Technologie gestellt werden, definiert:

- Einfach zu dokumentieren
- Einfache Einbindung einer Authentisierung
- Unterstützung für Microservices

10.4.3 Vergleich

Im Rahmen dieser Evaluation werden die folgenden APIs miteinander verglichen:

- REST [17]
- gRPC [18]
- GraphQL [19]

10.4.3.1 REST

Die klassische und traditionelle Variante wie APIs definiert werden, geschieht mithilfe von REST. REST steht hierbei für „Representational State Transfer“. Bei dieser Art von API werden Ressourcen definiert und Vorgänge mithilfe von Verlinkungen abgebildet, um die auf die API zugreifende Software durch den Lebenszyklus einer Ressource durchzuführen. REST-APIs bauen auf HTTP auf und benutzen auch die in HTTP festgelegten Methoden.

Erfüllung der Anforderungen

- Einfach zu dokumentieren
  - Die APIs muss separat dokumentiert werden.
- Einfache Einbindung einer Authentisierung
  - Einbindung kann einfach mit einem Bearer-Token oder Ähnlichem geschehen.
- Unterstützung für Microservices
  - Aufrufe sind komplett unabhängig, was die Unterstützung für Microservices fördert und die Skalierbarkeit ermöglicht.

Fazit
REST ist eine etablierte und die wahrscheinlich meistgenutzte API, welche im Moment existiert. Die Vorgaben sind strukturiert und gut durchdacht. Eine vollkommen konforme REST API zu erstellen, bedeutet allerdings viele Vorgaben und Anforderungen für den Entwickler.

10.4.3.2 gRPC

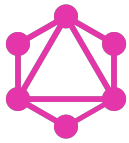
gRPC ist eine API Technik, welche primär auf Geschwindigkeit und Datensparsamkeit ausgerichtet ist. Ausserdem unterstützt gRPC eine Vielzahl von Plattformen und Betriebssystemen.

Erfüllung der Anforderungen

- Einfach zu dokumentieren
  - Die APIs muss separat dokumentiert werden.
- Einfache Einbindung einer Authentisierung
  - gRPC besitzt eine integrierte Authentisierung.
- Unterstützung für Microservices
  - Aufrufe sind komplett unabhängig, was die Unterstützung für Microservices fördert und die Skalierbarkeit ermöglicht.

Fazit
gRPC wurde für skalierbare und grosse Ökosysteme entwickelt, in welchen Geschwindigkeit und Bandbreite eine grosse Rolle spielen. Durch die breite Unterstützung diverser Sprachen und Systeme kann gRPC auf fast jedem Gerät eingesetzt werden. Die Unterstützung für das Web wurde leider erst spät angegangen, ist nun allerdings auch implementiert.

10.4.3.3 GraphQL



GraphQL ist eine Abfragesprache für APIs, welche spezifisch dafür konzipiert wurde, dem Client genau die Informationen zu liefern, welche dieser auch benötigt. Die Technik erlaubt es, verschachtelte und komplexe Anfragen zu definieren.

Erfüllung der Anforderungen

- Einfach zu dokumentieren
  - Das in der Sprache integrierte Schema ermöglicht die automatische Dokumentation der APIs. Dies setzt allerdings voraus, dass die Definition genügend Informationen enthält, um selbsterklärend zu sein.
- Einfache Einbindung einer Authentisierung
  - Einbindung kann einfach mit einem Bearer-Token oder Ähnlichem geschehen.
- Unterstützung für Microservices
  - Aufrufe sind komplett unabhängig und werden auch nicht zwischengespeichert. Das bedeutet, dass die Unterstützung für Microservices vorhanden ist. Allerdings ist GraphQL relativ komplex.

Fazit
GraphQL wurde entwickelt, um mit sich schnell ändernden APIs-Anforderungen und Daten umzugehen. Es wird allerdings auch das Problem des Overfetching sowie Underfetching gelöst. Ausserdem stellt GraphQL dem Client mächtige Tools zur Filterung der von ihm erhaltenen Daten zur Verfügung. GraphQL macht hierbei allerdings nicht von den in HTTP integrierten Funktionalitäten Gebrauch.

10.4.4 Ergebnis

GraphQL ist der neue Standard von APIs. Das Entwicklerteam hat auch bereits gute Erfahrungen damit gemacht und es erfüllt alle Anforderungen. Insgesamt ist es für eine Web Applikation etwas konformer als gRPC, da die Daten im JSON Format einsehbar sind. Durch das integrierte Schema ist die Dokumentation der APIs automatisch erstellt, was auch ein grosser Vorteil ist. Das Entwicklerteam hat sich aus diesen Gründen für GraphQL entschieden.

10.5 Microservice Architecture Evaluation

In diesem Kapitel werden die zwei Microservice Architekturen miteinander verglichen. Das Ziel ist hierbei, die am besten geeignete Architektur, für die im Rahmen dieser Arbeit entwickelte Applikation, zu finden. Als Microservice Architektur bezeichnet man eigentlich die Aufteilung des Backend. Das Backend wird auf verschiedene Microservices aufgeteilt, welche auf jeweils eigenen Instanzen laufen. Somit ist die Wartbarkeit des Backends sichergestellt.

Bei dieser Evaluation geht es um die Architektur des Frontends und hierbei gibt es zwei Varianten, wie man das machen könnte. Die Evaluation des Backends steht aufgrund der geforderten Microservice Architektur der Arbeit bereits fest.

10.5.1 Einschränkungen

Es gibt keine Einschränkungen, da es nur zwei Varianten gibt.

10.5.2 Anforderungen

Zu Beginn der Evaluation werden die Anforderungen, welche an die zu verwendende Architektur gestellt wird, definiert:

- Wartbarkeit des Codes
- Übersichtlichkeit des Codes

10.5.3 Vergleich

Im Rahmen dieser Evaluation werden die folgenden Möglichkeiten miteinander verglichen:

- Monolithisches UI [20]
- Zusammengesetztes UI [21]

10.5.3.1 Monolithisches UI

Ein Monolithisches UI besteht aus einer Instanz, welche mehrmals hochgefahren werden kann, um die Last der Anfragen zu verteilen. Allerdings ist die gesamte Logik des UIs nur auf einer Instanz am laufen.

Erfüllung der Anforderungen

- Wartbarkeit des Codes
  - Der Code kann wartbar gemacht werden, indem mit verschiedenen Klassen gearbeitet wird und so die Logik aufgeteilt werden kann. Allerdings führt eine Anpassung des Codes dazu, dass das gesamte UI neu gebildet und somit die Instanz neu gestartet werden muss.
- Übersichtlichkeit des Codes
  - Da der gesamte Code des UI in einer Repository gelagert ist, sind Grenzen bei der Übersichtlichkeit vorhanden. Für kleinere Applikationen reicht diese Variante völlig aus und die Übersichtlichkeit ist gegeben. Wird die Applikation allerdings grösser, wirkt sich das auf die Übersichtlichkeit aus.

Fazit
Das Monolitische UI eignet sich bestens für kleine Applikationen und die gesamte Logik des UIs befindet sich in einem Repository.

10.5.3.2 Zusammengesetztes UI

Beim zusammengesetzten UI wird, wie der Name schon sagt, das UI aus mehreren Instanzen zusammengesetzt. Diese Architektur ist das Pendant zur bereits festgelegten Backend-Architektur.

Erfüllung der Anforderungen

- Wartbarkeit des Codes
  - Der Code ist aufgrund der Architektur bereits sehr gut wartbar. Einzelne Komponenten können einfach ausgetauscht werden, ohne dass das gesamte UI ersetzt werden muss. Somit kann pro Microservices jeweils ein unterschiedliches Entwicklerteam das UI entwickeln, ohne dass die Teams sich gegenseitig in die Quere kommen.
- Übersichtlichkeit des Codes
  - Der Code ist bei kleinen Applikation eher unübersichtlich, da dass Zusammensetzen der Komponenten etwas komplizierter ist und Zusatzaufwand beim Entwickeln benötigt. Allerdings bleibt der Code bei grösseren Applikationen immer schön übersichtlich, da dieser pro Microservice in einer eigenen Repository liegt. Somit kann bei einer Anpassung des UI nur ein einzelner Microservice angepasst werden, ohne dass diese Änderungen einen Einfluss auf die restlichen Komponenten haben.

Fazit
Das zusammengesetzte UI ist vor allem für grössere Applikationen sehr gut geeignet. Ausserdem unterstützt es beim Entwicklen des UIs, dass verschiedene Teams jeweils gleichzeitig und unabhängig voneinander an einer Komponente arbeiten können.

10.5.4 Ergebnis

Wir haben uns bei dieser Arbeit für das zusammengesetzte UI entschieden, da es uns dabei unterstützt, die Arbeitsaufteilung auf einen Service machen zu können. Somit kann eine Person Full-Stack entwickeln, also man kann von der Datenbank bis zum UI alles selbst machen. Weiter ist dies für das Entwicklerteam eine neue Architekturvariante, welches es gerne ausprobieren würde.

10.6 Azure Deployment Evaluation

Im Rahmen dieser Arbeit möchten wir das optimale Deployment für unsere zu entwickelnde Applikation herausfinden.

In dieser Evaluation werden verschiedene Azure Services miteinander verglichen, mit welchen ein Deployment der Applikation realisiert werden kann. Mit Hilfe dieser Evaluation soll der beste Deployment Service auch für eine zukünftige Erweiterung der Applikation gefunden werden.

10.6.1 Einschränkungen

Aufgrund der bereits evaluierten Azure Cloud kommen für uns nur Azure spezifische Deployments in Frage. Um die Evaluation in einem vernünftigen Rahmen zu halten, werden wir nicht alle möglichen Azure Deployments anschauen, sondern haben uns auf drei verschiedene Services beschränkt.

Bei der Auswahl dieser Azure Services wurde primär auf deren Funktionsumfang und Einfachheit geschaut. Somit evaluieren wir nur Services, welche direkt ohne zusätzliche Installation ein Deployment unterstützen.

Eine weitere Einschränkung ist, dass wir als VCS uns für GitLab entschieden haben. Somit sollen über eine GitLab CI/CD Pipeline die Deployments verteilt werden können. Azure unterstützt auch ein eigenes VCS, Azure DevOps, welches stark in ihre Cloud integriert ist und gewisse Vorteile beim Deployment von Azure Services gegenüber GitLab hat. Aufgrund der Verbreitung von GitLab und weil das Projektteam dieses VCS bereits gut kennt, wird dieses verwendet und die Deployments werden für GitLab-Deployments geschrieben.

10.6.2 Anforderungen

Bei den Azure Services haben wir keine hart definierten Kriterien, können auf der „grünen Wiese“ starten und sind in der Auswahl relativ frei. Es gibt allerdings ein paar Punkte, welche aus unserer Sicht sehr wichtig sind und dementsprechend beachtet werden müssen. Dazu gehören:

- Direktes Deployment ohne zusätzliche Installation
- Deployment über GitLab CI/CD Pipelines möglich
- Verschiedene Deployments sollen sinnvoll gruppiert werden können, was die Multi-Tenancy Fähigkeit ermöglichen soll.

10.6.3 Vergleich

Im Rahmen dieser Evaluation werden die folgenden Azure Services miteinander verglichen:

- Azure App Services [22]
- Azure Kubernetes Services [23]
- Azure Red Hat OpenShift [24]
- Azure Functions [25]

10.6.3.1 Azure App Service

Azure App Service ist eine cloud-computing-basierte Plattform zum Deployen von Docker Containern und Hosten von Websites.

Vorteile

- Automatisches OS-Patching
- Sicherheit auf Unternehmensniveau
- Hohe Verfügbarkeit inkl. automatischem Scale-out/in und integriertem Load-Balancing
- Unterstützt viele Programmiersprachen sowie Plattformen
- Easy continuous deployment (continious delivery from third-party source control providers)
- App Service unterstützt ein einfaches Continuous Deployment. Dazu gehört auch ein Continious Delivery von Drittanbietern für die Versionskontrolle (z.B. GitLab).

Nachteile

- Beschränkungen bei der Installation von Software und Verwaltungstools von Drittanbietern
- Leistungszähler sind nicht verfügbar
- Starke Kopplung an Azure Cloud
- Integration von GitLab nur über Webhooks oder über CI/CD-Container mit Azure CLI möglich

Pricing

- Flexible Abrechnung, bei welcher nur die effektiv benötigten Ressourcen verrechnet werden.

Erfüllung der Anforderungen

- Direktes Deployment ohne zusätzliche Installation
  - Über den App Service kann der gewünschte Microservice direkt in die Cloud deployed werden.
- Deployment über GitLab CI/CD Pipelines möglich
  - Automatisches Deployment ist möglich. Dies kann entweder mit einer GitLab CI/CD Pipeline und der Azure CLI erreicht werden oder über die Integration des Repository in Azure und einem Webhock, welcher bei einem Push in den main Branch das Deployment aktualisiert.
- Verschiedene Deployments sollen sinnvoll gruppiert werden können
  - Die App Services können in einer eigenen Ressourcen-Gruppe deployed werden.

10.6.3.2 Azure Kubernetes Services

Kubernetes ist ein Open-Source Container-as-a-Service (CaaS)-Framework, das von Google-Entwicklern entwickelt wurde. Grundsätzlich ist Kubernetes ein portables, Open-Source Containerisierungssystem, mit dem Dienste und Arbeitslasten verwalten werden können.
Das System automatisiert die Bereitstellung, Skalierung und den Betrieb von Container-Anwendungen. Der Azure Kubernetes Service bietet serverloses Kubernetes in der Cloud an.

Vorteile

- Als Open-Source-Framework bietet Kubernetes sehr viel Flexibilität und kann auf fast jeder Plattform installiert werden. Aufgrund der Flexibilität kann praktisch jeder Use Case für das Deployment mit Kubernetes umgesetzt werden.
- Kubernetes verfügt über eine grosse und aktive Gemeinschaft von Entwicklern, die kontinuierlich an der Verbesserung der Plattform mitarbeiten. Ausserdem bietet sie Unterstützung für mehrere Frameworks und Programmiersprachen.
- Mit Helm-Charts können Kubernetes Deployments einfach verwendet werden und bieten ein hohes Mass an Flexibilität an.

Nachteile

- Das Standard Kubernetes Dashboard ist sehr komplex und muss zu Beginn installiert und mit einem zusätzlichen Komponenten, dem *kube-proxy*, verwendet werden. Das Dashboard hat leider auch keine Login-Seite, aufgrund dessen müssen Bearer-Tokens für die Authentifizierung und Authorisierung erstellt werden. Es ist allerdings möglich ein zusätzliches System wie den ELK-Stack oder Grafana zu installieren.
- Kubernetes verfügt nicht über integrierte Authentifizierungs- oder Autorisierungsfunktionen, so dass Entwickler Bearer-Token und andere Authentifizierungsverfahren manuell erstellen müssen.

Pricing

- Nur die virtuellen Maschinen und die verbrauchten Speicher- und Netzwerkressourcen müssen bezahlt werden.

Erfüllung der Anforderungen

- Direktes Deployment ohne zusätzliche Installation
  - Das Kubernetes Cluster kann direkt über Azure deployed und direkt orchestriert werden.
- Deployment über GitLab CI/CD Pipelines möglich
  - Das Kubernetes Cluster kann über einen GitLab-Agent mit GitLab verbunden werden. Über eine GitLab CI/CD Pipeline können Ressourcen im Cluster deployed werden.
- Verschiedene Deployments sollen sinnvoll gruppiert werden können
  - Das Kubernetes Cluster kann in einer eigenen Ressourcen-Gruppe deployed werden.

10.6.3.3 Azure Red Hat OpenShift

OpenShift ist eine Containerisierungssoftware, die vom Open-Source-Softwareanbieter Red Hat entwickelt wurde. Kubernetes ist dabei der Kern verteilter Systeme auf welcher OpenShift aufbaut, während OpenShift die Distribution selbst ist.

Im Kern ist OpenShift eine cloud-basierte Kubernetes-Container-Plattform, die sowohl als Containerisierungssoftware als auch als Platform-as-a-Service (PaaS) gilt. Sie basiert teilweise auch auf Docker, einer anderen beliebten Containerisierungsplattform.
OpenShift bietet konsistente Sicherheit, integrierte Überwachung, zentralisierte Richtlinienverwaltung und Kompatibilität mit Kubernetes-Container-Workloads. Es ist schnell, ermöglicht eine Self-Service-Bereitstellung und lässt sich mit einer Vielzahl von Tools integrieren. Der Azure Red Hat OpenShift Service bietet hochverfügbare, vollständig verwaltete OpenShift Cluster auf Abruf als Service in der Cloud an.

Vorteile

- Sehr gutes Management-Dashboard, welches auf die Kubernetes-API zugreift.
- OpenShift verfügt über eine intuitive Web-Konsole mit einer integrierten Anmeldeseite für die Authentifizierung und Authorisierung. Die Konsole bietet eine einfache Schnittstelle, welche auf die Kubernetes-API zugreift und über welche die Benutzer Ressourcen einfach verwalten können.
- Das Verwalten sowie Deployen von Applikationen ist einfacher aufgrund von vielen bereits vorkonfigurierten Workflows.
- OpenShift hat strenge Sicherheitsrichtlinien. So kann ein Container zum Beispiel nicht als root ausgeführt werden. Ausserdem bietet es eine Secure-by-default-Option zur Verbesserung der Sicherheit an.
- Bereits bei der LGT On Premise im Einsatz.

Nachteile

- OpenShift erfodert Red Hats proprietäres Red Hat Enterprise Linux Atomic Host (RHELAH), Fedora oder CentOS für die Installation. Open Shift wird allerdings von allen grossen Cloud Providern unterstützt und stellt somit keinen grossen Nachteil dar.
- Red Hat OpenShift hat eine viel kleinere Support-Community als Kubernetes, welche sich hauptsächlich auf Red Hat-Entwickler beschränkt.
- Templates sind bei OpenShift bei weitem nicht so flexibel und benutzerfreundlich wie Helm-Charts.

Pricing

- Flexibles On Demand Preismodell

Erfüllung der Anforderungen

- Direktes Deployment ohne zusätzliche Installation
  - Das OpenShift Cluster kann direkt über Azure deployed und direkt orchestriert werden.
- Deployment über GitLab CI/CD Pipelines möglich
  - Das Kubernetes Cluster kann über einen GitLab-Agent mit GitLab verbunden werden. Über eine GitLab CI/CD Pipeline können Ressourcen im Cluster deployed werden.
- Verschiedene Deployments sollen sinnvoll gruppiert werden können
  - Das OpenShift Cluster kann in einer eigenen Ressourcen-Gruppe deployed werden.

10.6.3.4 Azure Functions

Azure Functions ist ein cloud-basierter serverloser Dienst, der es ermöglicht, ereignisgesteuerten Code auf skalierbare Weise auszuführen, ohne Infrastruktur bereitzustellen oder zu verwalten.
Er ermöglicht die Ausführung eines Skripts oder Codes als Reaktion auf eine Vielzahl von Ereignissen und muss nicht kontinuierlich ausgeführt werden.

Vorteile

- Azure Functions sind einmalige, wiederverwendbare Codestücke, welche eine Eingabe verarbeiten und ein Ergebnis zurückgeben können (Single responsibility).
- Azure Functions bleiben nach der Ausführung nicht im Hintergrund und geben so Ressourcen für weitere Ausführungen frei (short lived).
- Azure Functions haben keinen dauerhaften Zustand und sind nicht auf den Zustand anderer Prozesse angewiesen (stateless).
- Azure Functions reagieren auf vordefinierte Ereignisse und werden sofort so oft wie nötig repliziert.

Nachteile

- Azure Functions werden speziell für die Azure Cloud entwickelt und können nur mit viel Aufwand in eine andere Cloud migriert werden (vendor lock-in).
- Komplexes Preismodell, welches bei falscher Konfiguration zu einer Kostenfalle werden kann. Mit der Einstellung „ always on instance“ wird der Kostenvorteil, der mit Functions as a Service verbunden ist, zunichte gemacht.
- Es gibt einige Einschränkungen bei der Skalierung mit gewissen Preismodellen. Zum Beispiel gibt es sogar mit dem Premium Plan eine Obergrenze für die Anzahl der Functions von 100 Stück.

Pricing

- Azure Functions werden nach der Execution Time abgerechnet. Somit muss nur die Zeit bezahlt werden, in welcher die Azure Function auch wirklich aktiv war.

Erfüllung der Anforderungen

- Direktes Deployment ohne zusätzliche Installation
  - Die Azure Functions können lokal oder in der Cloud entwickelt und direkt in Azure deployed werden. Zusätzliche Einstellungen und Konfigurationen sind über das Azure Portal oder die Azure CLI möglich.
- Deployment über GitLab CI/CD Pipelines möglich
  - Das Deployment kann entweder mit einer GitLab CI/CD Pipeline und einem Docker Container mit der Azure CLI erreicht werden oder über die Integration des Repository in Azure und einem Webhock, welcher bei einem Push in den main Branch das Deployment aktualisiert.
- Verschiedene Deployments sollen sinnvoll gruppiert werden können
  - Die App Functions können in einer eigenen Ressourcen-Gruppe deployed werden.

10.6.4 Preisvergleich

Beim Preisvergleich haben wir die jährlichen Kosten für das Deployment geschätzt. Beim App Service wurde eine Standard Instanz gewählt und die Gesamtanzahl an App Services auf fünf geschätzt.
Bei den Clustern (Kubernetes & OpenShift) wurde ebenfalls ein Standard Cluster mit mindestens zwei Nodes gewählt.

Produkt	**Zusammensetzung Kosten**	**Totale Kosten [CHF/Y.]**
Azure App Services	151.03 CHF/M. * 5 App Services * 12 M.	9′061.80
Azure Kubernetes Services	753.17 CHF/M. * 12 M.	9′038.04
Azure Red Hat OpenShift	1653.20 CHF/M. * 12 M.	19′838.40
Azure Functions	303.06 CHF/M. * 5 Functions * 12 M.	18′183.60

Tabelle 10.1: Preisvergleich Azure Services

10.6.5 Fazit

Aufgrund des vorgenommenen Vergleichs aus Kapitel 10.6 hat sich das Entwicklerteam für die Verwendung der, in Kapitel 10.6.3.2 evaluierten, Azure Kubernetes Service entschieden. Von den evaluierten Services waren schlussendlich neben dem Azure Kubernetes Service noch Red Hat OpenShift in der engeren Auswahl.

Bei der LGT ist bereits ein OpenShift On Premise Cluster im Einsatz und mit der Evaluation hat sich gezeigt, dass dies berechtigt ist. Für den reinen Betrieb bietet OpenShift mit einem sehr guten Management-Dashboard viele Vorteile. Für das Deployen sowie den Betrieb der Applikation, welche im Rahmen dieser Bachelorarbeit entwickelt wird, reicht ein normaler Kubernetes Cluster aber bei weitem aus. Da OpenShift auf einem Kubernetes aufbaut und das OpenShift Management-GUI auf die Kubernetes-API zugreift, können erstellte Kubernetes-Konfigurationsfiles sehr einfach für OpenShift adaptiert werden. Aus diesem Grund hat die Wahl für Kubernetes auf einen späteren, allfälligen Betrieb mit OpenShift keinen negativen Einfluss.

Die Entscheidung für den Azure Kubernetes Service ist auf nachfolgende Vorteile zurückzuführen:

- Beim Azure Kubernetes Service fallen tiefe Kosten an.
- Dem Entwicklerteam ist Kubernetes und dessen Funktionsweise bereits bekannt.
- Konfigurationsfiles können einfach und nur mit geringem Aufwand für OpenShift adaptiert werden.

10.7 Cloud IDE Evaluation

Im Rahmen dieser Arbeit wird eine Applikation entwickelt, welche mit einer Cloud IDE programmiert werden soll. Aufgrund dem Ready On Demand Ansatz soll die Cloud IDE jederzeit mit allen benötigten SDKs etc. bereit stehen. Dies ergibt für den Entwickler einen merklichen Zeitgewinn, da er nur noch programmieren und sich nicht mehr mit der Installation oder Updates der IDE sowie deren Komponenten herumschlagen muss. Weiter verwenden so alle Programmierer den genau gleichen Zustand und es entstehen keine Probleme aufgrund von veralteten Packages mehr. Schlussendlich soll die IDE aber alle gewohnten Funktionalitäten wie bei einer On Premise IDE unterstützen, da der Entwickler auf keine Funktion verzichten soll.

In diesem Kapitel werden verschiedene IDEs angeschaut und miteinander verglichen.

10.7.1 Einschränkungen

Es gibt viele IDEs auf dem Markt, welche unsere Anforderungen potentiell erfüllen könnten. Aufgrund der begrenzten Zeit und des marginalen Nutzens wurden hier nicht alle Cloud IDEs verglichen, sondern lediglich einige wenige.

Bei der Auswahl der in diesem Kapitel aufgeführten IDEs haben wir primär auf die Verbreitung und somit auf die Langlebigkeit und den Support geachtet.

10.7.2 Anforderungen

- Unterstützung von mehreren Programmiersprachen sowie Debugging
- Unterstützung von Plugins oder Einstellungen für Personalisierung der IDE
- Kann mit GitLab als VCS kombiniert werden

10.7.3 Vergleich

Im Rahmen dieser Evaluation haben wir drei verschiedene IDEs ausgewählt, welche in der Cloud betrieben werden können, sowie mehrere Programmiersprachen und Plugins für die Anpassung unterstützen. Diese werden im Folgenden aufgelistet und beschrieben.

- GitHub Codespaces [26]
- GitPod [27]
- Codeanywhere [28]

10.7.3.1 GitHub Codespaces

GitHub Codespaces ist eine Entwicklungsumgebung, die in der Cloud von GitHub gehostet und als SaaS-Service angeboten wird.

Vorteile

- SaaS
- Ready on Demand
- Pay-as-you-go
- Starke GitHub Integration

Nachteile

- Keine GitLab Unterstützung
- GitHub Codespaces gibt es nur als SaaS-Lösung und kann somit nicht selber gehosted werden. Die Daten befinden sich somit in der GitHub Cloud.

Pricing

- Abrechnung nach Aufwand („Pay as you go„)

Erfüllung der Anforderungen

- Unterstützung von mehreren Programmiersprachen sowie Debugging
  - Über diverse VS-Code Plugins können verschiedene Programmiersprachen inkl. Debugging unterstützt werden. Dabei handelt es sich aber nicht um den gleichen Store wie für die gewöhnlichen VS Code Extensions, sondern um die *Open VSX Registry* [29]. Es werden dabei aber ein Grossteil der Plugins aus dem VS Code Marketplace auch in diesem Store angeboten.
- Unterstützung von Plugins oder Einstellungen für Personalisierung der IDE
  - Über Einstellungen kann die IDE personalisiert sowie mit VS-Code Plugins ergänzt werden.
- Kann mit GitLab als VCS kombiniert werden
  - Unterstützt nur GitHub als VCS

10.7.3.2 GitPod SaaS

Gitpod ist eine Open-Source Kubernetes Anwendung für Ready-to-Code Entwicklungsumgebungen, die in sekundenschnelle neue, automatisierte Entwicklungsumgebungen für jede Aufgabe in der Cloud erstellt.
GitPod wird in zwei Varianten angeboten, SaaS und *self-hosted*. In der Evaluation gehen wir auf beide Varianten ein und beschreiben die jeweiligen Vor- und Nachteile.
Bei der SaaS Variante wird Gitpod als fertige Lösung in der Gitpod Cloud betrieben und als Service angeboten.

Vorteile

- Unterstützung von verschiedenen Programmiersprachen & Plugins
- GitHub, GitLab & Bitbucket Integration
- Browser Extension
- Kann stark modifiziert und pro Workspace angepasst werden

Nachteile

- Daten in der Cloud

Pricing
Bei der SaaS Lösung von GitPod gibt es insgesamt vier verschiedene Preismodelle. Diese gehen vom Gratis Modell mit eingeschränkter Funktionalität bis zur Unterstützung von sehr hohen Workloads. Die Preise sind fix pro Monat und nicht Pay-as-you-go, wie bei den GitHub-Codespaces.

Erfüllung der Anforderungen

- Unterstützung von mehreren Programmiersprachen sowie Debugging
  - Über diverse VS-Code Plugins können verschiedene Programmiersprachen inklusive Debugging unterstützt werden. Dabei handelt es sich aber nicht um den gleichen Store wie für die gewöhnlichen VS Code Extensions, sondern um die *Open VSX Registry* [29]. Es werden dabei aber ein Grossteil der Plugins aus dem VS Code Marketplace auch in diesem Store angeboten.
- Unterstützung von Plugins oder Einstellungen für Personalisierung der IDE
  - Über Einstellungen kann die IDE personalisiert sowie mit VS-Code Plugins ergänzt werden.
- Kann mit GitLab als VCS kombiniert werden
  - Unterstützt GitLab, GitHub & Bitbucket

10.7.3.3 GitPod self-hosted

Bei der *self-hosted* Variante wird Gitpod in einem Kubernetes Cluster oder einem Cloud Provider deployed und kann selber verwaltet und konfiguriert werden. Schlussendlich wird auch bei dieser Variante Gitpod als Service in der eigenen Cloud angeboten.

Vorteile

- Unterstützung von verschiedenen Programmiersprachen & Plugins
- GitHub, GitLab & Bitbucket Integration
- Kann selber gehosted werden auf GKE (Google Kubernetes Engine), EKS (Elastic Kubernetes Service), AKS (Azure Kubernetes Service) und Kubernetes
- Browser Extension
- Kann stark modifiziert und pro Workspace angepasst werden
- Gitpod kann auf jeder Kubernetes Installation deployed und gehosted werden. Dazu gehört auch OpenShift, welches von der LGT bereits On Premise betrieben wird.

Nachteile

- Grösserer initialer Aufwand mit Deployment
- Betrieb der Gitpod Kubernetes Infrastruktur

Pricing
Bei der *self-hosted*-Lösung von GitPod gibt es drei verschiedene Preismodelle. Diese gehen wie bei der SaaS-Variante von eingeschränkter Funktionalität bis zur Unterstützung von sehr vielem.

Erfüllung der Anforderungen

- Unterstützung von mehreren Programmiersprachen sowie Debugging
  - Über diverse VS-Code Plugins können verschiedene Programmiersprachen inkl. Debugging unterstützt werden.
- Unterstützung von Plugins oder Einstellungen für Personalisierung der IDE
  - Über Einstellungen kann die IDE personalisiert sowie mit VS-Code Plugins ergänzt werden.
- Kann mit GitLab als VCS kombiniert werden
  - Unterstützt GitHub, GitLab & Bitbucket

10.7.3.4 Codeanywhere

Codeanywhere ist eine plattformübergreifende cloud-integrierte Entwicklungsumgebung, die von Codeanywhere, Inc. erstellt wurde.

Vorteile

- Unterstützung von verschiedenen Programmiersprachen & Plugins
- Integrated Debugger and Refactor
- Live Collaboration

Nachteile

- Daten in der Cloud

Pricing
Codeanywhere bietet drei verschiedene Preismodelle an, welche nach Benutzer und pro Monat abgerechnet werden.

Erfüllung der Anforderungen

- Unterstützung von mehreren Programmiersprachen sowie Debugging
  - Über diverse VS-Code Plugins können verschiedene Programmiersprachen inkl. Debugging unterstützt werden.
- Unterstützung von Plugins oder Einstellungen für Personalisierung der IDE
  - Über Einstellungen kann die IDE personalisiert sowie mit VS-Code Plugins ergänzt werden.
- Kann mit GitLab als VCS kombiniert werden
  - GitHub, GitLab & Bitbucket Integration

10.7.4 Preisvergleich

Beim Preisvergleich haben wir die jährlichen Kosten für 100 Entwickler, welche 8h pro Tag und fünf Tage die Woche mit der IDE arbeiten, geschätzt. Bei der Leistung vergleichen wir die Kosten für acht Core-CPUs und 16 GB RAM.

Produkt	**Zusammensetzung Kosten**	**Totale Kosten [CHF/Y.]**
GitHub Codespaces	1920 h/Y. * 100 MA * 0.67 CHF/h.	128′640
Gitpod SaaS	35.83 CHF * 100 MA * 12 M.	42′996
Gitpod self-hosted	29.68 CHF/Y. * 100 MA * 12 M.	35′616 + $Hosting$
Codeanywhere	46.17 CHF/Y. * 100 MA * 12 M.	55′404

Tabelle 10.2: Preisvergleich Cloud IDE

10.7.5 Entscheidung

Aufgrund des vorgenommenen Vergleichs aus Kapitel 10.7 hat sich das Entwicklerteam für die Verwendung der, in Kapitel 10.7.3.3 evaluierten, Cloud IDE entschieden. Von den evaluierten IDEs waren schlussendlich noch GitPod SaaS, GitPod self-hosted sowie Codeanywhere in der engeren Auswahl. Die Entscheidung für GitPod self-hosted ist auf nachfolgende Vorteile zurückzuführen:

- Bei der Cloud IDE handelt es sich praktisch um VS Code, welches dem Entwicklerteam bereits bekannt und vertraut ist.
- Die IDE lässt sich personalisieren und ist mit Plugins stark erweiterbar.
- Gewünschte Einstellungen können pro Workspace und Programmiersprache konfiguriert werden.
- Aufgrund des self-hosted Ansatzes muss GitPod deployed und betrieben werden. Dadurch erhofft sich das Entwicklerteam ein grösseres Learning als bei einem SaaS-Ansatz.

11 Systemübersicht

In diesem Kapitel wird die geplante Architektur in Form eines Deployment-Diagramms dargestellt und die einzelnen Komponenten des Diagramms werden textuell erläutert. Dabei unterscheiden wir zwischen einem „High-Level“-Diagramm mit der Gesamtübersicht des Deployments sowie einem Diagramm mit Fokus auf das Deployment innerhalb des Clusters.

Bei der Detailansicht der Komponenten innerhalb des Clusters wird auf die spezifischen Kubernetes-Komponenten eingegangen, welche für das Deployment relevant sind. Alle bereits erwähnten Komponenten, wie die Persistenz, werden aufgrund der übersichtlichen Darstellung weggelassen.

11.1 Overview

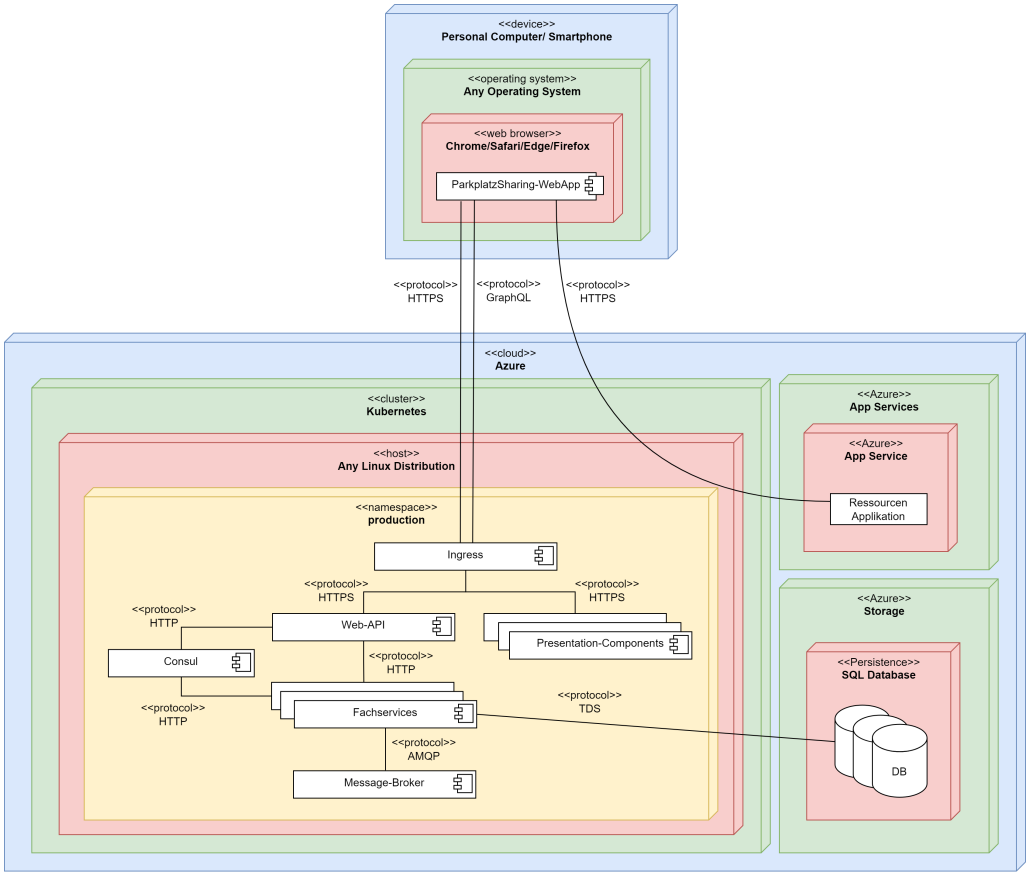


Abbildung 11.1: Deploymentdiagramm der Applikation

Komponente	Beschreibung
ParkplatzSharing WebApp	Stellt das User-Interface im Browser zur Verfügung. Kommuniziert mit dem Server.
Ingress	Leitet eingehende Anfragen an die Web-API oder die Presentation-Components weiter. Verwaltet TLS-Zertifikate.
Presentation-Components	Liefert das ParkplatzSharing-Frontend aus.
Web-API	Stellt die API für das Frontend zur Verfügung und handelt die API-Aufrufe für die verschiedenen Backend-Fachservices.
Consul	Stellt die Service Discovery der Fachservices von den Presentation-Components über den Web-API sicher.
Fachservices	Stellt das Backend der Applikation, bestehend aus verschiedenen Microservices, dar.
Message-Broker	Für die Asynchrone Kommunikation zwischen den Backend-Fachservices stellt der Message-Broker verschiedene Queues bereit.
Azure SQL Database	Persistiert die Daten des Backends. Für jeden Backend-Service gibt es eine eigene DB-Instanz.
Azure App Service	Stellt die Ressourcen Applikation dar, über welche auf das entwickelte Parkplatz-Modul (LGT Parkonomy) zugegriffen werden kann.

Tabelle 11.1: Komponenten Deploymentdiagramm

11.2 Cluster

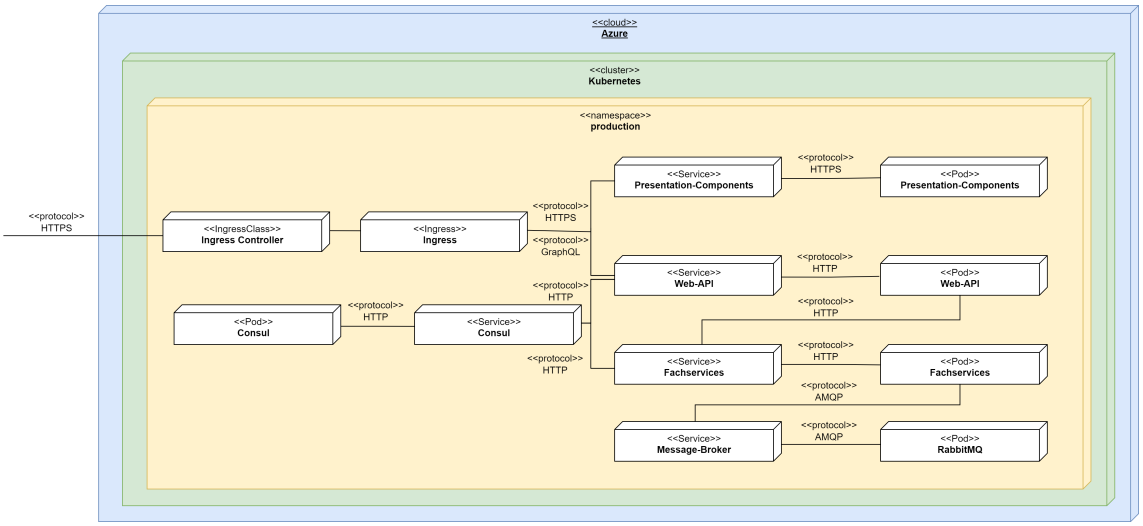


Abbildung 11.2: Deploymentdiagramm des Clusters

Komponente	**Beschreibung**
Namespace	Namespaces bieten einen Mechanismus zur Isolierung von Ressourcengruppen innerhalb eines einzelnen Clusters dar.
IngressClass	Der Ingress-Controller verwendet Ingress-Klassen, um Ingress-Objekte und andere Ressourcen zu filtern und zu routen. Der Ingress-Controller wird in einem separaten Namespace bereitgestellt und kann alle Ingress-Rules aus den verschiedenen Namespaces sehen. Anhand dieser Rules wird der eingehende Traffic entsprechend weitergeleitet.
Ingress	Ist im entsprechenden Namespace bereitgestellt und agiert als Reverse-Proxy. Traffic zwischen Browser und Applikation läuft immer über den Ingress, welcher die Anfrage an den entsprechenden Microservice weiterleitet.
Service	Ein Service ist ein Abstraktionslayer und über diesen kann eine Anwendung, welche auf einer Reihe von Pods läuft, als Netzwerkdienst dargestellt werden.
Pod	Ein Pod stellt eine Gruppe von einem oder mehreren Containern dar, auf welchem die Applikation resp. der Microservice ausgeführt wird.

Tabelle 11.2: Komponenten Kubernetes Cluster

12 Technologien

In diesem Kapitel werden die verwendeten Technologien und Softwares, welche zur Umsetzung der Applikation benötigt werden, dokumentiert und näher erläutert. Bei jeder verwendeten Software ist angegeben, zu welchem Zweck diese verwendet wird.

12.1 Allgemein

In diesem Kapitel sind Anforderungen gelistet, welche sowohl von Backend als auch Frontend erfüllt werden mussten. Entsprechend ist bei jeder Anforderung die eingesetzte Software in Front- und Backend aufgeteilt.

12.1.1 Codeprüfung

Um eine hohe Qualität für den gesamten Source Code zu gewährleisten, werden Entwicklungstools zur Codeüberprüfung benötigt. Zum Einen soll diese mit Hilfe von statischer Codeanalyse stattfinden, um eine frühe Fehlererkennung und die Durchsetzung eines einheitlichen Codestyles zu ermöglichen. Zum Anderen soll die Funktionalität des Codes stetig durch Unit- und Integrationstests überprüft werden, was ermöglicht, bei jeder Änderung des Codes die Lauffähigkeit der bestehenden Funktionalitäten sicherzustellen. Bei der Auswahl der entsprechenden Tools ist es wichtig zu garantieren, dass die Codeprüfung im Rahmen einer Continuous Integration automatisierbar ist.

12.1.1.1 Verwendete Software

- Frontend
  - ESLint [30]
    - Definiert Coderichtlinien, welche nach Bedarf angepasst werden können
- Backend
  - StyleCop Analyzers [31]
    - Definiert Coderichtlinien, welche nach Bedarf angepasst werden können
    - Ermöglicht die Überprüfung der Coderichtlinien innerhalb der IDE
  - Menees Analyzers [32]
    - Definiert zusätzliche Coderichtlinien, betreffend der erlaubten Länge von Klassen/Methoden/Zeilen, welche nach Bedarf angepasst werden können
    - Ermöglicht die Überprüfung der Coderichtlinien innerhalb der IDE

12.1.2 Code-Dokumentation

Da das Projekt nach Ende der Bachelorarbeit möglicherweise weiterentwickelt resp. als Referenzprojekt für diese Applikation sicher nochmals angeschaut wird, ist es sehr wichtig die Einarbeitung in das Projekt möglichst einfach zu gestalten. Aus diesem Grund wird ein Grossteil des Source Codes dokumentiert.

12.1.2.1 Verwendete Software

- Frontend
  - react-docgen [33]
    - Ermöglicht die Generierung einer Dokumentation aus den integrierten Kommentaren
- Backend
  - Doxygen [34]
    - Ermöglicht die Generierung einer Dokumentation aus den integrierten XML-Kommentaren
    - Ermöglicht verschiedene Ausgabeformate

12.1.3 Unit-Tests

Um eine gute Codequalität zu gewährleisten, muss der Code regelmässig getestet werden. Die am besten in die CI integrierbare ARt des Testens sind die *Unit-Tests*. Um sicherzustellen, dass der Code auch wirklich das macht, was der Entwickler sich bei der Implementation gedacht hat, werden sowohl im Frontend als auch im Backend Unit-Test verwendet. Diese helfen ausserdem nach vorgenommenen Änderungen zu prüfen, ob der Code immer noch erwartungesgemäss funktioniert.

12.1.3.1 Verwendete Software

- Frontend
  - Jest [35]
    - Test Framework für Assertion, Mocking und Test Coverage
- Backend
  - Xunit [36]
    - Test Framework für Assertion
    - Tests sind integriert in die IDE und können direkt ausgeführt werden.
    - Tests können auch über die Kommandozeile ausgeführt werden.
  - Nsubstitute [37]
    - Bietet Funktionalität an, um Mocks für Interfaces und auch gewisse Klassen zu erstellen
  - Coverlet [38]
    - Ermöglicht zu analysieren, wie hoch die Coverage ist und welche Codezeilen durch Unit-Tests abgedeckt sind

12.2 Frontend

Im Rahmen der Evaluationen wurde React als Frontend Technologie gewählt. Dadurch ist die Technologie bereits definiert. Andere Technologien spielen keine entscheidende Rolle in der Applikation und werden deshalb nicht dokumentiert.

12.2.1 Verwendete Software

- React [11]
  - Grundlage für das Frontend

12.3 Backend

Im Rahmen der Evaluationen wurde GraphQL als API Technologie, ASP.NET als Framework und Azure SQL-Datenbank als Datenbank gewählt. Dadurch sind diese Technologien bereits definiert. Für die in diesem Kapitel gewählten Technologien fanden wir eine Evaluation als nicht nötig, da sie keine entscheidende Rolle bei der Architektur spielen.

12.3.1 Verwendete Software

- ASP.NET [13]
  - Cross-platform Web-Framework, entwickelt von Microsoft
- HotChocolate [39]
  - Im Team bekannte Implementation von GraphQL
- Entity Framework [40]
  - Ist ein ORM-Mapper für .NET Applikationen von Microsoft
  - Besitzt eine gute Integration mit ASP.NET
- Azure SQL-Datenbank [41]
  - Der Azure SQL-Datenbank Service ist ein relationaler Datenbankdienst, welcher von Microsoft Azure zur Verfügung gestellt wird. Da wir das Entity Framework im Backend zur Kommunikation mit der Datenbank verwenden, kann die Datenbank einfach ausgetauscht werden, falls gewünscht. Weiter müssen wir uns mit einem Cloud Service nicht um die Persistenz der Daten kümmern, da dies vom Cloud Provider bereits gewährleistet wird. Die LGT verwendet hauptsächlich Oracle und MSSQL Datenbanken. Da der Azure SQL-Datenbank Service auf MSSQL basiert, haben wir uns für diesen Datenbank-Service entschieden.

13 Logische Architektur

In diesem Kapitel wird die interne Struktur der verschiedenen Komponenten beschrieben, sowie die Zusammenhänge zwischen den Komponenten dargestellt. Darüber hinaus werden die wichtigsten Abläufe innerhalb der Software erklärt.

13.1 Makroarchitektur

In der Makroarchitektur sehen wir die grobe Struktur unserer Architektur. Auf diese Architektur wird im Kapitel Mikroarchitektur noch genauer eingegangen.

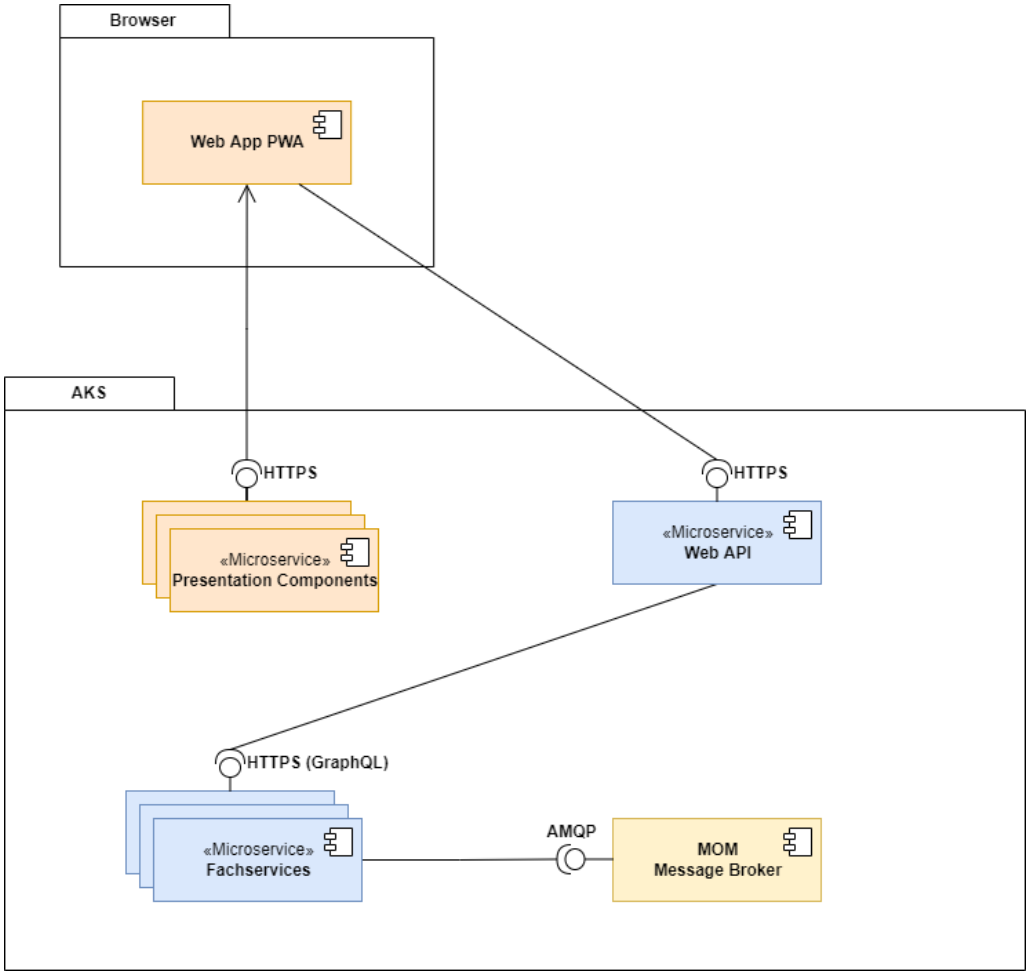


Abbildung 13.1: Makroarchitektur

Wie in der Abbildung ersichtlich ist, besteht die Architektur aus verschiedenen Komponenten. Als erstes wird vom Client eine HTTPS Anfrage an unseren WebApp Service geschickt, welche die gesamte Frontend Applikation an den Client zurückliefert. Sobald auf der PWA Applikation Informationen benötigt werden, wird eine Anfrage an die Web-API gesendet. Diese dient als zentrale Anlaufstelle für die Fachservices und stelle eine GraphQL Schnittstelle zur Verfügung. Im Hintergrund geschieht die Kommunikation zwischen den verschiedenen Fachservices mittels einem Message Broker. Falls ein Proxy Service bezogen werden muss, geschieht dies direkt und nicht über den Message Broker.

13.2 Mikroarchitektur

In der Mikroarchitektur ist ersichtlich, wie die genaue Struktur unserer Architektur im Detail aufgebaut ist.

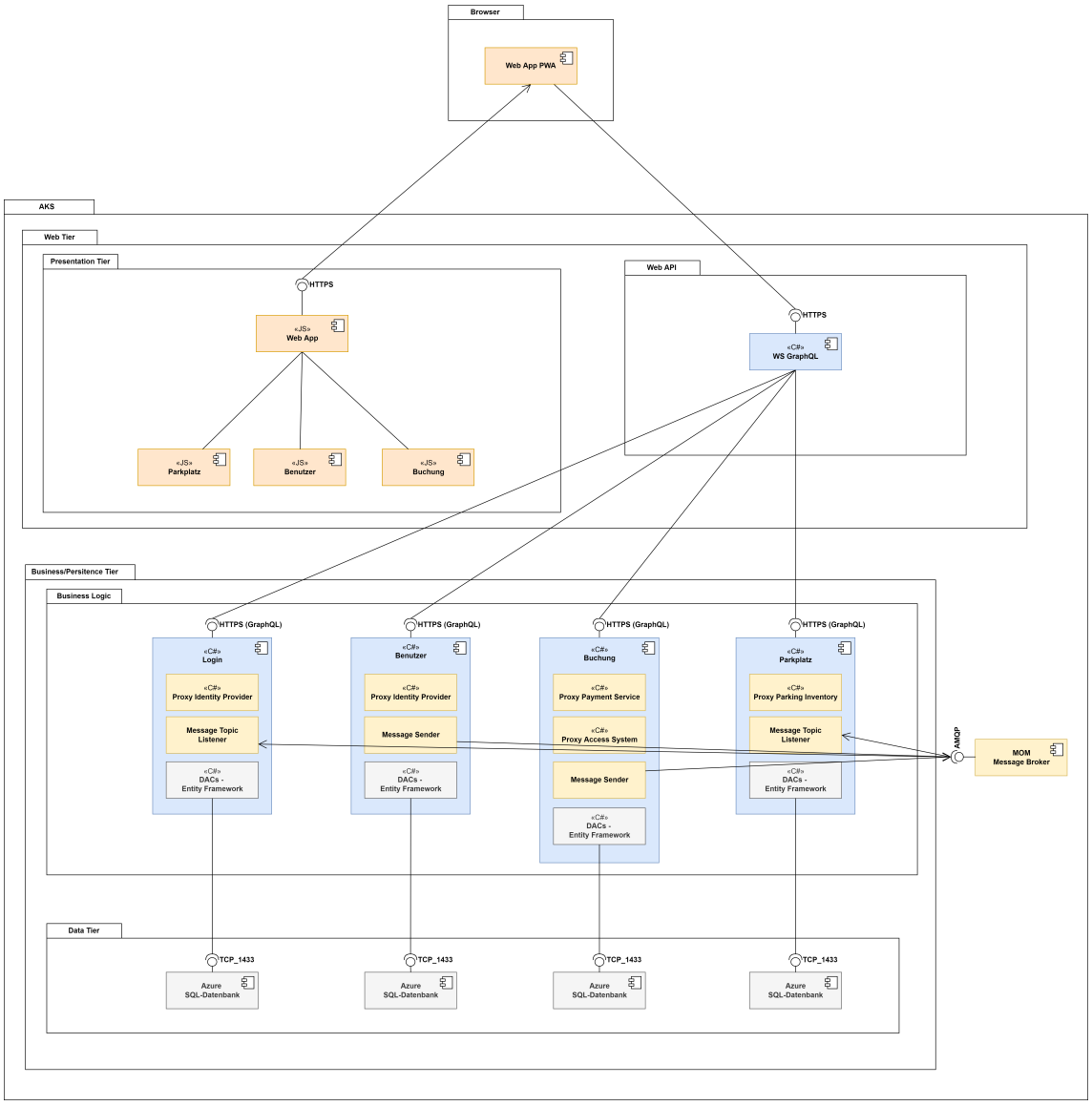


Abbildung 13.2: Mikroarchitektur

Im Presentation Tier werden die UI Komponenten in jeweils eigene Microservices unterteilt. Bei dieser Aufteilung vom Frontend spricht man von *Microfrontends*. Als zentrale Anlaufstelle dient dazu eine Web App, welche die verschiedenen Microfrontends zusammensetzt, damit nur eine JavaScript-Datei zum Client gesendet wird, wie das von SPA resp. PWA bekannt ist.

Der Client greift über die Web-API auf die Business Logic zu. Dafür gibt es wieder jeweils einzelne Microservices, welche aufgrund ihrer Domäne aufgeteilt wurden. Bei der Architektur im Backend spricht man nun definitiv von Microservices. Untereinander kommunizieren die Microservices mittels einem Message Broker. Dies aber nur falls eine Kommunikation von Nöten ist. Das bedeutet, dass die Kommunikation untereinander asynchron abläuft, was die Services voneinander entkoppelt.

Ein einzelner Microservice stellt eine GraphQL Schnittstelle zur Verfügung. Ausserdem beinhaltet dieser verschiedene Proxy Services, welche die Kommunikation mit einer externen Schnittstelle abstrahieren sollen. Zusätzlich kann auch ein Message Sender bzw. Listener implementiert sein, um mit dem Message Broker zu kommunizieren. Dieser speichert die gesendeten Nachrichten auf einer entsprechenden Queue ab, bis ein anderer Service diese Message abholt. Zusätzlich beinhaltet ein Microservice jeweils noch das Entity Framework von Microsoft, damit eine Verbindung zu einer Datenbank einfach geschehen kann.

Aus dem Diagramm kann auch entnommen werden, dass pro Microservice eine Datenbank existiert. Das heisst wenn ein Microservice nun mehrere Instanzen besitzt, greifen alle Instanzen auf dieselbe Datenbank zu. Somit kann die Datenqualität sichergestellt werden.

Ausserdem hat sich das Entwicklerteam für diese Architektur entschieden und gegen ein Self-Contained System-Ansatz, damit die Daten nicht dupliziert werden müssen. Mit dieser Architektur kann der Client über die Web-API die benötigten Daten anfragen. Mit der Variante SCS könnte er dies nicht, sondern die Daten müssten auf der jeweiligen Datenbank dupliziert werden.

Um einen besseren Überblick zu gewährleisten, haben wir im obigen Diagramm die Service Registry nicht aufgezeichnet. Diese wird im Kapitel Service Registry mit einem Sequenz Diagramm genauer beschrieben.

13.2.1 Service Registry

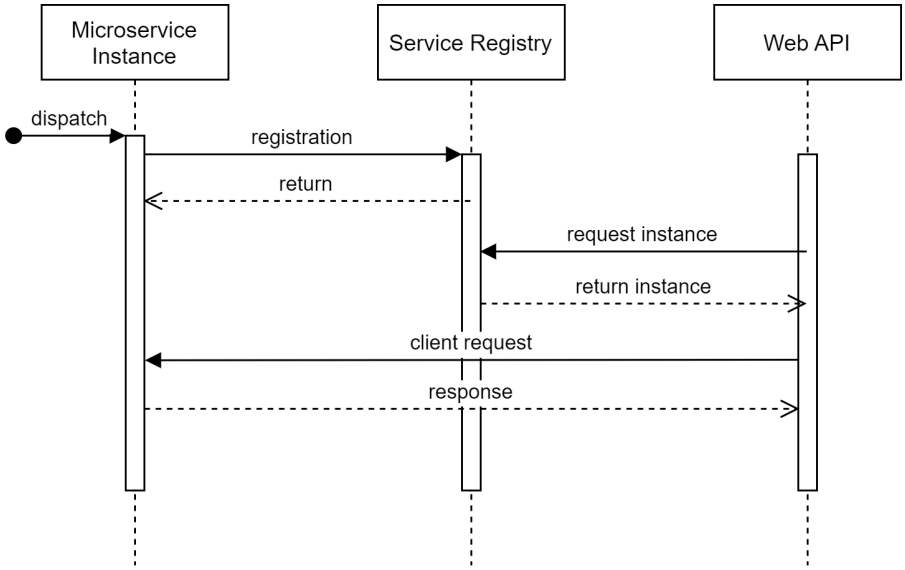


Abbildung 13.3: Service Registry

Die Service Registry ist für die Verwaltung der jeweiligen Microservice-Instanzen zuständig. Diese melden sich, wenn sie gestartet werden, bei der Service Registry und geben ihre IP Adresse sowie deren Port bekannt. Ausserdem müssen sie sich für einen spezifischen Microservice (zum Beispiel den Buchungsmicroservice) anmelden, für welchen sie verantworlich sind.

Somit weiss die Service Registry, welche Instanz für welchen Microservice zuständig ist. Es kann aber auch sein, dass mehrere Instanzen für einen Microservice gestartet werden. In diesem Fall ist die Service Registry zusätzlich auch fürs Load Balancing zuständig.

Sobald vom Client eine Anfrage über die Web-API abgesendet wird, kann die Web-API die Service Registry anfragen, um herauszufinden wohin dieser die Anfrage senden soll. Die Service Registry überprüft nun, welche Instanzen für die Anfrage zuständig ist und stellt der Web-API die entsprechende IP Adresse und deren Port zur Verfügung. Die Web-API kann die Anfrage schlussendlich an den richtigen Microservice weiterleiten.

14 Schnittstellen

In diesem Kapitel werden die Schnittstellen beschrieben, welche die Applikation nutzt. Ausserdem wird die vom Server selbst angebotene Schnittstelle, die GraphQL-API, visualisiert.

14.1 Frontend

Die PWA-Applikation kommuniziert über den Ingress des Kubernetes Clusters mit den Services der verschiedenen Microservices. Die API-Aufrufe ins Backend werden über einen Web-API an die API des jeweiligen Microservices weitergeleitet.

14.2 Backend

Das Backend ist so aufgebaut, dass jeder API-Aufruf ans Backend über den Web-API läuft, welcher die Anfrage ans API des benötigten Microservices weiterleitet. Jeder Backend-Microservice stellt eine eigene GraphQL-API zur Verfügung mit verschiedenen Querys sowie Mutationen.

14.2.1 API

Wir haben die Technologie GraphQL unter anderem gewählt, da durch das Schema die API dokumentiert werden kann. Alle Endpunkte und Parameter der API wurden dokumentiert und dies kann mittels dem Abrufen des Schemas direkt angeschaut werden. Zusätzlich wird nachfolgend eine vereinfachte Visualisierung aller Endpunkte dargestellt.

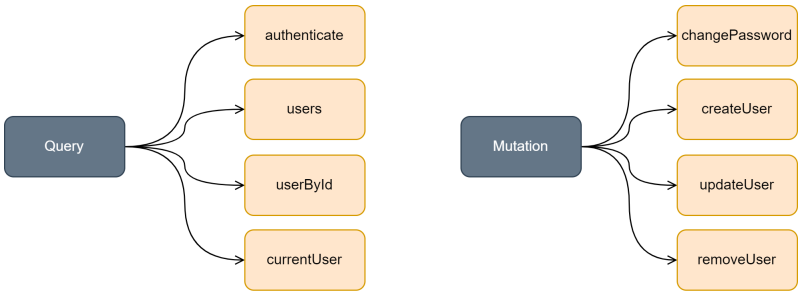


Abbildung 14.1: Übersicht über die GraphQL-API

In dieser Grafik sind die API-Aufrufe für den Benutzer-Microservice abgebildet. Da jeder Microservice grundsätzlich gleich aufgebaut ist, werden die Queries und Mutationen aller Microservices nicht aufgelistet.

15 | Cloud-IDE

Bei der Cloud-IDE haben wir uns, wie in Kapitel 10.7 beschrieben, für Gitpod self-hosted entschieden. Dabei gibt es von Gitpod ein fertiges Installationsscript [42] für die Azure Cloud, welches das gesamte Deployment übernimmt.

15.1 Übersicht

In der folgenden Grafik ist ersichtlich, was für Azure-Komponenten alles involviert sind und wie diese zusammenspielen.

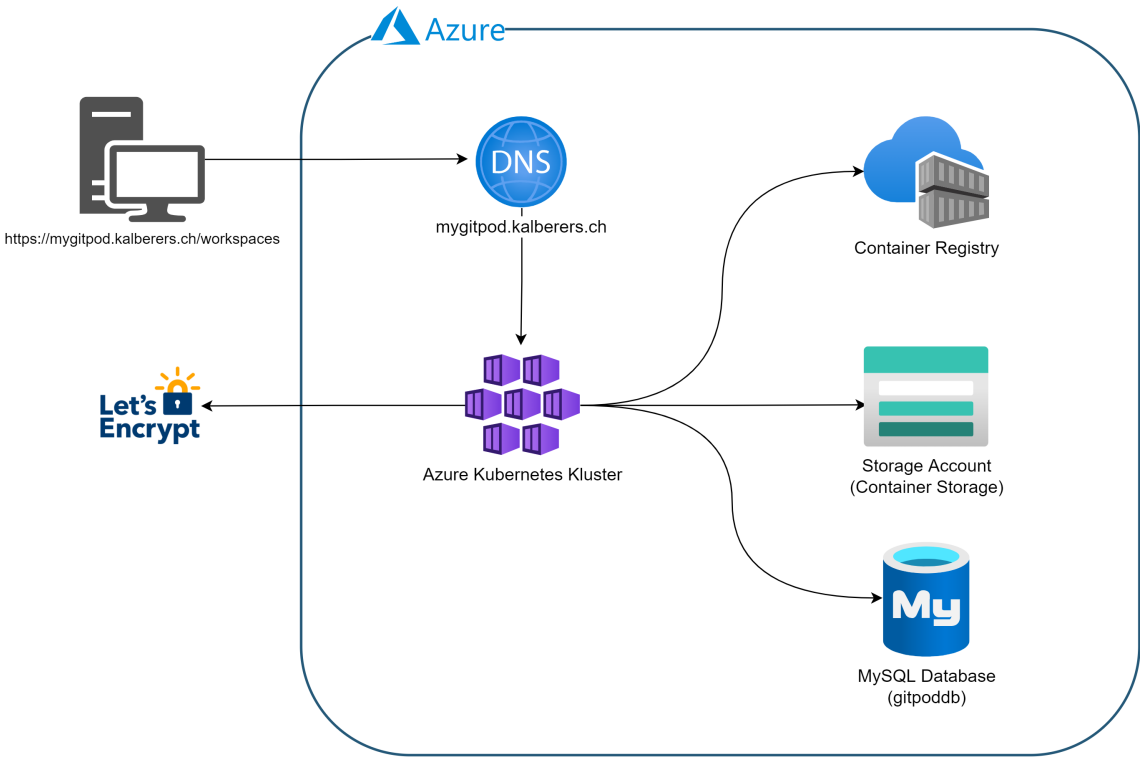


Abbildung 15.1: Deploymentübersicht Gitpod

Komponente	Beschreibung
DNS zone	DNS Einträge, damit die Gitpod-Instanz über die gewünschte Domain erreichbar ist.
AKS	Azure Kubernetes Cluster, in welchem die Gitpod Applikation deployed ist.
Container Registry	Die von Gitpod benötigten Docker Container werden in der Container Registry abgelegt, damit die Ladezeit verkürzt wird.
Storage Account	Gitpod verwendet für die Persistierung von Workspaces Container Storages. Dabei wird jeder Workspace in einer einzelnen Archivdatei gespeichert und innerhalb des Container Storage gespeichert.
Azure MySQL Database	Die Persistenz der Benutzerdaten von Gitpod wird über eine Azure MySQL Datenbank gewährleistet.

Tabelle 15.1: Komponenten Cloud IDE

15.2 Customization

Gitpod Workspaces können nach Belieben weiter konfiguriert werden. Wir haben dabei die Workspace Config angepasst und diese um Prebuilds sowie eigene Docker Images erweitert.

15.2.1 Workspace Config

Über ein *.gitpod.yml*-File können zusätzliche Workspace Einstellungen [43] vorgenommen werden. Dazu gehören unter anderem eigene Docker Images sowie VS-Code Extensions.

15.2.2 Prebuilds

Prebuilds [44] verringern die Wartezeit, indem sie Abhängigkeiten installieren oder Builds ausführen, bevor ein neuer Arbeitsbereich gestartet wird. Dabei werden die Prebuilds ebenfalls in der *.gitpod.yml*-Datei festgelegt und umfassen die Tasks *before, init & command*, welche beim starten eines Workspaces auch in dieser Reihenfolge gestartet werden. Die folgende Grafik illustriert die Performance-Optimierung von Prebuilds.

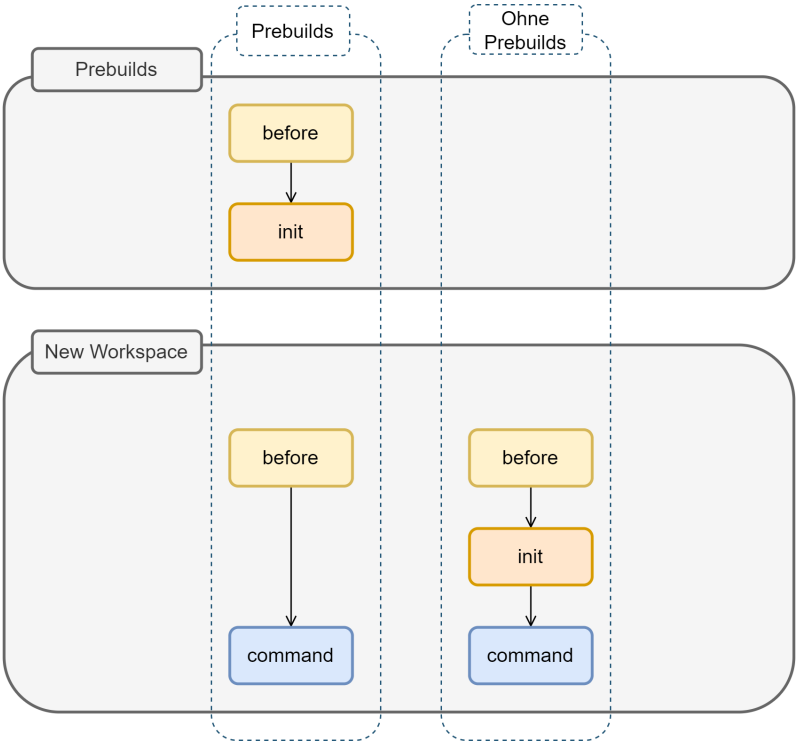


Abbildung 15.2: Gitpod Prebuilds

15.2.2.1 before

Ein *before*-Task wird empfohlen, wenn Anpassungen am Terminal oder Installationen von globaler Projektabhängigkeiten vorgenommen werden müssen. Der *before*-Task wird vor dem *init*- und vor dem *command*-Task ausgeführt.

15.2.2.2 init

Der *init*-Task wird für ressourcenintensive Aufgaben, wie das Herunterladen von Abhängigkeiten oder Kompilieren von Code, empfohlen.

15.2.2.3 command

Mit dem *command*-Task kann ein Datenbank- oder Entwicklungsserver gestartet werden.

15.2.3 Browser Settings

Für eine bessere User-Experience sollten die empfohlenen Browser Settings [45] ergänzt werden. Bei einer produktiven Umgebung müssen diese Einstellungen nicht von jedem Benutzer selber durchgeführt werden, sondern können z.B. über GPOs global festgelegt werden.

16 | Continuous Integration

Um eine gute Code-Qualität zu garantieren, wird CI verwendet. Es ermöglicht uns, den in der Versionverwaltung eingecheckten Code, permanent zu überprüfen. Nachfolgend sind die Abläufe der Pipeline abgebildet, welche für alle Microservices gleich aufgebaut sind. Dabei zu beachten ist, dass jeder Schritt eine erfolgreiche Durchführung des Schritts davor voraussetzt.

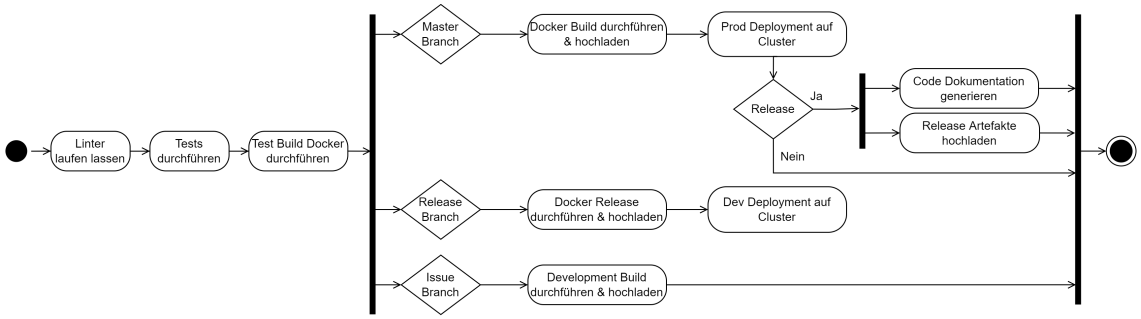


Abbildung 16.1: Übersicht CI/CD Pipeline

17 End of Elaboration

Das nachfolgende Kapitel befasst sich mit der Auflösung von Unklarheiten und Unsicherheiten. Dies erfolgt durch die Erstellung eines technischen Prototypen, in welchem die wichtigsten Funktionen rudimentär implementiert werden, um die technische Machbarkeit des Projekts sicherzustellen und projektkritische Risiken zu beseitigen. Darüber hinaus wird dies aus administrativer Sicht durch die Prüfung der End of Elaboration Checklist sichergestellt.

17.1 Technischer Prototyp

Das Ziel des technischen Prototyps ist es sicherzustellen, dass das Projekt mithilfe der evaluierten Technologien durchgeführt werden kann. Dafür wurden unter Berücksichtigung der Anforderungsspezifikation die nachfolgenden Features ausgewählt, welche betreffend der Implementation eine Herausforderung darstellen könnten:

- Kommunikation zwischen Browser und Server mittels der GraphQL-API
- Benutzer-Authentifikation über die GraphQL-API
- Kommunikation der verschiedenen Microservices untereinander
- Darstellung eines Composed Frontends
- Automatisiertes Deployment über die GitLab CI/CD Pipeline
- Entwicklung und Deployment über Cloud IDE möglich

Aufgrund dieser Eigenschaften wurde entschieden, den Prototyp in Form einer PWA-Applikation zu gestalten, welche den Zugriff und eine geeignete Darstellung von einem Smartphone sowie einem Computer gewährleistet. Jeder Frontend Microservice hat einen entsprechenden Backend-Microservice, welche über die Web-API miteinander kommunizieren. Das komplette Frontend wird wiederum von einem Microservice dargestellt, indem dieser die Inhalte von den restlichen Frontend-Microservices holt. Sobald eine Änderung an einem Microservice durchgeführt wurde, kann dieser über die Pipeline direkt im Cluster über eine Update-Strategie deployed werden.

17.2 Checkliste

Nach der erfolgreichen Implementation des technischen Prototyps wird zur bereits erwähnten Checkliste Stellung genommen:

- Wir haben die Kundenbedürfnisse eingeholt. Scope (=grober Funktionsumfang) ist vereinbart.
  - Die Anforderungen des Kunden (LGT) sind in der Aufgabenstellung aufgeführt.
  - In der Anforderungsanalyse wurden diese über mehrere Iterationen spezifiziert und erweitert.
  - Die finale Anforderungsspezifikation wurde vom Kunden und der Betreuungsperson überprüft und abgenommen.
- Wir beherrschen alle Werkzeuge (IDE, Versionskontrolle, Build Server, Deployment, Unit Testing, Workflow Tools, Wiki, . . .).
  - Sämtliche Repositories wurden inkl. CI/CD aufgesetzt.
  - Das Deployment der verschiedenen Microservices wurde im Rahmen des technischen Prototyps bereits getestet.
- Cloud IDE wurde aufgesetzt und eingerichtet.
  - Benötigte GitLab Repositories können über die Cloud IDE zugegriffen und bearbeitet werden.
  - Deployments können über die GitLab CI/CD Pipeline gemacht werden.
  - Cloud IDE kann nach eigenen Bedürfnissen modifiziert und angepasst werden.
- Wir wissen, wie die Architektur aussehen wird (Architektur skizziert sowie Architektur-Prototypen gemacht).
  - Die grobe Architektur wurde im Rahmen des technischen Prototyps ausgearbeitet.
  - Die aus der Arbeit resultierende Architektur ist im Abschnitt III dokumentiert.
- Für das User Interface gibt es einen ersten Entwurf (Grafiken sowie Wireframes), der dem Kunden gefällt.
  - Der Entwurf des User Interfaces (8.5.3) wurde während der Anforderungsspezifikation zusammen mit dem Kunden ausgearbeitet.
  - Die finale Version des Entwurfs wurde mit dem Kunden besprochen und entsprechend abgenommen.
- Alle grossen Risiken und Unklarheiten sind minimiert.
  - Die Anforderungen wurden so weit wie möglich spezifiziert.
  - Der erfolgreiche Abschluss des technischen Prototyps gilt als Beweis für die technische Machbarkeit des Projekts.
  - Mögliche Risiken wurden in Kapitel 6 zeitlich geschätzt und eingeplant.

Teil IV

Umsetzung

18 Herausforderungen

In diesem Kapitel wird beschrieben, wie die geplante Softwarearchitektur umgesetzt und die Infrastruktur aufgebaut wurde. Es soll einen Einblick in die Strukturen und Abläufe der Software sowie den Deploymentkomponenten gewährt werden. Während der Umsetzungsphase können oft unerwartete Situationen eintreten, welche in zusätzlichen Herausforderungen enden. Aus diesem Grund werden in diesem Kapitel die Key Points der Umsetzungsphase sowie spezielle Implementationsdetails genauer erläutert.

18.1 Architecture

Um einen besseren Überblick über entstandene Probleme und Key Points während der Umsetzungsphase zu geben, möchten wir zu Beginn die aufgebaute Architektur beschreiben. Im folgenden Diagramm ist ein Big-Picture der aufgebauten Azure-Architektur ersichtlich.

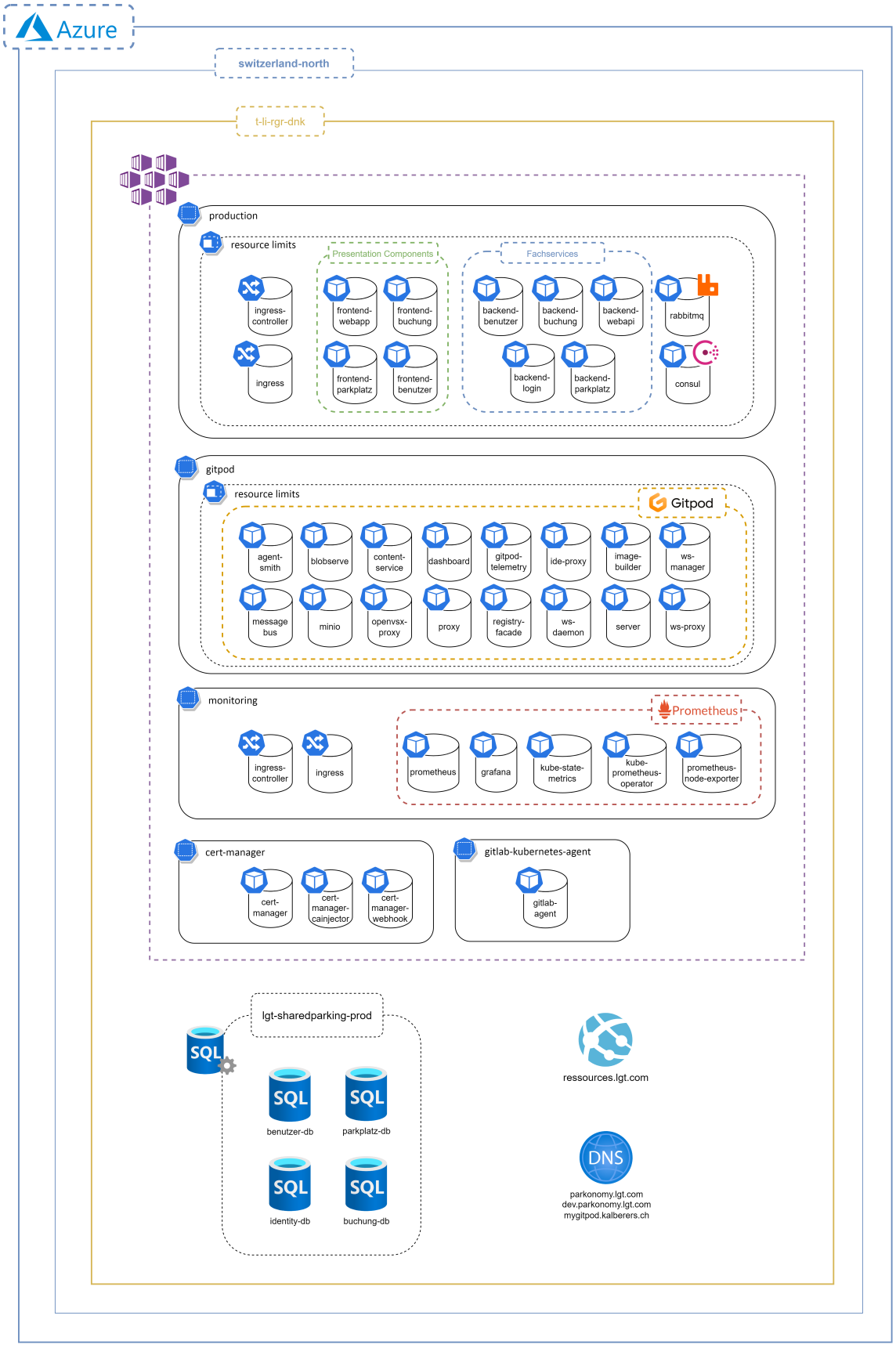


Abbildung 18.1: Architektur Azure

18.2 Domain Model

Insgesamt konnten wir uns gut an das ursprünglich geplante Domain Model halten. Aufgrund der Microservice-Architektur können aber die Beziehungen der Klassen untereinander nicht mehr wie geplant abgebildet werden. Um dies besser zu veranschaulichen haben wir das ursprüngliche UML-Diagramm um die Microservices ergänzt.

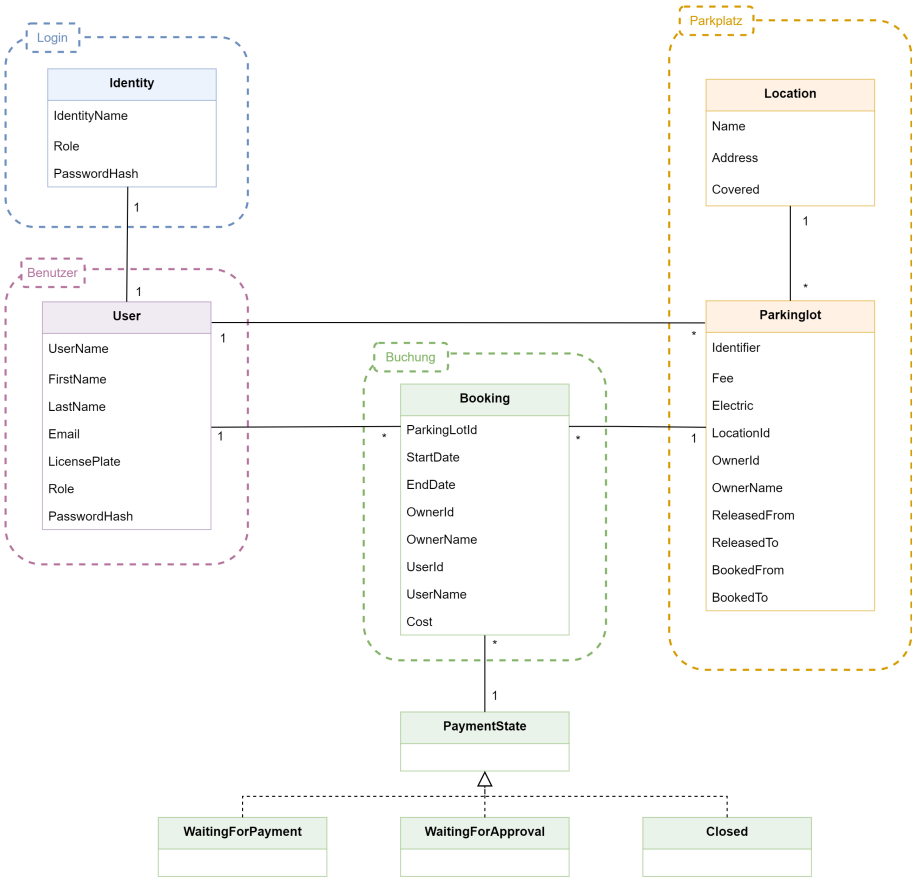


Abbildung 18.2: Übersicht umgesetztes Domain Model

Alle mit Microservices eingezeichneten Klassen und somit der grösste Teil an Use-Cases konnte umgesetzt werden. Aufgrund der zeitlichen Beschränkung der Bachelorarbeit und dem hohen initialen Aufwand für den Aufbau der Deployments, der Architektur sowie den verschiedenen Microservices, konnten wir die Bezahlungs Use-Cases 8.3.2.2 & 8.3.3.2 leider nicht mehr umsetzen. Aktuell basiert dieser Punkt auf dem Vertrauen, dass ein Mitarbeiter dem jeweiligen Parkplatzbesitzer für den Zeitraum der Buchung den entsprechenden Betrag überweist.

In einem allfällig weiteren Sprint würde dieser Use-Case im erstellten Issue umgesetzt werden können.

18.3 Allgemein

In diesem Abschnitt gehen wir auf die Herausforderungen ein, welche nicht direkt mit dem Frontend oder dem Backend zusammenhängen, aber dennoch wichtig sind.

18.3.1 GitLab Runner

Wir haben uns als VCS für die public GitLab-Instanz entschieden, weil wir so alle für uns relevanten Funktionen benutzen können und gleichzeitig das Projekt sehr einfach zu einem späteren Zeitpunkt auf die private GitLab-Instanz der LGT migriert werden kann. Leider ist bei der öffentlichen Instanz ein Limit der CI-Minuten pro Projekt auf 400 Minuten pro Monat beschränkt. In der Umsetzungsphase haben wir diese Limite vor allem aufgrund der Erstellung von Docker-Images schnell erreicht. Aus diesem Grund haben wir bei der OST einen Server bestellt, auf welchem wir unseren eigenen GitLab-Runner betreiben können. Dieser wurde als Docker-Container gestartet und in unser GitLab-Projekt integriert. Somit mussten wir bei der Überschreitung der CI-Minuten auf keine Funktionalitäten verzichten und konnten wie gewohnt Docker-Images über die Pipeline erstellen und schlussendlich aufs Cluster deployen.

18.3.1.1 Konfiguration

Ein externer Runner kann sehr schnell und einfach eingebunden werden. Als erstes muss der GitLab-Runner gemäss der offiziellen Anleitung installiert sowie registriert werden. Nun muss auf dem GitLab-Projekt ein „Registration Token “ erstellt und im Konfigurationsfile angegeben werden.

```
    [[runners]]
      name = "544152f17800"
      url = "https://gitlab.com/"
      token = "y4we_HpMs8Z-uz67nyXX" # Registration Token
      executor = "docker"
    [runners.docker]
      image = "docker:stable"
      volumes = ["/var/run/docker.sock:/var/run/docker.sock",
         "/cache"]
```

Auflistung 18.1: config.toml

Dieses Konfigurationsfile wird schlussendlich in den Docker-Container gemountet und der Docker-Container kann gestartet werden. Der Runner registriert sich nun bei der spezifizierten GitLab-Instanz und ist dort im entsprechenden Projekt oder Repository als dedizierter Runner ersichtlich.

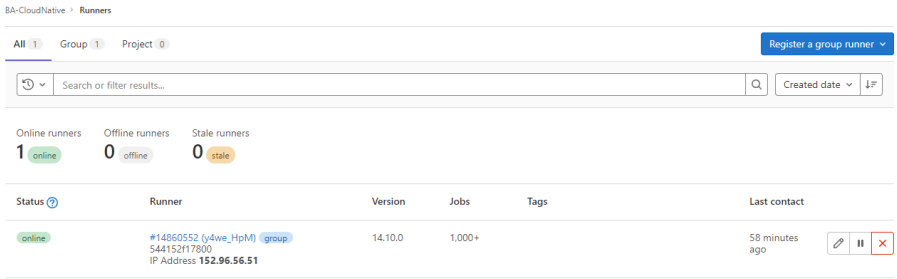


Abbildung 18.3: Dedizierter GitLab Runner

18.4 Cloud IDE (Gitpod)

In diesem Abschnitt werden alle Probleme und Key-Points, welche mit der Gitpod Cloud IDE in Verbindung stehen beschrieben.

18.4.1 Installation

Für die Installation der self-hosted Gitpod Instanz existiert bereits ein Installationsscript für Azure, welches von Gitpod zur Verfügung gestellt wird. Zu Beginn lief dieses jedoch nicht, da wir zu wenig verfügbare Ressourcen in Azure alloziert hatten. Wir mussten die benötigten Ressourcen erhöhen. Im Folgenden sind die benötigten Ressourcen aufgelistet:

- 1x Network Watcher
- 24x Standard DSv2 Family vCPUs

Mit diesen vorhandenen Ressourcen kann Gitpod über das Installationsscript deployed werden. Insgesamt werden die folgenden Azure Services erstellt, welche für Gitpod benötigt werden:

- AKS Cluster (Kubernetes v1.21)
- Azure Load Balancer
- Azure MySQL Database
- Azure Blob Storage
- Azure DNS Zone
- Azure Container Registry
- Calico als CNI und NetworkPolicy Implementation
- Cert-Manager für self-signed SSL Zertifikate

Um schlussendlich auf Gitpod zugreifen zu können, ist eine eigene Domain notwendig. Die DNS Zone muss wie in der Installationsanleitung beschrieben, konfiguriert werden und auf die eigene Domain zeigen.

Gitpod ist unter `https://<domain>/workspaces` erreichbar.

18.4.2 Customized Docker Images

Wenn ein Docker Image von Gitpod (z.B. `gitpod/workspace-full`) benutzt wird und die komplette LaTeX-Installation (`texlive-full`) zusätzlich darauf installiert wird, wird das Image sehr gross (ca. 12 GB). Dies führt zu einer *Low Memory on Node* Fehlermeldung oder zu einem Problem beim Upload des Images auf den Azure Blob.

Es wird allerdings die komplette Installation von *texlive-full* benötigt, damit die LaTeX-Documentation wie bisher ohne Fehler und mit allen benötigten Libraries kompiliert werden kann.

Aus diesem Grund wurde ein eigenes Docker Image mit dem zusätzlichen Package erstellt:

```
FROM gitpod/workspace-full

# Install LaTeX
RUN sudo apt-get update \
    && sudo apt-get install -y \
    texlive-full \
    && sudo rm -rf /var/lib/apt/lists/*
```

Auflistung 18.2: Dockerfile für customized Gitpod-LaTeX-Image

Nun kann das Docker-Image lokal gebildet werden und ein Repository auf einer Container-Registry gepusht werden. Wir haben uns dabei für den Docker-Hub entschieden.

```
docker build -f <Dockerfile> -t
   gianfluetschost/gitpod-latex .
docker push gianfluetschost/gitpod-latex:latest
```

Auflistung 18.3: Docker Image builden & auf Docker-Hub pushen

Im Gitpod Konfigurationsfile kann dieses angepasste Image vom Docker-Hub angegeben werden.

```
image: gianfluetschost/gitpod-latex
```

Auflistung 18.4: Ausschnitt .gitpod.yml

18.4.3 Login Button

Ein weiteres aufgetauchtes Problem war, dass der Login-Button für Gitpod nicht mehr vorhanden war. Dies war bei der *self-hosted* Gitpod Instanz ein bereits bekanntes Problem. In diesem Fall musste als Workaround ein Pod neu gestartet werden.

```
kubectl delete pod -n gitpod -l component=server
```

Auflistung 18.5: Restart Pod

Mittlerweile wurde dieses Problem in den GitHub Issues 8927 & 9046 behoben.

18.5 Deployment

In diesem Abschnitt werden alle Probleme und Key-Points, welche mit dem Deployment oder der Infrastruktur zusammenhängen, beschrieben.

18.5.1 Kustomize

Beim Deployment haben wir uns entschieden, Kustomize einzusetzen. Kustomize erstellt modifizierte Kubernetes-Manifeste durch Überlagerung von deklarativen Spezifikationen über bestehende Kubernetes-Manifeste. Änderungen werden in einer separaten Datei namens `kustomization.yaml` gespeichert. Kustomize liest die Datei `kustomization.yaml` und die Kubernetes-Spezifikationsdateien (Manifeste) und gibt die fertigen Ressourcen aus, wobei die Änderungen aus `kustomization.yaml` zusätzlich zu den Standard-Kubernetes-Dateien angewendet werden.

Wir haben uns dabei an die folgende Ordnerstruktur gehalten:

```
/
├── base
│   ├── configmap.yaml
│   ├── service.yaml
│   ├── deployment.yaml
│   └── kustomize.yaml
├── development
│   └── kustomize.yaml
└── production
    └── kustomize.yaml
```

Im *base*-Ordner ist jeweils die Standardkonfiguration, welche der Konfiguration im Namespace development entspricht. Im *development*-Ordner kann die Konfiguration für den development-Namespace noch spezifischer definiert werden. Im *production*-Ordner werden die nötigen Konfigurationen für den production-Namespace definiert und angewendet.

Im `kustomization.yaml`-File wird jeweils definiert, welche Konfigurationsfiles angewendet werden sollen oder welche Konfigurationen überschrieben werden müssen.

Kustomize ist bereits im Befehlszeilenprogramm `kubectl` integriert und alle definierten Kustomize-Konfigurationen können wie folgt angewendet werden:

```
1    kubectl apply -k deployment/production
```

Auflistung 18.6: Kustomize Deployment

18.5.2 Rolling Updates

Als Update-Strategie des Deployments wurde *RollingUpdate* gewählt. Damit kann definiert werden, dass immer mindestens ein Pod läuft und bei einem Update auf eine neue Version die alte noch laufende Version erst terminiert wird, wenn der neue Pod im *Running State* ist. Sobald das Deployment angepasst und neu deployed wird, wird diese Update Strategie angewendet.

Wenn aber nur der Source-Code verändert und ein neues Docker-Image gebildet wird, erkennt Kubernetes keine Änderung zum aktuell deployten Stand und führt standardmässig keine Änderung durch.

Damit wir das gesamte Deployment über die Pipeline redeployen können, ohne Downtime der Pods zu haben, gibt es zwei Möglichkeiten. Zum einen könnte über die Pipeline ein Timestamp oder eine Art Versioning über einen Tag oder eine Annotation ins Deployment-File dynamisch eingefügt werden. Die zweite Möglichkeit ist, ein `Rollout Restart` durchzuführen. Damit wird das gesamte Deployment neu deployed und die gewünschte Update-Strategie wird dabei berücksichtigt.

```
1    deploy:
2    ...
3        script:
4        - kubectl apply -k deployment/production
5        - kubectl rollout restart deployment -n $PROD_NS
             $DEPLOYMENT_NAME
```

Auflistung 18.7: Ausschnitt Pipeline Production Kubernetes-Deployment

Wir haben uns für die zweite Variante mit dem `Rollout Restart` entschieden, weil wir so kein zusätzliches Versioning in der Pipeline benötigen und nach dem gewöhnlichen Deployment noch den Rollout Restart Job mitschicken.

18.5.3 Ingress Indexierung von neu deployten Pods

Wir haben initial einen Ingress-Controller im *default*-Namespace deployed und in den *development* & *production*-Namespaces einen Ingress. Die eingehenden Anfragen an den Ingress Controller soll dieser anhand des Hosts an den entsprechenden Ingress weiterleiten. Dies hat grundsätzlich funktioniert, allerdings wurden neu deployte Pods nicht mehr richtig vom Ingress indexiert und die Anfrage wurde nicht an den Pod weitergeleitet.

Aus diesem Grund haben wir in beiden Namespaces neben dem Ingress auch einen eigenen Ingress-Controller deployed. Damit werden auch neu deployte Pods richtig erkannt und die Anfragen werden wie gewünscht weitergeleitet.

```
gianfluetsch@XPS15:~$ k get ingress -n production ingress-lgt
NAME          CLASS              HOSTS               ADDRESS          PORTS     AGE
ingress-lgt   nginx-production   parkonomy.lgt.com   20.203.204.173   80, 443   3d21h
gianfluetsch@XPS15:~$ k get ingressclass -n production nginx-production
NAME               CONTROLLER                                          PARAMETERS   AGE
nginx-production   production.kalberers.ch/ingress-nginx-production   <none>       49d
gianfluetsch@XPS15:~$
```

Abbildung 18.4: Ingress & Ingress Controller

18.6 Frontend/ Presentation Components

In diesem Kapitel werden die Herausforderungen beschrieben, welche im Frontend stattgefunden haben.

18.6.1 Microfrontends

Da wir uns im Frontend für die Microservice und gegen die Monolithische Architektur entschieden haben, mussten wir uns darüber schlau machen. Diesen Ansatz kannten wir bis jetzt noch gar nicht. Im Internet fanden wir zudem nicht viele Beispiele bzw. Artikel dazu. Die Idee von uns war eine Webseite mit verschiedenen Komponenten aufzubauen. Das stellte sich als schwierig heraus, da wir sehr viele kleine Komponenten entwickeln mussten und dies schnell unübersichtlich wurde. Ausserdem gibt es wenige Erfahrungberichte zu diesem Thema und somit auch nur begrenzt Hilfestellungen. Daher entschieden wir uns jeweils nur eine Navigation mit einer Komponente einblenden zu lassen. Das war zudem für die Entwicklung einfacher, da ganze Seiten einfach eingeblendet werden können und weniger Abhängigkeiten existieren.

Um dies umzusetzen, haben wir uns an diesem Artikel orientiert. Dabei geht es um den Ansatz, dass die einzelnen Microservices als SPA entwickelt werden und schlussendlich in eine JavaScript Datei kompiliert werden. Diese JavaScript Dateien können in einem zentralen Frontend eingebunden und somit kann die Funktionalität erweitert werden. Das ist ein ähnlicher Ansatz wie die Web API, welche wir im Backend verwendet haben. Diese fungiert auch als eine zentrale Schnittstelle zum Backend. Der Unterschied ist die zentrale Web App im Frontend, welche die gesamte Funktionalität bei sich einbettet, was im Backend nicht der Fall ist. Dort werden die Anfragen „nur“ weitergeleitet.

Eine weiteres Problem trat beim Routing der verschiedenen Seiten auf. Dabei wussten wir nicht genau wie wir die richtige Seite einblenden können, weil ein Microservice mehrere Seiten besitzt. Ausserdem können diese nicht wie gewohnt als React Komponente abgerufen werden. Dieses Problem konnten wir mit dem React Router lösen. Wir haben verschiedene Routen definiert, um die jeweiligen Seiten einzublenden. Somit konnten wir die Navigation im Web App so bauen, dass diese einfach den URL Pfad ändern musste um eine neue Seite einzublenden. Hinzu kam, dass nun die URL nicht identisch sein durfte, wie die URL der einzelnen Microservices. Da ansonsten die Anfrage an die Microservices weitergeleitet wird, wir aber von dort nur die JavaScript Datei abholen müssen um diese einzubinden. Die Anfrage muss vom Web App behandelt werden. Die URL Pfade sind nun nicht nach dem Best-Practive-Prinzip, aber sie erfüllen den Zweck.

In den folgenden Zeilen haben wir kurz beschrieben, wie die Microservices in der Web App eingebunden werden:

Damit die Funktionalität der Microservices eingebettet werden kann, müssen diese eine Funktion anbieten. Hier im Beispiel der Buchung Microservice.

```
1 window.renderBuchung = (containerId, history) => {
2   const container = document.getElementById(containerId);
3   const root = createRoot(container);
4   root.render(<App history={history} />);
5 };
```

Auflistung 18.8: Bereitstellung der Methoden von Microfrontends in index.js

In der Web App muss ein Platzhalter in der Webseite definiert werden, damit die Funktionalität da eingebunden werden kann.

```
    <div id="Buchung-container"></div>
```

Auflistung 18.9: Platzhalter für Einbettung der Funktionalität

Nun kann mittels folgendem Code die JavaScript Datei der jeweiligen Microservices eingebettet werden.

```
const renderMicroFrontend = () => {
    window[`render${componentName}`](
    `${componentName}-container`, history, pathName
    );
};

fetch(`${host}${componentPathName}/asset-manifest.json`)
.then((res) => res.json())
.then((manifest) => {
    const script = document.createElement("script");
    script.id = scriptId;
    script.crossOrigin = "";
    script.src = `${host}${manifest.files["main.js"]}`;
    script.onload = () => {
    renderMicroFrontend();
    };
    document.head.appendChild(script);
});
```

Auflistung 18.10: Aufruf der Methoden von Microfrontends in Web App

18.6.2 Nginx Environment Variables

Um Ressourcen zu sparen, wollten wir die Frontend Container in jeweils einem Nginx Container laufen lassen. Allerdings mussten wir feststellen, dass die Environment Variablen nicht so funktionierten, wie wir das von der lokalen Ausführung gewohnt waren. Nach einigen Recherche wurde uns schnell klar, dass das am Nginx Container lag. Nginx ignoriert die Environment Variablen und somit konnte unsere geplante Lösung mit .env Dateien zu arbeiten nicht ohne Weiteres umgesetzt werden. Im Internet fanden wir einen hilfreichen Artikel, welcher einen Lösungsansatz aufführte. Dabei wird ein Skript ausgeführt bevor der Nginx Container gestartet ist, um die Environment Variablen auszulesen und in einer Datei zu speichern. Diese Datei wird dann im index.html eingebetten und somit werden die Variablen für die Applikation zugänglich gemacht.

```
<script src="%PUBLIC_URL%/env-config.js"></script>
```

Auflistung 18.11: Einbettung Environment Variablen in index.html

18.7 Backend/ Fachservices

In diesem Kapitel werden die Herausforderungen beschrieben, welche hauptsächlich im Backend stattgefunden haben oder zumindest die grösste Herausforderung bei der Umsetzung darstellten.

18.7.1 RabbitMQ

Eine spezielle Herausforderung bei allen Microservices, welche Messages über *RabbitMQ* empfangen müssen, war, dass ein Background Prozess durchgehend die Queue auf neue Messages abfragen muss. Da dieser Service nur einmalig instanziert und über die gesamte Laufzeit des Microservices aktiv sein soll, muss dieser als scoped Service in einem Singleton verwendet werden [46].
Ein scoped Service bedeutet, dass in jedem Scope eine neue Instanz des Services erstellt wird. Der Service verhält sich so, wie sie innerhalb eines Scopes ein Singleton wäre.
Dies ist allerdings ein *Code-Smell* und sollte grundsätzlich vermieden werden.

Der Versuch, `scoped` Services innerhalb von Singletons zu verwenden, kann zu sogenannten Captive Dependencies führen, die alle möglichen unangenehmen Fehler und Speicherlecks verursachen können. Für unser Problem ist dies jedoch die beste und „schönste“ Lösung.

Der ASP.NET Core Dependency Injection Container hat einen Root-`IServiceProvider`, der zur Auflösung von Singleton-Services verwendet wird. Für Scoped Services muss der Container zunächst einen neuen Scope erstellen und jeder hat wiederum seinen eigenen `IServiceProvider`. Auf scoped Services kann nur vom `IServiceProvider` innerhalb des eigenen Scopes zugegriffen werden und nicht vom Root-`IServiceProvider`.

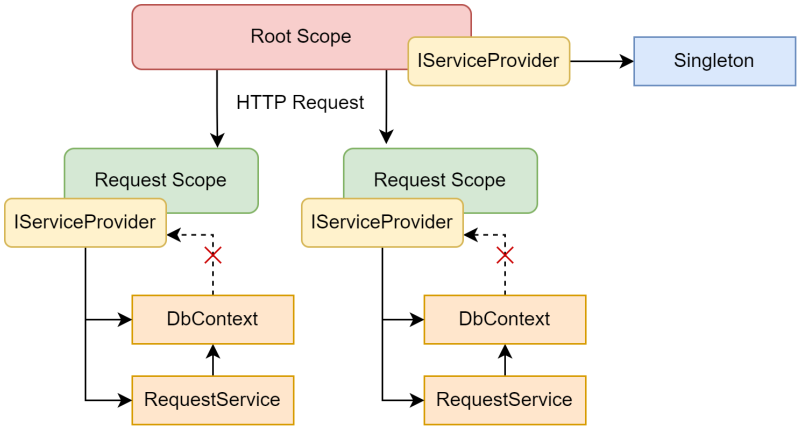


Abbildung 18.5: Problem Scoped Services mit Singletons

Jede Anfrage erstellt einen neuen Scope sowie einen eigenen `IServiceProvider`. Scoped Services, welche vom scoped Provider aufgelöst werden, sind spezifisch für die jeweiligen Anfrage und werden entfernt sobald die Anfrage endet.

Um scoped-Services innerhalb eines Singletons verwenden zu können, muss ein Scope manuell erstellt werden. So ein Scope kann erstellt werden, indem ein `IServiceScopeFactory` in den Singleton Service injected wird. Dies funktioniert aus dem Grund, weil `IServiceScopeFactory` selbst auch ein Singleton ist.

```
public class RabbitMQBackgroundService : BackgroundService
{
    private readonly IServiceProvider serviceProvider;
    private async Task ExecuteReceiveAsync(
            CancellationToken stoppingToken, LoginIdentity
               loginIdentity)
    {
        using (IServiceScope scope =
           this.serviceProvider.CreateScope())
        {
            IRabbitMQService rabbitMQService =
                scope.ServiceProvider.
                GetRequiredService<IRabbitMQService>();
            await rabbitMQService.ExecuteReceiveAsync(
                stoppingToken, loginIdentity);
        }
    }
}
```

Auflistung 18.12: Scoped Service innerhalb von einem Singleton erstellen

Der erstellte Scope verfügt nun über einen eigenen `IServiceProvider`, auf welchen zugegriffen werden kann, um den Scoped-Service aufzulösen.

Dabei ist es wichtig sicherzustellen, dass der Scope nur so lange wie nötig existiert. Damit können Probleme mit unverlierbaren Abhängigkeiten vermieden werden. Dazu gehört auch, dass eine `using`-Anweisung verwendet wird. Dadurch wird sichergestellt, dass der Scope ordnungsgemäss entfernt wird, sobald dieser nicht mehr benötigt wird.

18.7.2 Consul

Ein Problem, welches beim *Consul* während der Umsetzung aufgetreten war, ist, dass im *Production*-Namespace die Frontend Pods den Consul nicht erreichen konnten. Somit konnte die Abfrage, nach dem jeweilgen Port über welchen der benötigte Backend-Service erreichbar ist, nicht ausgeführt werden.

Im *development*-Namespace hat allerdings alles wunschgemäss funktioniert.

Die Konfiguration für den *Consul* wird im *Web-API* Pod mitgegeben. Aufgrund dessen, dass beide Namespaces genau gleich aufgebaut und somit auch deren Konfigurationen gleich sind, musste das Problem bei einer nicht oder falsch angewendeten Konfiguration liegen. In den Logs konnten wir keine Hinweise darauf finden. Es gibt allerdings von Kustomize einen Befehl, mit welchem die Konfiguration überprüft werden kann, bevor diese angewendet wird.

```
    kubectl kustomize <kustomize-folder>
```

Auflistung 18.13: Verifizierung der Kustomize Konfiguration

Damit wird die anzuwendende Konfiguration aufgelistet und kann mit der Soll-Konfiguration verglichen werden.

```
apiVersion: v1
data:
    aspnetcore_environment: Development
    consul_host: consul-server.development
    consul_port: "8500"
    consul_scheme: http
kind: ConfigMap
metadata:
    name: backend-webapi-config
    namespace: development
```

Auflistung 18.14: Development-Konfiguration

```
apiVersion: v1
data:
    aspnetcore_environment: Production
    consul_host: consul-server.production
    consul_port: '"8500"'
    consul_scheme: http
kind: ConfigMap
metadata:
    name: backend-webapi-config
    namespace: production
```

Auflistung 18.15: Production-Konfiguration

Wenn diese zwei Konfigurationen nun verglichen werden, ist ersichtlich, dass der Port im *production*-Namespace nicht richtig gesetzt wird.

In der ursprünglichen ConfigMap, welche im *development*-Namespace direkt angewendet wird, ist der Port in Anführungs- und Schlusszeichen angegeben. In der Kustomize Konfiguration, welche die Angaben für den *production*-Namespace überschreibt, wurde der Port initial auch so angegeben.

Aufgrund der falschen Syntax mussten die Anführungs- und Schlusszeichen dort entfernt werden. Mit dieser Änderung konnten die Frontend-Pods die Backend-Pods wieder wie gewünscht erreichen, resp. deren Anfragen richtig weiterleiten.

18.8 Vergleich Wireframes & Applikation

Im folgenden Kapitel möchten wir die wichtigsten geplanten Funktionalitäten anhand der in Kapitel 8.5.3.2 erstellten Wireframes mit der effektiv umgesetzten Applikation gegenüberstellen.

18.8.1 Booking Confirmation

18.8.2 Login

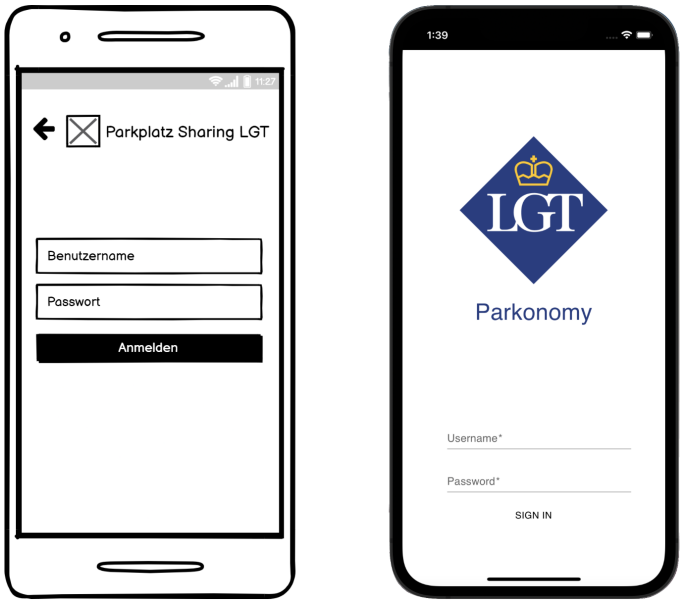


Abbildung 18.6: Vergleich Login

18.8.3 Navigation

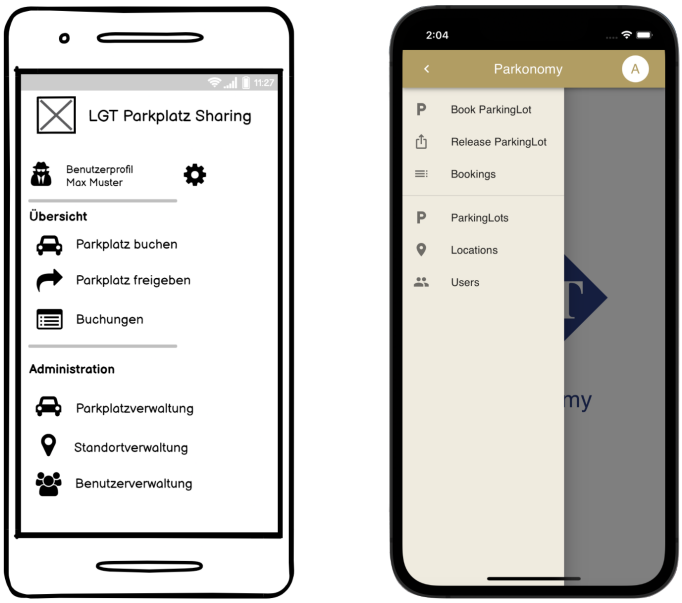


Abbildung 18.7: Vergleich Navigation

18.8.4 Release ParkingLot

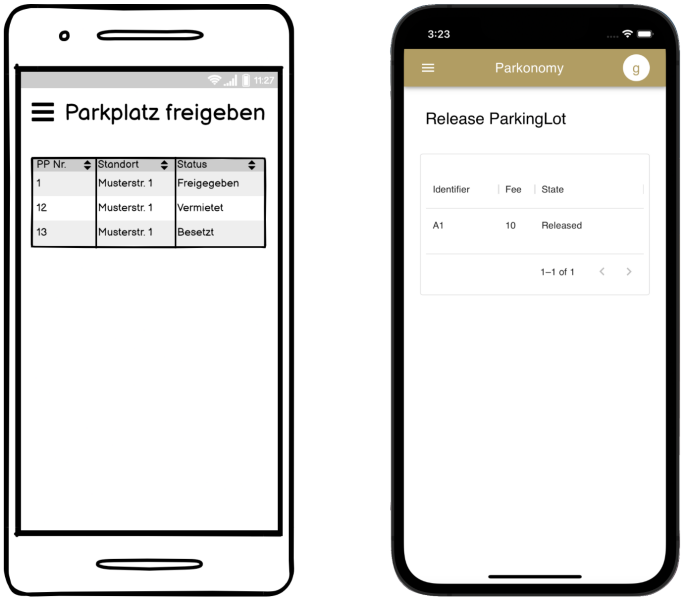


Abbildung 18.8: Vergleich Parkplatz freigeben

18.8.5 Search ParkingLot

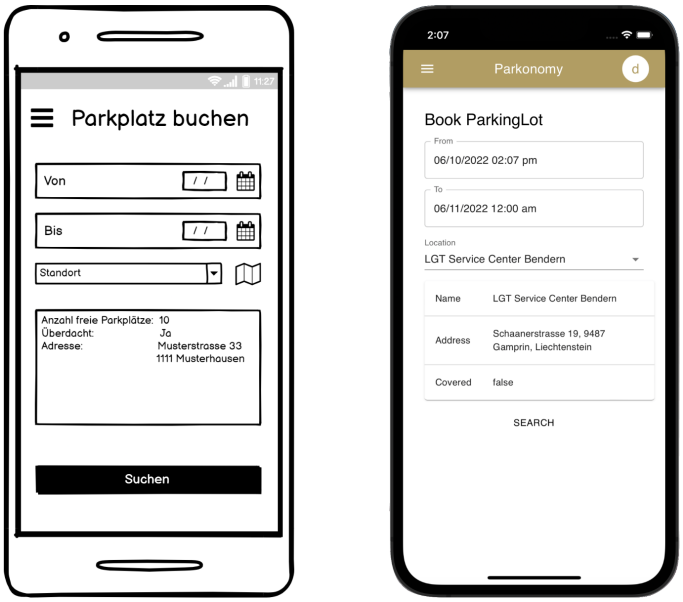


Abbildung 18.9: Vergleich Parkplatz suchen

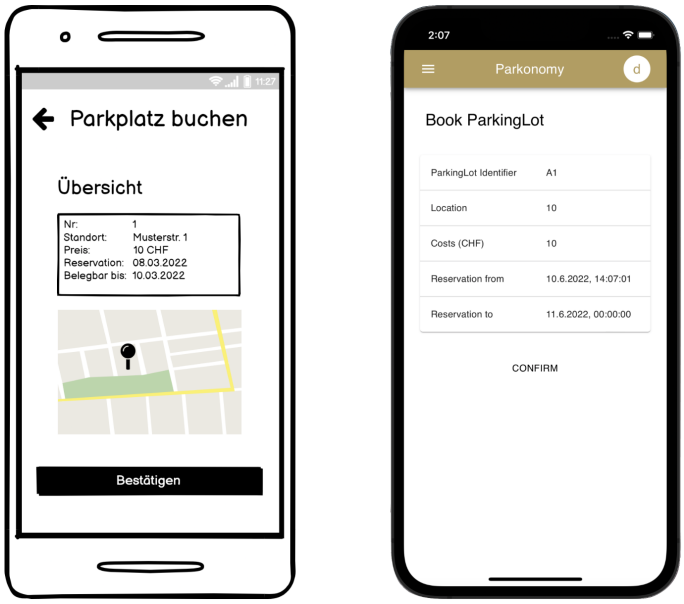


Abbildung 18.10: Vergleich Parkplatz buchen

18.8.6 Book ParkingLot

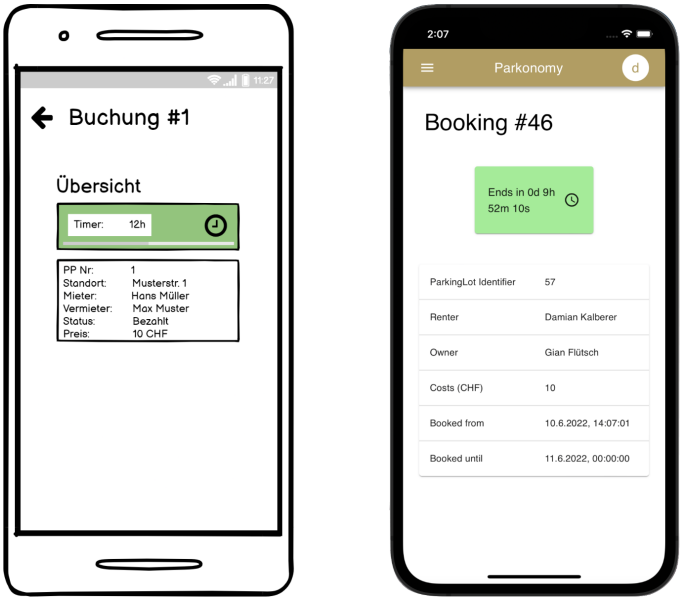


Abbildung 18.11: Vergleich Buchungsbestätigung

18.8.7 Fazit

Abschliessend können wir sagen, dass wir uns insgesamt sehr gut an die zu Beginn geplanten Wireframes halten konnten. Wie bereits erwähnt, konnten aufgrund der beschränkten Zeit für diese Arbeit, nicht ganz alle Funktionalitäten umgesetzt oder nur beschränkt umgesetzt werden.

Wir sind aber dennoch sehr zufrieden mit dem erreichten Resultat.

19 Gitpod Erfahrungsbericht

In diesem Kapitel wird beschrieben, wie die Entwicklung und Anwendung mit der evaluierten GitPod self-hosted Cloud IDE funktioniert hat. Weiter möchten wir unsere gemachten Erfahrungen erläutern, sowie ein Fazit dazu abgeben. Damit soll der LGT ein Erfahrungsbericht aus erster Hand angeboten werden können.

19.1 Allgemein

Die Installation und initiale Konfiguration von Gitpod ging sehr einfach vonstatten und alle wichtigen Einstellungen sind über das UI direkt ersichtlich oder in ihrer Dokumentation ausführlich beschrieben.

19.1.1 GitLab Provider

Zu Beginn muss der gewünschte VCS Provider mit Gitpod verbunden werden. In unserem Fall ist dies GitLab. Diese Providerintegration ist in der Dokumentation sehr gut beschrieben und aus diesem Grund gehen wir im Rahmen dieser Arbeit nicht mehr genauer darauf ein.

19.2 User Experience

Das UI ist sehr übersichtlich aufgebaut und man findet alle relevanten Einstellungen schnell. Repositories aus dem verlinkten VCS-Provider können über *New Workspace* ausgewählt und mit dem hinterlegten Docker Image geöffnet werden.

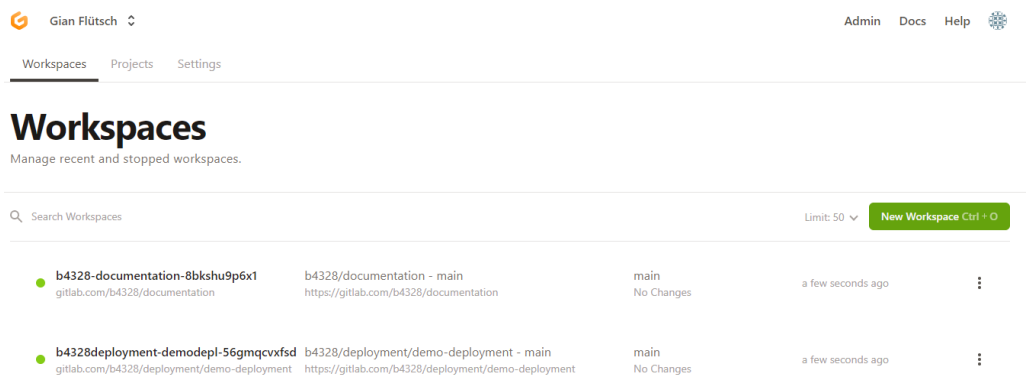


Abbildung 19.1: Gitpod UI

19.2.1 Browser Settings

Um eine bessere User-Experience gewährleisten zu können und unter anderem das Copy/ Paste zu unterstützen, ohne dass jedesmal ein Popup bestätigt werden muss, kann die Gitpod URL wie in der Gitpod-Dokumentation beschrieben gewhitelistet werden.

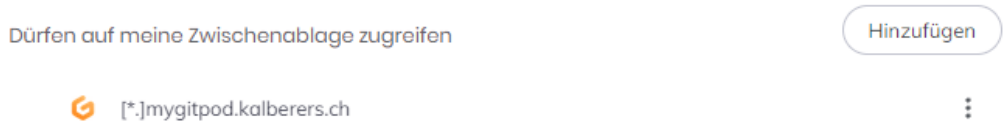


Abbildung 19.2: Gitpod Clipboard Browser Settings

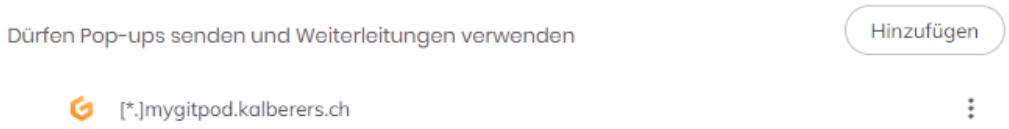


Abbildung 19.3: Gitpod Popups Browser Settings

19.3 Entwicklung

Bei der Entwicklung stellt man fast keine Unterschiede zu einer lokalen Installation von VS Code fest. Zudem können über das *.gitpod.yml*-File unterschiedliche Gitpod Standardkonfigurationen erstellt werden, welche für alle User gelten. So können unter anderem direkt Extensions installiert werden oder das Projekt kann direkt schon gebildet werden. Weiter können Benutzerspezifische Anpassungen vorgenommen werden, welche zentral in der Cloud gespeichert und danach auf alle Gitpod Workspaces angewendet werden.

19.3.1 Docker

Gitpod bietet verschiedene Docker-Images an, in welchen direkt entwickelt werden kann. Dies ist sehr praktisch, weil so alle wichtigen Befehle für einen gewissen Tech-Stack bereits vorhanden sind und unterstützt werden. Falls kein Docker-Image von Gitpod für eine Technologie angeboten wird, kann selber ein Docker-Image erstellt werden und dieses über eine Container-Registry in den Workspace gepullt werden.

Ein weiterer wichtiger Punkt ist, dass weitere Docker-Container im Gitpod Workspace gestartet werden können. Dies vor allem wenn Abhängigkeiten zu externen Systemen resp. zu anderen Containern von Nöten sind. Dabei kann wie gewohnt ein einzelner Docker-Container oder direkt mehrere über Docker Compose gestartet werden.

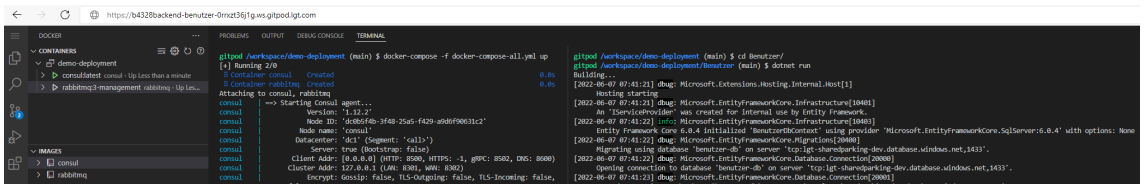


Abbildung 19.4: Entwicklung mit zusätzlichen Docker Containern in Gitpod

Auf diese Container können aus der aktuellen Umgebung wie gewohnt zugegriffen oder über eine API eingebunden werden.

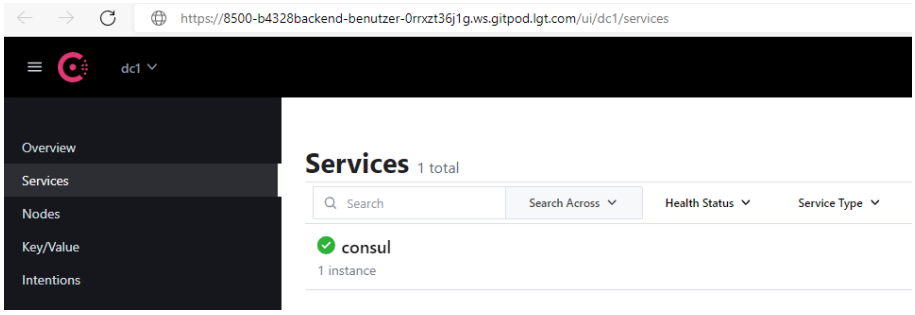


Abbildung 19.5: Zugriff auf UI eines Dockers innerhalb von Gitpod

19.4 Extensions

Wie bei der lokalen VS Code Installation können auch in Gitpod Extensions hinzugefügt und somit die IDE dem Bedürfnis entsprechend angepasst werden. Es handelt sich allerdings nicht um den gleichen Extension-Marketplace und es können nicht alle Extensions, welche gewohnt in VS Code vorhanden sind, installiert werden. VS Code greift auf den VS Code Marketplace zu, während Gitpod auf die Open VSX Registry zugreift.

Ein Grossteil der Extensions aus dem VS Code Marketplace werden auch in der Open VSX Registry angeboten. Für die Umsetzung unserer Bachelorarbeit haben wir einzig eine Extension für das Testing von ASP.NET vermisst, was aber nicht weiter schlimm war.

19.5 Workspace Konfiguration

Für jedes Repository können im `.gitpod.yml`-File Konfigurationen erstellt werden, welche beim Start vom Workspace und für alle Entwickler angewendet werden.

Zum einen kann ein spezifisches Docker-Image angegeben werden, in welchem entwickelt werden soll. Zudem können vordefinierte Tasks (15.2.2) beim Starten des Gitpod-Workspaces ausgeführt werden. Weiter können VS Code-Extensions installiert werden. Alle möglichen `gitpod.yml` Konfigurationen sind in der Gitpod-Dokumentation beschrieben.

```
image: gitpod/workspace-dotnet

# Commands to start on workspace startup
tasks:
  - init: dotnet build
    command: dotnet run

# VS Code Extensions (Open VSX Marketplace)
vscode:
    extensions:
      - ms-azuretools.vscode-docker
```

Auflistung 19.1: .gitpod.yml

Zusätzlich zu diesen globalen Einstellungen pro Repository können Benutzereinstellungen von den Entwicklern selbst gesetzt werden. Diese Einstellungen werden in einem Azure Container Storage gespeichert und wiederum global auf alle Gitpod-Workspaces des jeweiligen Benutzers angewendet.

19.6 Fazit

Insgesamt haben wir einen sehr positiven Eindruck von der evaluierten Cloud IDE erhalten. Es gibt, abgesehen von der geringeren Auswahl an Extensions, praktisch keinen Unterschied im Vergleich zur lokalen VS Code Installation. Aufgrund der guten Unterstützung und Integration von Docker muss praktisch auf nichts verzichtet werden.

Weiter wurden wir von der schnellen Startzeit eines neuen Workspaces positiv überrascht. Wenn ein Repository mit einem noch nicht gespeicherten Docker-Image gestartet wird, dauert es initial ein wenig länger. Sobald das Image aber in der Azure Container Registry gespeichert ist, ist der Ladevorgang sehr schnell und vergleichbar mit der Startdauer der lokalen VS Code Installation. Ein weiterer Vorteil ist, dass die Cloud IDE von jedem Gerät aus On-Demand verfügbar ist.

Bei der Entwicklung mit Gitpod muss auf nichts verzichtet werden und gleichzeitig fällt der hohe Aufwand für das Betriebsteam weg. Alle Extensions, Images und SDKs können zentral in der Cloud verwaltet und nichts muss mehr lokal auf den Geräten der Entwickler aktualisiert werden.

Wir sind überzeugt, dass Gitpod eine gute Alternative zu den gewöhnlichen lokalen Installationen von IDEs ist und der LGT, insbesondere ihrem Betriebsteam, einen Mehrwert bringt.

20 Usability Test

In diesem Kapitel geht es darum, die Usability-Tests der Applikation zu dokumentieren und Schlüsse daraus zu ziehen. Der Usability-Test prüft die Wahrnehmung, die Oberfläche, die Nutzbarkeit sowie die Funktionalität aus Sicht der Benutzer.

20.1 Ziele

Unser Ziel ist es, mit dem Usability-Test herauszufinden, wie gut die Benutzer mit der Applikation klar kommen und ob für sie direkt ersichtlich ist, was sich wo befindet. Dabei stellen wir den Testpersonen kleine Fragen und Aufgaben. Somit können wir beobachten, wo noch mögliche Schwierigkeiten vorhanden sind und was sich bewährt. Die genauen Aufgaben sind unter 20.3 ersichtlich.

Für den Usability-Test setzen wir unsere Hauptziele darauf, dass die Aufgaben *spezifisch* sowie *messbar* sind und nach *Priorität* geordnet werden können. Weiter möchten wir den Usability-Test auf verschiedenen Plattformen (Browser, iOS oder Android) durchführen. Somit kann sich der Benutzer besser auf die Applikation konzentrieren und eine mögliche Umgewöhnung von der Darstellung zwischen den beiden Betriebssystemen hat keinen Einfluss.

20.2 Teilnehmer

Den Usability-Test führen wir direkt mit Mitarbeitenden der LGT durch. Dabei ist es uns wichtig, dass wir nicht nur mit Mitarbeitern, welche bei der Entwicklung schon dabei waren den Test durchführen, sondern mit Personen welche die Applikation noch gar nicht kennen. Damit wir die verschiedenen Rollen und deren Funktionen innerhalb der Applikation testen können, möchten wir den Test mit *Mitarbeitern*, *Parkplatzbesitzern* sowie *Administratoren* durchführen.

Die genauen Beschreibungen und deren Funktionen sind unter 8.3.1 ersichtlich.

Ein grosser Vorteil eines Usability-Tests vor Ort ist es, dass viel mehr Rückschlüsse aus dem Test gezogen werden können. Dazu gehört nicht nur die direkte Rückmeldung des Testers, sondern auch seine Mimik, seine Vorgehensweise sowie mögliche Stolpersteine. Zudem kann der Test dynamisch angepasst und erweitert oder der Testbenutzer bei Problemen unterstützt werden.

Zum Schluss können wir mit einem persönlichen Gespräch noch zusätzliche wünschenswerte Features aufnehmen.

20.3 Planung

Nummer	Beschreibung	Testperson	Erwartetes Ergebnis
UT01	Applikation wird auf Browser, iOS und Android Geräten korrekt dargestellt und ist nutzbar.	Benutzer	PWA Applikation kann heruntergeladen werden.
UT02	Die Mobile-First optimierte Applikation ist auch im Browser nutzbar.	Benutzer	Darstellung sauber
UT03	Farbenblinde können die Applikation nutzen.	Farbenblinder	Alle Funktionen sind vollständig durch sehbehinderte Personen nutzbar.
UT04	Sehbeeinträchtigte können die Applikation ohne Probleme nutzen.	Sehbeeinträchtigter	Alle Funktionen und Informationen können genutzt werden. Verschiedene Hilfsmittel können in der Applikation eingesetzt werden, wie zum Beispiel das Vergrössern.
UT05	Die Funktionsweise der Oberfläche ist den Benutzern ohne Einführung klar.	Benutzer	Die Benutzer finden sich in der Applikation zurecht.
UT06	Die Funktionsweise der Oberfläche ist zur Nutzung der Applikation ausreichend.	Benutzer	Die Benutzer sind zufrieden mit den bestehenden Funktionen und würden die Applikation im Berufsalltag benutzen.
UT07	Die Applikation ist einfach gehalten und der Benutzer kommt mit wenigen Schritten zum Ziel.	Benutzer	Die Benutzer müssen keine überflüssigen Eingaben tätigen.
UT08	Die Applikation vereinfacht den Arbeitsalltag.	Benutzer	Die Benutzer haben durch die Nutzung der Applikation einen Mehrwert.
UT09	Fehler in der Applikation werden korrekt dargestellt.	Benutzer	Die Benutzer erhalten bei einem Fehler aussagekräftige Rückmeldungen.

UT10	Benutzer, Parkplätze und Standorte können erstellt werden.	Administrator	Der Administrator kann einzelne oder mehrere Benutzer, Parkplätze sowie Standorte in einer für ihn angemessenen Variante erstellen.
UT11	Benutzer können ihre Profilangaben selbstständig ändern.	Benutzer	Ein Benutzer kann sein Passwort bzw. Benutzername, E-Mail, Vorname und Nachname einfach ändern.
UT12	Ein Parkplatz kann für eine bestimmte Zeitdauer freigegeben werden.	Parkplatzbesitzer	Der Parkplatzbesitzer kann seinen Parkplatz ohne Probleme freigeben.
UT13	Ein Parkplatz kann für eine bestimmte Zeitdauer gebucht werden.	Mitarbeiter	Der Mitarbeiter kann einen freigegebenen Parkplatz ohne Probleme buchen.
UT14	Die Buchungen können übersichtlich dargestellt und abgerufen werden.	Benutzer	Der Benutzer kann seine Buchungen abrufen und findet die gewünschten Informationen.

Tabelle 20.1: Planung Usability-Test

20.4 Ablauf

Der Usability-Test wird mit einem Moderator durchgeführt, damit eine bessere Interaktion, direktes Feedback eingeholt sowie die Gedanken der Testbenutzer während den verschiedenen Aufgaben nachvollzogen werden können. Dabei ist es wichtig, dass der Moderator dem Testbenutzer hilft, aber nicht die Führung übernimmt.

Um die Gedankengänge des Testbenutzers möglichst gut zu kennen, wird dieser vom Moderator animiert:

- laut zu denken
- sagen, wenn etwas gesucht aber nicht gefunden wird
- eine Rückmeldung zu geben, ob das Resultat der Erwartung entspricht

Der genaue Ablauf des Usability-Tests sowie die jeweiligen Funktionen der Mitarbeiter sind nachfolgend ersichtlich.

Funktion	**Beschreibung**
Mitarbeiter	1. Login 2. Parkplatz buchen 3. Buchungen einsehen 4. Profilangaben ändern 5. Logout
Parkplatzbesitzer	1. Login 2. Parkplatz freigeben 3. Parkplatz buchen 4. Buchungen einsehen 5. Profilangaben ändern 6. Logout
Administrator	1. Login 2. Benutzer erstellen / editieren / löschen 3. Standort erstellen / editieren / löschen 4. Parkplatz erstellen / editieren / löschen 5. Parkplatz freigeben 6. Parkplatz buchen 7. Buchungen einsehen 8. Profilangaben ändern 9. Logout

Tabelle 20.2: Ablauf Usability-Test

20.5 Resultate

In diesem Abschnitt werden die Resultate des Usability-Tests ausgewertet. Der Usability-Test konnte mit verschiedenen Testbenutzern der LGT durchgeführt werden. Dazu gehörten drei Administratoren, ein Parkplatzbesitzer sowie zwei Mitarbeiter. Wir konnten dabei viele *qualitative Daten* sammeln, um Einblicke in die von den Teilnehmern eingeschlagenen Wege sowie die aufgetretenen Probleme zu erhalten. In einem kurzen Abschlussgespräch konnten wir über zusätzliche Ideen für Funktionen oder sonstige Verbesserungen mit den Mitarbeitern der LGT brainstormen.

20.5.1 Auswertung

Die Usability-Tests wurden im Kapitel 20.3 bereits genau beschrieben und aufgelistet. Aus diesem Grund werden im Folgenden nur die Bewertung sowie die erstellten Notizen zusammengefasst aufgelistet, welche wir noch nicht umsetzen konnten. Die kompletten Ergebnisse des Usability-Test sind im Anhang H ersichtlich.

Nummer	Bewertung (1-5)	Bemerkungen
UT05	4.2	• Zeit- und Datumauswahl nicht intuitiv für iOS Benutzer • Tabellenklick mit Button kennzeichnen, da nicht von Anfang an klar • App Bar geht nur über Button weg und nicht mit Klick in den Hintergrund wie gewohnt • Alles gefunden ohne Hilfe
UT06	4.2	• Karte für Standort des Parkplatzes
UT07	5	• Sehr zufrieden
UT08	4.6	• Mehrwert in Form von Geld
UT09	4.7	• Beim Import via CSV ist Error Handling nicht gut gelöst • Passwortanforderung nicht klar • Meist aussagekräftige Nachrichten

UT10	4.7	• Eingaben bei einem Fehler in der Erstellung/ Bearbeitung sind weg • Intuitiv gelöst
UT11	5	• Passwortwechsel mit Rückmeldung, dass es funktioniert hat
UT12	4.8	• EndDate nicht automatisch geändert bei einer Anpassungen des StartDate • Eingabe des Zeitraums wird nicht geprüft
UT13	4.7	• Bei Buchung über eine grössere Zeitdauer, Möglichkeiten einblenden die man hat • Mit wenigen Schritten zum Ziel
UT14	4.4	• Zu viele Informationen auf der Übersicht • Parkplatzname, Überdachung sowie Name des Standorts fehlen bei den Buchungsinformationen

Tabelle 20.3: Auswertung Usability-Test

20.5.1.1 Weitere mögliche Funktionen/Verbesserungen

- Homepage aussagekräftiger machen mit bspw. Übersicht Buchungen, aktive Buchungen oder Funktionalitäten der Applikation auflisten.
- Authentisierung via Anbindung an Active Directory z.B. Azure AD
- Bei Profil statt erster Buchstabe des Benutzernamen, der erste vom Vor- und Nachname anzeigen
- Besitz von mehreren Nummernschilder
- Wiederholende Freigabe, da teilweise immer am gleichen Tag Home-Office
- Mehrere Freigaben/ Buchungen gleichzeitig
- Bei Parkplatz ist möglicherweise mehr möglich als nur elektrisch ja/nein, z.B. Ampere
- Stornieren einer Buchung/ Freigabe
- Hilfestellung in App mit Kontaktinformationen, Copyright usw.
- Funktionsweise bei einem geteilten Parkplatz
- Import CSV besser erklären und Buttons ersichtlicher gestallten
- Karte um Parkhaus/ Parkplatz anzuzeigen
- Bei Buchungsübersicht Sortierung nach Aktiven, da möglicherweise in Zukunft reserviert wird welche noch nicht interessant sind
  - Aus der Sicht des Entwicklerteams macht es durchaus Sinn, die neusten Buchungen zuoberst zu haben. Auch wenn so eine im Moment aktive Buchung in diesem Fall nicht zuoberst wäre. Es ist aber trotzdem möglich, die Buchungen nach einem anderen Kriterium zu Filtern. Zudem könnte in einer weiteren Ausbaustufe die aktiven Buchungen nochmals grafisch gekennzeichnet werden.
- Benutzer sollen sich selber registrieren können
  - Da der Administrator zuerst prüfen muss, ob es diesen Benutzer überhaupt gibt und dieser einen Parkplatz hat, ist es schwierig diesen Punkt umzusetzen. Wir schätzen, dass dieser Punkt sich erübrigt, wenn die Benutzer in einer nächsten Ausbaustufe automatisch erfasst werden.

21 Qualitätssicherung

In diesem Kapitel gehen wir auf die in Kapitel 6 analysierten Risiken ein. Der Fokus liegt dabei, ob ein Risiko eingetreten ist und wie mit einem allfällig entstandenen Schaden umgegangen wurde oder wie die Risiken mitigiert werden konnten.

Abschliessend möchten wir auch auf die nicht-funktionalen Anforderungen, welche in Kapitel 8.4 definiert wurden und deren Einhaltung nach Abschluss des Projektes eingehen.

21.1 Risikomanagement

Im Kapitel 6 wurden die möglichen Risiken und ihre Auswirkungen auf das Projekt analysiert und dokumentiert. In diesem Kapitel soll aufgezeigt werden, welche vorhersehbaren Risiken eingetreten sind und welche unvorhergesehene dazukamen.

21.1.1 Probleme bei der Umsetzung

Das vorgesehene Risiko „Probleme bei der Umsetzung“ aus Kapitel 6.1.1 ist während dem Projekt zu einem gewissen Grad eingetreten. Bei der Umsetzung von Microservices im Frontend war das Risiko lange nicht mitigiert. Wir fanden wenige Hilfestellungen zu dieser Thematik und investierten verhältnismässig viel Zeit, um verschiedene Ansätze auszuprobieren. Schlussendlich haben wir das Problem so gelöst, dass wir alle Komponenten in der Web App einbinden und somit eine zentrale Anlaufstelle für das Frontend haben. Mit zusätzlichem Aufwand konnten wir den Zeitplan des Projekts halten und mussten keine weiteren Massnahmen ergreifen.

Insgesamt entstand ein Schaden von 24h, was unter dem maximalen Schaden von 48h liegt.

21.1.2 Unterschätzung des Aufwandes

Das vorgesehene Risiko „Unterschätzung des Aufwandes“ aus Kapitel 6.1.3 ist während dem Projekt eingetreten. Aufgrund dem Einsatz von neuen Technologien sowie moderner Architekturen, benötigten wir mehr Einarbeitungszeit als zuerst angenommen. Ausserdem haben wir uns für die Umsetzungsphase sehr viel vorgenommen und den Aufwand unterschätzt sowie die Reserven zu gering geplant. Glücklicherweise konnten wir mit zusätzlichem Aufwand den Zeitplan des Projekts halten und mussten deshalb keine weiteren Massnahmen ergreifen.

Insgesamt entstand ein Schaden von 48h, was dem geschätzten maximalen Schaden entspricht.

21.1.3 Fazit

Da ansonsten keine Risiken eingetreten sind, sind wir sehr zufrieden mit unseren Vorbeugungsmassnahmen. Der zeitliche Verlust bzw. Mehraufwand der eingetretenen Risiken ist im Rahmen und konnte gut bewerkstelligt werden.

21.2 Nicht-funktionale Anforderungen

Wie schon im Kapitel 8.4 haben wir die nicht-funktionalen Anforderungen in die jeweilige Kategorie gruppiert. Dies ermöglicht eine bessere Übersicht und zusammengehörende NFRs sind besser erkennbar.

Damit der Rahmen dieses Berichts nicht gesprengt wird, gehen wir in diesem Kapitel nicht mehr auf alle NFRs ein, sondern nur noch auf die wichtigsten erreichten sowie die nicht erreichten NFRs.

21.2.1 Cloud

In dieser Kategorie befinden sich alle NFRs, welche mit der Cloud in Verbindung stehen.

21.2.1.1 NFR02

Um über die GitLab CI/CD Pipeline Docker Images zu builden sowie in die Cloud deployen zu können, sollen stets genügend Runners zur Verfügung stehen.

Die Shared-Runner Limite wurde während der Umsetzung schnell erreicht. Aufgrund eines dedizierten Runners, welcher auf einem Server der OST gehosted wurde, konnte dieses NFR stets eingehalten werden.

21.2.1.2 NFR03

Der Azure Tenant soll stetig genügend Ressourcen vorhanden haben, um die nötigen Deployments starten zu können.

Aufgrund von *Autoscaling* skaliert der Tenant die benötigten Ressourcen automatisch. Falls ein definiertes Limit erreicht wird, kann dieses über einen *Increase Quota Limit* Request trotzdem weiter erhöht werden.

21.2.1.3 NFR04

Die benötigten Azure Ressourcen für die Evaluation sowie den Betrieb sollen ohne Kosteneinschränkungen benutzt werden können.

Dies wurde durch den privaten Azure Tenant der LGT und den darauf hinterlegten Zahlungsinformationen gewährleistet.

21.2.2 Cloud Native Applikation

In dieser Kategorie befinden sich alle NFRs, welche mit der Applikation sowie deren Funktionalität zu tun haben.

21.2.2.1 NFR07

Dieses NFR besagt, dass das Auslesen von Daten eine erfolgreiche Anmeldung benötigt. Dies wurde durch die Implementation einer Autorisierung in der Applikation umgesetzt. Dabei wird der generierte JWT-Token im *local Storage* vom Browser gespeichert und bei jedem Request auf die Gültigkeit geprüft. Dies wird auf jedem Backend-Microservice überprüft.

21.2.2.2 NFR09

Die verschiedenen Komponenten sollen möglichst wenige Abhängigkeiten untereinander besitzen, damit einzelne Komponenten ohne Auswirkung auf die anderen ausgetauscht werden könnten.
Dieses NFR wurde aufgrund der Microservice-Architektur eingehalten. Die einzelnen Microservices haben keine direkte Abhängigkeiten untereinander.

Der Pfad zu notwendigen Schnittstellen wie andere Microservices, der Datenbank sowie zu RabbitMQ wurden alle über Umgebungsvariablen in die Pods gemappt. Diese können somit einfach und ohne Anpassung des Docker Images angepasst werden.

21.2.2.3 NFR11

Wenn die Ressourcenauslastung der Applikation einen gewissen Schwellenwert erreicht, sollen automatisch zusätzliche Ressourcen deployed werden und wenn diese nicht mehr benötigt werden die zusätzlichen Replicas wieder gelöscht werden. Bei diesem NFR muss zwischen *Cluster Autoscaling* und *Horizontal Pod Autoscaling (HPA)* unterschieden werden.

Unter *Cluster Autoscaling* wird die dynamische Skalierung von den Nodes innerhalb eines Clusters verstanden. Dies kann mit Azure sehr einfach aktiviert werden und wenn die bestehenden Nodes über einen gewissen Zeitraum zu stark ausgelastet sind, wird automatisch ein oder mehrere zusätzliche Nodes deployed und ins Kubernetes Cluster integriert.

Als *Horizontal Pod Autoscaling* wird das dynamische Skalieren von einzelnen Pods bezeichnet. Dafür kann mit jedem Deployment eine zusätzliche `HorizontalPodAutoscaler` Ressource deployed werden, welche die Auslastung überwacht. Bei der Überschreitung des definierten Threshold der CPU-Auslastung, werden automatisch weitere Pods deployed, solange die Replicaanzahl im definierten Range liegt.

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: backend-benutzer-hpa
  namespace: production
spec:
  maxReplicas: 10
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: backend-benutzer
  targetCPUUtilizationPercentage: 70
```

Auflistung 21.1: HorizontalPodAutoscaler

Dieses NFR haben wir aus Gründen der zu geringen Auslastung nicht deployed. Aufgrund der stateless aufgebauten Microservices sind alle Vorgaben dafür erfüllt und es könnte ein HPA wie beschrieben schnell und einfach integriert werden.

21.2.2.4 NFR15

Dieses NFR besagt, dass Klassen innerhalb der Applikation nicht länger als 300 Zeilen lang sein dürfen, um die Lesbarkeit von Klassen zu garantieren.

Frontend
Im Frontend wurde mit React gearbeitet und dabei `Function Components` verwendet. Somit haben diese Funktionen die Zeilenlänge überschritten. Da diese aber wie Klassen fungieren, haben wir die Zeilenlänge für Funktionen auf 300 erhöht und somit den NFR eingehalten.

Backend
Dieses NFR wurde im Backend weitestgehend eingehalten. Wo es nicht eingehalten werden konnte, wurde dies mit einem Kommentar signalisiert und begründet. Zumeist wurden Klassen mit starker Kohäsion durch Dokumentations-Kommentare oder durch die JWT-Validierung aufgeblasen und so grösser als 300 Zeilen. In solchen Fällen wurde eine Ausnahme definiert.

21.2.2.5 NFR16

Dieses NFR besagt, dass Methoden nicht länger als 30 Zeilen lang sein dürfen, um die Lesbarkeit von Methoden zu garantieren.

Frontend
Da, wie beim vorherigen NFR erwähnt, mit React `Function Components` gearbeitet und diese als Klasse definiert wurde, ist dieses NFR im Frontend hinfällig.

Backend
Diese Anforderung wurde im Backend eingehalten. Vereinzelt wurden bei starker Kohäsion innerhalb der Methode oder einfachen Konstrukten wie Switch, welche lange Methoden zur Folge haben, Ausnahmen definiert und dokumentiert.

21.2.2.6 NFR18

Dieses NFR besagt, dass der McCabe-Wert jeder Methode unter 10 liegen muss.

Frontend
Im Frontend konnte dieses NFR eingehalten werden.

Backend
Über alle Backend-Microservices haben nur 16 Methoden einen McCabe-Wert über fünf. Von diesen Methoden, welche einen McCabe-Wert über fünf haben, sind lediglich drei über dem vorgegebenen Limit von zehn.

Dies sind die `Message`-Methode in der Klasse ErrorExtensions (McCabe-Wert 12) und die `Update`-Methode in der Klasse `UserManager` sowie `ParkingLotManager` (McCabe-Wert 19 resp. 22). Der McCabe-Wert wurde in beiden Fällen aufgrund des enthaltenen Switch-Statements sowie bei den Manager-Methoden aufgrund der vielen GraphQL Argumente in die Höhe getrieben.

21.2.2.7 NFR19

Dieses NFR besagt, dass Methoden innerhalb der Applikation höchstens fünf Argumente haben dürfen.

Frontend
Im Frontend konnte dieses NFR nur bei Methoden nicht eingehalten werden, bei welchen die GraphQL-API mehrere Parameter fürs Backend benötigt. Da es sich dabei nur um wenige Methoden handelt, wurde dies vom Entwicklerteam so akzeptiert.

Backend
Im Backend konnte dieses NFR über alle verschiedenen Microservices bei nur sechs Methoden nicht eingehalten werden. Hierbei handelt es sich bei allen sechs Methoden um solche, welche direkt von der GraphQL-API aufgerufen werden. Ausserdem arbeiten alle von ihnen intern mit Managern zusammen, um ihre Aufgabe zu erfüllen.

Da es sich bei den betroffenen Methoden um API-Methoden handelt, wird die Argumentliste zusätzlich durch die Dependency Injection aufgebläht. Aufgrund dieses Umstandes und aufgrund der Tatsache, dass alle Argumente für einen erfolgreichen API-Call gebraucht werden, wurde beschlossen, dass diese Methoden so bestehen bleiben können.

21.2.2.8 NFR20

Dieses NFR besagt, dass die Testabdeckung des Codes mindestens 80% betragen muss.

Frontend
Im Frontend beträgt die Testabdeckung für alle Microservices über 80%.

Die Testabdeckung über alle Frontend Microservices beträgt 84.8%, was über dem angestrebten Wert von 80% liegt.

Backend
Im Backend beträgt die Testabdeckung über alle Microservices des testbaren, nicht automatisch generierten Codes 81%. Somit wurde der gewünschte Wert erreicht. Einzig beim `Buchung`-Microservice wurde der Wert mit 71.24% nicht erreicht. Der gewünschte Wert wurde damit unglücklicherweise nicht erreicht, dennoch konnten die wichtigsten Funktionalitäten, namentlich die Business Logik, grössenteils getestet werden. Somit kann trotzdem eine ausreichende Testabdeckung gewährleistet werden.

Das Entwicklerteam ist aber trotzdem der Meinung, da es sich dabei nur um einen einzigen Microservice handelt und die wichtigsten Funktionalitäten getestet wurden, dass NFR als erreicht betrachtet werden kann.

21.2.2.9 NFR22

Dieses NFR besagt, dass eine Installations- sowie eine Wartungsanleitung erstellt werden soll. Es wurde für alle erstellten Microservices eine Installationsanleitung in der Form einer *ReadMe*-Datei im jeweiligen Repository angelegt.

21.2.3 Cloud IDE

In dieser Kategorie befinden sich alle NFRs, welche mit der Cloud IDE zu tun haben.

21.2.3.1 NFR27

Die Cloud IDE kann durch einen Entwickler um zusätzliche Plugins und Einstellungen angepasst und erweitert werden.

Aufgrund der VS Code Integration in Gitpod können zusätzliche Plugins über das `.gitpod.yml`-File zentral für das Repository aber auch in den persönlichen Benutzereinstellungen angepasst werden. Alle persönlichen Einstellungen werden in einem Azure Storage Container gespeichert und werden auf jeden Gitpod Workspace des entsprechenden Benutzers angewendet.

22 Schlussfolgerungen

In diesem Kapitel geben wir abschliessend unser Fazit über das gesamte Projekt ab und möchten noch ein paar Worte zum Ausblick und weiteren Vorgehen für die entwickelte Applikation erläutern.

22.1 Kostenauswertung

Wir haben unsere komplette Infrastruktur in der Azure Cloud betrieben. Wir möchten nun nochmals auf die Kosten eingehen, welche beim Betrieb unserer Applikation in dieser Cloud anfallen. Bei Kosten in einer Cloud müssen verschiedene Punkte berücksichtigt werden. Es können verschiedene Ressourcen auf Knopfdruck hochgefahren und benutzt werden. Es können allerdings auch Ressourcen über einen längeren Zeitraum reserviert und somit die Kosten reduziert werden. Dies aus dem Grund, da der Cloud Provider eine konstante Auslastung besser planen kann und somit die Ressourcen günstiger anbietet. In unserem Fall haben wir die Ressourcen On-Demand gestartet und nicht über einen längeren Zeitraum reserviert, deshalb fiel unsere Betriebsrechnung auch eher teuer aus. Der Preis könnte noch ein wenig gesenkt werden, wenn das Modell gewechselt wird. Wenn nun die Kosten einer Cloud angeschaut werden, besteht zu Beginn immer das Gefühl, dass es extrem teuer sei. Dabei muss aber immer in Relation gesetzt werden, wie viel der Betrieb der ganzen Infrastruktur On Premise kosten würde. Dazu gehören Geräte-, Miet-, Strom- und Personalkosten (Liste nicht abschliessend). Es ist also sehr schwierig die Kosten vom lokalen Betrieb mit dem der Cloud zu vergleichen. Wir würden sogar fast behaupten, dass die meisten Firmen das gar nicht können.

Im Kapitel 10 haben wir bereits einen Kostenvoranschlag für die Cloud IDE sowie die Container Orchestrierung erstellt. Wir haben nun für unsere Arbeit folgende Ressourcen deployed, wobei auf dem Kubernetes Cluster unsere Applikation sowie die Cloud IDE lief. Bei der Cloud IDE hatten wir jedoch die gratis Lizenz. Deshalb geht es hier nur um die Kosten des Hostings.

Produkt	**Deployment**	**Kosten**
Kubernetes Cluster	Cloud IDE + Parkonomy	CHF 1′642.00
SQL Datenbanken	Parkonomy	CHF 35.50
DNS Zonen	Cloud IDE + Parkonomy	CHF 1.50
MySQL Server	Cloude IDE	CHF 167.50
Container Registry	Cloude IDE	CHF 50.00
Total		*CHF* 1′896.50

Tabelle 22.1: Kostenauswertung Monat Mai

Im April haben wir ein OpenShift Cluster für eine Woche getestet und nur Teile der Applikation darauf laufen gelassen. Man kann aus der Rechnung entnehmen, dass alleine schon der Cluster die Rechnung viel teurer macht.

Produkt	**Kosten**
OpenShift Cluster	CHF 584.00 * 4 = CHF 2′336.00
DNS Zonen	CHF 0.10 * 4 = CHF 0.40
MySQL Server	CHF 36.50* 4 = CHF 146.00
Container Registry	CHF 11.00 * 4 = CHF 44.00
Total	*CHF* 2′526.40

Tabelle 22.2: Kostenauswertung Monat April Hochrechnung

Aus den obenstehenden Tabellen kann entnommen werden, was ein Hosting der Cloud IDE bzw. der Applikation pro Monat in der Cloud bedeuten würde. Dabei war während unserer Bachelorarbeit die Auslastung auf den Services sehr klein. Das bedeutet, wenn nun die Applikation produktiv eingesetzt würde und somit die Auslastung steigt, entstehen auch höhere Kosten, da die Cloud die Ressourcen automatisch skaliert. Als nächster Schritt müsste nun eine Rechnung für den On Premise Betrieb erstellt werden. Da dies aber wie erwähnt sehr schwierig ist und es den Rahmen unserer Arbeit sprengen würde, haben wir darauf verzichtet.

22.2 Fazit

Die Frontend sowie die Backend Microservices, welche gemeinsam die Applikation „LGT Parkonomy“ darstellen und im Rahmen dieser Arbeit entwickelt wurden, erfüllen die wichtigsten Bedürfnisse des Kunden. Zusätzlich bieten sie eine solide Basis für eine nachhaltige Weiterentwicklung der Applikation.

Obschon nicht alle ursprünglich definierten Anforderungen der LGT umgesetzt wurden, konnten doch mit der Benutzerverwaltung, Parkplatzverwaltung sowie der Buchungsfunktionalität die wichtigsten Funktionalitäten korrekt und solide implementiert werden. Auch war von Beginn an klar, dass nicht alle definierten Szenarien im Rahmen dieser Arbeit implementiert werden. Dies vor allem aufgrund der vielen Evaluationen und der aufgebauten Infrastruktur in der Azure Cloud.

Besonders zu beachten gilt, dass bei der Implementierung auf eine gute Wartungsfähigkeit, Testabdeckung und Erweiterbarkeit geachtet wurde. Somit wurde der Grundstein für eine erfolgreiche Weiterführung des Projekts gelegt. Mit der Microservice Architektur sind die einzelnen Microservices nicht voneinander abhängig und können ohne Auswirkung auf die anderen Microservices ausgetauscht werden.

Aufgrund der modular aufgebauten Microservices konnten wir uns an die Empfehlungen der IDEAL Cloud Application Properties halten und sind überzeugt, einen soliden Grundstein für die Weiterentwicklung der Applikation gelegt zu haben.

Weiter konnte mit der evaluierten und im Kubernetes Cluster selbst gehosteten Cloud IDE ein neuer Ansatz für die Entwicklung bei der LGT aufgebaut und getestet werden. Ein detailliertes Fazit zu Gitpod ist im Kapitel 19 vorhanden.

Erwähnenswert ist ausserdem, dass die Applikation über die LGT-Domain unter `parkonomy.lgt.com` verfügbar ist.

22.2.1 On Premise Betrieb

Die „LGT Parkonomy“-Applikation wurde in einem Kubernetes Cluster in der Azure Cloud entwickelt. Aufgrund der Microservice-Architektur und den geringen Abhängigkeiten zu weiteren Cloud-Services, kann die Applikation mit nur kleinen Anpassungen auch auf einem On Premise Cluster betrieben werden. Die LGT setzt aktuell On Premise verschiedene OpenShift Cluster ein. Da OpenShift auf Kubernetes aufbaut, können auch gewöhnliche Kubernetes-Deployments auf OpenShift deployed werden. Somit können wir die Funktionsweise unserer Applikation für Kubernetes und OpenShift Cluster gewährleisten.

Die einzige Abhängigkeit zu Cloud-Services ist die Datenbank, welche im Entity Framework über eine Umgebungsvariable als `Connection String` hinterlegt ist. Somit kann die Datenbank einfach ausgetauscht werden, ohne dass ein neues Docker-Image gebildet werden muss.

22.2.1.1 Stats

Zum Schluss möchten wir noch ein paar Zahlen zu „LGT Parkonomy“ auflisten.

Insgesamt wurden von uns acht Microservices selber entwickelt, je vier im Frontend und vier im Backend. Zusätzlich sind mit dem `WebAPI`, `RabbitMQ` & `Consul` noch drei weitere Microservices involviert. Das ganze Deployment ins Kubernetes Cluster wurde mit insgesamt 94 `YAML`-Konfigurationsfiles automatisiert.

In der folgenden Tabelle sind die Codezeilen (Lines of Code) für die jeweiligen Microservices im Front- sowie Backend ersichtlich.

Frontend		Backend	
Microservice	**Lines of Code**	**Microservice**	**Lines of Code**
Web App	1'062	Login	4'559
Benutzer	2'315	Benutzer	6'265
Parkplatz	2'766	Parkplatz	6'235
Buchung	1'410	Buchung	3'081
Total	*7'553*		*20'140*

Tabelle 22.3: Übersicht Lines of Code

Dabei ist anzumerken, dass die vielen Codezeilen auf die Microservice Architektur zurückzuführen sind und sich der Code dadurch ein wenig „aufbläst". Wir sind aber trotzdem sehr zufrieden mit der evaluierten und aufgebauten Architektur und sind überzeugt, dass dies der richtige Ansatz für so eine Applikation ist.

Insgesamt wurden viele verschiedene Kubernetes Ressourcen erstellt und deployed. Wir möchten im Folgenden noch einen kurzen Überblick über die Anzahl der deployten Ressourcen innerhalb des *development* resp. *production*-Namespaces geben.

Kubernetes Ressource	**Anzahl**
Pods	21
Service	17
Daemonset	1
Deployments	10
Statefulset	2
Configmap	30
Secret	19
Ingress	2
Total	*102*

Tabelle 22.4: Übersicht Kubernetes Ressourcen

Diese Ressourcen sind je im *development* sowie *production* Namespace deployed.

22.3 Ausblick

Im Rahmen unserer Arbeit wurde die Applikation in beiden Namespaces des Kubernetes Clusters deployed und umfasst die für den täglichen Betrieb wichtigsten Funktionalitäten. Nach Abschluss dieser Arbeit werden alle erarbeiteten Unterlagen sowie das gesamte GitLab-Projekt der LGT zur Verfügung gestellt. Damit soll bei einer allfälligen Weiterentwicklung möglichst viel Know-How erhalten bleiben.

Alle noch offenen Punkte und zusätzlich wünschenswerten Features wurden als Issues im GitLab Issue Board erfasst. In einem nächsten Sprint ist die Anbindung an das Zugangssystem ein wichtiges Thema. Damit soll bei einer Buchung die Smartcard des entsprechenden Mitarbeiters für den gebuchten Parkplatz aktiviert werden, damit dieser die Barriere öffnen kann. Bei Ablauf der Buchung soll diese wieder deaktiviert werden. Weiter sollen die Benutzer automatisch in die Applikation importiert werden können oder noch besser die Benutzerverwaltung durch die Anbindung des Azure-AD ausgetauscht werden. Analog zu den Benutzern, sollen auch die Parkplätze aus einer schon bestehenden Datenbank automatisch importiert werden können. Ein wünschenswertes Feature ist zudem eine Karte, über welche der genaue Standort des Parkplatzes ersichtlich ist.

Weiter ist neben dem „LGT Parkonomy“-Modul ein Weiteres fürs Desksharing interessant. Dabei könnte ein Grossteil der Funktionalität von „LGT Parkonomy“ übernommen und auf den Desksharing Use-Case adaptiert werden. Für das „LGT Desksharing“-Modul müssten nur geringfügige Anpassungen durchgeführt werden.

Wir sind überzeugt, die Applikation der LGT in einem guten Zustand übergeben zu können und denken, dass durch die bessere Auslastung der Parkplätze ein Mehrwert für die Mitarbeitenden generiert werden konnte.

23 Zeitauswertung

In diesem Abschnitt wird der Zeitplan ausgewertet. Dies beinhaltet eine Analyse des Zeitaufwandes und die Evaluation der Einhaltung der einzelnen Meilensteine.

23.1 Meilensteine

Nachfolgend wird die Einhaltung der Meilensteine geprüft und eventuelle Eigenheiten der einzelnen Arbeitsprodukte erläutert. Hier gilt zu beachten, dass als Referenz zur Fertigstellung des Meilensteins nicht der Abschluss des Meilensteins in GitLab genommen wurde.

23.1.1 Projektplan

Der Meilenstein Projektplan wurde auf den **27.03.2022** geplant und konnte wie geplant fertiggestellt werden. Da die Erstellung eines Projektplans bereits aus der Studienarbeit bekannt war, konnte dieser zügig erstellt und mit dem Betreuer besprochen werden.

23.1.2 Requirements

Der Meilenstein Requirements wurde auf den **13.03.2022** geplant und konnte termingerecht abgeschlossen werden. Die definierten Use-Cases konnten mit dem Betreuer sowie dem Kunden zeitgerecht besprochen und abgenommen werden.

23.1.3 Architecture Prototype

Der Meilenstein Architecture Prototype wurde auf den **10.04.2022** geplant und konnte eingehalten werden. Ein Prototyp wurde mit den evaluierten Technologien und Architekturen aufgebaut und mit dem Kunden sowie dem Betreuer besprochen und abgenommen. Damit konnte eine solide Grundlage für die Umsetzungsphase gelegt werden.

23.1.4 End of Elaboration

Der Meilenstein End of Elaboration wurde auf den **24.04.2022** geplannt und konnte fristgerecht abgeschlossen werden. Es wurden noch kleine Anpassungen an der geplanten Architektur vorgenommen sowie Kriterien für das Endprodukt definiert.

23.1.5 End of Construction

Der Meilenstein End of Construction wurde auf den **29.05.2022** geplant und konnte mit einer kleinen Verzögerung am **01.06.2022** abgeschlossen werden. Die Verzögerung ist den eingetretenen Risiken zuzuschreiben. Da es sich bei der Verzögerung aber nur um drei Tage handelt, sind wir der Meinung, dass dies vernachlässigt werden kann.

23.1.6 End of Documentation

Der Meilenstein End of Documentation wurde auf den **12.06.2022** geplant und konnte erst am **14.06.2022** abgeschlossen werden. Dies ist auf die Verzögerung der Construction-Phase im Meilenstein End of Construction zurückzuführen.

23.1.7 Abgabe

Der Meilenstein Abgabe wurde auf den **17.06.2022** geplant und konnte einen Tag früher, am **16.06.2022**, abgeschlossen werden. Trotz der kleinen Verzögerung während der Construction-Phase, konnte die Dokumentation zügig fertiggestellt werden, was eine termingerechte Abgabe ermöglicht hat.

23.2 Zeitrapport

Für das Projekt wurden insgesamt 720h budgetiert, was pro Person 360h ergibt. Dies entspricht dem vorgesehenen Aufwand von acht ECTS Punkte des Moduls „Bachelorarbeit“.

Tatsächlich aufgewendet wurden im Rahmen der Bachelorarbeit **809h**, welche sich wie folgt auf die Projektmonate, Projektmitglieder und Aufwandsarten aufteilen:

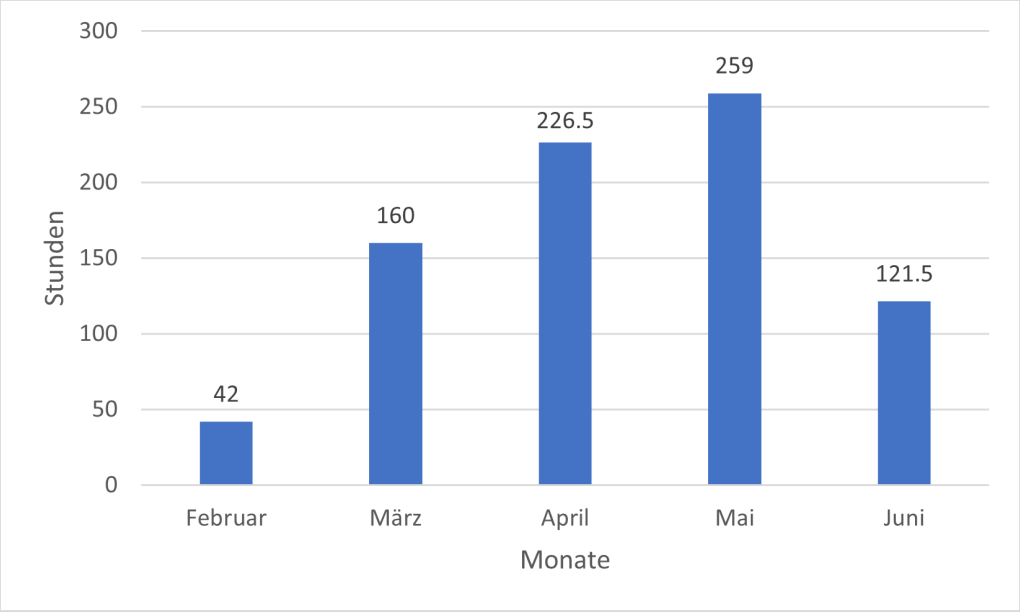


Abbildung 23.1: Zeitrapport, ausgewertet nach Monat

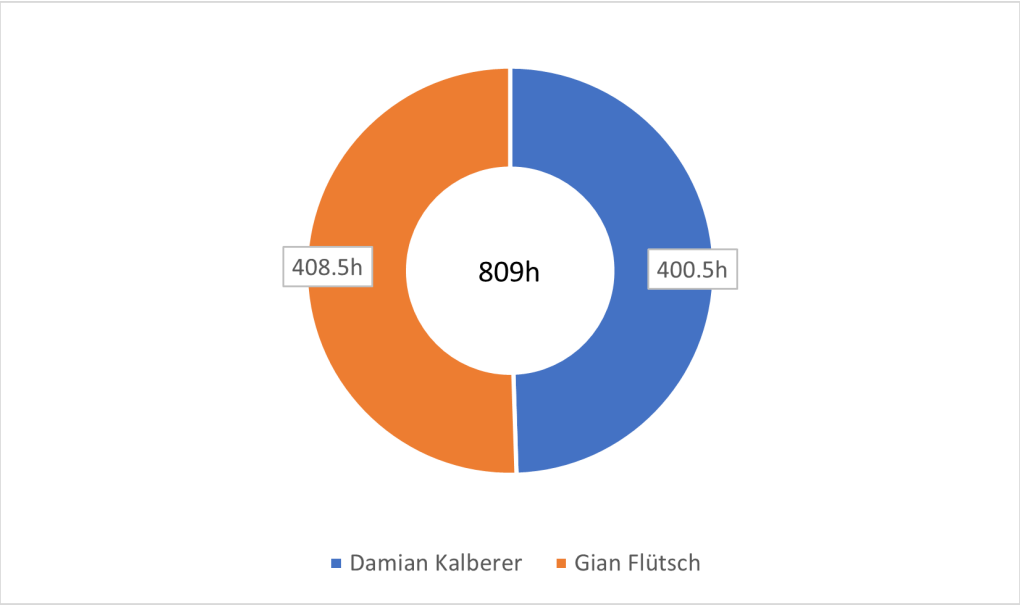


Abbildung 23.2: Zeitrapport, ausgewertet nach Personen

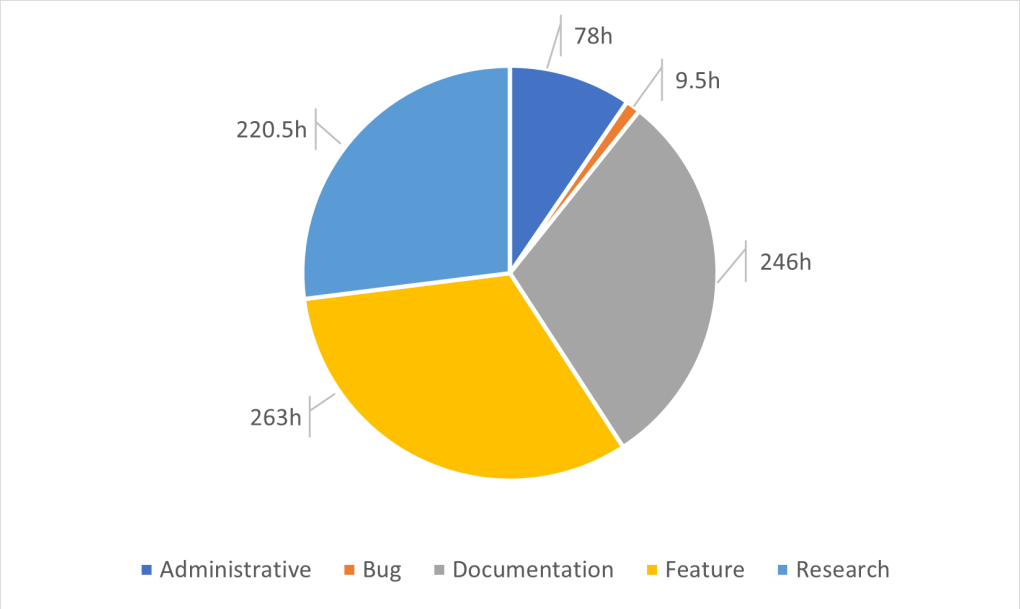


Abbildung 23.3: Zeitrapport, ausgewertet nach Kategorie

Insgesamt wurde mehr Zeit aufgewendet, als ursprünglich geplant war. Dies ist auf die eingetretenen Risiken 21.1 und den daraus entstandenen zusätzlichen Aufwand zurückzuführen. Insgesamt verursachten die eingegangenen Risiken einen Verlust von 78 Stunden, der durch zusätzlichen Aufwand in der Umsetzungsphase kompensiert wurde. Zieht man diesen Schaden ab, ergibt sich eine Differenz von **11 Stunden**. Somit sind wir sehr zufrieden mit unserer ursprünglichen Planung.

Teil V

Appendix

A Abschlussberichte

A.1 Damian Kalberer

Auf der Suche nach einem Thema habe ich bei meiner aktuellen Arbeitgeberin der LGT nachgefragt, ob das Interesse vorhanden ist, die Bachelorarbeit für sie zu machen. Die Rückmeldung war sehr positiv, da sie bereits mit früheren Arbeiten Erfahrungen gesammelt haben und immer viel daraus lernen konnten. Somit konnten wir auf die Suche nach einem geeigneten Thema gehen. Dabei ist uns viel Freiheit gelassen worden und wir konnten somit unsere eigenen Ideen einbringen. Das fand ich sehr spannend und ich konnte bereits sehr viel Lernen bevor die Bachelorarbeit überhaupt offiziel gestartet hat.

Zusammen mit Gian Flütsch diese Arbeit zu bewältigen, fand ich sehr hilfreich, da wir uns beide sehr gut in unserem Wissen ergänzen und bereits die Studienarbeit zusammen geschrieben haben.

Die Arbeit war zu Beginn wie erwartet sehr herausfordernd. Wir hatten es mit für uns unbekannten Technologien, wie OpenShift, Kubernetes, Cloud IDE, usw. zu tun, für welche wir uns allerdings auch sehr interessierten. Mit vielen Informationen im Web sowie den gleichzeitig belegten Modulen in der Schule konnte ich mir viel Wissen über diese Technologien aneignen. Auf dieses Wissen werde ich in Zukunft bestimmt zählen können, da ich in meinem zukünftigen Beruf als DevOps Enigneer viele Berührungspunkte mit den ausgewählten Technologien haben werde.

Die Zusammenarbeit mit unserem Betreuer Mirko und den beiden Mitarbeitern von mir, Markus und Christian von der LGT, schätzte ich sehr. Sie waren stets hilfsbereit und unkompliziert. Dabei stand für mich die Tatsache heraus, dass uns bei der Wahl der Azure Services vollstes Vertrauen gegeben wurde und wir verschiedene Produkte ohne Rücksicht auf die Kosten ausprobieren konnten. Das war sehr interessant und motivierte mich noch mehr um ein gutes Resultat bei der Arbeit abzuliefern.

Zum Abschluss kann ich sagen, dass ich die Bachelorarbeit als sehr interessante und herausfordernde Arbeit empfand, welche wir aus meiner Sicht gut gemeistert haben. Ich konnte mir neues Wissen aneignen und bestehendes Wissen festigen, welches ich wie bereits erwähnt in der Arbeitswelt gut gebrauchen kann.

A.2 Gian Flütsch

Bei der Suche nach einem spannenden Thema für unsere Bachelorarbeit stiessen wir bei der LGT und somit Damian's Arbeitgeberin auf grosses Interesse. Aufgrund unserer Interessen an der Cloud sowie Cloud Native Technologien, war ein Thema in diesem Gebiet unser Ziel. Bei der LGT wird aktuell stark in Richtung Cloud tendiert und deshalb war es für sie auch von Interesse, Learnings für kommende Projekte aus unserer Bachelorarbeit ziehen zu können.

Bei der genauen Themen- und Technologienwahl wurden uns sehr viele Freiheiten gelassen und es gab praktisch keine Vorgaben. Wir konnten somit auf der sogenannten „grünen Wiese" starten und verschiedene Technologien ausprobieren und evaluieren.

Die Entwicklung von einzelnen Microservices kannte ich bereits aus früheren Modulen sowie der Studienarbeit im letzten Semester. Diesmal wollten wir aber eine Applikation entwickeln, welche auf Microservices basiert und in einem Kubernetes Cluster in der Cloud betrieben wird. Mit der Entwicklung von einer solchen Applikation sowie deren Abhängigkeiten untereinander, investierten wir viel Zeit in die Evaluation der entsprechenden Architektur. Uns war es wichtig, eine gute Grundlage zu legen und nicht später aufgrund von Limitierungen gewisse Funktionalitäten nicht umsetzen zu können.

Die entwickelte Applikation sowie die zusätzlich benötigten Cloud-Services konnten wir im Azure Tenant der LGT deployen. Ein wichtiger Service dabei war der Azure Kubernetes Service (AKS). Dabei wird ein gewöhnliches Kubernetes Cluster in der Cloud zur Verfügung gestellt. Kubernetes kannte ich bereits aus früheren Modulen, lernte es aber während der Bachelorarbeit nochmals auf einem ganz anderen Level kennen.

Mit der in der Studienarbeit bereits erfolgreich eingesetzten Projektmanagementmethode Scrum+ konnten wir unser Wissen während der Bachelorarbeit nochmals vertiefen. Mit der gewählten agilen Vorgehensweise konnten wir sehr flexibel und effizient am Projekt arbeiten. Weiter wurde das Increment nach jedem Sprint auf dem Kubernetes Cluster redeployed und konnte vom Product Owner getestet und weiteres Feedback eingeholt werden. Dieses Feedback wurde wiederum direkt in den nächsten Sprint eingeplant und umgesetzt. Somit war der Kunde stets über den aktuellsten Stand informiert und die Entwicklung ging in die richtige Richtung.

Die Zusammenarbeit mit Damian war wie bereits bei der Studienarbeit sehr angenehm. Da wir bereits mehrere Projekte im Studium zusammen durchgeführt haben, kannten wir die jeweiligen Stärken und konnten uns sehr gut ergänzen.

Abschliessend gilt es zu sagen, dass ich die Arbeit als eine durchaus herausfordernde, aber auch als sehr spannende Aufgabe empfand. Ich konnte das im Studium erlernte Wissen anwenden, vertiefen und viele neue Erfahrungen sammeln. Mit dem Endprodukt bin ich sehr zufrieden und überzeugt, dass die gemachten Learnings der LGT einen Mehrwert bringen und dieses Wissen wiederum in zukünftige Cloud-Projekte fliessen wird.

B Literaturverzeichnis

[1] „Git Feature Branch Workflow.“ Englisch, Atlassian. (2022), Adresse: https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow (besucht am 26. 02. 2022).

[2] „ISO/IEC 9126-1:2001.“ Englisch, International Organization for Standardization. (2001), Adresse: https://www.iso.org/standard/22749.html.

[3] „Microsoft Azure Well-Architected Framework.“ Englisch, Microsoft. (2022-03-09), Adresse: https://docs.microsoft.com/en-us/azure/architecture/framework/.

[4] „Modulbeschreibung Cloud Solutions.“ Deutsch, OST - Ostschweizer Fachhochschule. (2022-03-04), Adresse: https://studien.rj.ost.ch/allModules/37167_M_CldSol.html.

[5] „Cloud Computing aus der Sicht des Anwendungsarchitekten.“ Deutsch, Institute for Software, HSR FHO. (2022-03-10), Adresse: https://www.ifs.hsr.ch/fileadmin/user_upload/customers/ifs.hsr.ch/Home/projekte/ZIO-CHOpenDay-CCaSAWAv10p.pdf.

[6] „Server-side web frameworks.“ Englisch, Mozilla. (2022-04-09), Adresse: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks.

[7] „Web Frameworks mit JavaScript und HTML5.“ Deutsch, Hyperformers. (2022-04-09), Adresse: https://hyperformers.com/web-frameworks/.

[8] „Single-page application.“ Englisch, Wikipedia. (2022-04-09), Adresse: https://en.wikipedia.org/wiki/Single-page_application.

[9] „Progressive web application.“ Englisch, Wikipedia. (2022-04-09), Adresse: https://en.wikipedia.org/wiki/Progressive_web_application.

[10] „Angular.“ Englisch, Google. (2022-04-09), Adresse: https://angular.io/.

[11] „React.“ Englisch, Meta Platforms. (2022-04-09), Adresse: https://reactjs.org/.

[12] „The Progressive JavaScript Framework.“ Englisch, VueJS. (2022-04-09), Adresse: https://vuejs.org/.

[13] „ASP.NET.“ Englisch, Microsoft. (2022-04-09), Adresse: https://dotnet.microsoft.com/en-us/apps/aspnet.

[14] „Express - Node.js web application framework.“ Englisch, OpenJS Foundation. (2022-04-09), Adresse: https://expressjs.com/.

[15] „Spring.“ Englisch, VMware. (2022-04-09), Adresse: https://spring.io/.

[16] „LGT - Private Banking und Asset Management.“ Deutsch, LGT. (2022-03-02), Adresse: https://www.lgt.com/.

[17] „Representational state transfer.“ Englisch, Wikipedia. (2022-04-09), Adresse: https://en.wikipedia.org/wiki/Representational_state_transfer.

[18] „gRPC.“ Englisch, Google. (2022-04-09), Adresse: https://grpc.io/.
[19] „GraphQL | A query language for your API.“ Englisch, GraphQL Foundation. (2022-04-09), Adresse: https://graphql.org/.
[20] „Monolithic Architecture.“ Englisch, Microservices.io. (2022-04-09), Adresse: https://microservices.io/patterns/monolithic.html.
[21] „Microservice Architecture.“ Englisch, Microservices.io. (2022-04-09), Adresse: https://microservices.io/patterns/microservices.html.
[22] „App Service.“ Englisch, Microsoft. (2022-03-23), Adresse: https://azure.microsoft.com/en-us/services/app-service/.
[23] „Azure Kubernetes Service (AKS).“ Englisch, Microsoft. (2022-03-23), Adresse: https://azure.microsoft.com/en-us/services/kubernetes-service/.
[24] „Azure Red Hat OpenShift.“ Englisch, Microsoft. (2022-03-23), Adresse: https://azure.microsoft.com/en-us/services/openshift/.
[25] „Azure Functions.“ Englisch, Microsoft. (2022-03-25), Adresse: https://azure.microsoft.com/en-us/services/functions/.
[26] „Blazing fast cloud developer environments.“ Englisch, GitHub. (2022-03-21), Adresse: https://github.com/features/codespaces.
[27] „Always Ready to Code.“ Englisch, GitPod. (2022-03-21), Adresse: https://www.gitpod.io/.
[28] „Codeanywhere is a Cloud IDE.“ Englisch, Codeanywhere Inc. (2022-03-21), Adresse: https://codeanywhere.com/.
[29] „Open VSX Registry.“ Englisch, Eclipse. (2022-03-23), Adresse: https://open-vsx.org/.
[30] „ESLint.“ Englisch, ESLint. (2022-04-24), Adresse: https://eslint.org/.
[31] „StyleCopAnalyzers.“ Englisch, StyleCopAnalyzers. (2022-04-24), Adresse: https://github.com/DotNetAnalyzers/StyleCopAnalyzers.
[32] „Menees Analyzers.“ Englisch, Menees. (2022-04-24), Adresse: https://github.com/menees/Analyzers.
[33] „React docgen.“ Englisch, Facebook. (2022-04-24), Adresse: https://github.com/reactjs/react-docgen.
[34] „Doxygen.“ Englisch, Doxygen. (2022-04-24), Adresse: https://www.doxygen.nl/index.html.
[35] „Jest.“ Englisch, Meta. (2022-04-24), Adresse: https://jestjs.io/.
[36] „Xunit.“ Englisch, .NET Foundation. (2022-04-24), Adresse: https://github.com/xunit/xunit.
[37] „Nsubstitute.“ Englisch, Nsubstitute. (2022-04-24), Adresse: https://nsubstitute.github.io/.
[38] „Coverlet.“ Englisch, .NET Foundation. (2022-04-24), Adresse: https://github.com/coverlet-coverage/coverlet.
[39] „Hot Chocolate.“ Englisch, ChilliCream. (2022-04-24), Adresse: https://github.com/ChilliCream/hotchocolate.
[40] „Entity Framework.“ Englisch, Microsoft. (2022-04-24), Adresse: https://docs.microsoft.com/en-us/ef/.
[41] „Azure SQL-Datenbank.“ Englisch, Microsoft. (2022-04-24), Adresse: https://azure.microsoft.com/de-de/products/azure-sql/database/.
[42] „gitpod-microsoft-aks-guide.“ Englisch, GitHub. (2022-04-25), Adresse: https://github.com/gitpod-io/gitpod-microsoft-aks-guide.

[43] „.gitpod.yml Reference.“ Englisch, Gitpod. (2022-04-25), Adresse: https://www.gitpod.io/docs/references/gitpod-yml.
[44] „Prebuilds.“ Englisch, Gitpod. (2022-04-25), Adresse: https://www.gitpod.io/docs/prebuilds.
[45] „Browser Settings.“ Englisch, Gitpod. (2022-04-25), Adresse: https://www.gitpod.io/docs/configure/browser-settings.
[46] „Using scoped services inside singletons.“ Englisch, DevDoc. (2022-05-03), Adresse: https://samwalpole.com/using-scoped-services-inside-singletons.
[47] „LaTeX– A document preparation system.“ Englisch, The LaTeXProject. (2022-02-26), Adresse: https://www.latex-project.org/.
[48] „Programmierschnittstelle.“ Deutsch, Wikipedia. (2022-02-28), Adresse: https://de.wikipedia.org/wiki/Programmierschnittstelle.
[49] „Continuous integration.“ Englisch, Wikipedia. (2022-04-09), Adresse: https://en.wikipedia.org/wiki/Continuous_integration.
[50] „Clockify Time Tracking.“ Englisch, Clockify. (2022-02-24), Adresse: https://clockify.me/.
[51] „Free and Open Search.“ Englisch, Elastic. (2022-06-12), Adresse: https://www.elastic.co/.
[52] „FURPS.“ Deutsch, Wikipedia. (2022-02-28), Adresse: https://en.wikipedia.org/wiki/FURPS.
[53] „GitLab.“ Englisch, GitLab. (2022-02-21), Adresse: https://gitlab.com/.
[54] „CI/CD pipelines.“ Englisch, GitLab. (2022-04-09), Adresse: https://docs.gitlab.com/ee/ci/pipelines/.
[55] „Grafana: The open observability platform.“ Englisch, Grafana. (2022-06-12), Adresse: https://grafana.com/.
[56] „Cohesion (computer science).“ Englisch, Wikipedia. (2022-06-12), Adresse: https://en.wikipedia.org/wiki/Cohesion_(computer_science).
[57] „Kustomize - Kubernetes native configuration management.“ Englisch, Kustomize. (2022-06-06), Adresse: https://kustomize.io/.
[58] „Lint (Programmierwerkzeug).“ Deutsch, Wikipedia. (2022-02-28), Adresse: https://de.wikipedia.org/wiki/Lint_(Programmierwerkzeug).
[59] „McCabe-Metrik.“ Deutsch, Wikipedia. (2022-02-28), Adresse: https://de.wikipedia.org/wiki/McCabe-Metrik.
[60] „Microsoft Teams.“ Deutsch, Microsoft. (2022-02-26), Adresse: https://www.microsoft.com/de-ch/microsoft-teams/group-chat-software.
[61] „Open Source Initiative.“ Englisch, opensource. (2022-03-04), Adresse: https://opensource.org/licenses/MIT.
[62] „Open Source Initiative.“ Englisch, opensource. (2022-03-04), Adresse: https://opensource.org/osd.
[63] „OST - Ostschweizer Fachhochschule.“ Deutsch, Gitpod. (2022-06-02), Adresse: https://www.ost.ch/de/.
[64] „SCS: Self-Contained Systems.“ Englisch, INNOQ. (2022-04-09), Adresse: https://scs-architecture.org/.
[65] „Single-responsibility principle.“ Englisch, Wikipedia. (2022-04-09), Adresse: https://en.wikipedia.org/wiki/Single-responsibility_principle.
[66] „The Twelve-Factor App.“ Deutsch, Heroku. (2022-03-04), Adresse: https://12factor.net/de/.

[67] „Usability-Test.“ Deutsch, Wikipedia. (2022-02-28), Adresse: https://de.wikipedia.org/wiki/Usability-Test.

[68] „Visual Studio Code.“ Englisch, Microsoft. (2022-06-07), Adresse: https://code.visualstudio.com/.

C Abbildungsverzeichnis

D Tabellenverzeichnis

E Auflistungsverzeichnis

F | Glossar

LaTeX

Textsystem zur Erstellung von technischen und wissenschaftlichen Dokumenten. Verfügbar unter [47]. 18

API

API steht für „application programming interface “, was auf Deutsch übersetzt Programmierschnittstelle bedeutet. Die API fungiert als Anbindungsstelle, über welche andere Softwares auf die angebotenen Funktionen zugreifen können. Mehr dazu: [48]. 11, 47, 67–70, 78, 81, 89, 94, 96–99, 104, 138, 155

CI

Steht für ”Continuous integration” und ist eine Methode, bei welcher verschiedene Komponenten fortlaufend zu einer Applikation zusammengefügt werden. Siehe auch: [49]. 92, 103, 160

Clockify

Tool für die Zeitschätzung und Erfassung. Verfügbar unter [50]. 17, 18, 27

Closed-Source

Als Closed-Source wird Software bezeichnet, bei welcher der Quellcode nicht öffentlich zugänglich gemacht wird. Diese Variante wird häufig durch Firmen verwendet, welche mit dieser Massnahme ihr intellektuelles Eigentum schützen wollen. 161

Cloud Native

Cloud Native ist ein Ansatz der sicherstellen soll, dass eine Applikation für die Cloud-Computing-Architektur konzipiert und entwickelt wird. Dabei werden die von der Cloud bereitgestellten Services genutzt und die Applikation basiert auf Microservices. 1, 10, 11, 15, 19, 54, 150

Definition of Done

Eine Liste von Aufgaben, welche erledigt sein müssen, bevor ein Projekt oder Arbeitspaket als abgeschlossen angesehen werden kann. 17, 27, 28

Dependency Injection

Beschreibt eine Software-Engineering-Technik, in welcher ein Objekt andere Objekte, welche es selbst verwendet, übergeben bekommt. Somit kann eine globale Instanz

die Abhängigkeiten verwalten und an die entsprechenden Objekte übergeben. Mehr dazu: **dependency-injection**. 117, 138

ELK-Stack

Der ELK-Stack besteht aus den Open-Source-Projekten *Elasticsearch*, *Logstash* und *Kibana*. Elasticsearch ist eine Suchmaschine und Analytics Engine. Logstash ist eine serverseitige Datenverarbeitungspipeline, die Daten aus unterschiedlichen Quellen gleichzeitig ingestiert, sie umwandelt und dann an einen Speicherort, z. B. an Elasticsearch, sendet. Kibana ermöglicht die Visualisierung von Daten durch Diagramme und Tabellen in Elasticsearch [51]. 76

End of Elaboration

Meilenstein in der RUP-Methodik, bei welchem die grössten Risiken behoben sein müssen. Bei diesem Meilenstein wird entschieden, ob mit dem Projekt so fortgefahren werden kann oder nicht. 19, 20, 160, 162

End of Elaboration Checklist

Eine Liste von Bedingungen, welche zum End of Elaboration erfüllt worden sein müssen. 20, 23, 104

Feature Branch Workflow

Workflow in Git, bei welchem die Entwicklung von Features in separaten Branches erfolgt. Dokumentiert unter [1]. 17

Feature-Freeze

Zeitpunkt in einem Projekt, ab welchem keine neuen Features mehr implementiert werden. 20, 21

FURPS

FURPS ist ein Modell aus dem Bereich der Softwarequalität. Es entstammt dem Englischen und fasst Qualitätsmerkmale von einer Software zusammen. Siehe auch [52]. 44

GitLab

Dienst zur Versionsverwaltung von Softwareprojekten. Verfügbar unter [53]. 12, 18, 27, 33, 35, 74, 75, 77, 79, 82–86, 105, 110, 123, 144, 145, 160, 163

GitLab CI/CD Pipeline

Eine Softwarekomponente von GitLab, welche es dem Benutzer erlaubt CI-Pipelines zu erstellen. Siehe auch: [54]. 1, 14, 27, 44, 45, 74, 75, 77, 79, 80, 104, 105, 135

Grafana

Grafana ist eine plattformübergreifende Open-Source-Anwendung zur grafischen Darstellung von Daten aus verschiedenen Datenquellen wie InfluxDB, MySQL, PostgreSQL, Prometheus und Graphite. Die erfassten Rohdaten können in verschiedenen Anzeigeformaten ausgegeben werden [55]. 76

IDEAL Cloud Application Properties

Die IDEAL Cloud Application Properties sind generisch gehaltene Empfehlungen für die Entwicklung einer guten Cloud Applikation [5]. Diese werden im Modul CldSol [4] vermittelt. 45, 142

Kohäsion

Als „Kohäsion“ wird in der Programmierung die Eigenschaft von Code bezeichnet, genau eine wohldefinierte Aufgabe abzubilden. Klassen/ Methoden mit starker Kohäsion bilden entsprechend genau eine Aufgabe ab, so wie dies vom Single-Responsibility-Prinzip vorgegeben wird [56]. 137, 138

Kustomize

Kustomize durchläuft ein Kubernetes-Manifest, um Konfigurationsoptionen ohne Forking hinzuzufügen, zu entfernen oder zu aktualisieren. Es ist sowohl als eigenständige Binärdatei als auch als native Funktion von kubectl verfügbar [57]. 113, 118

LGT

Die LGT ist ein international tätiges Finanzdienstleistungsunternehmen mit Sitz in Vaduz im Fürstentum Liechtenstein [16]. 1, 2, 10–12, 29, 31, 35–38, 44, 46, 50, 78, 81, 85, 94, 105, 110, 126, 127, 131, 135, 142, 144, 149, 150

Linter

Software zur statischen Code-Analyse. Wird oft zur Überprüfung der Code-Formatierung verwendet. Mehr dazu: [58]. 27

McCabe-Wert

Der McCabe-Wert, auch „Cyclomatic Complexity“ genannt, ist eine Messgrösse in der Softwareentwicklung, mithilfe dessen die Komplexität einer Software angegeben werden kann. Siehe auch: [59]. 46, 138

Microsoft Teams

Plattform für Kommunikation in Form von Chats und Anrufen. Verfügbar unter [60]. 18

MIT-Lizenz

Die MIT-Lizenz ist eine Open-Source-Softwarelizenz. Sie ermöglicht die Wiederverwendung der Software in Open-Source und Closed-Source Projekten. Der Lizenztext ist verfügbar unter: [61]. 36

NFR

Abkürzung für den englischen Begriff „non-functional requirement„ was auf Deutsch nicht-funktionale Anforderung heisst. plural 44, 135–139

Open-Source

Als Open-Source wird grundsätzlich eine Software bezeichnet, bei welcher der Quellcode öffentlich verfügbar ist. Zu Open-Source im eigentlichen Sinn gehört allerdings noch mehr. Die vollständige Definition kann hier eingesehen werden: [62]. 64, 76, 78, 160, 161, 163

OST

Die OST - Ostschweizer Fachhochschule ist eine Fachhochschule in der Ostschweiz, welche aus einem Zusammenschluss der Fachhochschulen FHS St. Gallen, HSR Rapperswil und NTB Buchs entstanden ist [63]. 110, 135

potentially shippable product

Softwareprodukt, welches in dieser Form an den Kunden ausgeliefert werden könnte. Alle implementierten Features sind komplett funktionsfähig. 20

PWA

Eine progressive Web-App verknüpft Eigenschaften von Websites mit vielen Merkmalen nativer mobiler Apps. Dadurch soll die Anwendung losgelöst vom Betriebssystem gleichermassen als Website sowie auch als App funktionieren [9]. 1, 11, 58, 59, 96, 97, 99, 104, 128

RUP

Rational Unified Process 19, 160, 162

SaaS

Software-as-a-Service bezeichnet einen Teilbereich des Cloud Computings. Dabei wird grundsätzlich eine Software in der Cloud des Anbieters betrieben und dem Kunden angeboten. 83–85, 87

Scrum+

Projektmethodik, welche Scrum mit dem End of Elaboration von RUP verbindet. 19, 150

Self-Contained System

Das Konzept des in sich geschlossenen Systems (Self-contained System) ist eine Architektur, bei der der Schwerpunkt bei der Aufteilung der Funktionalität in viele unabhängige Systeme liegt, sodass das vollständige logische System aus vielen kleineren Softwaresystemen zusammengesetzt ist. Auf diese Weise wird das Problem grosser Monolithen vermieden, die ständig wachsen und schliesslich nicht mehr gewartet werden können [64]. 97

Single-Responsibility-Prinzip

Ein Prinzip aus der Softwareentwicklung welches besagt, dass Softwarekomponenten nur eine fest definierte Aufgabe zu erfüllen haben. Mehr dazu: [65]. 53, 161

SPA

Eine Single-Page-Applikation (SPA) ist eine Website, welche neuen Daten vom Webserver dynamisch umschreibt, anstatt wie standardmässig üblich die ganze neue Seiten im Webbrowser zu laden. Das Ziel sind schnellere Übergänge, so dass sich die Website eher wie eine native Anwendung anfühlt [8]. 1, 11, 57, 58, 97, 115

Twelve-Factor App

Die Twelve-Factor App sind Empfehlungen für die Entwicklung einer guten Cloud Applikation. Diese wurden ursprünglich für die Heroku-Platform erstellt [66]. 45

Usability-Test

Ein Usability-Test ist ein Test, welcher mit Benutzern einer Applikation durchgeführt wird, um deren Bedienbarkeit für den Endbenutzer zu evaluieren. Mehr dazu: [67]. 21, 28, 127, 129, 131

VCS

Version Control System ist ein Softwareprogramm, welches Änderungen an einem Dateisystem verfolgt und verwaltet. Ein VCS bietet ausserdem Funktionen zur Zusammenarbeit, mit denen VCS-Benutzer diese Änderungen an Dateisystemen untereinander teilen und integrieren können. Bekannte VCS Systeme sind GitHub, GitLab sowie BitBucket. 74, 82–86, 110, 123

VS Code

Visual Studio Code ist ein kostenloser Open-Source-Editor von Microsoft. Visual Studio Code ist auf allen Plattformen verfügbar und basiert auf dem Electron-Framework [68]. 83, 84, 87, 124–126, 139

G Zusatzinformationen

G.1 Anforderungsanalyse

G.1.1 Wireframes

G.1.1.1 App

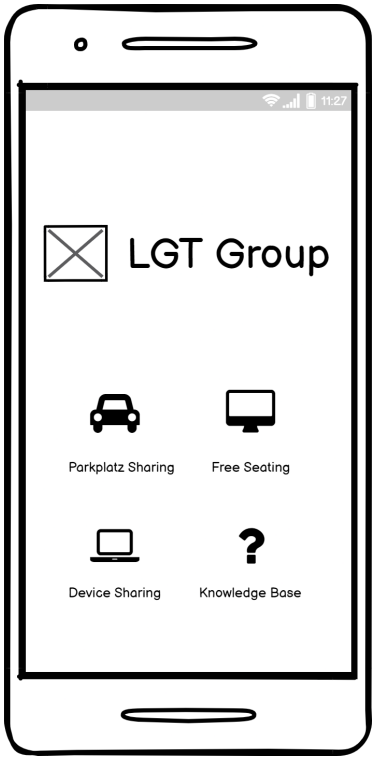


Abbildung G.1: Wireframe Ressourcen Applikation

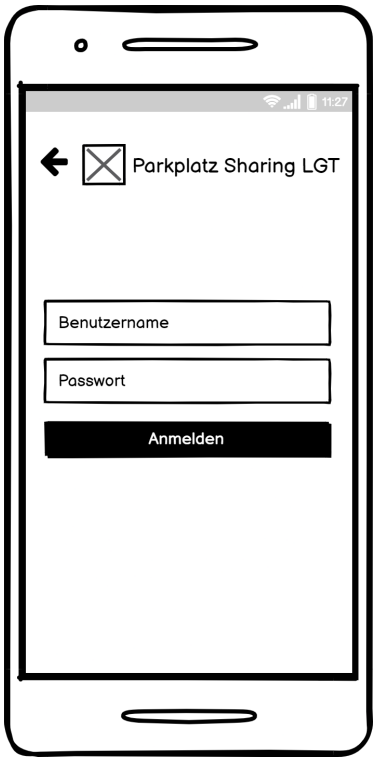


Abbildung G.2: Wireframe Login

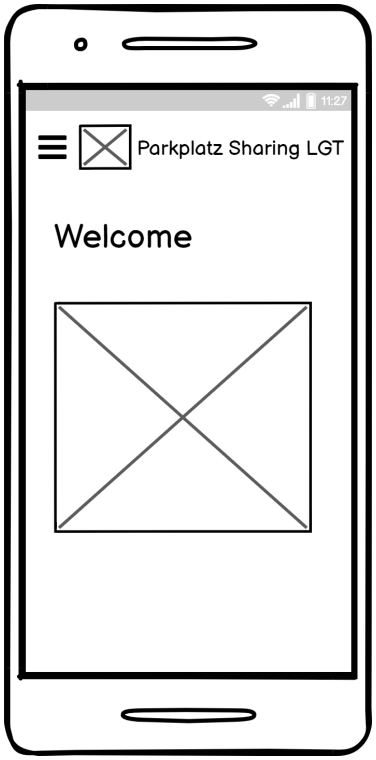


Abbildung G.3: Wireframe Landing Page

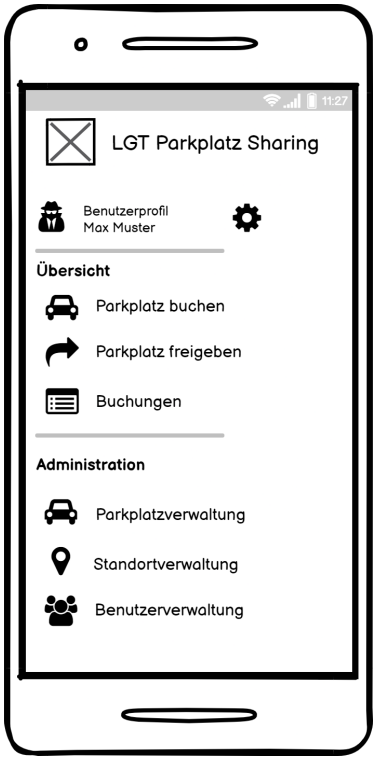


Abbildung G.4: Wireframe Navigation Menu

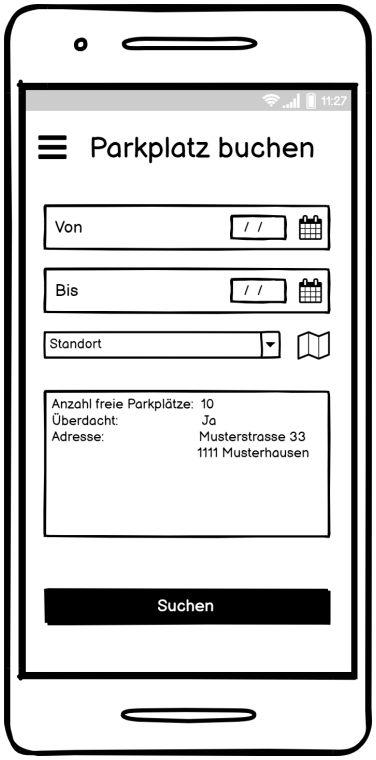


Abbildung G.5: Wireframe Übersicht Parkplatz buchen

Abbildung G.6: Wireframe Karte der Standorte

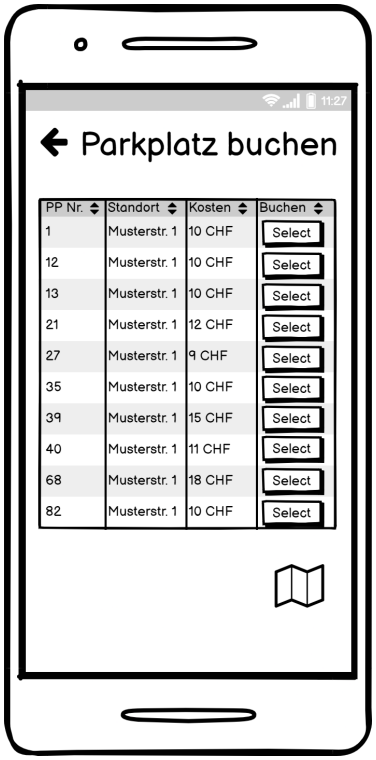


Abbildung G.7: Wireframe verfügbare Parkplätze

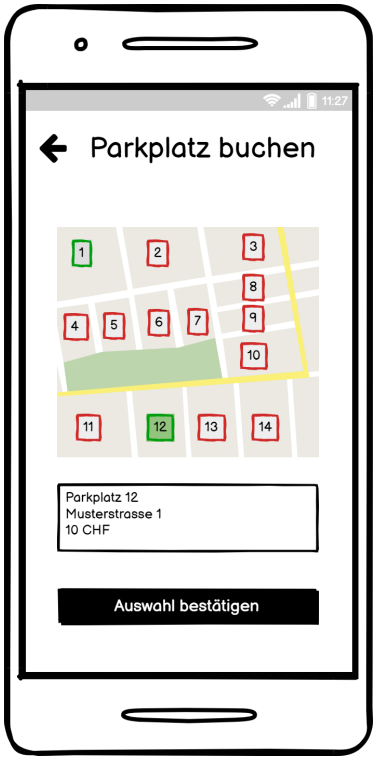


Abbildung G.8: Wireframe Karte der Parkplätze

Abbildung G.9: Wireframe Parkplatz Buchung bestätigen

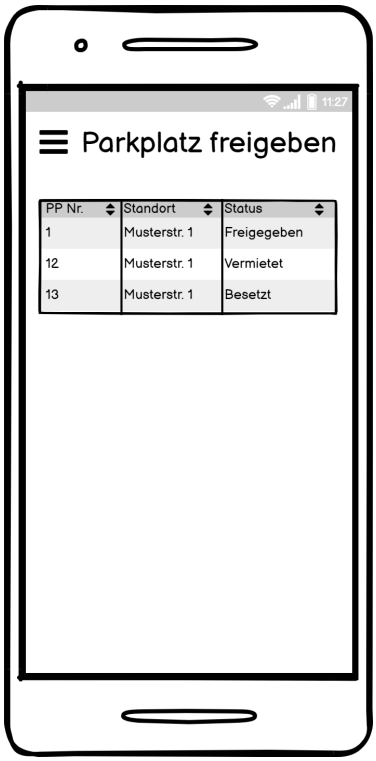


Abbildung G.10: Wireframe Übersicht Parkplatz freigeben

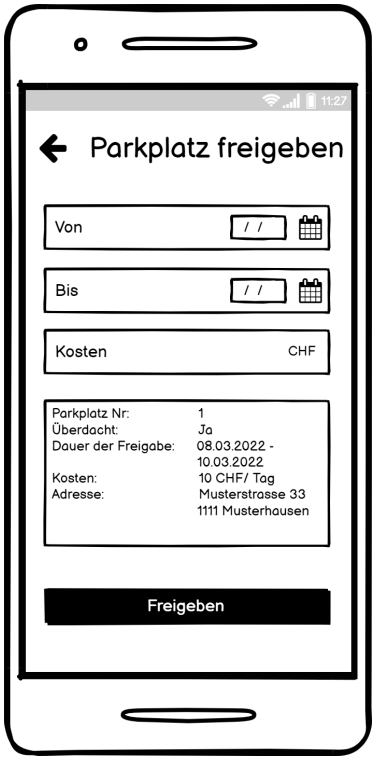


Abbildung G.11: Wireframe Parkplatz freigeben

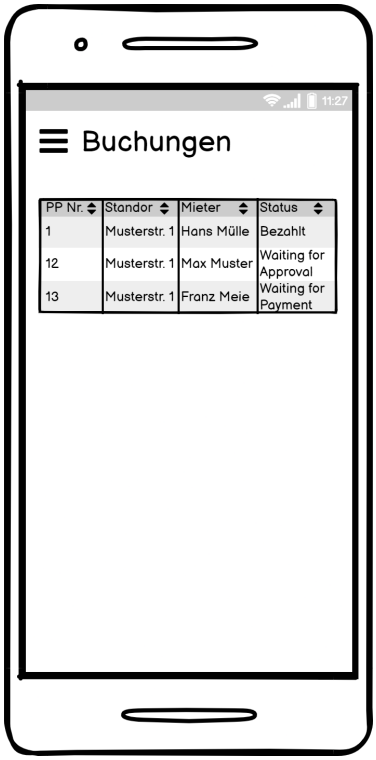


Abbildung G.12: Wireframe Buchungsübersicht

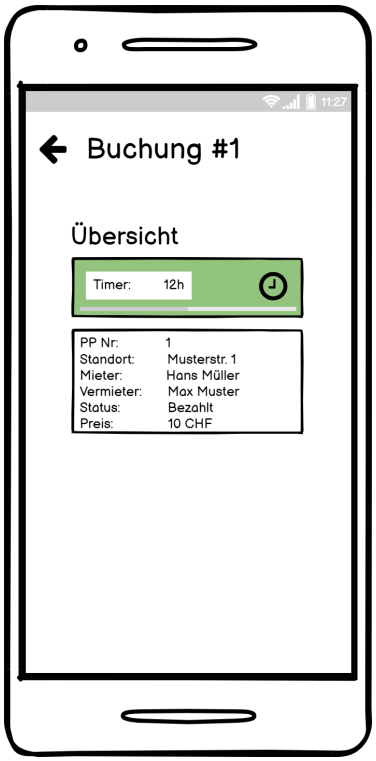


Abbildung G.13: Wireframe Buchung ansehen

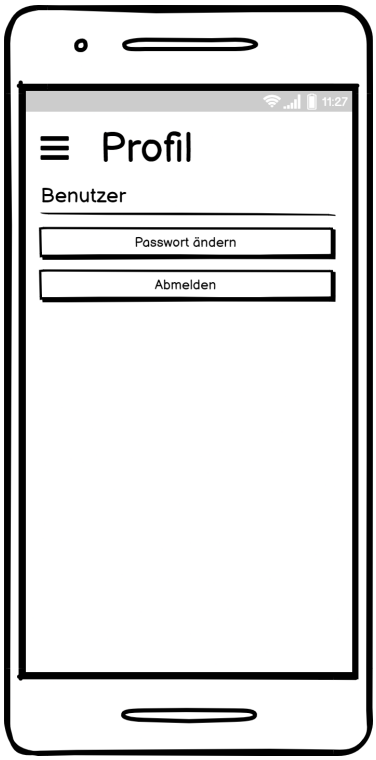


Abbildung G.14: Wireframe Profilübersicht

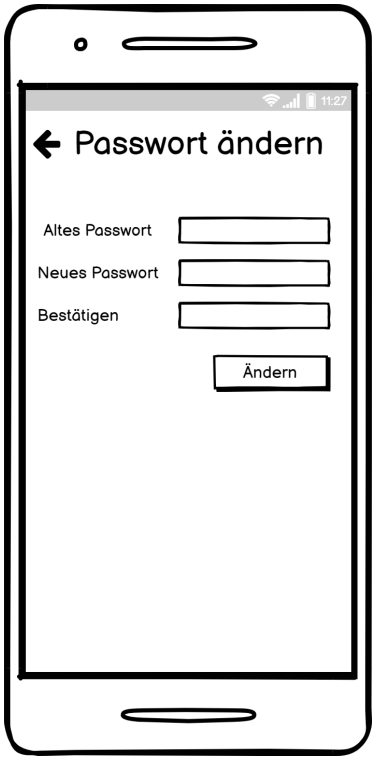


Abbildung G.15: Wireframe Passwort ändern

G.1.1.2 Admin

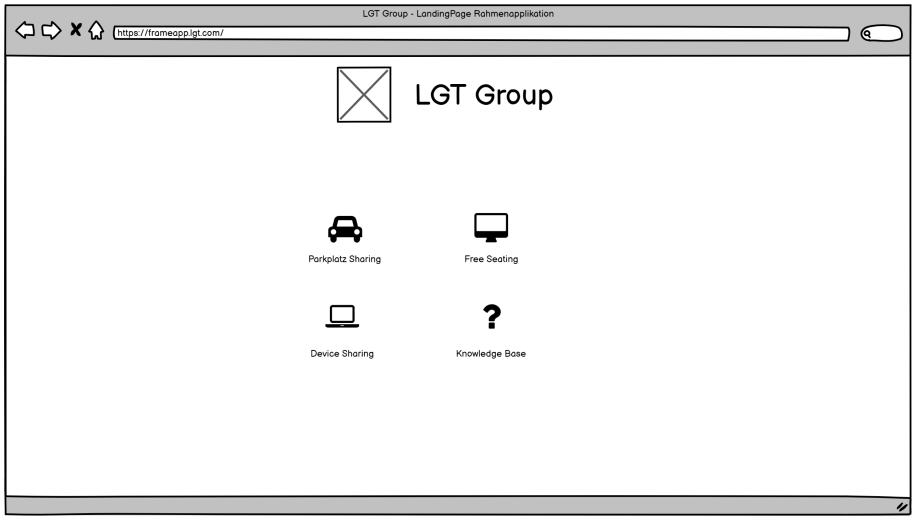


Abbildung G.16: Wireframe Ressourcen Applikation

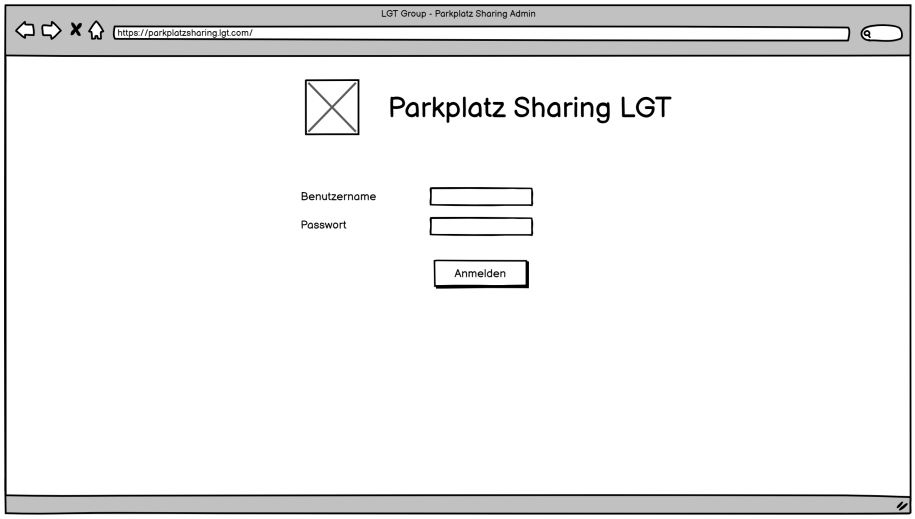


Abbildung G.17: Wireframe Login

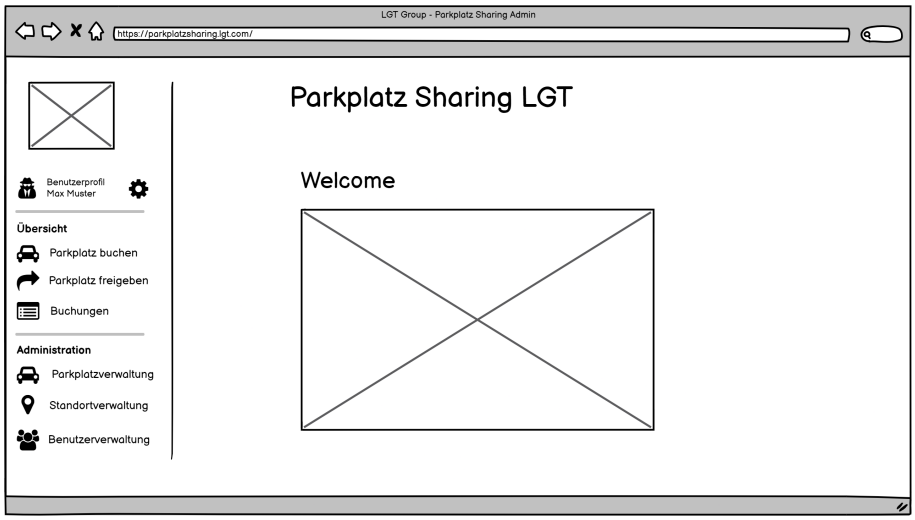


Abbildung G.18: Wireframe Landing Page

Abbildung G.19: Wireframe Parkplatzverwaltung

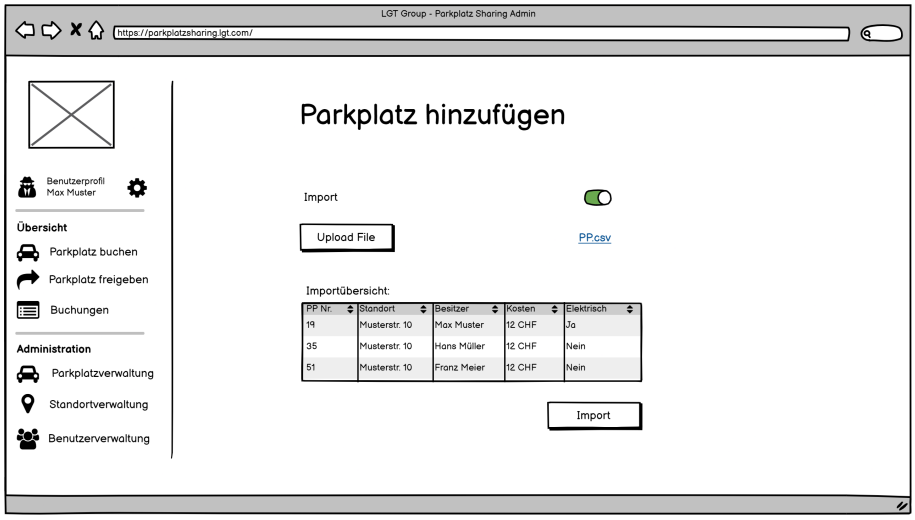


Abbildung G.20: Wireframe Parkplatz hinzufügen (Import)

Abbildung G.21: Wireframe Parkplatz hinzufügen (Manuell)

Abbildung G.22: Wireframe Standortverwaltung

Abbildung G.23: Wireframe Standort hinzufügen

Abbildung G.24: Wireframe Benutzerverwaltung

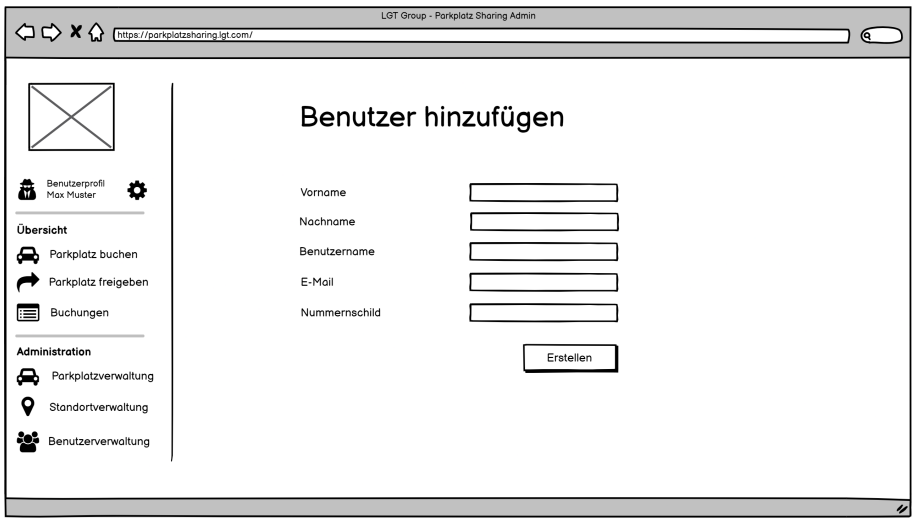


Abbildung G.25: Wireframe Benutzer hinzufügen

H Resultate Usability Test

Name: Christian Albrecht, Administrator

Nummer	Beschreibung	Erwartetes Ergebnis	Bewertung (1-5)	Notizen
UT05	Die Funktionsweise der Oberfläche ist den Benutzern ohne Einführung klar	Die Benutzer finden sich zurecht.	3	Release wurde als Freigabe der Buchung verstanden
UT06	Die Funktionsweise der Oberfläche ist ausreichend zur Nutzung der App	Die Benutzer sind zufrieden mit den bestehenden Funktionen und würden die App im Berufsalltag benutzen.	4	Karte für Parkplatz fehlt
UT07	Die App ist einfach gehalten und man kommt mit wenigen Schritten zum Ziel.	Die Benutzer müssen keine überflüssigen Eingaben tätigen.	5	
UT08	Die App vereinfacht den Arbeits-Alltag	Die Benutzer haben durch die Nutzung der App einen Mehrwert.	5	Mehrwert in Form von Geld
UT09	Fehler in der App werden korrekt dargestellt	Die Benutzer bekommen bei einem Fehler aussagekräftige Meldungen.	4	Import CSV könnte Error Handling besser sein
UT10	Benutzer, Parkplätze und Standorte können erstellt werden	Der Administrator kann einzelne oder mehrere Benutzer, Parkplätze und Standorte in einer für ihn angemessenen Variante erstellen.	5	Intuitiv gelöst
UT11	Benutzer können ihr Profilangaben selbstständig ändern	Ein Benutzer kann sein Passwort bzw. Benutzername, E-Mail, Vorname und Nachname einfach ändern.	5	
UT12	Ein Parkplatz kann für eine bestimmte Zeitdauer freigegeben werden	Der Parkplatzbesitzer kann seinen Parkplatz ohne Probleme freigeben.	5	Material Design nicht gewöhnt von iOS, Zeitraum prüfen und wenn ungültig abfangen
UT13	Ein Parkplatz kann für eine bestimmte Zeitdauer gebucht werden	Der Mitarbeiter kann einen freigegebenen Parkplatz ohne Probleme buchen.	5	
UT14	Die Buchungen können übersichtlich dargestellt und abgerufen werden	Der Benutzer kann seine Buchungen abrufen und findet die gewünschten Informationen.	5	

Verbesserungsvorschläge/Notizen:

Azure AD Anbindung von Vorteil

Homepage mit Übersicht über Buchungen oder ähnliches

Kein Boolean bei Überdacht anzeigen, sondern Text

Buchungen nur Tag anzeigen und ID entfernen

Buchung stornieren fehlt

Bei Klick auf Titel wieder auf Startseite gelangen

Rückmeldung bei einem Update

Import CSV besser erklären und Buttons ersichtlicher gestallten

Name: Reto Calonder, Parkplatzbesitzer

Nummer	Beschreibung	Erwartetes Ergebnis	Bewertung (1-5)	Notizen
UT05	Die Funktionsweise der Oberfläche ist den Benutzern ohne Einführung klar	Die Benutzer finden sich zurecht.	5	Alles gefunden ohne Hilfe
UT06	Die Funktionsweise der Oberfläche ist ausreichend zur Nutzung der App	Die Benutzer sind zufrieden mit den bestehenden Funktionen und würden die App im Berufsalltag benutzen.	3	Karte, um Parkhaus auszuwählen fehlt
UT07	Die App ist einfach gehalten und man kommt mit wenigen Schritten zum Ziel.	Die Benutzer müssen keine überflüssigen Eingaben tätigen.	5	
UT08	Die App vereinfacht den Arbeits-Alltag	Die Benutzer haben durch die Nutzung der App einen Mehrwert.	4	Vereinfacht nicht unbedingt, aber ist von Vorteil
UT09	Fehler in der App werden korrekt dargestellt	Die Benutzer bekommen bei einem Fehler aussagekräftige Meldungen.	5	Passwort ändern und Login wurde korrekt angezeigt
UT10	Benutzer, Parkplätze und Standorte können erstellt werden	Der Administrator kann einzelne oder mehrere Benutzer, Parkplätze und Standorte in einer für ihn angemessenen Variante erstellen.	-	-
UT11	Benutzer können ihr Profilangaben selbstständig ändern	Ein Benutzer kann sein Passwort bzw. Benutzername, E-Mail, Vorname und Nachname einfach ändern.	5	Passwort Wechsel mit Rückmeldung, dass es funktioniert hat
UT12	Ein Parkplatz kann für eine bestimmte Zeitdauer freigegeben werden	Der Parkplatzbesitzer kann seinen Parkplatz ohne Probleme freigeben.	5	
UT13	Ein Parkplatz kann für eine bestimmte Zeitdauer gebucht werden	Der Mitarbeiter kann einen freigegebenen Parkplatz ohne Probleme buchen.	5	
UT14	Die Buchungen können übersichtlich dargestellt und abgerufen werden	Der Benutzer kann seine Buchungen abrufen und findet die gewünschten Informationen.	4	Zu wenig Platz auf einem Handy, aber sinnvolle Attribute gewählt.

Verbesserungsvorschläge/Notizen:

Buchung erstellen

- Reservation until
- Verbesserungen von Parameter Namen
- Buchung wechselt Farbe bei Timer

LocationID mit Location Name ersetzen

Funktionsweise bei einem geteilten Parkplatz nicht umgesetzt

Name: Mathias Freund, Administrator

Nummer	Beschreibung	Erwartetes Ergebnis	Bewertung (1-5)	Notizen
UT05	Die Funktionsweise der Oberfläche ist den Benutzern ohne Einführung klar	Die Benutzer finden sich zurecht.	4	Zeitauswahl nicht intuitiv für iOS Benutzer
UT06	Die Funktionsweise der Oberfläche ist ausreichend zur Nutzung der App	Die Benutzer sind zufrieden mit den bestehenden Funktionen und würden die App im Berufsalltag benutzen.	4	
UT07	Die App ist einfach gehalten und man kommt mit wenigen Schritten zum Ziel.	Die Benutzer müssen keine überflüssigen Eingaben tätigen.	5	
UT08	Die App vereinfacht den Arbeits-Alltag	Die Benutzer haben durch die Nutzung der App einen Mehrwert.	4	Würde die App nutzen
UT09	Fehler in der App werden korrekt dargestellt	Die Benutzer bekommen bei einem Fehler aussagekräftige Meldungen.	5	Passwort Wechsel, Passwort Eingabe
UT10	Benutzer, Parkplätze und Standorte können erstellt werden	Der Administrator kann einzelne oder mehrere Benutzer, Parkplätze und Standorte in einer für ihn angemessenen Variante erstellen.	4	Angaben bei Fehler des Benutzers erstellen sind weg
UT11	Benutzer können ihr Profilangaben selbstständig ändern	Ein Benutzer kann sein Passwort bzw. Benutzername, E-Mail, Vorname und Nachname einfach ändern.	5	
UT12	Ein Parkplatz kann für eine bestimmte Zeitdauer freigegeben werden	Der Parkplatzbesitzer kann seinen Parkplatz ohne Probleme freigeben.	4	EndDate Datum nicht automatisch geändert bei Änderung des StartDate
UT13	Ein Parkplatz kann für eine bestimmte Zeitdauer gebucht werden	Der Mitarbeiter kann einen freigegebenen Parkplatz ohne Probleme buchen.	4	Bei Buchung über eine grössere Zeitdauer, Möglichkeiten einblenden die man hat
UT14	Die Buchungen können übersichtlich dargestellt und abgerufen werden	Der Benutzer kann seine Buchungen abrufen und findet die gewünschten Informationen.	4	Buchungsübersicht wird nicht alles angezeigt auf dem Bildschirm

Verbesserungsvorschläge/Notizen:

Bei Profil statt erster Buchstabe des Benutzernamen, der erste vom Vor- und Nachname anzeigen

Mehrere Nummernschilder nicht möglich

Header id beim Freigeben eines Parkplatzes zu gross

Status Open beim Freigeben eines Parkplatzes ist unklar

Location ist nicht wichtig bei der Freigabe eines Parkplatzes

Wiederholende Freigabe gewünscht, da teilweise immer am gleichen Tag HomeOffice

Mehrere Freigaben gleichzeitig nicht möglich

Die App Bar kann nur über Button geschlossen werden

Bei Parkplatz ist möglicherweise mehr möglich als nur elektrisch ja/nein, z.B. Ampere

Bei Buchung anzeigen ob überdacht und Name von Location

Bei Buchungsübersicht nur Tag und nicht auch Zeit einblenden

Bei Buchungsübersicht Sortierung nach aktiven, da vlt in Zukunft reserviert wird welche noch nicht interessant sind

Stornieren einer Buchung nicht möglich

Hilfestellung in App mit Kontaktinformationen, Copyright usw.

Homepage ist nichts aussagend, z.B. Übersicht mit der nächsten Buchung

Homepage muss zugänglich gemacht werden

Parkplatzfreigabe kann nicht abgebrochen werden

Name: Christian Keel, Administrator

Nummer	Beschreibung	Erwartetes Ergebnis	Bewertung (1-5)	Notizen
UT05	Die Funktionsweise der Oberfläche ist den Benutzern ohne Einführung klar	Die Benutzer finden sich zurecht.	5	
UT06	Die Funktionsweise der Oberfläche ist ausreichend zur Nutzung der App	Die Benutzer sind zufrieden mit den bestehenden Funktionen und würden die App im Berufsalltag benutzen.	4	Benutzer sollen sich selber registrieren können
UT07	Die App ist einfach gehalten und man kommt mit wenigen Schritten zum Ziel.	Die Benutzer müssen keine überflüssigen Eingaben tätigen.	5	
UT08	Die App vereinfacht den Arbeits-Alltag	Die Benutzer haben durch die Nutzung der App einen Mehrwert.	5	
UT09	Fehler in der App werden korrekt dargestellt	Die Benutzer bekommen bei einem Fehler aussagekräftige Meldungen.	4	0 characters in Passwort Anforderungen
UT10	Benutzer, Parkplätze und Standorte können erstellt werden	Der Administrator kann einzelne oder mehrere Benutzer, Parkplätze und Standorte in einer für ihn angemessenen Variante erstellen.	5	Sehr gut gelöst
UT11	Benutzer können ihr Profilangaben selbstständig ändern	Ein Benutzer kann sein Passwort bzw. Benutzername, E-Mail, Vorname und Nachname einfach ändern.	5	
UT12	Ein Parkplatz kann für eine bestimmte Zeitdauer freigegeben werden	Der Parkplatzbesitzer kann seinen Parkplatz ohne Probleme freigeben.	5	
UT13	Ein Parkplatz kann für eine bestimmte Zeitdauer gebucht werden	Der Mitarbeiter kann einen freigegebenen Parkplatz ohne Probleme buchen.	5	
UT14	Die Buchungen können übersichtlich dargestellt und abgerufen werden	Der Benutzer kann seine Buchungen abrufen und findet die gewünschten Informationen.	4	Zu viele Informationen in der Übersicht

Verbesserungsvorschläge/Notizen:

App Bar geht nur über Zurück Button weg und nicht mit Klick irgendwo hin

Bei Eingaben Erfolgsmeldung fehlt

Homepage ist nichtsausagend, vlt mit Buchungen oder aktive Buchung anzeigen

Authentisierung einfacher – SSO

Name: Max Pfiffner, Mitarbeiter

Nummer	Beschreibung	Erwartetes Ergebnis	Bewertung (1-5)	Notizen
UT05	Die Funktionsweise der Oberfläche ist den Benutzern ohne Einführung klar	Die Benutzer finden sich zurecht.	4	Meist intuitiv, Button zum Freigeben des Parkplatzes, da sonst statisch und man weiss vlt nicht ob man klicken kann
UT06	Die Funktionsweise der Oberfläche ist ausreichend zur Nutzung der App	Die Benutzer sind zufrieden mit den bestehenden Funktionen und würden die App im Berufsalltag benutzen.	5	Würde es nutzen
UT07	Die App ist einfach gehalten und man kommt mit wenigen Schritten zum Ziel.	Die Benutzer müssen keine überflüssigen Eingaben tätigen.	5	
UT08	Die App vereinfacht den Arbeits-Alltag	Die Benutzer haben durch die Nutzung der App einen Mehrwert.	5	
UT09	Fehler in der App werden korrekt dargestellt	Die Benutzer bekommen bei einem Fehler aussagekräftige Meldungen.	5	
UT10	Benutzer, Parkplätze und Standorte können erstellt werden	Der Administrator kann einzelne oder mehrere Benutzer, Parkplätze und Standorte in einer für ihn angemessenen Variante erstellen.	-	
UT11	Benutzer können ihr Profilangaben selbstständig ändern	Ein Benutzer kann sein Passwort bzw. Benutzername, E-Mail, Vorname und Nachname einfach ändern.	5	
UT12	Ein Parkplatz kann für eine bestimmte Zeitdauer freigegeben werden	Der Parkplatzbesitzer kann seinen Parkplatz ohne Probleme freigeben.	-	
UT13	Ein Parkplatz kann für eine bestimmte Zeitdauer gebucht werden	Der Mitarbeiter kann einen freigegebenen Parkplatz ohne Probleme buchen.	4	Zeitdauer auswählen könnte intuitiver sein, automatisch Tag bei EndDate ändern wenn StartDate gesetzt wird
UT14	Die Buchungen können übersichtlich dargestellt und abgerufen werden	Der Benutzer kann seine Buchungen abrufen und findet die gewünschten Informationen.	5	

Verbesserungsvorschläge/Notizen:

Parkplatzname fehlt bei Buchung

Rückmeldung fehlt beim Update

Name: Riccardo Somma, Mitarbeiter

Nummer	Beschreibung	Erwartetes Ergebnis	Bewertung (1-5)	Notizen
UT05	Die Funktionsweise der Oberfläche ist den Benutzern ohne Einführung klar	Die Benutzer finden sich zurecht.	4	Button besser signalisieren Tabelle wissen nicht alle das man reindrücken kann
UT06	Die Funktionsweise der Oberfläche ist ausreichend zur Nutzung der App	Die Benutzer sind zufrieden mit den bestehenden Funktionen und würden die App im Berufsalltag benutzen.	4	Karte für Parkplatz
UT07	Die App ist einfach gehalten und man kommt mit wenigen Schritten zum Ziel.	Die Benutzer müssen keine überflüssigen Eingaben tätigen.	5	
UT08	Die App vereinfacht den Arbeits-Alltag	Die Benutzer haben durch die Nutzung der App einen Mehrwert.	-	Besitzt kein Auto
UT09	Fehler in der App werden korrekt dargestellt	Die Benutzer bekommen bei einem Fehler aussagekräftige Meldungen.	5	Passwort Eingabe, Profiländerungen
UT10	Benutzer, Parkplätze und Standorte können erstellt werden	Der Administrator kann einzelne oder mehrere Benutzer, Parkplätze und Standorte in einer für ihn angemessenen Variante erstellen.	-	
UT11	Benutzer können ihr Profilangaben selbstständig ändern	Ein Benutzer kann sein Passwort bzw. Benutzername, E-Mail, Vorname und Nachname einfach ändern.	5	
UT12	Ein Parkplatz kann für eine bestimmte Zeitdauer freigegeben werden	Der Parkplatzbesitzer kann seinen Parkplatz ohne Probleme freigeben.	-	
UT13	Ein Parkplatz kann für eine bestimmte Zeitdauer gebucht werden	Der Mitarbeiter kann einen freigegebenen Parkplatz ohne Probleme buchen.	5	Mit wenigen Schritten zum Ziel
UT14	Die Buchungen können übersichtlich dargestellt und abgerufen werden	Der Benutzer kann seine Buchungen abrufen und findet die gewünschten Informationen.	4	Zu viele Informationen auf der Übersicht

Verbesserungsvorschläge/Notizen:

Homepage ist nichtaussagend, könnte besser verwendet werden

Bestätigung bei Änderung ohne Seiten Wechsel