Entwicklung eines Fahrradverleihsystems unter Verwendung der Microservice-Architektur

Abschlussarbeit

zur Erlangung des akademischen Grades:

Bachelor of Science (B.Sc.)

an der

Hochschule für Technik und Wirtschaft (HTW) Berlin
Fachbereich 4: Informatik, Kommunikation und Wirtschaft
Studiengang *Angewandte Informatik*

1. Gutachter_in: Prof. Dr. Elena Schüler
2. Gutachter_in: Prof. Dr.-Ing. Hendrik Gärtner

Eingereicht von Quang Anh Nguyen

06.10.2021

Zusammenfassung

Fahrradverleihsysteme bieten Kunden die Möglichkeit, bei der Nutzung einer mobilen Anwendung ein Fahrrad zu mieten. Nebenbei können Kunden Bonuspunkte sammeln, wenn sie defekte Fahrräder melden oder wenn die zurückgelegte Entfernung mehr als 5 km beträgt. Die vorliegende Bachelorarbeit befasst sich mit der Entwicklung des von Fahrradverleihsystemen, die aus einem Backendserver, einer mobilen Anwendung für Kunden und einer mobilen Anwendung für Fahrräder bestehen.

Abstract

Bike rental systems offer customers the option of renting a bike when using a mobile application. Along the way, customers can earn bonus points if they report defective bikes or if the distance traveled is more than 5 km. This bachelor thesis deals with the development of bicycle rental systems consisting of a backend server, a mobile application for customers and a mobile application for bicycles.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

.

1 Einleitung

1.1 Motivation

Heutzutage wird eine Microservice-Architektur immer mehr zum Standard-Architektur, die eine große Softwareanwendung in eine Reihe von lose gekoppelten Microservices zerlegt. Sie hat durch die große Übernahme durch Unternehmen wie Amazon, Netflix, SoundCloud und Spotify an Bedeutung gewonnen.

Radfahren ist nicht nur gut für die Gesundheit, sondern auch für die Umwelt. Mit dem Fahrrad können Menschen aller Altersgruppen kurze und mittellange Strecken zeit- und kostengünstig zurücklegen. In Kombination mit den öffentlichen Verkehrsmitteln kann das Fahrrad auch auf langen Wegen mit dem Auto konkurrieren. In der Zukunft kann der Radverkehr einen Teil des motorisierten Verkehrs ersetzen. Um den Radverkehr und das Radfahren weiter zu fördern sollen aktive Radfahrer belohnt werden.

Ein solches Fahrradverleihsystem mit Belohnung für Kunden kann unter Verwendung der Microservice-Architektur realisiert werden.

1.2 Zielsetzung

Hauptziel dieser Arbeit ist es, die Konzipierung und die Implementierung eines Backendsystems unter Nutzung einer Microservice-Architektur für ein öffentliches Fahrradverleihsystem zu verdeutlichen. Der Backend-Server soll auf HTTP-Anfragen reagieren und die geforderten Daten mit einer angemessenen Geschwindigkeit und Stabilität liefern. Zusätzlich werden Android-basierte Anwendungen für Kunden und Mietfahrräder entwickelt.

1.3 Aufbau der Arbeit

Kapitel 1: Einleitung
Im ersten Kapitel werden Motivation und Zielsetzung für diese Arbeit erläutert. Außerdem wird die Aktualität des gewählten Themas kurz dargelegt.

Kapitel 2: Grundlagen
In Kapitel 2 werden die grundlegenden Begriffe und Technologien erläutert. Dabei stehen die Kenntnisse über Webservices im Vordergrund.

Kapitel 3: Anforderungsanalysen
In diesem Kapitel werden die funktionale Anforderung und nicht-funktionale Anforderung definiert.

Kapitel 4: Entwurf des Backendservers
Das Kapitel erläutert die Architektur des Backendsystems. Zusätzlich werden alle verarbeiteten Anfragen beschrieben..

Kapitel 5: Implementierung des Backendservers
Dieses Kapitel beschäftigt sich mit der Umsetzung sowie der vorher skizzierten Architektur. Auch der Aufbau aller Microservices wird in dem Kapitel dargestellt.

Kapitel 6: Teste des Backendservers
In diesem Kapitel werden die verwendeten Testverfahren für den Server berücksichtigt, die zur Sicherung der Software-Qualität genutzt werden.

Kapitel 7: Installation des Backendservers
Dieses Kapitel beschäftigt sich mit der Installation der Backend-Anwendung auf dem Server.

Kapitel 8: Entwurf und Implementierung der mobilen Anwendung für Kunden
In Kapitel 7 geht es um den Entwurf und die Implementierung der mobilen Anwendung für Kunden.

Kapitel 9: Entwurf und Implementierung der mobilen Anwendung für Leihfahrräder
Dieses Kapitel beschäftigt sich mit dem Entwurf und der Implementierung der mobilen Anwendung für Leihfahrräder.

Kapitel 10: Fazit
Im letzten Kapitel erfolgt ein Fazit, das einen Überblick über die Ergebnisse der Arbeit geben soll und ein kurzer Ausblick für zukünftige Erweiterungen.

2 Grundlagen

2.1 Client Server Modell

Heutzutage ist das Client-Server-System sehr beliebt. So wird es für verschiedene Anwendungen genutzt. Einige der standardisierten Protokolle, die Client und Server zur Kommunikation verwenden, sind unter anderem: File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP) und das Hypertext-Übertragungsprotokoll (HTTP).

Das Client-Server-Modell bezeichnet die Beziehung, in der zwei Programme zueinanderstehen. Der Client stellt eine Anfrage an den Server, eine gegebene Aufgabe zu erledigen. Der Server erledigt die Aufgabe und liefert das Ergebnis beim Client ab. Ein Kommunikationsdienst verbindet Client und Server miteinander. Ein Client-Server-System besteht also aus einem oder mehreren Clients (Auftraggebern), einem Server (Auftragnehmer) und einem Kommunikationsdienst (Netzwerk, Vermittler). Client und Server sind Programme, die auch auf gleichem Rechner laufen können, aber müssen nicht unbedingt über ein Rechnernetz verbunden sein. Natürlich werden Client und Server in der Praxis auf unterschiedlichen Rechnern verteilt, um die Vorteile eines verteilten Systems zu ziehen. [2, S. 7]

Die Interaktion zwischen Client und Server kann asynchron oder synchron erfolgen. Im synchronen Fall kann der Client andere Aktivitäten erst dann durchführen, wenn er nach Absenden der Anfrage an den Server eine Antwort des Servers bekommt. Bei der asynchronen Kommunikation arbeitet der Client sofort wieder, nachdem die an den Server gesendet wird. Beim Ankommen der Antwort werden dann bestimmte Ereignisbehandlungsroutinen beim Client aufgerufen. [2, S. 8]

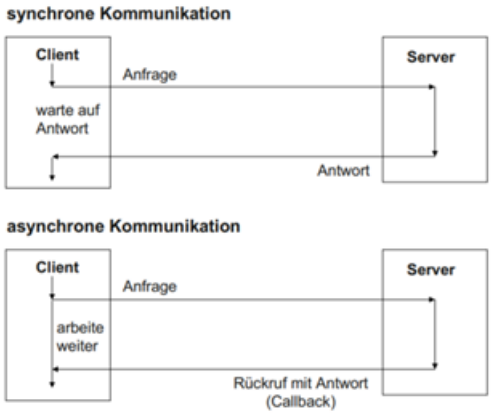

Abbildung 2.1: *Interaktionsarten; Bildquelle [1]*

2.2 REST

REST steht für Representational State Transfer und ist ein Architekturstil für die Entwicklung verteilter Netzwerkanwendungen. Roy Fielding prägte den Begriff REST in seiner Dissertation und schlug die sechs folgenden Einschränkungen bzw. Grundsätze als Grundlage vor.

- Client-Server-Concerns: Clients und Server sollten getrennt werden. Diese Trennung ermöglicht die voneinander unabhängigen Weiterentwicklungen von Client- und Server-Komponenten und die Skalierung des Systems. [20, S. 1]

- Zustandslos: Die Kommunikation zwischen Client und Server sollte zustandslos sein. Der Server muss sich nicht an den Status des Clients erinnern. Stattdessen müssen die Clients alle notwendigen Informationen in die Anfrage mitschicken, damit der Server sie verstehen und verarbeiten kann. [20, S. 1]

- Mehrschichtiges System: Zwischen Client und Server können mehrere hierarchische Schichten wie Gateways, Firewalls und Proxys bestehen. Schichten können hinzugefügt, geändert, neu geordnet oder entfernt werden, um die Skalierbarkeit zu verbessern. [20, S. 1]

- Cache: Antworten vom Server müssen als cachefähig oder nicht cachefähig deklariert werden. Dies würde dem Client oder seinen zwischengeschalteten Komponenten ermöglichen, Antworten zwischenzuspeichern und sie für spätere Anfragen wiederzuverwenden. Das verringert die Belastung des Servers und trägt zur Verbesserung der Leistung. [20, S. 1]

- Einheitliche Schnittstelle: Alle Interaktionen zwischen Client, Server und zwischengeschalteten Komponenten beruhen auf der Einheitlichkeit ihrer Schnittstellen. Dies vereinfacht die Gesamtarchitektur, da sich die Komponenten unabhängig voneinander weiterentwickeln können, solange sie den vereinbarten Vertrag implementieren. [20, S. 2]

- Code auf Anfrage: Clients können ihre Funktionalität durch das Herunterladen und Code bei Bedarf ausführen. Beispiele hierfür sind JavaScript-Skripte, Java-Applets und Silverlight. [20, S. 2]

2.3 HTTP

Jede HTTP-Anfrage und -Antwort besteht aus den folgenden Komponenten:

- Header: Informationen über Kodierung, Länge des Bodys, Herkunft, Inhaltstyp usw.
- Body: Inhalt, normalerweise Parameter oder Daten, die dem Server übergeben oder an einen Client zurückgeschickt werden.

Die wichtigsten HTTP-Methoden sind:

- GET: Mit GET wird die Repräsentation einer Ressource abgefragt. Die Ressource auf dem Server darf nicht verändert werden.
- POST: Mit POST kann eine neue Ressource mit einem vom Server bestimmten URI erstellt werden und auch eine beliebige Verarbeitung auf dem Server angestoßen werden.
- PUT: PUT wird verwendet, um eine Ressource, deren URI bereits bekannt ist, zu ändern oder um eine neue Ressource zu erstellen.
- DELETE: Mit DELETE wird eine Ressource gelöscht..

HTTP ermöglicht die Nutzung von Status Codes in Responses, welche von einer großen Menge von HTTP-Clients unterstützt werden.

- Staus Codes 1xx informiert, dass die Bearbeitung der Anfrage noch andauert.
- Status Codes 2xx informiert, dass die Anfrage erfolgreich war, die Antwort verwertet werden kann.
- Status Codes 3xx informiert, dass die Anfrage umgeleitet wurde.
- Status Codes 4xx sind Fehler, die im Zusammenhang mit einer fehlerhaften Nutzung des Requests stehen.

Code	Nachricht	Bedeutung
400	Bad Request	Die Anfrage-Nachricht war fehlerhaft aufgebaut.
401	Unauthorized	Die Anfrage kann nicht ohne gültige Autorisierung durchgeführt werden.
402	Payment Required	Die Bezahlung benötigt.
403	Forbidden	Die Anfrage wurde mangels Berechtigung des Clients nicht durchgeführt.
404	Not Found	Die angeforderte Ressource wurde nicht gefunden.
405	Method Not Allowed	Die Anfrage darf nur mit anderen HTTP-Methoden (zum Beispiel GET statt POST) gestellt werden.
406	Not Acceptable	Die angeforderte Ressource steht nicht in der gewünschten Form zur Verfügung.

Abbildung 2.2: *Client Error; Quelle [12]*

- Status Codes 5xx sind schwerwiegende Fehler, die serverseitig aufgetreten sind.

2.4 Spring

Das Spring-Framework ist eine umfangreiche Sammlung von Bibliotheken und Tools zur Vereinfachung der Softwareentwicklung, nämlich Dependency Injection, Datenzugriff, Validierung, Internationalisierung und aspektorientierte Programmierung. Es ist eine beliebte Wahl für Java-Projekte und funktioniert auch mit anderen Java Virtual Machine basierten Sprachen wie Kotlin und Groovy. Der Grund für die Beliebtheit von

Spring liegt darin, dass es viel Zeit spart, weil es integrierte Implementierungen für viele Aspekte der Softwareentwicklung bietet. [11, S. 8]

Die anderen Frameworks aus Spring sind:

- Spring Data vereinfacht den Datenzugriff für relationale und NoSQL-Datenbanken.
- Spring Batch bietet eine leistungsstarke Verarbeitung für große Mengen von Datensätzen.
- Spring Security ist ein Sicherheits-Framework, das Sicherheitsfunktionen für Anwendungen abstrahiert.
- Spring Cloud bietet Entwicklern Tools, mit denen sie schnell einige der gängigen Muster in verteilten Systemen erstellen.
- Spring Integration ist eine Implementierung von Enterprise Integration Muster. Sie erleichtert die Integration mit anderen Unternehmensanwendungen mithilfe von leichtgewichtigem Messaging und deklarativen Adaptern.

2.5 Spring Boot

Spring Boot ist ein Framework, das Spring nutzt, um schnell eigenständige Anwendungen in Java-basierten Sprachen zu erstellen. Es hat sich zu einem beliebten Werkzeug für die Erstellung von Microservices entwickelt. Trotz vieler Bemühungen zur Vereinfachung der Konfiguration von Spring müssen Entwickler immer noch Zeit aufwenden, um alle nötige Einstellung einzurichten. Und manchmal benötigen Entwickler dieselbe. Der Vorteil von Spring Boot ist, dass dieser Prozess größtenteils entfällt, da es Standardkonfigurationen und -tools bereitstellt, die automatisch für die Entwicklung eingerichtet werden. Der größte Nachteil besteht darin, dass Entwickler möglicherweise beim Verlassen der Standardeinstellung die Kontrolle und das Bewusstsein für das Geschehen verlieren. [11, S. 10]

Außerdem bietet Spring Boot nicht nur vordefinierte Starterpakete für Spring-Module, sondern kann auch mit einigen Bibliotheken und Tools von Drittanbietern zusammen fungieren. Ein Beispiel hierfür ist das spring-boot-starter-web Paket, das bei der Erstellung einer eigenständigen Webanwendung hilft. Es gruppiert die Spring Core Web-Bibliotheken mit Jackson (JSON-Verarbeitung), Validierung, Protokollierung, Autokonfiguration und sogar einem eingebetteten Tomcat-Server, unter anderem Werkzeugen. Neben den Starterpaketen spielt auch die Autokonfiguration eine wichtige Rolle in Spring Boot. Diese Funktion macht das Hinzufügen von Funktionen zur Anwendung einfach. Die Autokonfigurationsklassen scannen den Klassenpfad, die Eigenschaften, sowie Komponenten und laden auf dieser Grundlage einige zusätzliche Beans und Verhaltensweisen. Um die verschiedenen Konfigurationsoptionen für Spring Boot Anwendungen zu verwalten, führt das Framework Profile ein. Diese Profile können verwendet werden, um unterschiedliche Werte für den Host festzulegen, mit dem eine Verbindung hergestellt werden soll. Zusätzlich kann ein anderes Profil für Tests festgelegt werden. [11, S. 11]

2.6 NoSQL-Datenbank

NoSQL-Datenbanken sind quelloffene, nicht-relationale, verteilte Datenbanken. Die Unternehmen ermöglichen, riesige Datenmengen zu analysieren. [4, S. 2]

Hier sind die Vorteile der Verwendung von NoSQL-Datenbank:

- Skalierbarkeit: Aufgrund der verteilten Natur können mehrere Server und Rechenzentren über redundante Daten verfügen.[14, S. 33]
- Hohe Leistung: MongoDB ist sehr effektiv zum Speichern und Abrufen von Daten, was teilweise auf die Abwesenheit von Beziehungen zwischen Elementen und Sammlungen in der Datenbank zurückzuführen ist. [14, S. 33]
- Flexibilität: Ein Key-Value-Speicher ist ideal für Prototypen, weil die Entwickler das Schema nicht kennen müssen, und es besteht keine Notwendigkeit für feste Datenmodelle oder komplexe Migrationen. [14, S. 33]

2.7 Microservices

Microservices sind eine Technik der Softwareentwicklung - eine Variante der serviceorientierten Architektur (SOA), die eine Anwendung als eine Sammlung lose gekoppelten Diensten strukturiert. In einer Microservice-Architektur sind die Dienste feinkörnig und die Protokolle sind leichtgewichtig. Der Vorteil der Zerlegung einer Anwendung in verschiedene kleinere Dienste besteht darin, dass sie die Modularität verbessert. Dadurch ist die Anwendung leichter zu verstehen, zu entwickeln und zu testen und wird widerstandsfähiger gegen die Erosion der Architektur. Die Microservice-Architektur parallelisiert die Entwicklung, indem sie kleinen autonomen Teams ermöglicht, ihre jeweiligen Dienste unabhängig voneinander zu entwickeln, bereitzustellen und zu skalieren. Außerdem kann sich die Architektur eines einzelnen Dienstes durch kontinuierliche Refaktorisierung herausbilden. [5, S. 34]

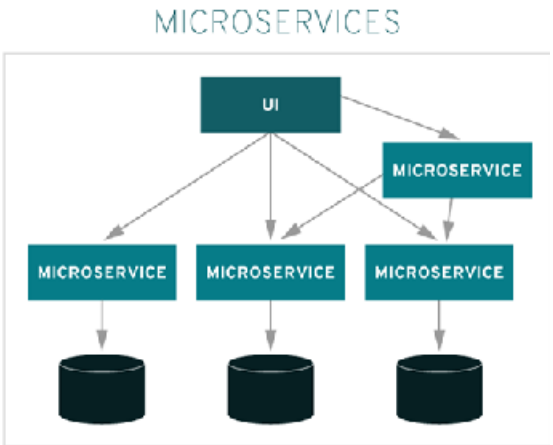

Abbildung 2.3: *Microservice-Architektur; Bildquelle [1]*

2.8 Volley Android

Volley ist eine HTTP-Bibliothek, die die Vernetzung von Android-Anwendungen einfacher und vor allem schneller macht. Volley ist auf GitHub verfügbar.

Volley bietet die folgenden Vorteile [7]:

- Automatische Planung von Netzwerkanfragen.
- Mehrere gleichzeitige Netzwerkverbindungen.
- Transparentes Festplatten- und Speicherantwort-Caching mit Standard-HTTP-Cache-Kohärenz.
- Unterstützung für die Priorisierung von Anfragen.
- API für die Stornierung von Anfragen. Sie können eine einzelne Anforderung abbrechen, und Blöcke oder Bereiche von Anforderungen zum Abbrechen festlegen.
- Leichte Anpassbarkeit, z. B. für Wiederholungen und Backoff.
- Starke Ordnung, die es einfach macht, ihre Benutzeroberfläche mit asynchron aus dem Netzwerk abgerufenen Daten korrekt zu füllen.
- Debugging- und Verfolgungswerkzeuge.

Volley eignet sich hervorragend für RPC-ähnliche Operationen, die zum Auffüllen einer Benutzeroberfläche verwendet werden, z. B. das Abrufen einer Seite mit Suchergebnissen als strukturierte Daten. Volley lässt sich problemlos in jedes Protokoll integrieren und bietet standardmäßig Unterstützung für Raw Strings, Bilder und JSON. Mithilfe der integrierten Unterstützung für die benötigten Funktionen befreit Volley die Entwickler vom Schreiben von Standardcode und ermöglicht, sich auf die Logik zu konzentrieren, die für die Anwendung spezifisch ist. [7]

Allerdings ist Volley nicht für große Download- oder Streaming-Vorgänge geeignet, da Volley alle Antworten während des Parsens im Speicher hält. Für große Download-Vorgänge sollte eine Alternative wie Download-Manager verwendet werden. Die Kernbibliothek von Volley wird auf GitHub entwickelt und enthält die Hauptpipeline für den Request Dispatch sowie eine Reihe von allgemein anwendbaren Dienstprogrammen, die in der Volley-Toolbox verfügbar sind. [7]

2.8.1 Senden einer Anfrage

Eine Anfrage wird zunächst erstellt und dann in der RequestQueue mit der add() Funktion hinzugefügt. Nach dem Hinzufügen läuft die Anfrage in die Pipeline durch. Danach wird sie bearbeitet und ihre Rohantwort wird geparst und zugestellt. [7]

Wenn die add() Funktion aufgerufen wird, führt Volley einen Cache-Verarbeitungsthread und einen Pool von Netzwerk-Dispatch-Threads aus. Wenn eine Anfrage zur Warteschlange hinzufügen wird, wird sie vom Cache-Thread abgeholt und bewertet. Wenn die Anfrage vom Cache bearbeitet werden kann, wird die im Cache gespeicherte Antwort im Cache-Thread geparst und die geparste Antwort wird im Hauptthread zugestellt. Wenn die Anfrage nicht aus dem Cache bedient werden kann, wird sie in die Netzwerkwarteschlange gestellt. Der erste verfügbare Netzwerk-Thread nimmt die

Anfrage aus der Warteschlange, führt die HTTP-Transaktion durch, parst die Antwort im Worker-Thread, schreibt die Antwort in den Cache und sendet die geparste Antwort zurück an den Haupt-Thread zur Zustellung. [7]

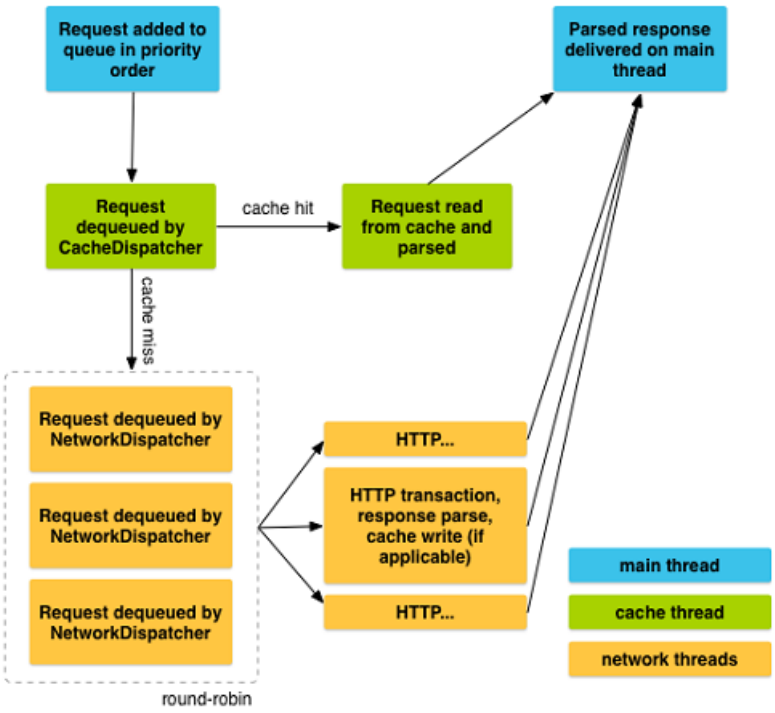

Abbildung 2.4: *Leben einer Anfrage; Bildquelle [9]*

2.8.2 Abbrechen einer Anfrage

Zum Abbrechen einer Anfrage wird die cancel-Funktion für das Request-Objekt aufgerufen. Nach dem Abbrechen garantiert Volley, dass der Response-Handler niemals aufgerufen wird. Um dieses Verhalten zu nutzen, müssen normalerweise alle laufenden Anfragen verfolgt werden, damit sie zum richtigen Zeitpunkt abgerechnet werden können. Zusätzlich gibt es für jede Anfrage ein Tag-Objekt, mit dem ein Bereich von zu stornierenden Anfragen angegeben wird. Zum Beispiel können alle Anfragen mit der Aktivität versehen werden, für die sie gemacht werden, und die requestQueue.cancelAll(this)- Funktion von der onStop-Funktion aus aufrufen. [7]

2.9 GPS-Daten

Das Global Positioning System (GPS) ist ein satellitengestütztes Funknavigationssystem, das 1973 vom US-Verteidigungsministerium eingeführt wurde und heute von der United States Space Force betrieben wird. GPS steht jedem kostenlos zur Verfügung, der ein Gerät besitzt, das Übertragungen von den zahlreichen GPS-Satelliten in der mittleren Erdumlaufbahn empfangen kann. [16]

Preisgünstige GPS-Geräte für den Endverbraucher haben oft eine Genauigkeit von etwa fünf Metern, so dass sie für die Bestimmung der Entfernung zwischen Objekten in unmittelbarer Nähe nicht wirklich nützlich sind. GPS sendet Daten über Funkwellen, die von einem GPS-Empfangsgerät aufgefangen werden, das die Satellitendaten in ein maschinenlesbares binäres Format umwandelt. Im Allgemeinen beschreiben GPS-Daten den Standort eines Geräts in Form von Längen- und Breitengrad, Höhe und Empfangszeit der Übertragung. [16]

3 Anforderungsanalysen

3.1 Funktionale Anforderungen

Das Gesamtsystem soll über folgende Funktionalitäten verfügen.

3.1.1 Mobile Anwendung für Kunden

Zuerst kann der Benutzer ein neues Konto erstellen oder sich über seinem E-Mail und sein Password anmelden. Der Benutzer hat auch die Möglichkeit, sich abzumelden. Nach erfolgreicher Anmeldung sollen dem Benutzer alle Funktionen der Anwendung gewährt werden. Der Kunde kann seinen aktuellen Standort finden, um die nächsten verfügbaren Leihfahrräder zu suchen und den Weg zu diesen zu navigieren. Die Hauptfunktion ist die Vermietung vom Fahrrad, indem der Kunde die Fahrradnummer eingibt und die PIN für die Entschlüsselung des Fahrrades bekommt. Außerdem können alle Reisen in Vergangenheit angeschaut werden. Beim Schadensfall hat der Kunde auch die Möglichkeit, die Fehler zu melden.

3.1.2 Mobile Anwendung für Leihfahrräder

Zur Sicherheit muss das Fahrrad nach einer festgelegten Zeit seinen letzten Standort an den Server schicken. Die zentrale Aufgabe der Anwendung für Leihfahrräder ist das Senden des von der Kunde eingegeben PIN-Codes. Bei der positiven Antwort vom Server wird das Schloss entschlüsselt. Während der Fahrt rechnet die Anwendung die zurückgelegte Strecke. Zum Ende der Fahrt wird der Server dann informiert.

3.1.3 Backendserver

Der Server muss die HTTP-Anfrage der Anwendung für Kunden und Leihfahrräder genau antworten sowie bei Fehlern den Fehlercode zurückmelden. Die notwendigen Dienste vom Server sind Authentifizierung, Verwaltung des Bonuskontos, Verleihung des Fahrrades, Fehlermeldung und Lokalisierung des Leihfahrrades. Für jeden Dienst werden die entsprechenden Informationen in der Datenbank gespeichert und bei Bedarf aufgerufen.

3.2 Nicht-funktionale Anforderungen

Sicherheit
Sensible Daten sollen in der zu entwickelnden Datenbank gespeichert werden. Bei der Entwicklung der Kommunikation sollte keine sensiblen Daten wie Kundendaten unverschlüsselt versendet werden. Der Zugriff auf sensible Daten und Dienste sollte nicht

autorisierten Benutzern verweigert werden. Kundenpasswörter sollten verschlüsselt in der Datenbank gespeichert werden.

Benutzbarkeit
Unter Benutzbarkeit ist die Benutzbarkeit der Backend API in Bezug auf die mobilen Anwendungen gemeint. Sollten Fehler während einer Anfrage unterlaufen, so soll die mobile Anwendung eine Fehlermeldung erhalten.

Performance
Um dem Benutzer das Gefühl zu vermitteln, dass das System sofort auf seine Aktionen reagiert, soll die Bearbeitungsdauer einer Anfrage möglichst schnell sein.

Wartbarkeit und Erweiterbarkeit
Die Funktionalitäten des zu entwickelnden Backendsystems sollen modular entwickelt werden. Es soll möglich sein, das Backendsystem zur Erweiterung von weiteren Funktionalitäten, ohne die bereits bestehenden Module zu beeinflussen.

4 Entwurf des Backendservers

In Abbildung 4.1 wird die Interaktion des Backendservers mit der Umgebung dargestellt.

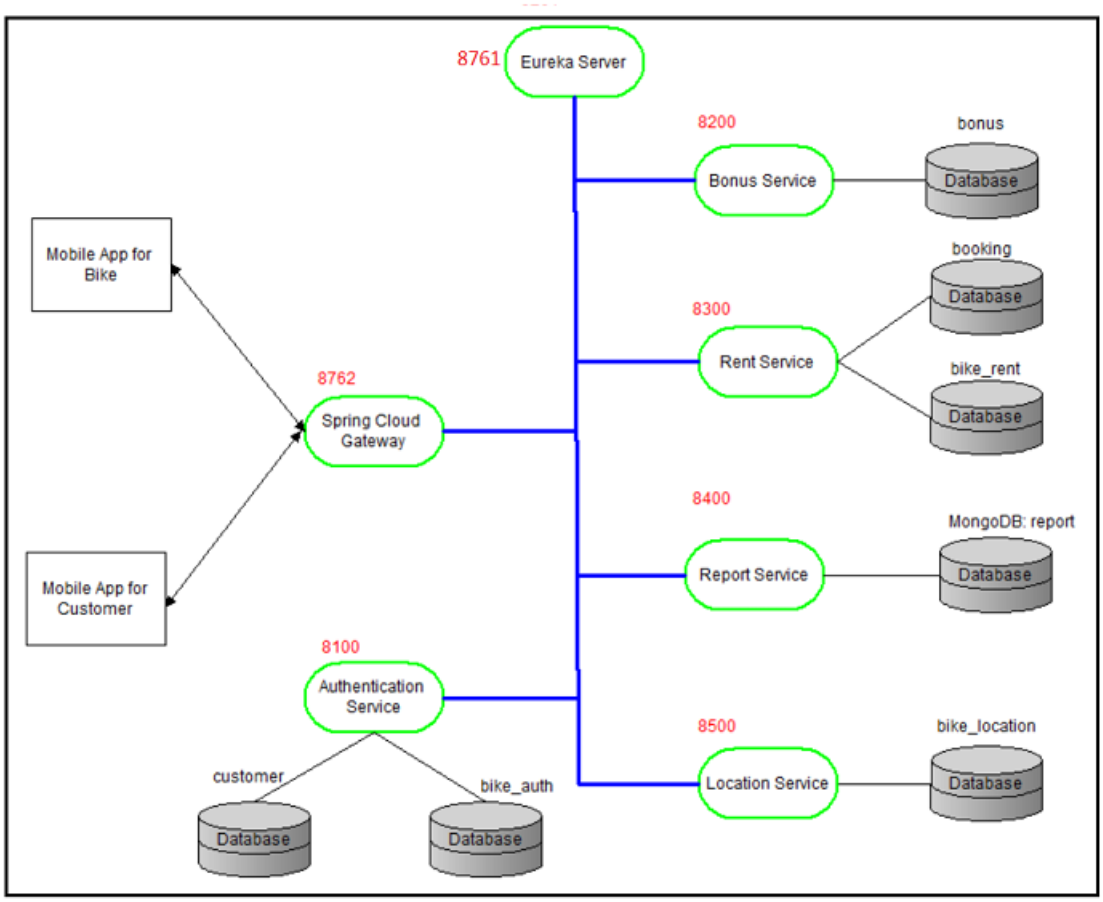

Abbildung 4.1: *Backendsystem*

Der Backendserver wird mit der Microservice-Architektur aufgebaut, indem jeder Dienst in einem Microservice zugeordnet wird. Jeder Microservice (Dienst) hat seine eigene Aufgabe zu erledigen, die unabhängig von anderen Microservices ist, und verwaltet deshalb ihre zugehörige Datenbank. Neben den fünft Microservices besteht der Backendserver aus einem Service-Register und einem zentralen Gateway, das alle REST-Anfragen der mobilen Anwendungen für Leihfahrräder und Kunden empfängt und beantwortet.

4.1 Authentifizierung- und Autorisierungsdienst

Der Dienst muss den Authentifizierungsprozess durchführen. Wenn der Kunde die genauen Logindaten übergibt oder ein neues Konto anlegt, erstellt der Dienst einen Token, der im Zwischenspeicher gespeichert wird, und schickt ihn zurück. Bei jeder HTTP -Anfrage muss ein Token mitgeliefert und bei diesem Dienst überprüft werden.

In Tabelle 4.1 enthält alle Anfragen, die für die Authentifizierung und die Autorisierung erforderlich sind.

Nr.	Pfad	Methode	Funktion
1	/auth/check/{token}	GET	-überprüft die Gültigkeit des übermittelten Tokens.
2	/auth/bikelogin	POST	- übermittelt Logindaten für Leihfahrrad. - erstellt den Token.
3	/auth/login	POST	- übermittelt Logindaten für Kunde. - erstellt den Token.
4	/auth/register	POST	- übermittelt Logindaten für Kunde. - erstellt ein neues Konto für Kunde.

Tabelle 4.1: *Auth-Service*

Die customer Tabelle,die in der MySQL-Datenbank erstellt wird, beinhaltet alle notwendigen Zugangsdaten von Kunden, wie der Name, das Passwort und die E-Mail-Adresse. Zur Identifizierung wird eine Kundennummer automatisch vergeben. Das Passwort, das in der Tabelle gespeichert ist, soll ein gehashtes Passwort sein.

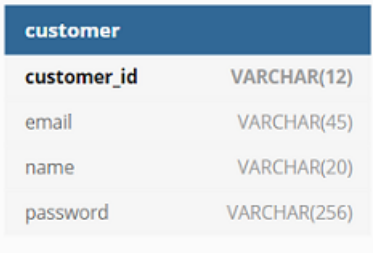

Abbildung 4.2: *customer Tabelle*

Die bike-auth Tabelle,die in der MySQL-Datenbank erstellt wird, enthält die Fahrradnummer und das Passwort, was notwendig für den Techniker ist, der das Anmeldeverfahren für das Leihfahrrad durchgeführt.

Abbildung 4.3: *bike-auth Tabelle*

4.2 Bonuskonto-Service

Jeder registrierte Kunde besitzt ein eigenes Bonuskonto. Das Bonuskonto wird zusammen mit dem Kundenkonto angelegt. Bei einer Fehlermeldung ohne Bild des Leihfahrrades bekommt der Kunde 5 Punkte und bei einer Fehlermeldung mit Bild bekommt er 5 weitere Punkte. Bei jeder Fahrt, die mehr als 5km ist, bekommt der Kunde für jeden weiteren Kilometer 10 Punkte. Die letzte Aufgabe von diesem Dienst ist, Kunden über den aktuellen Stand des Bonuskontos zu informieren.

In Tabelle 4.2 wurden alle Anfragen berücksichtigt, die für die Bearbeitung vom Bonuskonto der Kunden notwendig sind.

Nr.	Pfad	Methode	Funktion
1	/bonus/{customerId}	GET	-übermittelt die Kundennummer. -ruft die Daten eines Bonuskontos einer bestimmten Kunde auf.
2	/bonus/plus/{customerId}/{distance}	PUT	-übermittelt die Kundennummer und die Distanz von einer neuen Fahrt. -addiere die Punkt für die Fahrt. -ruft die neuen Daten eines Bonuskontos der Kunde auf.
3	/bonus/create	POST	-übermittelt die Kundennummer. -erstellt ein neues Bonuskonto und gibt dieses zurück.
4	/bonus/add/{customerId}/{point}	PUT	-übermittelt die Kundennummer und die Punkte. -addiere die Punkt für die Fehlermeldung. -ruft die neuen Daten eines Bonuskontos der Kunde auf.

Tabelle 4.2: *Bonus-Service*

Die bonus Tabelle, die in der MySQL-Datenbank erstellt ist, umfasst die Kundennummer und die Anzahl von Bonuspunkten.

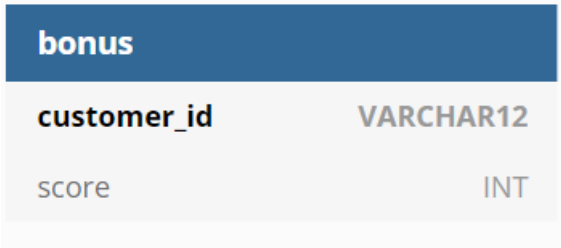

Abbildung 4.4: *bonus Tabelle*

4.3 Verleih-Service

Dieser Service bietet Kunden die Möglichkeit, anhand seiner aktuellen Positionen die verfügbaren Leihfahrräder zu finden. Der Service erstellt die neue Buchung und aktualisiert sie zum Fahrende. Bei der Erstellung der Buchung wird ein vierstelliger Code erzeugt und an Kunden gesendet.

In Tabelle 4.3 wurden alle Anfragen dargestellt, die für die Vermietung des Leihfahrrades notwendig sind.

Nr.	Pfad	Methode	Funktion
1	/rent/locatebike	POST	- übermittelt die GPS-Daten eines Kunden. - ruft die Sammlung von GPS-Daten der Leihfahrräder in der Nähe auf.
2	/rent/create	POST	- übermittelt die Kundennummer und die Fahrradnummer. - erstellt eine neue Buchung und gibt den PIN-Code zurück.
3	/rent/pincheck	POST	- übermittelt die Kundennummer, die Fahrradnummer und den PIN-Code. - überprüft die Richtigkeit des PIN-Codes und gibt das Ergebnis zurück.
4	/rent/end	PUT	- übermittelt die Daten einer Buchung und die Fahrradnummer. - ändert die Fahrt zum Ende und gibt die vollständigen Daten von der Buchung zurück.
5	/rent/updatebikelocation	PUT	- übermittelt die Fahrradnummer und die GPS-Daten eines Leihfahrrades. - ändert den aktuellen Standort des Leihfahrrades in der Datenbank und gibt das Ergebnis zurück.
6	/rent/history/{customerId}	GET	- übermittelt die Kundennummer. - ruft alle Buchungen von Kunden auf.
7	/rent/now/{customerId}	GET	- übermittelt die Kundennummer. - ruft die aktuelle Buchung von Kunden auf.

Tabelle 4.3: *Rent-Service*

Die bike-rent Tabelle, die in der MySQL-Datenbank erstellt ist, umfasst die Fahrradnummer, den aktuellen Pin-Code, die aktuellen GPS-Daten des Leihfahrrades und den Status. Der mögliche Status des Leihfahrrades kann „reserviert“, „verfügbar“ oder „beschädigt“ sein.

Abbildung 4.5: *bike-rent Tabelle*

Die booking Tabelle, die in der MySQL-Datenbank erstellt ist, umfasst die Buchungsnummer, die Kundennummer, die Fahrradnummer, den Anfangszeitpunkt, den Endzeitpunkt, die Reiseentfernung und den Status der Buchung. Der Status der Buchung kann „reseviert“,„läuft“ oder „abgeschlossen“ sein.

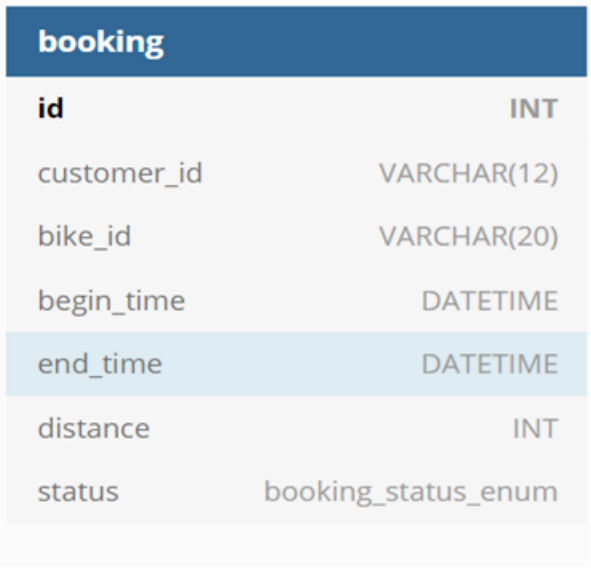

Abbildung 4.6: *booking Tabelle*

4.4 Fehlermeldung-Service

Der Service bietet Kunden die Möglichkeit, das defekte Fahrrad zu melden und sein Bild hochzuladen.

In Tabelle 4.4 wurden alle Anfragen zusammengefasst, die für die Bearbeitung von der Fehlermeldung notwendig sind.

Nr.	Pfad	Methode	Funktion
1	/report/	POST	- übermittelt die Fahrradnummer, die Kundennummer, die Notiz und das Bild als Binärtext. - erstellt eine neue Meldung in der Datenbank und gibt das Ergebnis zurück.
2	/report/{reportId}	PUT	- übermittelt die Meldungsnummer und das Bild. - aktualisiert die Meldung in der Datenbank

Tabelle 4.4: *Report-Service*

Die bike-rent Sammlung, die in der MongoDB-Datenbank erstellt ist, beinhaltet die Nummer von Meldung, die Kundennummer, die Fahrradnummer, den Status der Reparatur, den Erklärungstext und das Bild als Binärtext. Im Erklärungstext beschreibt der Kunde, wo und wie das Leihfahrrad beschädigt ist. Das Bild, das vom Kunden hochgeladen ist, wird zum Binärtext konvertiert.

4.5 Lokalisierung-Service

Um Diebstähle zu vermeiden, wird die aktuelle Position des Leihfahrrades alle 10 Sekunden an den Server gesendet.

In Tabelle 4.5 wurden die Anfrage erarbeitet, die für die Lokalisierung von Leihfahrrädern erforderlich ist.

Nr.	Pfad	Methode	Funktion
1	/locate/	POST	- übermittelt die Fahrradnummer und die GPS-Daten vom Leihfahrrad. - erstellt einen neuen Eintrag in der Datenbank.

Tabelle 4.5: *Location-Service*

Die bike-location Tabelle, die in der MySQL-Datenbank erstellt ist, beinhaltet die Fahrradnummer, die GPS-Daten und den Zeitpunkt.

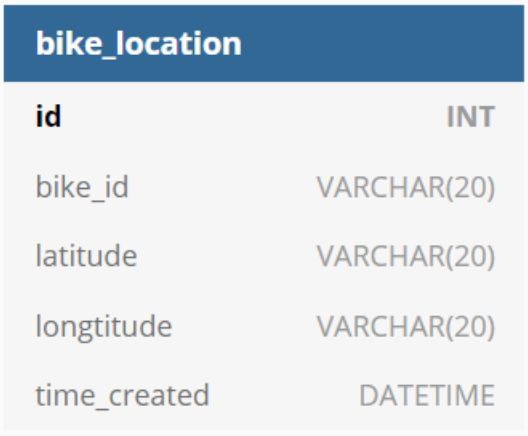

Abbildung 4.7: *bike-location Tabelle*

4.6 Gateway

Um den Autorisierungsvorgang durchzuführen, wird ein Spring Cloud Gateway genutzt, das eine Bibliothek zur Erstellung von API-Gateways auf der Grundlage von Spring und Java ist.

Das Spring Cloud Gateway wird auf Spring Framework 5, Project Reactor und Spring Boot 2.0 aufgebaut. Es ist integriert mit Circuit Breaker und Spring Cloud Discovery Client. Dazu kann es die Anfragerate begrenzen und den Pfad umschreiben. Besonders kann das Gateway Routen für jedes Anforderungsattribut abgleichen. [17]

Alle HTTP-Anfragen werden ein zentrales Gateway gesendet. Das Gateway schickt den übergebenen Token an den Auth-Service, um diesen zu überprüfen. Wenn er korrekt ist, wird die Anfrage weiter an den gewünschten Service geleitet. Wenn der Token ungültig oder gefehlt ist, wird der Dienst verweigert.

4.7 Dienstregister

Die Service-Discovery ermöglicht den Diensten, einander zu finden und miteinander zu kommunizieren, ohne Hostname und Port fest zu kodieren. Der einzige „Fixpunkt“ in einer solchen Architektur besteht aus einem Dienstregister, bei dem sich jeder Dienst registrieren muss. Ein Nachteil ist, dass alle Clients eine bestimmte Logik implementieren müssen, um mit diesem festen Punkt zu interagieren. Dies bedeutet, dass vor der eigentlichen Anfrage ein zusätzlicher Netzwerktrip erforderlich ist. Mit Netflix Eureka kann jeder Client gleichzeitig als Server fungieren, um seinen Status an einen verbundenen Peer zu replizieren. Mit anderen Worten, ein Client ruft eine Liste aller angeschlossenen Peers eines Dienstregisters ab und stellt alle weiteren Anfragen an

alle anderen Dienste über einen Lastausgleichsalgorithmus. Um über die Anwesenheit eines Clients informiert zu werden, muss dieser ein Heartbeat-Signal an die Registry senden. [3]

Um alle Ziele zu erreichen, müssen drei Microservices implementiert werden: eine Dienstregistrierung (Eureka Server), einen REST-Dienst, der sich bei der Registry registriert (Eureka Client) und eine Webanwendung, die den REST-Dienst als Registry-aware Client nutzt (Spring Cloud Netflix Feign Client). [3]

Im Rahmen dieser Abschlussarbeit werden nur Eureka-Server und Eureka-Client für alle Services implementiert.

5 Implementierung des Backendservers

5.1 Annotation

Die @SpringBootApplication Annotation entspricht der Verwendung von @Configuration, @EnableAutoConfiguration und @ComponentScan mit ihren Standardattributen.

- Die @EnableAutoConfiguration Annotation aktiviert den Autokonfigurationsmechanismus von Spring Boot. [19]
- Die @Component Annotation im Paket, wo sich die Anwendung befindet, wird von der @ComponentScan Annotation aktiviert. [19]
- Die @Configuration Annotation ermöglicht die Registrierung zusätzlicher Beans im Kontext oder den Import zusätzlicher Konfigurationsklassen zu haben. [19]

Die @AutoWired Annotation informiert Spring, wo es mittels Injection Objekte in andere Klassen eingefügt werden soll. Die Injektion erfolgt über den Typ des Objekts. Normalerweise werden diese Abhängigkeiten als erforderlich angesehen, mittels @Autowired(required=false) kann man dieses Verhalten jedoch ausschalten. [18]

Für die Deklarierung der Parameter der Funktionen sind folgende Annotationen relevant.

- Die @RequestMapping (path=ÜRL", method=RequestMethod. GET | POST | PUT) Annotation bildet URLs auf Klassen oder Methoden ab. Bei Anwendung auf einer Klasse definiert sie den Basispfad für Anfragen des Controllers. Angewandt auf eine Methode, definiert sie den URI, durch den die Methode ausgeführt wird. Typischerweise bilden die Klassenannotationen auf einem Form-Controller ab. [18]
- Die @PathVariable Annotation wird auf ein Methodenargument angewandt, um den Wert einer URI-Templatevariable auf das Attribut zu übertragen. [18]
- Die @RequestParam Annotation bildet Anfrageparameter in der URL auf Methodenvariablen im Controller ab. [18]
- Die @RequestBody Annotation markiert Methodenargumente von Anfragehandlern und deutet an, dass diese mit den Werten des HTTP Anfrage-Bodys gebunden werden sollen. [18]
- Die @RequestHeader Annotation markiert Methodenargumente von Anfragehandlern und deutet an, dass diese mit den Werten des HTTP Anfrage-Headers gebunden werden sollen. [18]

Für die Implementierung eines Eureka-Servers sind zwei folgende Annotationen erforderlich.

- Die @EnableEurekaServer Annotation aktiviert den Discovery-Service, der den Microservices erlaubt, sich gegenseitig zu finden. Diese Annotation wird auf die Hauptklasse angewandt. [18]
- Die @EnableDiscoveryClient Annotation stellt das Gegenstück zu @EnableEurekaServer dar. Die Annotation wird auf die Einstiegsklasse der Anwendung angewandt. Damit wird dem Service gesagt, dass er sich bei Eureka zu registrieren hat. [18]

5.2 Konfiguration

In der application.properties Datei werden die Portnummer, der Anwendungsname, die Zugangsdaten der Datenbank und Eigenschaften von Hibernate konfiguriert.

```
# serivce name
spring.application.name=bonus-service
# port
server.port=8200
# eureka server url
eureka.client.service-url.default-zone=http://localhost:8761/eureka

## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
spring.datasource.url = jdbc:mysql://178.254.24.xxx:3306/bikeverleih
spring.datasource.username =quang
spring.datasource.password = password

## Hibernate Properties
# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQLDialect
spring.jpa.properties.hibernate.show_sql= true
```

Listing 5.1: *Ausschnitt aus application.properties Datei im bonus-service*

Das gesamte Projekt wird durch Maven, die ein auf Java basierendes Build-Management-Tool der Apache Software Foundation ist, durchgeführt. Deswegen wird in jedem Projekt eine pom.xml Datei ausgestattet, in der alle notwendigen Abhängigkeiten verwaltet und Profile, Plugins und Eigenschaften festgelegt werden.

5.3 Services des Backendservers

Im diesen Unterkapitel wird die Realisierung der Authentifizierung-Services, Bonuskonto-Services, Verleih-Services, Fehlermeldung-Services und Lokalisierung-Services beschrieben.

Jeder Service hat drei Pakete, bei denen die Entitäten, die Controller und die Repository implementiert werden. Außerdem gibt es eine ServiceApplication-Klasse, die die Springboot-Anwendung ausführt.

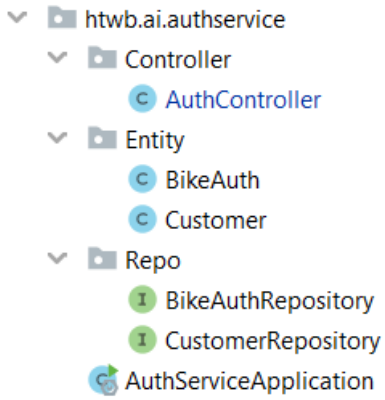

Abbildung 5.1: *Aufbau der Microservice Auth-Service*

5.3.1 Entitätsklassen

Die Entitätsklassen, die mit der @Entity und @Table Annotation markiert sind, beziehen sich auf die Tabellen in der MySQL-Datenbank oder die Sammlung in der Mongo-Datenbank. Das Schlüsselattribut wird zusätzlich mit der @Id Annotation markiert. Die Klasse hat die entsprechenden Getter und Setter Funktionen sowie die toString() Funktion. Die isAcceptable() Funktion kann auch implementiert werden, um zu überprüfen, ob die notwendigen Attribute nicht leer sind.

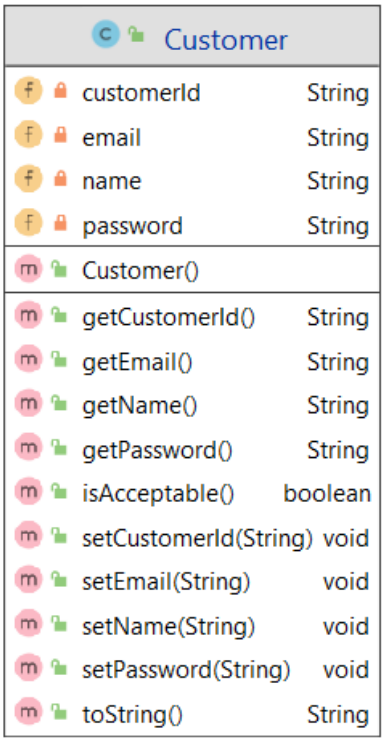

Abbildung 5.2: *Entität Customer*

5.3.2 Repository Pattern

Die Repository Schnittstelle in jedem Microservice erweitert die CrudRepository Schnittstelle mit entsprechenden Typen von der Entitätsklasse und dem Schlüsselattribut. Die CrudRepository Schnittstelle, die eine Schnittstelle von der Spring Data Bibliothek ist, bietet eine ausgefeilte CRUD-Funktionalität für die zu verwaltende Entitätsklasse. Die bereits implementierten Methoden sind count, delete, deleteById, save, saveAll, findById und findAll. Zur Erweiterung dieser Schnittstelle müssen der Entitätstyp und der Identifikationstyp deklariert werden.

```
1  public interface CustomerRepository extends CrudRepository<Customer,String>
       {
2    boolean existsCustomByEmail(String email);
3    Customer findCustomByEmail(String email);
4  }
```

Listing 5.2: *CustomerRepository-Schnittstelle in MS Auth-Service*

5.3.3 Controllerklassen

Die Controllerklassen, die mit den @RestController und @RequestMapping Annotationen markiert sind, besitzen eine oder zwei Variablen der Repository Schnittstellen, die mit der @Autowired Annotation markiert werden.

Die Anfragen aus dem Entwurf in Kapitel 4 werden durch Funktionen in der Klasse verarbeitet. Zuerst muss die HTTP-Methode durch die @GetMapping, @PostMapping oder @PutMapping Annotationen deklariert werden. Der Endpunkt kann auch in Klammern von der Mapping-Annotation beschrieben werden. Die Parameter in Payload oder in Headerteil der Anfrage werden mithilfe der @RequestParam, @HeaderParams oder @PathVarible Annotation zu der Funktion übergegeben. Dank des Objekts von der CrudRepository Schnittstelle sind die logischen Fragen einfach zu verarbeiten. Der Rückgabewert der Funktion in den meisten Fällen ist ein Objekt der ResponseEntity Klasse. Das Objekt beinhaltet den Statuscode und einen Inhalt, der ein gewünschtes Objekt tragen kann.

```
1  // Request 2
2  @PostMapping("/bikelogin")
3  public ResponseEntity getTokenForBike(@RequestParam String bikeId,
4  @RequestParam String password) {
5    if (bikeId.isEmpty() || password.isEmpty()) {
6      return ResponseEntity.badRequest().body("Wrong format");
7    }
8    if (!bikeAuthRepository.existsById(bikeId)) {
9      return ResponseEntity.status(401).body("Bike cannot be authenticated");
10   } else {
11     BikeAuth bikeAuth = bikeAuthRepository.findById(bikeId).get();
12     if(bikeAuth.getPassword().equals(password)){
13       return ResponseEntity.ok().contentType(MediaType.TEXT_PLAIN).body(
       generateAuthTokenForBike(bikeAuth));
```

```
14      } else {
15        return ResponseEntity.status(401).body("Bike cannot be authenticated");
16      }
17    }
18  }
```

Listing 5.3: *Ausschnitt aus der AuthController.java Datei*

5.4 Service Registry Eureka Server

In diesem Service wird nur die EurekaServerApplication Klasse implementiert. Sie wird mit der @SpringBootApplication Annotation und der @EnableEurekaServer Annotation markiert. In der Konfigurationsdatei soll die „register-with-eureka“ Eigenschaften auf „false“ gesetzt werden, damit sich der Eureka-Server nicht als Client registriert.

5.5 Spring Cloud Gateway

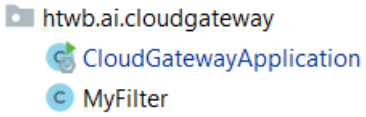

Abbildung 5.3: *Aufbau der Spring Cloud Gateway*

Die CloudGatewayApplication Klasse, die mit der @SpringBootApplication Annotation markiert wird und eine Hauptmethode beinhaltet, startet die Spring-Anwendung. Darüber hinaus wird die gatewayRoutes() Funktion beschrieben, die alle kommenden Anfragen außer der Anfrage, die an den Auth-Service gesendet wird, durch die MyFilter Klasse filtert.

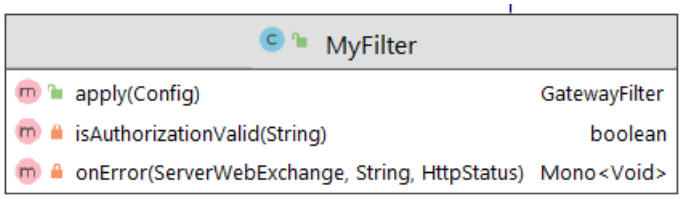

Abbildung 5.4: *MyFilter Klasse*

Neben der CloudGatewayApplication Klasse ist die MyFilter Klasse zu implementieren. In der Klasse sendet die isAuthorizationValid() Funktion eine GET-Anfrage an den Auth-Service, um den Token zu überprüfen. Diese interne Kommunikation nutzt die RestTemplate Klasse. Die apply() Funktion wendet die isAuthorizationValid() Funktion und hängt die Identifikationsnummer an Headerteil der Anfrage an, wenn der Token gültig ist. Ohne den gültigen Token wird die Anfrage mit der Antwort gegeben, deren Statuscode 401 und deren Meldung „Unauthorized“ ist.

6 Teste des Backendservers

Die CrudRepository-Klasse wurde nicht überschrieben, daher braucht sie keinen Test zu schreiben. Die Teste der Controllerklassen aller Microservices werden implementiert.

Die Testklasse muss mit der @SpringBootTest Annotation markiert werden. Die Klasse beinhaltet ein TestResTemplate- Objekt, das auch mit der @Autowired Annotation markiert wird. Alle Testfunktionen basieren auf dem RestTemplate-Objekt, um die Anfrage zu schicken. Neben dem HTTP-Anfragetyp kann auch der gewünschte Rückgabetyp eingestellt werden. Die Header-Parameter und ein Objekt als JSON-Typ können der Anfrage beigefügt werden.

```
1  @Test
2  void postReportTest200() {
3    HttpHeaders headers = new HttpHeaders();
4    headers.setContentType(MediaType.APPLICATION_JSON);
5    headers.add("currentId", "Tester");
6    HttpEntity<String> entity = new HttpEntity<String>(asJsonString(initReport
       ()), headers);
7    ResponseEntity<Report> response = restTemplate.exchange("/report",
       HttpMethod.POST, entity, Report.class);
8    System.out.println("Result: "+response);
9    assertTrue(response.getStatusCode().value() == 200);
10 }
```

Listing 6.1: *Ausschnitt aus der ReportServiceApplicationTest.java Datei*

Aus einer Testfunktion in der Klasse ReportServiceApplicationsTests.java wird eine POST-Anfrage erstellt. Der Header-Parameter wird als JSON-Typ eingerichtet. Die HTTP-Entität beinhaltet das Report-Objekt als JSON-Typ und wird zusammen mit dem Header-Parameter an den Report-Service gesendet. Aus der Antwort wird der Statuscode mitgeteilt und der Inhalt ausgenommen.

7 Installation des Backendservers

Nach der Testphase im lokalen Rechner wird die gesamte Backend-Anwendung auf einem öffentlichen Ubuntu-Server installiert, damit die mobilen Anwendungen den Backendserver über den Mobilfunk erreichen kann.

7.1 Microservice als Systemd-Service

Das Build-Management-Tool Maven erstellt nach der Kompilierung des Projekts eine JAR-Datei im Target-Ordner.

Neben der Ausführung von Spring Boot-Anwendungen mit Kommandobefehl java -jar ist es auch möglich, vollständig ausführbare Anwendungen für Unix-Systeme zu erstellen. Eine vollständig ausführbare jar-Datei kann wie jedes andere ausführbare Binärprogramm ausgeführt werden oder kann mit init.d-Service oder systemd-Service registriert werden. Dies hilft bei der Installation und Verwaltung von Spring Boot-Anwendungen in gängigen Produktionsumgebungen. [13]

Systemd ist ein System- und Dienstmanager für Linux-Betriebssysteme. Er ist abwärtskompatibel zu SysV-Init-Skripten und bietet eine Reihe von Funktionen, wie z. B. den parallelen Start von Systemdiensten beim Booten, die Aktivierung von Dämon bei Bedarf oder eine abhängigkeitsbasierte Dienststeuerungslogik. Systemd führt das Konzept der systemd-Units ein. Diese Units werden durch Unit-Konfigurationsdateien repräsentiert, die sich in /etc/systemd/system-Verzeichnis befinden. Die Systemd-Konfigurationsdatei sammelt Informationen über Systemdienste, lauschende Sockets und andere Objekte, die für das Init-System relevant sind. [15]

Die Service-Unit von Systemd, die einem Microservice entspricht, wird genutzt, um die einzelne Springboot-Anwendung auszuführen. Für jeder Microservice wird eine .service-Datei erstellt und wie unten konfiguriert.

```
1  [Unit]
2  Description=BikeVerleih Eureka-Server
3  After=syslog.target
4  [Service]
5  User=root
6  ExecStart=/usr/bin/java -jar /root/BA/BikeVerleih/eureka-server/target/
     eureka-server-0.jar
7  SuccessExitStatus=143
8  Restart=always
9  [Install]
10 WantedBy=multi-user.target
```

Listing 7.1: *Ausschnitt aus der eureka-server.service Datei*

Beim Ausführen folgendes Befehls startet der Systemd-Service, der unabhängig laufen und nach dem Boot automatisch starten kann.

```
sudo systemctl start auth-service.service
```

Listing 7.2: *Systemd-Service starten*

7.2 Firewall

Eine Firewall ist ein Sicherheitssystem, das den Netzwerkverkehr auf der Grundlage einer Reihe von Sicherheitsregeln überwacht und kontrolliert. Firewalls befinden sich in der Regel zwischen einem vertrauenswürdigen Netzwerk und einem nicht vertrauenswürdigen Netzwerk. So verwenden beispielsweise Büronetzwerke häufig eine Firewall, um ihr Netzwerk vor Online-Bedrohungen zu schützen. Firewalls entscheiden, ob eingehender oder ausgehender Datenverkehr durchgelassen wird. Sie können in Hardware, Software oder einer Kombination aus beidem eingebaut sein. Der Begriff „Firewall“ stammt eigentlich aus dem Bauwesen, wo Wände zwischen oder durch die Mitte von Gebäuden gebaut werden, um einen Brand einzudämmen. [10]

In diesem Backendserver wird die ufw-Anwendung (uncomplicated firewall) genutzt. Durch die Ausführung des folgenden Befehls erlaubt die Firewall nur den Zugriff zu den Cloud Gateway in Port 8762.

```
sudo ufw allow 8762
```

Listing 7.3: *Firewall einstellen*

Der Zugang zu allen anderen Ports für die Microservices wie 8100,8200,8300,8400 wird verweigert. Somit besteht es keine Möglichkeit, ohne die Authentifizierung den Fahrradverleih-Service zu nutzen.

8 Entwurf und Implementierung der mobilen Anwendung für Kunden

8.1 Modell

Booking
+ distance : String + endTime : String + beginTIme: String + id : String
+ toString() : String + setBikeId(bikeId : String) : void + getBikeId() : String + setDistance(distance : String) : void + getDistance() : String + getEndTime() : String + setEndTime(endTime : String) : void + setBeginTime(begintTime : String) : void + getBeginTime() : String + setId(id : String) : void + getId() : String + Booking(id: String, bikeId:String, beginTime:String, endTime:String, distance:String) + Booking()

Tabelle 8.1: *Booking Modell*

Das erste Modell ist Booking-Modell.Die Attribute des Booking-Modells sind die Buchungsnummer, die Fahrradnummer, den Endzeitpunkt, den Anfangszeitpunkt und die Fahrtentfernung. Die Getter Funktionen, Setter Funktionen und die toString() Funktion werden ebenfalls implementiert.

Bike
+ longtitude : String + latitude: String + bikeId : String
+ toString() : String + setBikeId(bikeId : String) : void + getBikeId() : String + setLongtitude(longtitude: String) : void + getLongtitude() : String + getLatitude() : String + setLatitude(latitude : String) : void + Bike()

Tabelle 8.2: *Bike Modell*

Das zweite Modell der Anwendung ist Bike-Modell.Die Attribute des Bike-Modells sind die Fahrradnummer und die GPS-Daten als Latitude und Longtitude.

User
- password : String - email: String - name : String - userId: String
+ toString() : String + isAccepted() : boolean + setPassword(password : String) : void + getPassword() : String + setEmail(mail : String) : void + getEmail() : String + setName(name : String) : void + getName() : String + setUserId(userId : String) : void + getUserId() : String + User(email : String, password : String) + User()

Tabelle 8.3: *User Modell*

Das dritte Modell ist User-Modell. Die Attribute des User Modells sind die E-Mail-Adresse, der Name, das Passwort und die Kundennummer. Die Getter Funktionen, Setter Funktionen und die toString() Funktion werden ebenfalls implementiert. Zusätzlich soll die isAccepted() Funktion realisiert werden, wenn die notwendigen Attribute im erstellten User-Objekt nicht leer sind.

8.2 Benutzerschnittstelle

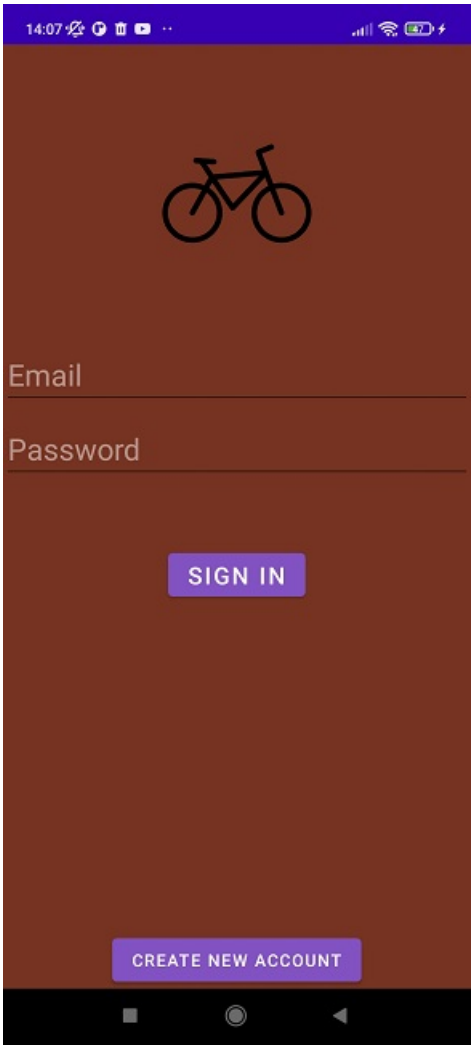

Abbildung 8.1: *Activity-login.xml Datei*

Die activity-login.xml Layoutdatei für Einloggen enthält zwei EditText Elemente für die Eingabe von der E-Mail-Adresse und das Passwort, zwei Schaltflächen für die Bestätigung zum Einloggen und für die Erstellung eines neuen Kontos.

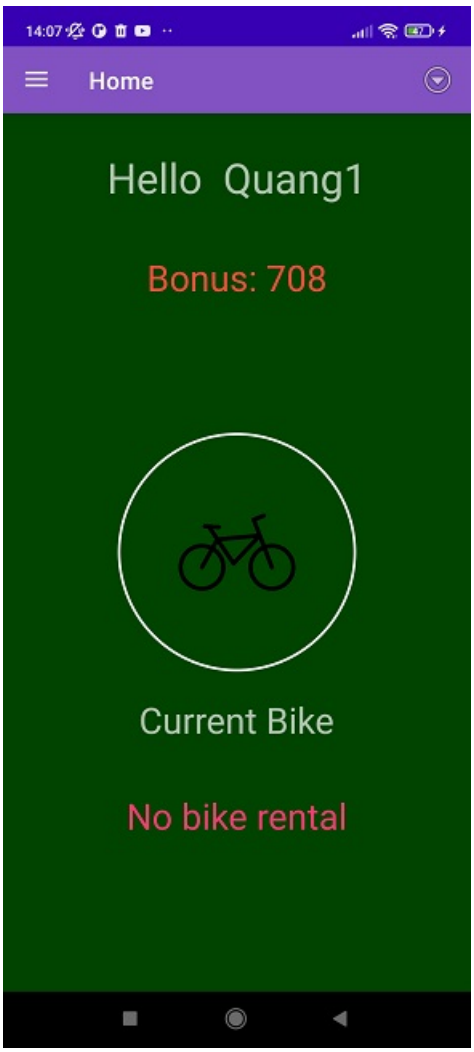

Abbildung 8.2: *Fragment-home.xml Datei*

Die fragment-home.xml Layoutdatei enthält drei TextView-Elemente, die die Begrüßung, den aktuellen Stand der Bonuspunkte und den PIN-Code für das aktuelle Leihfahrrad anzeigen.

Abbildung 8.3: *Fragment-history.xml Datei*

Die fragment-history.xml Layoutdatei enthält ein ListView Element, das nötig für die Anzeige einer Sammlung von allen Buchungen der Kunden ist.

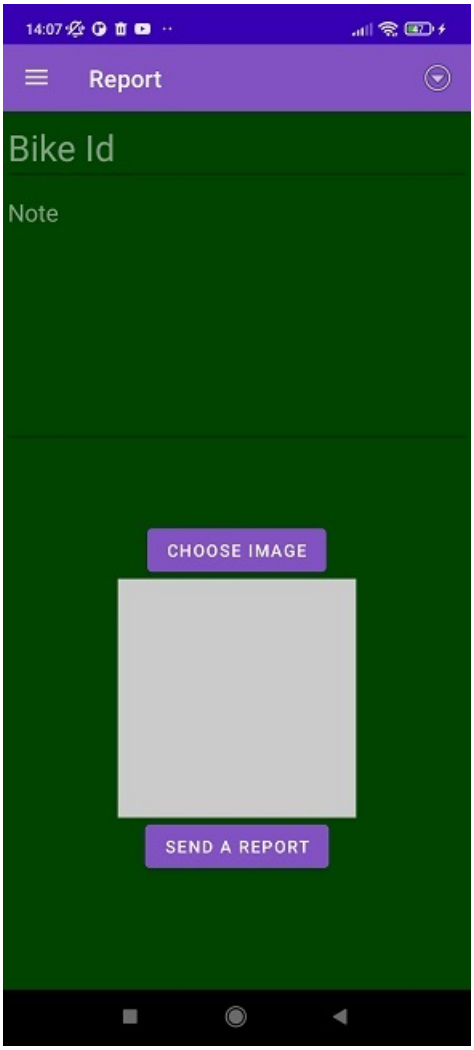

Abbildung 8.4: *Fragment-report.xml Datei*

Die fragment-report.xml Layoutdatei hat zwei TextView-Elemente, die für die Eingabe der Fahrradnummer und die Beschreibung des Schadens nötig sind, ein ImageView Element, das das hochgeladene Bild ausstellt, und zwei Schaltflächen für die Auswahl der Bilddatei und die Bestätigung zum Senden der Meldung.

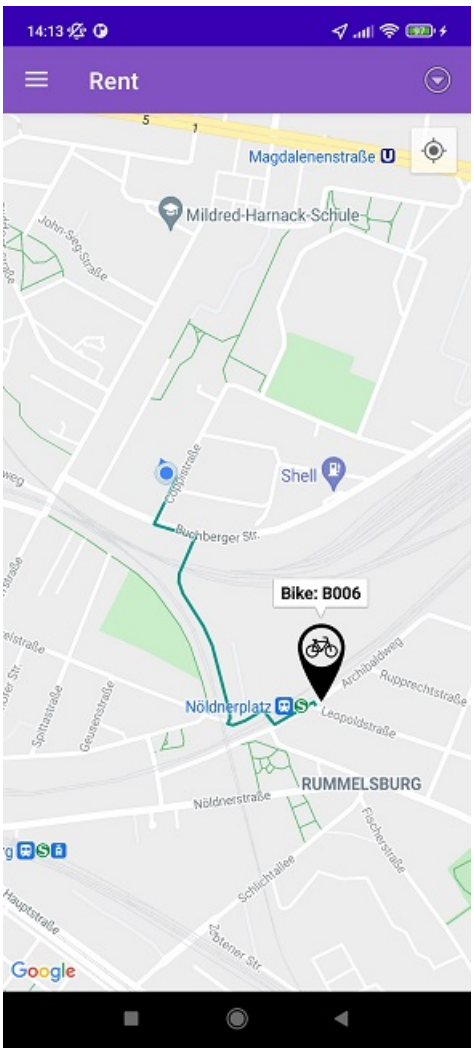

Abbildung 8.5: *Fragment-rent.xml Datei*

Die fragment-rent.xml Layoutdatei nutzt die Standard-Karte von Google Maps, um den aktuellen Standort der Kunden, die Orte der verfügbaren Leihfahrräder sowie die Reiseroute anzuzeigen.

8.3 Controller

8.3.1 MainActivity

MainActivity.java
onCreate(savedInstanceState : Bundle) : void # onResume() : void + onCreateOptionsMenu(menu : Menu) : boolean + onOptionsItemSelected(item : MenuItem) : boolean + onSupportNavigateUp() : boolean + gotoLoginActivity() : void + getAccessPermission() : void - checkAndRequestPermissions() : boolean - getLocationPermission() : void + onRequestPermissionsResult(requestCode : int, permissions : String[], grantResults : int[]) : void - updateUIAfterPermissionRequest() : void

Tabelle 8.4: *MainActivity.java*

Die MainActivity steuert die actvity-main.xml Layoutdatei, die als Erstes gestartet wird.

Bei der ersten Öffnung der mobilen Anwendung muss der Kunde die Genehmigungen für den Zugriff auf Speicher, auf das Internet und die Lokalisierung geben, indem die getAccessPermission() Funktion aufgerufen wird. Nach der Erteilung der Genehmigung werden die Kundendaten, die beim erfolgreichen Einloggen im SharedPreference Speicher gespeichert sind, abgerufen. Wenn diese Daten leer sind, navigiert die Anwendung zur LoginActivity, damit sich der Kunde anmelden kann.

8.3.2 LoginActivity

LoginActivity.java
+ onCreate(savedInstanceState : Bundle) : void + hashPassword(password : String) : String + setUIAfterRegister(response JSONObject, user : User, context : Context) : void + postRegister(user : User, context : Context) : void + setUIAfterLogin(response JSONObject, user : User, context : Context) : void + postLogin(user : User, context : Context) : void - showLoginFailed(error : VolleyError) : void + gotoMainActivity() : void

Tabelle 8.5: *LoginActivity.java*

Die LoginActivity steuert die activity-login.xml Layoutdatei.

```
// SEND Post Request to Login
public void postLogIn(User user, Context context) throws JSONException {
  System.out.println("postLogIn Func");
  RequestQueue requestQueue = Volley.newRequestQueue(context);
  JSONObject object = new JSONObject();
  object.put("email",user.getEmail());
  object.put("password",hashPassword(user.getPassword()));

  JsonObjectRequest request = new JsonObjectRequest(Request.Method.POST, URLs
    .URL_LOGIN, object, new Response.Listener<JSONObject>() {
    @Override
    public void onResponse(JSONObject response) {
      System.out.println("RESPONSE in postLogIn: "+response.toString());
      try {
        setUIAfterLogin(response,user,context);
      } catch (JSONException e) {
        e.printStackTrace();
      }

      gotoMainActivity();
    }
  }, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
      System.out.println("Error: " + error);
      showLoginFailed(error);
    }
  });
  requestQueue.add(request);
}

```

Listing 8.1: *postLogin() Funktion in LoginActivity.java Datei*

Nachdem der Kunde die E-Mail-Adresse und das Passwort eingegeben hat, wird die postLogin() Funktion aufgerufen. Danach wird das Passwort gehackt, indem die hashPassword() Funktion aufgerufen wird, um diese Sicherheitswörter auf keinen Fall im Klartext im Internet zu senden. Diese postLogin() Funktion nutzt die Bibliothek Volley. Sie schickt eine POST-Anfrage mit der E-Mail-Adresse und dem gehackten Passwort vom Kunden an den Backendserver und wartet auf Antwort ein JSON-Objekt, die einen Token enthält. Der Token muss später bei allen Anfragen angefügt werden. Bei der erfolgreichen Anmeldung werden ebenfalls die Kundendaten im SharedPrefernce Speicher gespeichert.

Auf Wunsch kann der Kunde ein neues Konto anlegen, bei dem ein neues EditText Element auftritt, das für den Namen eingegeben wird. Die postRegister() Funktion wird aufgerufen und sendet eine POST-Anfrage an den Backendserver, was der postLogin() Funktion mit zusätzlich dem Namen des Kunden ähnelt.

Am Ende beendet die LoginActivity und die Anwendung kommt zur MainActivity zurück.

In der MainActivity können das HomeFragment, das RentFragment und das HistoryFragment ausgewählt werden.

8.3.3 HomeFragment

HomeFragment.java
+ onCreate(inflater LayoutInflater, container:ViewGroup, savedInstanceState : Bundle) : View + onResume() : void + onDestroyView() : void + sendRequestToUpdateUI() : void + changeToRentFragment() : void - rotation() : void + setTextViewBonusScore(text : String) : void + getBonusScore() : void + setUI(response : JSONObject) : void + setUIByError() : void + getCurrentBooking() : void

Tabelle 8.6: *HomeFragment.java*

Das HomeFragment steuert die fragment-home.xml Layoutdatei.

Die getBonusScore() Funktion sendet eine GET-Anfrage mit der Kundennummer und bekommt den aktuellen Stand des Bonuskontos der Kunden.

Die getCurrentBooking() Funktion endet eine GET-Anfrage mit der Kundennummer und bekommt die aktuellen Buchungsdaten, mit denen die Fahrradnummer und der PIN-Code genau angezeigt werden.

8.3.4 RentFragment

RentFragment.java
+ onCreateView(inflater : LayoutIntflater, container : Container : ViewGroup , savedlnstanceState : Bundle) : View + onDestroyView(): void + onMapReady(googleMap : GoogleMap) : void + mapListener(): void + findRoutes(Begin : LatLng, End : LatLng) : void + onRoutingCancelled(): void + onRoutingFailure(e : RouteException) : void + onRoutingStart(): void + onRoutingSuccess(routes : ArrayList<Route>, shortestRoutelndex : int) : void + createParametersGetBikeLocation(latitude : String, longtitude : string) : Map<string, string> + getBikeLocations(params : Map<string, string>, context : Context) : void + rentBike(bikeld : String, customerld : String, context : Context) : void + createListOfBikeFromJSonArray(jsonArray : JSONArray) : List<Bike> + createMarkerFromBikeLocation(list : List<Bike>) : List<MarkerOptions> + createMarker(bikeld : string, latitude : string, longtitude : string) : MarkerOptions + addPointer(listOtMarker : List<MarkerOptions>) : void - getDeviceLocation(): void - updateLocationUI(): void + createDialog(marker : Marker) : void + createAlertDialogForPIN(bikeld : String, pin : String) : void

Tabelle 8.7: *RentFragment*

Das RentFragment steuert die fragment-rent.xml Layoutdatei.

Die wichtigste Funktion in diesem Fragment ist die getDeviceLocation() Funktion, die den aktuellen Standort der Kunden nimmt und anschließend die getBikeLocations() Funktion aufruft.

Die getBikeLocations() Funktion sendet eine GET-Anfrage an den Backendserver, um Standortpunkte von Leihfahrrädern, die gerade verfügbar im Umkreis von 5 km sind, zu finden. Natürlicherweise muss auch der aktuelle Standort der Kunden als Parameter mitgeliefert werden. Die Funktion wartet auf die Antwort, die einer Sammlung von Standortpunkten ist. Danach werden die Standortpunkte der Leihfahrräder auf der Karte dargestellt.

Die rentBike() Funktion wird aufgerufen, wenn der Kunde ein Leihfahrrad auf der Karte ausgewählt hat. Sie schickt eine POST-Anfrage an den Backendserver und wartet auf einen PIN-Code, der für die Entsperrung des Leihfahrrades benötigt ist.

Die anderen wichtigen Funktionen des RentFragments werden in Kapitel 8.4 beschrieben.

8.3.5 ReportFragment

ReportFragment.java
+ onCreateView(inflater : Layoutlnflater, container : ViewGroup, savedlnstanceState : Bundle) : View + onDestroyView(): void - fileChoosing() : void + onActivityResult(requestCode : int, resultCode : int, data : Intent) : void + sendReport(bikeId : String, note : String) : void + uploadImage(bitmap : Bitmap, reportId : String) : void + getPath(uri : Uri) : String + getFileDataFromDrawable(bitmap : Bitmap) : byte[]

Tabelle 8.8: *ReportFragment.java*

Das ReportFragment steuert die Layoutdatei „fragment-report.xml“.

Nach dem Klick der Schaltfläche „Choose Image“ wird die fileChoosing() Funktion aufgerufen. Die besondere Activity „Intent.createChoose“ wird durchgeführt, damit der Kunde ein Bild aus der Bildgalerie auswählen kann.

Nach dem Klick der Schaltfläche „Send report“ wird die sendReport() Funktion aufgerufen. Diese Funktion schickt eine POST-Anfrage an den Backendserver, um eine neue Fehlermeldung zu erstellen. Danach wird die uploadImage() Funktion aufgerufen, falls der Kunde zuvor ein Bild ausgewählt hat.

Die uploadImage() Funktion ruft zuerst die getFileDataFromDrawable() Funktion auf, die das Bild zum Binärtext umwandelt, und schickt danach eine PUT-Anfrage zusammen mit dem Binärtext an den Backendserver, um das ausgewählte Bild als Binärtext zum Server hochzuladen und dort zu speichern.

8.3.6 HistoryFragment

HistoryFragment.java
+ HistoryFragment() + newInstance(columnCount: int) : HistoryFragment + onCreate(savedInstanceState : Bundle) : void + onCreateView(inflater : LayoutInflater, container : ViewGroup, savedInstanceState : Bundle) : View + createListOfBookingFromJSonArray(jsonArray: JSONArray) : List<Booking> + updateUI(response : JSONArray) : void + getBookingList() : void

Tabelle 8.9: *HistoryFragment.java*

Das HistoryFragment steuert die Layoutdatei „fragment-history.xml“.

Die getBookingList() Funktion sendet eine GET-Anfrage mit der Kundennummer an den Backendserver, um alle Buchungen von Kunden zu bekommen. Der Antworttyp ist eine Sammlung von JSON-Objekten, die zu einer Sammlung von Booking Objekt umgewandelt ist. Diese Sammlung wird dank des ListView Elements dargestellt.

8.4 Bibliothek

8.4.1 Map SDK for Android

Mit der Bibliothek „Maps SDK for Android“ kann die Android-App Karten hinzufügen, die auf Google Maps-Daten sowie -Kartendarstellungen basieren und auf Kartensteuerungsgesten reagieren. Darüber hinaus können die Karte Markierungen, Polygone und Overlays hinzugefügt werden und so zusätzliche Informationen zu Standorten bieten sowie die Interaktion mit der Karte ermöglichen. Die Bibliothek unterstützt sowohl die Programmiersprache Kotlin als auch Java und stellt zusätzliche Bibliotheken sowie erweiterte Funktionen und Programmiertechniken zur Verfügung. [8]

Die build.gradle Datei auf App-Ebene enthält folgende Kartenabhängigkeit, die für die Bibliothek „Maps SDK for Android“ erforderlich ist:

```
1  dependencies {
2    implementation 'com.google.android.gms:play-services-maps:17.0.1'
3  }
```

Listing 8.2: *Dependecy Konfiguration für MapSDK Android*

Diese Bibliothek wird im RentFragment genutzt. Das Fragment implementiert die OnMapReadyCallBack Klasse. Die onMapReady() Funktion, die erforderlich zu überschreiben ist, hat die Aufgaben, anhand der GPS-Daten die Markierungspunkte zu zeichnen und die Kameraansicht vernünftig zu ändern. Dabei werden die updateLocationUI() Funktion, die getDeviceLocation() Funktion und die mapListener() Funktion aufgerufen.

```
1  @Override
2  public void onMapReady(@NonNull GoogleMap googleMap) {
3    map = googleMap;
4    map.setOnMyLocationButtonClickListener(new GoogleMap.
       OnMyLocationButtonClickListener() {
5      @Override
6      public boolean onMyLocationButtonClick() {
7        // Turn on the My Location layer and the related control on the map.
8        updateLocationUI();
9        // Get the current location of the device and set the position of the
       map.
10       getDeviceLocation();
11       mapListener();
12       return false;
13     }
14   });
15   // Turn on the My Location layer and the related control on the map.
16   updateLocationUI();
17   // Get the current location of the device and set the position of the map.
18   getDeviceLocation();
19   mapListener();
20 }
```

Listing 8.3: *onMapReady() Funktion im RentFragment*

8.4.2 Directions API

Directions API ist ein Webdienst, der mittels einer HTTP-Anfrage JSON- oder XML-formatierte Wegbeschreibungen zwischen Orten zurückgibt. Es können Wegbeschreibungen für verschiedene Verkehrsmittel erhaltet werden, z. B. für den Nahverkehr, das Autofahren, das Gehen oder das Radfahren. [6]

Für Richtungsberechnungen, die in Echtzeit auf Benutzereingaben reagieren (z. B. in einem Element der Benutzeroberfläche), können auch diese Schnittstellen verwendet werden. Für die serverseitige Nutzung können Java Client, Python Client, Go Client und Node.js Client für Google Maps Services genutzt werden. [6]

8.4.3 Google Direction Android

Eine Bibliothek von Drittanbietern auf Basis von der Bibliothek „Directions API“ heißt „Google-Directions-Android“, die von Joel Dean entwickelt wurde, wird in der Anwendung verwendet.

Die build.gradle Datei auf App-Ebene soll die folgende Zeile hinzufügen, um die Bibliothek zu importieren.

```
1  dependencies {
2    implementation 'com.google.android.gms:play-services-maps:17.0.1'
3  }
```

Listing 8.4: *Dependecy Konfiguration für Google Direction Android*

Das RentFragment nutzt diese Bibliothek und die RoutingListener Schnittstelle wird implementiert, indem die folgenden vier Funktionen überschrieben werden.

```
1  @Override
2  public void onRoutingStart() {  }
3
4  @Override
5  public void onRoutingSuccess(ArrayList<Route> routes, int
     shortestRouteIndex){}
6
7  @Override
8  public void onRoutingCancelled() {  }
9
10 @Override
11 public void onRoutingFailure(RouteException e) { }
```

Listing 8.5: *die zu implementierte Funktionen im RentFragment*

In der findRoute() Funktion wird ein Routing Objekt erzeugt, das die Anfangspunkte (die GPS-Daten von Kunden) und die Endpunkte (die GPS-Daten von dem gewünschten Leihfahrrad) besitzt. Das Routing Objekt wird aufgebaut und dann ausgeführt, um den Weg zum Leihfahrrad zu finden. Beim erfolgreichen Wegfinden wird die onRoutingSuccess() Funktion aufgerufen, in der der Fahrtweg auf der Karte gezeichnet wird. Im Gegenfall wird die onRoutingFailure() Funktion aufgerufen, um die Kunden zu benachrichtigen.

```
1    // find Routes by Using Google-Direction-Android
2   public void findRoutes(LatLng Begin, LatLng End) {
3     if(Begin==null || End==null) {
4       Toast.makeText(getContext(),"Unable to get location", Toast.LENGTH_LONG
      ).show();
5     } else {
6       Routing routing = new Routing.Builder()
7       .travelMode(AbstractRouting.TravelMode.WALKING)
8       .withListener(this)
9       .alternativeRoutes(true)
10      .waypoints(Begin, End)
11      .key(getString(R.string.google_maps_key))
12      .build();
13      routing.execute();
14    }
15  }
```

Listing 8.6: *Ausschnitt aus der RentFragment.java Datei*

9 Entwurf und Implementierung der mobilen Anwendung für Leihfahrräder

9.1 Benutzerschnittstelle

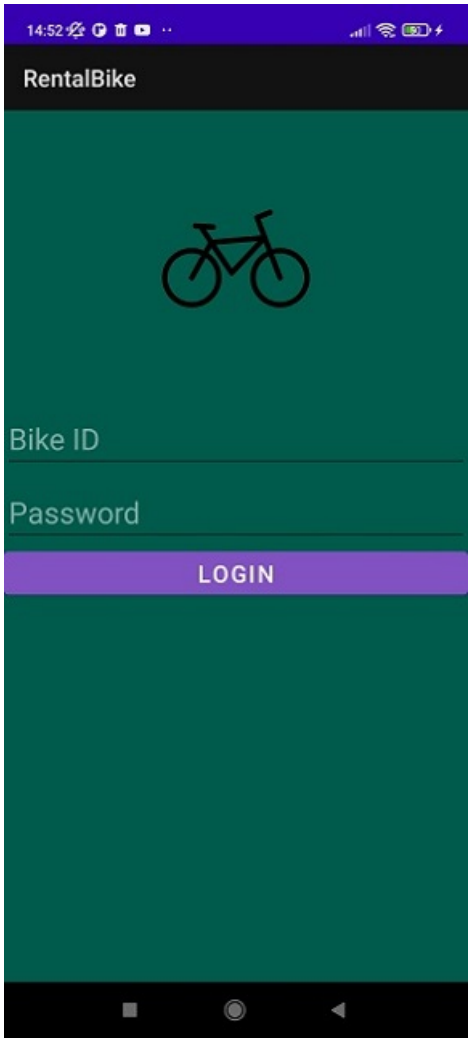

Abbildung 9.1: *activity-login.xml Datei*

Bevor der Kunde ein Fahrrad verleihen kann, muss das Fahrrad von dem Techniker in dem Server eingeloggt werden. Die activity-login.xml Layoutdatei beinhaltet zwei Edit-Text Elemente, die für die Eingabe der Fahrradnummer und des Passwortes zuständig sind, und eine Schaltfläche für die Bestätigung des Einloggens.

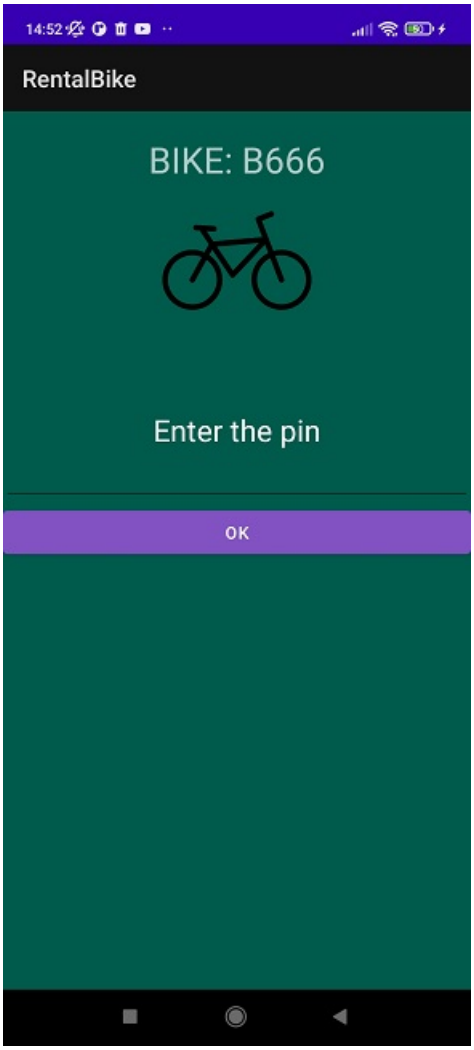

Abbildung 9.2: *activity-main.xml Datei*

Die activity-main.xml Layoutdatei beinhaltet ein EditText Element, in dem der Kunde den PIN-Code eingeben muss, ein TextView-Element, das für die Anzeige der Fahrradnummer zuständig ist, und eine Schaltfläche für die Bestätigung des Sendens vom PIN-Code.

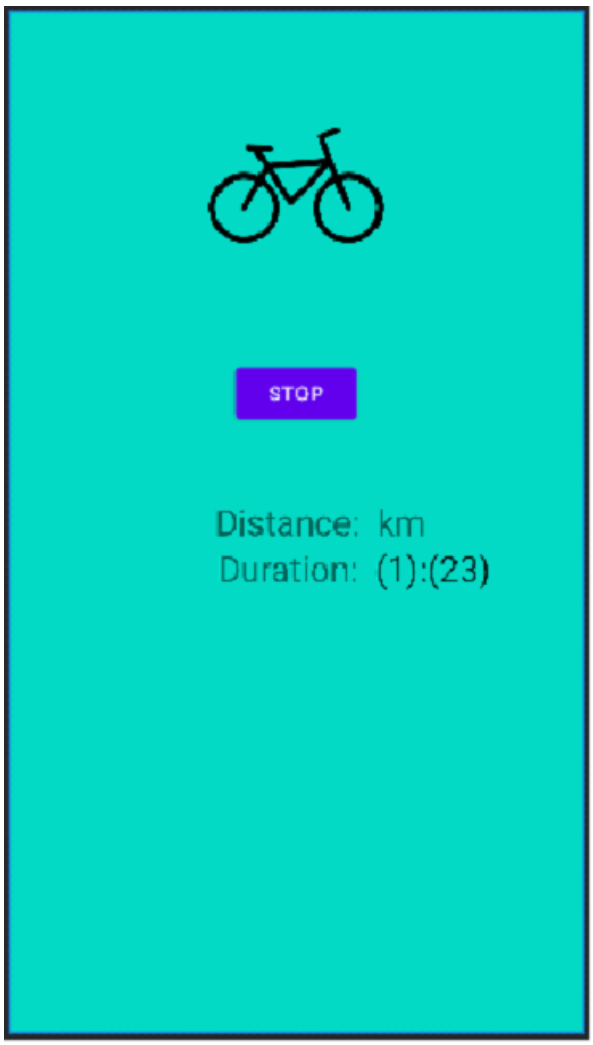

Abbildung 9.3: *activity-rent.xml Datei*

Die activity-rent.xml Layoutdatei beinhaltet eine Schaltfläche für das Ende der Fahrt und zwei TextView-Elemente, die die Fahrtentfernung und die Fahrtdauer berichten.

9.2 Controller

9.2.1 LoginActivity

LoginActivity.java
+ gotoMainActivity() : void + bikeLogin(Context : context, bikeId : String, password : String) : void # onCreate(savedInstanceState : Bundle) : void

Tabelle 9.1: *LoginActivity.java*

Die LoginActivity steuert die activity-login.xml Layoutdatei.

Nachdem der Kunde die Schaltfläche „Sign In“ geklickt hat, wird die bikeLogin() Funktion aufgerufen. Diese Funktion sendet eine POST-Anfrage zusammen mit der Fahrradnummer und dem Passwort an den Backendserver, um einen Token zu bekommen. Der Token muss auch mit allen Anfragen angehängt werden, um die Authentifizierung beim Server durchzuführen. Schließlich navigiert die Anwendung zur MainActivity.

9.2.2 MainActivity

MainActivity.java
+ onRequestPermissionsResult(requestCode : int, permissions : String[], grantResults : int[]) : void - getLocationPermission() : void + sendCurrentLocation(url : String, x : int) : void + gotoRentActivity() : void + gotoLoginActivity() : void + setUI() : void + pinCheck(context : Context, bikeId : String, pin : String) : void # onResume() : void # onCreate(savedInstanceState : Bundle) : void

Tabelle 9.2: *MainActivity.java*

Die MainActivity steuert die Layoutdatei „activity-main.xml“.

Die gotoLoginActivity() Funktion wird aufgerufen, falls keine Logindaten des Leihfahrrades im Speicher gefunden sind.

Die sendCurrentLocation() Funktion erstellt einen zeitlich Ablauf, bei dem der aktuelle Standort des Leihfahrrades alle 10 Sekunden durch eine POST-Anfrage an den Backendserver geschickt wird.

Die pinCheck() Funktion sendet eine GET-Anfrage mit dem von Kunden eingegebenen PIN-Code, um diesen Code zu überprüfen. Im korrekten Fall leitet die Anwendung zur RentActivity um.

9.2.3 RentActivity

RentActivity.java
+ sendAndUpdateLocation(url : String, x : String) : void # onResume() : void + endJourney() : void # onCreate(savedInstanceState : Bundle) : void

Tabelle 9.3: *RentActivity.java*

Die RentActivity steuert die Layoutdatei „activity-rent.xml“

Die sendAndUpdateLocation() Funktion ähnelt der sendCurrentLocation() Funktion, die den aktuellen Standort des Leihfahrrades an den Server sendet und berechnet die zurückgelegte Strecke durch die updateDistance() Funktion.

Am Ende der Fahrt wird die endJourney() Funktion aufgerufen. Sie sendet eine PUT-Anfrage mit der Fahrradnummer, der Buchungsnummer, dem Endzeitpunkt und der zurückgelegten Entfernung an den Backendserver.

```
1  // Send PUT-Request to Server
2  public void endJourney(){
3    chronometer.stop();
4    Toast.makeText(getApplicationContext(), "Timed: "+chronometer.getText(),
     Toast.LENGTH_SHORT).show();
5    timer.cancel();
6
7    JSONObject jsonObject = new JSONObject();
8    try {
9      jsonObject.put("bikeId",retrieveBikeID(getApplicationContext()));
10     jsonObject.put("id",Integer.valueOf(bookingId));
11     jsonObject.put("endTime", Helper.simpleDateFormat.format(new Timestamp(
     System.currentTimeMillis())));
12     jsonObject.put("distance",(Integer)Math.round(distance));
13   } catch (JSONException e) {
14     e.printStackTrace();
15   }
16
17   // Instantiate the RequestQueue.
18   RequestQueue queue = Volley.newRequestQueue(this);
19   // Request a string response from the provided URL.
20   JsonObjectRequest request = new JsonObjectRequest(Request.Method.PUT,
     URLs.URL_ROUTE_END, jsonObject,
21   new Response.Listener<JSONObject>() {
```

```
22        @Override
23        public void onResponse(JSONObject response) {
24          System.out.println("Response is: "+ response);
25        }
26      }, new Response.ErrorListener() {
27        @Override
28        public void onErrorResponse(VolleyError error) {
29          System.out.println(error.toString());
30          Toast.makeText(getApplicationContext(), "Login failed", Toast.
      LENGTH_LONG).show();
31        }
32      }){
33        @Override
34        public Map<String, String> getHeaders() throws AuthFailureError {
35          Map<String, String> headers = new HashMap<String, String>();
36          headers.put("Authorization",retrieveToken(getApplicationContext()));
37          return headers;
38        }
39      };
40      // Add the request to the RequestQueue.
41      queue.add(request);
42    }
```

Listing 9.1: *endJourney() Funktion in der RentActivity*

10 Fazit

10.1 Zusammenfassung

Das Ziel dieser Arbeit war es, ein gesamtes System, das aus einem Backendserver, einer mobilen Anwendung für Kunden und einer mobilen Anwendung für Leihfahrräder besteht, zu entwickeln.

Dafür wurden in Anlehnung an das REST-Paradigma aus den Anforderungen Anfragen abgeleitet, auf die der Backendserver mit HTTP-Antwort reagieren soll. Eine Antwort liefert der mobilen Anwendung Informationen in Form von Parametern mit den entsprechenden Werten im JSON-Format oder Plain Text-Format. Durch Anfragen können Daten aus der Datenbank abgefragt und verändert werden. Darüber hinaus können durch Anfragen Datenbankeinträge erstellt werden. Mittels der Bibliothek Volley Android wird der Kommunikationsaufbau zwischen mobilen Anwendungen und dem Backendserver realisiert. Bei der Nutzung der Bibliotheken MapSDK for Android und Directions API werden die Navigationsfunktion und Positionierfunktion bequem implementiert. Somit wurden alle funktionalen Anforderungen erfüllt.

Die Backendanwendung wird auf dem öffentlichen Server installiert und antwortet auf Anfrage von den mobilen Anwendungen über mobiles Netzwerk in einer angemessenen Zeit. Die Kundendatenbank speichert nur das gehashte Passwort und keine sensiblen Daten der Kunden. Die Benutzerschnittstellen der mobilen Anwendungen sind sehr leicht zu bedienen. Darum werden auch alle nicht funktionalen Anforderungen umgesetzt.

10.2 Ausblick

Bei der Betrachtung der Ergebnisse sind ein paar Ideen für zukünftige Erweiterungen entstanden. Ein vollständiges Fahrradverleihsystem in der Praxis braucht noch eine mobile Anwendung für den Techniker, der alle defekten Fahrräder finden kann, um sie zu reparieren. Eine wichtige Erweiterung ist ein zusätzlicher Abrechnungsdienst, der aus der Fahrtdauer und den Bonuspunkten berechnet wird. Eine mögliche Optimierung ist die Nutzung des QR-Codes auf dem Bildschirm des Leihfahrrades, somit der Kunde schneller die Reise starten kann.

Mein persönliches Feedback ist daher, dass ich die Implementierung von größeren Projekten mehr üben muss, um schneller zu werden. Vorher habe ich noch kein umfangreiches Projekt, das von Frontend bis zu Backend ist, mitgemacht. Deshalb war diese Teilnahme eine wertvolle Erfahrung für mich.

Quellenverzeichnis

[1] Online: https://pretius.com/blog/benefits-of-microservices/.

[2] Dietmar Abts. *Masterkurs Client/Server-Programmierung mit Java. Anwendungen entwickeln mit Standard-Technologien*. 5. Auflage. Wiesbaden, Germany: Springer Vieweg, 2019. ISBN: 978-3-658-25925-9.

[3] *Baeldung Spring Cloud Netflix Eureka*. Online: https://www.baeldung.com/spring-cloud-netflix-eureka.

[4] Subhashini Chellappan und Dharanitharan Ganesan. *MongoDB Recipes. With Data Modeling and Query Building Strategies*. 1. New York,USA: Apress, 2020. ISBN: 978-1-4842-4890-4.

[5] Binildas Christudas. *Practical Microservices Architectural Patterns. Event-Based Java Microservices with Spring Boot and Spring Cloud*. 1. New York,USA: Apress, 2019. ISBN: 978-1-4842-4501-9.

[6] *Direction API Documentation*. Online: :https://developers.google.com/maps/documentation/directions/overview?hl=de.

[7] *Documentation Android Volley*. Online: https://developer.android.com/training/volley; Apache.

[8] *Documentation Map SDK for Android*. Online: :https://developers.google.com/maps/documentation/android-sdk/overview?hl=de.

[9] *Documentation Volley*. Online: https://developer.android.com/images/training/volley-request.png.

[10] *Firewall*. Online: :https://www.cloudflare.com/learning/security/what-is-a-firewall.

[11] Moises Macero Garcia. *Learn Microservices with Spring Boot. A Practical Approach to RESTful Services Using an Event-Driven Architecture, Cloud-Native Patterns, and Containerization*. 2. Auflage. New York,USA: Apress, 2020. ISBN: 978-1-4842-6131-6.

[12] *Hypertext Transfer Protocol – HTTP/1.1*. Online: https://datatracker.ietf.org/doc/html/rfc2616.

[13] *Installing Spring Boot Applications*. Online: https://docs.spring.io/spring-boot/docs/current/reference/html/deployment.html.

[14] Azat Mardan. *Fullstack JavaScript. Learn Backbone.js, Node.js and MongoDB*. 2. New York,USA: Apress, 2018. ISBN: 978-1-4842-3718-2.

[15] *Red Hat Enterprise Linux 7. Chapter 10*. Online: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7.

[16] Bob Reselman. *An architect's guide to GPS and GPS data formats*. Online: `https://www.redhat.com/architect/architects-guide-gps-and-gps-data-formats`. 2021.

[17] *Spring Cloud Gateway*. Online: `https://spring.io/projects/spring-cloud-gateway`.

[18] *Spring Konzept Annotation*. Online: `:http://www.matthiassommer.it/programming/spring-konzepte-annotationen/`.

[19] *Springboot Application Annotation*. Online: `https://docs.spring.io/spring-boot/docs/2.0.x/reference/html/using-boot-using-springbootapplication-annotation.html`.

[20] Balaji Varanasi und Sudha Belida. *Spring Rest. Rest And Webservices Development Using Spring*. 1. New York,USA: Apress, 2015. ISBN: 978-1-4842-0824-3.

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

_____

Datum, Ort, Unterschrift