

# DESIGN UND IMPLEMENTIERUNG EINER SMARTPHONE-APP FÜR DEN BIKELIN-NAVIGATOR

Abschlussarbeit  
zur Erlangung des akademischen Grades:  
**Bachelor of Science**

an der

Hochschule für Technik und Wirtschaft Berlin  
Fachbereich 4: Informatik, Kommunikation und Wirtschaft  
Studiengang Angewandte Informatik

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

Eingereicht von Karl Schulz

27. Februar 2023

# Abstract

Rad fahren ist gesund, gut für die Umwelt und spart Zeit und Kosten.[1] Dabei gibt es oft nur ein Problem. Unsere Städte sind weitestgehend für das Auto gebaut und meist Radfahrer:innen unfreundlich. Um Radfahrer:innen im Alltag eine Abhilfe zu schaffen, bietet die Webanwendung „Bikelin Navigator“ Funktionen zur Planung von Radfahrten in Berlin.

Damit Benutzer:innen der Anwendung von den zur Verfügung gestellten Informationen auch von unterwegs profitieren können, beschäftigt sich diese Bachelorarbeit mit der Planung und Implementierung einer App für den Bikelin Navigator. Dabei wird besonders Fokus auf die Benutzer:innenfreundlichkeit sowie eine Unterstützung beim Absolvieren der Route gelegt.

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
<b>2 Grundlagen</b>	<b>2</b>
2.1 Native App Entwicklung . . . . .	2
2.2 Cross-Plattform App Entwicklung . . . . .	2
2.3 Gegenüberstellung . . . . .	3
2.3.1 Performance . . . . .	3
2.3.2 Entwicklungszeit & Kosten . . . . .	3
2.3.3 Sicherheit . . . . .	3
2.3.4 Testbarkeit & Wartbarkeit . . . . .	4
2.3.5 Benutzer:innenerfahrung & Individuelle Anpassungen . . . . .	4
2.3.6 Zusammenfassung . . . . .	5
2.4 Cross-Plattform Frameworks . . . . .	6
2.4.1 React Native . . . . .	6
2.4.2 Flutter . . . . .	7
2.4.3 Gegenüberstellung . . . . .	7
2.5 Expo . . . . .	8
<b>3 Analyse</b>	<b>9</b>
3.1 Ist-Zustand . . . . .	9
3.2 Soll-Zustand . . . . .	10
3.3 Marktanalyse . . . . .	11
3.3.1 Google Maps . . . . .	12
3.3.2 Komoot . . . . .	14
3.3.3 Ergebnis . . . . .	15
3.4 Funktionale Anforderungen . . . . .	16
3.5 Nicht funktionale Anforderungen . . . . .	17
<b>4 Konzeption</b>	<b>20</b>
4.1 Scrum . . . . .	20
4.2 Zeitplan . . . . .	20

4.3	Wireframes	20
<b>5</b>	<b>Implementation</b>	<b>22</b>
5.1	Sprint 1	22
5.1.1	Projekt Struktur	22
5.1.2	Navigation	23
5.1.3	Anlegen einer Route	23
5.2	Sprint 2	24
5.2.1	Karte	24
5.2.2	Marker	25
5.2.3	Context API	25
5.2.4	Speichern von Routen	25
5.3	Sprint 3	26
5.3.1	Routen Berechnung	26
5.3.2	Darstellen einer Route	26
5.3.3	Kamerafokus	26
5.4	Sprint 4	27
5.4.1	Anzeigen von gespeicherten Routen	27
5.4.2	Live Navigation	28
<b>6</b>	<b>Tests</b>	<b>29</b>
<b>7</b>	<b>Fazit und Ausblick</b>	<b>32</b>
<b>Abbildungsverzeichnis</b>		I
<b>Tabellenverzeichnis</b>		I
<b>Literatur- und Quellenverzeichnis</b>		II

# 1 Einleitung

Rad fahren ist gesund, gut für die Umwelt und spart Zeit und Kosten.[1] Dabei gibt es oft nur ein Problem. Unsere Städte sind weitestgehend für das Auto gebaut und meist Radfahrer:innen unfreundlich. Der Alltag im Straßenverkehr ist daher geplagt durch schlechte Luftqualität sowie nicht adäquat ausgebauten Fahrradwegen, welche immer wieder lebensgefährliche Situationen verursachen.[2] Faktoren, wie Schnee und Regen oder bei Nacht schlecht beleuchtete Radwege sind ebenfalls Risikofaktoren. Mithilfe der Bikelin Navigator Webanwendung sollen diese Risikofaktoren minimiert (oder reduziert) werden. Durch die Einbeziehung von Wetterdaten sowie Unfallstatistiken unterstützt die Anwendung bei der Planung und Vorbereitung von Fahrten mit dem Rad.

Um diese Anwendung von unterwegs sowie während der Fahrt nutzen zu können, beschäftigt sich diese Bachelorarbeit mit dem Design und der Entwicklung einer passenden Smartphone-App. Eine Welt ohne Smartphones und Apps scheint heutzutage schon fast undenkbar. Laut einer Prognose von Statista gibt es im Jahr 2023 weltweit ca. 6,92 Milliarden Smartphone Benutzer:innen. Dies entspricht ungefähr 86,41% der aktuellen Weltbevölkerung.[3][4] Durch die stets steigende Relevanz von Smartphones in unserem Alltag wächst gleichzeitig die Nachfrage an Software für mobile Endgeräte. Allein 2022 wurden rund 234 Milliarden Apps heruntergeladen und ca. 385 Milliarden Euro Umsatz durch Anwendungen auf dem Handy gemacht.[5][6]

Gleichzeitig kann die Entwicklung einer App bei schlechter Planung sehr viel Zeit und Geld kosten. Entsprechend wichtig ist die vorgelagerte Wahl eines Techstacks, welcher an die Anforderungen der Anwendung angepasst ist.[7] Dementsprechend, besteht die Bachelorarbeit aus verschiedenen Phasen und Iterationen von Design und Entwicklung. Zunächst werden in den Grundlagen wesentliche Technologien erklärt und verglichen. Anschließend werden in der Analyse und Konzeption die Anforderungen definiert und Vorgehensweisen gewählt. Nach dem theoretischen Teil wird die App implementiert und dokumentiert, um am Ende der Arbeit eine erste funktionierende Version zu erhalten, welche fortlaufend weiter entwickelt werden kann.

# **2 Grundlagen**

Das folgende Kapitel beschäftigt sich mit der Definition grundlegender Technologien und Werkzeugen für die Entwicklung der Bikelin Navigator App. Dabei werden verwandte Technologien mit einander verglichen und anschließend abgewogen, welche sich für die Anwendung am besten eignen. Das Resultat dieses Abschnitts soll ein geeigneter Techstack als Grundlage für die folgende Entwicklung sein.

## **2.1 Native App Entwicklung**

Native Apps werden speziell für ein bestimmtes Betriebssystem entwickelt und in der Regel mit einer spezifischen Programmiersprache, die für das jeweilige Betriebssystem optimiert ist, geschrieben. Die beiden meist verbreitetsten Betriebssysteme für Smartphones sind Android von Google und iOS von Apple und haben zusammen einen Marktanteil von ungefähr 99 %.<sup>[8]</sup> Eine native App sollte, um den Großteil des Marktes abzudecken, für diese beiden Betriebssysteme entwickelt werden.

Native Applikationen können sehr einfach mit der Hardware oder vorinstallierter Software des Endgerätes interagieren. Dies ermöglicht einen direkten Zugriff auf die Kamera, das GPS oder weitere Sensoren.<sup>[8]</sup> Die Benutzung von eingebauten Frameworks, wie zum Beispiel von Apple das Vision-Framework ist durch die Native Entwicklung ebenso leicht zugänglich.

## **2.2 Cross-Plattform App Entwicklung**

Die Cross-Plattform App Entwicklung bezeichnet die Erstellung von Anwendungen, die auf mehreren Plattformen, wie zum Beispiel iOS, Android und Windows lauffähig sind. Sie werden in der Regel mithilfe von Tools und Frameworks entwickelt, die es ermöglichen, den Code auf verschiedenen Plattformen zu übersetzen. Dies bietet den Vorteil, dass anders als bei der Nativen Entwicklung der Code nur einmal geschrieben werden muss. Beispiele für bekannte Cross-Plattform Frameworks sind React Native, Flutter und Xamarin.<sup>[9]</sup>

## 2.3 Gegenüberstellung

### 2.3.1 Performance

Native Apps werden direkt auf dem Betriebssystem ausgeführt und können damit die Fähigkeiten und Funktionen der Plattform vollständig nutzen. Sie müssen keine Extra-Schritte ausführen, um die Benutzer:innenoberfläche darzustellen und können direkt auf die native Oberfläche des jeweiligen Betriebssystems zugreifen. Dies trägt dazu bei, dass Native Apps schnellere Ladezeiten haben.[10] Des Weiteren kann die Kommunikation zwischen nativen und nicht nativen Komponenten bei den plattformübergreifenden Anwendungen zu Leistungseinbrüchen führen.[11]

Zwar gibt es bei der Cross-Plattform Entwicklung Tools oder Frameworks wie zum Beispiel Flutter oder React Native, die darauf ausgelegt sind, die Performanz von nativen Apps so weit wie möglich zu replizieren. Allerdings bleiben sie in Bezug auf die Leistung hinter den nativen Apps und stehen daher an zweiter Stelle.[12]

Wichtig zu beachten ist, dass die Leistung von Apps auch von anderen Faktoren abhängt. Zum Beispiel die Qualität des Codes, der verwendeten Hardware oder die Art der ausgeführten Aufgaben. Eine sorgfältige Planung und Optimierung des Codes trägt dazu bei, die Leistung von Nativ-Apps und Cross-Plattform Apps zu verbessern.[13]

### 2.3.2 Entwicklungszeit & Kosten

In der Regel ist die Entwicklungszeit einer App durch Cross-Plattform Frameworks deutlich schneller, als die einer nativen App. Ursache hierfür ist, dass bei der Cross-Plattform Entwicklung lediglich eine Codebase geschrieben wird und nicht wie beim nativen Entwickeln für jedes Betriebssystem eine separate.[12] Dies ist zugleich häufig ein Grund dafür, dass die Entwicklung von plattformübergreifenden Apps kostengünstiger ist. Aus wachsender Entwicklungszeit resultieren auch steigende Kosten.[14] Des Weiteren könnte es sinnvoll sein, bei der Entwicklung einer nativen App, separate Entwickler:innenteams für jede Plattform zu beschäftigen, um die Entwicklung zu beschleunigen und durch gezieltere Expertisen, die Qualität der App zu verbessern. Jedoch hat der erhöhte Personalaufwand höhere Entwicklungskosten zur Folge.[15]

### 2.3.3 Sicherheit

Neben den grundlegenden Faktoren, wie die Qualität des Codes oder die implementierten Sicherheitsmaßnahmen, sowie eine sorgfältige Wahl von verwendeten Bibliotheken und einem sicheren Backend, spielt auch die Wahl des Frameworks eine Rolle bei der

Sicherheit von Apps.[16] Nativ entwickelte Apps gelten meist als sicherer, da sie direkten Zugriff auf die integrierten Sicherheitsfunktionen haben. Cross-Plattform Frameworks hingegen müssen öfter für sonst native Funktionen auf Drittanbieter Bibliotheken zugreifen, was eine größere Angriffsfläche bietet. Jedoch besitzen auch Cross-Plattform Frameworks ernstzunehmende Sicherheitsvorkehrungen.[12] Infolgedessen sind plattformübergreifende Anwendungen in der Regel nicht weniger sicher als native Anwendungen. Jedoch ist ein höherer Aufwand von Nöten, um den gleichen Sicherheitsgrad von nativen Apps zu erreichen.[16]

### **2.3.4 Testbarkeit & Wartbarkeit**

Wie jede Anwendung sollte eine App ausführlich getestet und regelmäßig gewartet werden. Vor allem die Testbarkeit einer App geht Hand in Hand mit ihrer Sicherheit sowie der Benutzer:innenerfahrung und ist dementsprechend ein nicht zu vernachlässigender Faktor.[17]

Die Testbarkeit von Cross-Plattform Apps und nativen Apps ist von den verwendeten Werkzeugen und Technologie abhängig. Cross-Plattform Apps haben den Vorteil, dass sie auf mehreren Plattformen laufen und daher nur einmal getestet werden müssen. Somit kann die Testzeit verringert und die Entwicklung beschleunigt werden.[14] Allerdings können Cross-Plattform Apps in einigen Fällen weniger gut mit der Plattform kompatibel sein und mit den spezifischen Funktionen und Tools der einzelnen Plattformen umgehen, was in der Regel einen erhöhten Testaufwand zur Folge hat.[18] Dennoch ist die Cross-Plattform Entwicklung hier deutlich effizienter und damit im Vorteil.[14].

Bei der Wartung von Apps sollten besonders zwei verschiedene Aspekte beachtet werden. Zum einen erfordert die Entwicklung nativer Apps im Allgemeinen, aufgrund der verschiedenen Codebasen, mehr Ressourcen für die Wartung als bei einem Cross-Plattform Framework. Zum anderen ermöglicht ein nativer Ansatz die sofortige Unterstützung für neue Betriebssystemversionen. Das bedeutet, dass es für Entwickler:innen einfacher ist, neue Versionen der App zu erstellen und bestehende Apps auf eine neue Version des Betriebssystems zu migrieren, ohne dass sie große Änderungen am Code vornehmen müssen.[12]

### **2.3.5 Benutzer:innenerfahrung & Individuelle Anpassungen**

Sowohl bei der Benutzer:innenerfahrung als auch bei individuellen Anpassungen, hat die native App Entwicklung einen natürlichen Vorteil gegenüber der Cross-Plattform Entwicklung, da sie spezifisch für ein Betriebssystem entwickelt wurde und somit in der Lage

ist, direkt auf die nativen UI (User Interface) Elemente des Betriebssystems zuzugreifen. Dies kann dazu beitragen, dass die Benutzer:innenoberfläche intuitiver ist und die Interaktionen für die Benutzer:innen reibungsloser verlaufen.[19] Dasselbe gilt für individuelle Anpassungen der jeweiligen Plattform. Native Apps sind besser in der Lage, die Hardwarefunktionen eines Geräts wie die Kamera, GPS und den Touchscreen zu nutzen. Zwar haben Frameworks wie zum Beispiel React Native ebenfalls Schnittstellen, um auf diese Funktionen zuzugreifen, sie sind jedoch meist mit mehr Konfigurationsaufwand verbunden.[14]

Es ist jedoch wichtig zu beachten, dass die Benutzer:innenerfahrung nicht nur von der Wahl der Entwicklungsmethode abhängt. Es gibt diverse weitere Faktoren, wie beispielsweise die Zugänglichkeit, Funktionalität oder Leistung der App.[20] Die Wahl der richtigen Entwicklungsmethode ist daher nur ein Teil bei der Gestaltung einer benutzer:innenfreundlichen Anwendung.

### **2.3.6 Zusammenfassung**

Ein bedeutender Vorteil der Cross-Plattform Entwicklung besteht darin, dass man mit nur einer einzigen Anwendung auf mehreren Betriebssystemen arbeiten kann, ohne jedes Mal eine separate App programmieren zu müssen. Stattdessen wird die Anwendung lediglich für die jeweilige Plattform übersetzt. Das spart Zeit und Geld, was für viele Unternehmen oder Entwickler:innen ein ausschlaggebendes Argument bei der Wahl der Entwicklungsmethode sein kann. Ein weiterer Punkt ist, dass führende plattformübergreifende Frameworks, wie React Native oder Flutter viele der genannten Schwächen bereits weitestgehend beheben. In Fällen, in denen die Entwicklungsgeschwindigkeit eine wichtige Rolle spielt, empfiehlt sich somit die Wahl von einem der Frameworks der Cross-Plattform Entwicklung.[21]

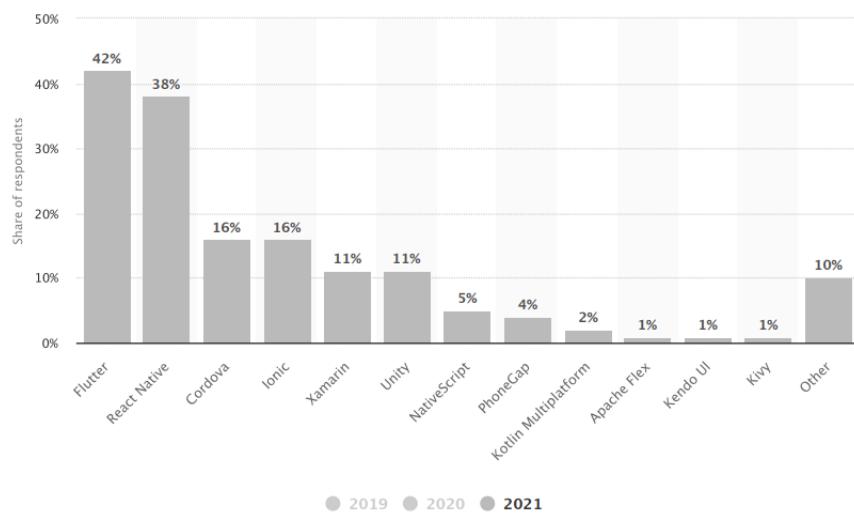
Wenn jedoch Zeit und Geld eine untergeordnete Rolle spielt und besonders hoher Wert auf die Individualisierung der App auf dem jeweiligen Betriebssystem oder die Schnelligkeit der Anwendung gelegt wird, empfiehlt es sich, die App auf jedem Betriebssystem nativ zu entwickeln.[15]

Die Wahl der Entwicklung bei der Anwendung dieser Arbeit fällt allerdings sehr leicht aus. Die mit Abstand ausschlaggebendsten Aspekte sind die Zeit und Kosten. Da dieses Projekt weder über viel Zeit, Budget noch ein großes Entwickler:innenteam verfügt, wäre die Entwicklung von zwei nativen Apps für iOS und Android schwer umzusetzen. Performance und Sicherheit sollten zwar nie vernachlässigt werden, benötigen bei dieser Anwendung jedoch keine besonderen Fokus. Ebenso die Benutzer:innenfreundlichkeit sollte mit guter Planung und unter Verwendung von Gestaltungsrichtlinien ohne die native Entwicklung angemessen umsetzbar sein. Dementsprechend empfiehlt sich auf eines

der vielen Cross-Plattform Frameworks für die Entwicklung der Bikelin Navigator App zurückzugreifen.

## 2.4 Cross-Plattform Frameworks

Es gibt viele verschiedene Cross-Plattform Frameworks. Laut der Statistik aus Abbildung 2.1 sind Flutter und React Native, die mit Abstand am meisten benutzen. Folglich wird sich in diesem Kapitel auf eine Gegenüberstellung dieser beiden Frameworks beschränkt.[22]



**Details:** Worldwide; 2019 to 2021; 31,743 respondents; software developers

Abbildung 2.1: Von Softwareentwickler:innen weltweit verwendete plattformübergreifende mobile Frameworks in 2021 [22]

### 2.4.1 React Native

React Native ist eine Javascript Bibliothek, die von Meta entwickelt wurde und es ermöglicht, native mobile Anwendungen für iOS und Android zu erstellen. Im Gegensatz zu mobilen Anwendungen, die in einer spezifischen Programmiersprache wie Swift, Java oder Kotlin geschrieben werden, basiert React Native auf dem Webframework React. Dadurch können dieselben Kenntnisse und Fähigkeiten der Webentwicklung, auch für die Entwicklung von mobilen Anwendungen genutzt werden. Ähnlich wie bei React, werden React Native Anwendungen mit einer Mischung aus JavaScript und JSX geschrieben.[23]

Damit der geschriebene Code auf die jeweilige Plattform übersetzt werden kann, sind folgende Schritte nötig. Beim Starten der App wird von React Native ein nativer Haupt-

prozess erstellt, welcher dann eine Javascript virtuelle Maschine in einem weiteren Prozess startet. Beide Prozesse kommunizieren über die sogenannte *Bridge*. Der Javascript Prozess bestimmt, was und an welcher Position, etwas angezeigt werden soll. Diese Informationen werden in Form einer JSON Datei über die *Bridge* an den nativen Hauptprozess übermittelt. Der Hauptprozess hingegen ist dafür zuständig jede Aktion, die Benutzer:innen auf der Benutzer:innenoberfläche tätigen, wahrzunehmen und auszuführen und sie dann ebenfalls über die *Bridge* an den Javascript Prozess zu schicken, um sie dort zu verarbeiten.[24]

## 2.4.2 Flutter

Flutter wurde von Google entwickelt und wird mit der Programmiersprache Dart geschrieben. Es basiert auf sogenannten *Widgets*. Sie werden verwendet, um sowohl die visuellen Komponenten einer App, wie beispielsweise Knöpfe oder Texte, als auch die funktionalen Elemente zu erstellen.[25] Flutter nutzt die sogenannte *Skia Engine*. Die *Skia Engine* ist eine Grafikbibliothek, welche den Dart-Code in nativen Code für die jeweilige Plattform übersetzt und die Benutzer:innenoberfläche auf dem Zielgerät rendert.[26][27] Auf diese Weise kann Flutter auf Android und iOS Geräten native Apps erstellen.[28]

## 2.4.3 Gegenüberstellung

Sowohl Flutter als auch React Native sind hervorragende Frameworks für die Entwicklung plattformübergreifender mobiler Anwendungen. Beide besitzen eine hohe Code Wiederverwendbarkeit, sowie eine Hot-Reload-Funktion, welche es den Entwickler:innen ermöglicht Änderungen am Code direkt umgesetzt zusehen.[21] Obwohl sie zahlreiche Gemeinsamkeiten aufweisen, existieren einige signifikante Unterschiede. Einer der offensichtlichsten liegt in der Programmiersprache. React Native verwendet Javascript, Flutter dagegen Dart.[29] Da Dart eine höhere Kompilierungsrate als Javascript besitzt, bietet sie von Natur aus eine höhere Leistung. Allerdings wird Dart laut Statista von nur 6,54 % aller Programmierer:innen verwendet.[30] Javascript mit ca. 65 % auf Platz 1 hat hier einen klaren Vorteil und bietet einen guten Einstieg in React Native für Entwickler:innen.[31] React Native ist älter und hat eine größere Community, die bereits eine große Menge an fertigen Paketen und Bibliotheken entwickelt hat. Der Vorteil von Flutter hingegen ist, dass es von Anfang an umfassend von Google Ingenieur:innen entwickelt wurde. Es verfügt über eine viel bessere Dokumentation und ist weniger von Drittanbieter Bibliotheken abhängig als React Native.[21]

Als Cross-Plattform Framework für die Bikelin Navigator App würden sich sowohl React Native als auch Flutter anbieten. Die jedoch deutlich größere Beliebtheit von Javascript

gegenüber Dart, ermöglicht zukünftigen Entwickler:innen einen leichteren Einstieg in die Weiterentwicklung der App, weshalb die Entscheidung für das Framework unserer App auf React Native fällt.

## 2.5 Expo

*Expo* ist ein Open-Source-Framework für die Entwicklung von React Native-Apps. Es ermöglicht viele wichtige Funktionen für die Erstellung und Skalierung von Apps wie Live-Updates, sofortiges Teilen der App und Web Unterstützung. Das *Expo* npm-Paket bringt diese Funktionen in jedes React Native Projekt.[32]

# 3 Analyse

Die folgende Analyse dient zur Definition des Ist-Zustandes der aktuellen Bikelin Navigator Web Anwendung. Dabei wird untersucht, welche funktionalen Anforderungen und Eigenschaften erfüllt werden und welchen Zweck diese haben. Aus den daraus resultierenden Ergebnissen wird anschließend der Soll-Zustand des Praxisteils dieser Bachelorarbeit formuliert. Es wird versucht eine möglichst sinnvolle Erweiterung zur Webanwendung zu finden, welche nicht ausschließlich bestehende Features übernimmt, sondern zugleich einen Mehrwert für das Projekt darstellt.

## 3.1 Ist-Zustand

Der Bikelin Navigator ist eine Webanwendung und ein Forschungsprojekt von Prof. Dr. Schüler an der HTW Berlin. Die Anwendung dient der Planung und Erstellung von Radtouren und Radrouten in und um Berlin und soll durch eine Kombination von verschiedenen Microservices den Alltag mit dem Rad sicherer und praktikabler machen.

Das Backend bündelt diese Services in Form einer Microservice-Architektur. Dies ist ein Architekturstil zur Softwareentwicklung, bei dem große Anwendungen in kleinere, unabhängige Elemente (Microservice) mit einem eigenen Zuständigkeitsbereich aufgeteilt werden.[33] Beispiele von Microservicen des Bikelin Navigators sind, das Sammeln von Niederschlagsdaten in Berlin, die Berechnung der besten Radrouten von Punkt A zu Punkt B sowie der Open-Source Authentifizierungsdienst Keycloak.[34]

Ein weiterer essenzieller Teil der Anwendung ist das Web Frontend. Geschrieben wurde das Frontend mit dem Web Framework Angular. Angular wurde von Google entwickelt und basiert auf der Programmiersprache TypeScript. Es ermöglicht eine Entwicklung auf Basis von Komponenten und bietet eine Vielzahl von Werkzeugen und Funktionen, die es Entwickler:innen erleichtert, die Anwendungsarchitektur zu gestalten, Daten zu verwalten, Routing zu implementieren und Anwendungen für unterschiedliche Plattformen anzupassen.[35] Das Frontend des Bikelin Navigator visualisiert die gesammelten Daten der Mikrodienste aus dem Backend und ermöglicht es den Benutzer:innen mit der Anwendung zu interagieren. Sie können sich dort von verschiedenen Routenplaner-Anbietern

mögliche Fahrradstrecken zwischen zwei beliebigen Punkten anzeigen und unter ihnen wählen. Anschließend werden weitere Daten für die gewählte Strecke zur Verfügung gestellt. Unter Anderem der geschätzte Niederschlag inklusive Regenradar, die Luftqualität oder besondere Informationen auf der Route. Des Weiteren kann die berechnete Route gespeichert und zu einem späteren Zeitpunkt wieder aufgerufen werden.

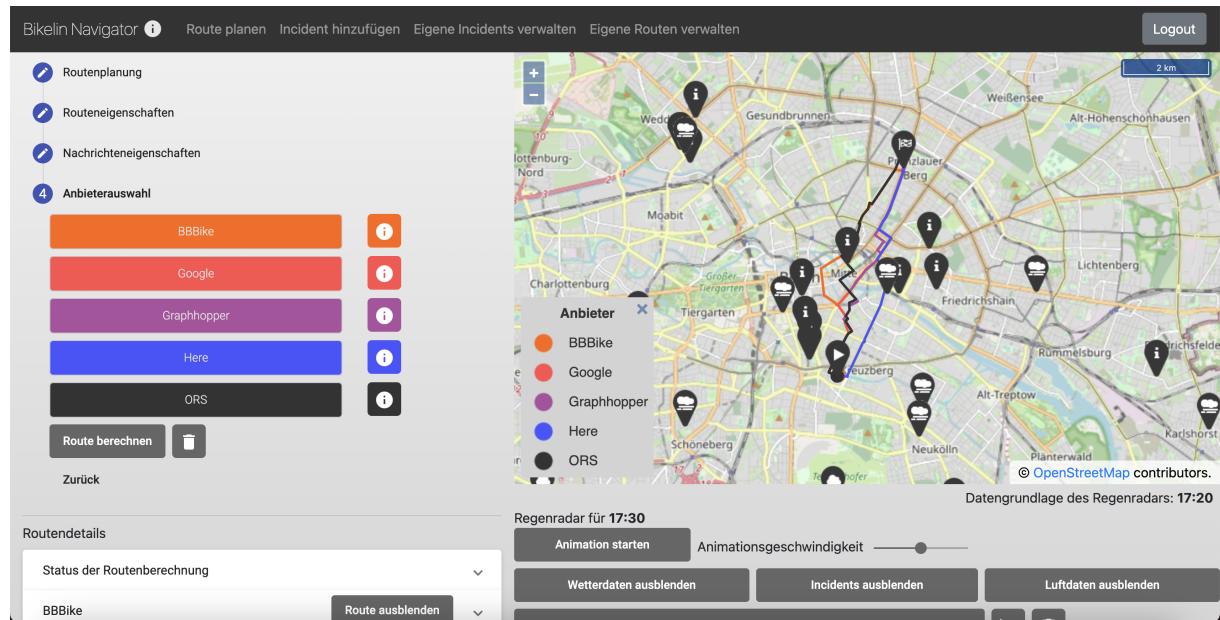


Abbildung 3.1: Benutzer:innenoberfläche der Web Anwendung Bikelin Navigator

Zusammenfassend ist die Bikelin Navigator Anwendung ein optimales Werkzeug zum Berechnen und Planen von Fahrten mit dem Rad. Die Stärke der Anwendung ist die Vielzahl an angebotenen Informationen, welche für die Routenplanung vorliegen. Somit wird den Benutzer:innen vor Antritt der Fahrt eine ideale Vorbereitung ermöglicht.

## 3.2 Soll-Zustand

Um einen sinnvollen Soll-Zustand zu definieren, muss zunächst die Frage beantwortet werden, welche Vorteile eine App gegenüber der webbasierten Anwendung hat. Eines der Vorteile ist, dass die bestehende Webanwendung zwar alle Mittel für die Planung einer Fahrt mit dem Fahrrad zur Verfügung stellt, jedoch den Nutzer:innen bisher keine Unterstützung beim tatsächlichen Absolvieren der Strecke bietet. Die Webanwendung kann während der Fahrt, im Browser des mobilen Endgerätes aufgerufen werden, jedoch ist dafür, wie in Abbildung 3.2 zu sehen, weder die Benutzer:innenoberfläche geeignet, noch eine live Navigationshilfe implementiert. Dies bedeutet, dass die Nutzer:innen unterwegs keinen geeigneten Zugriff auf die vorher zusammen gestellten Informationen haben. Eine App eignet sich, um dieses Problem zu lösen und ist somit eine sinnvolle Erweiterung für

den Bikelin Navigator.

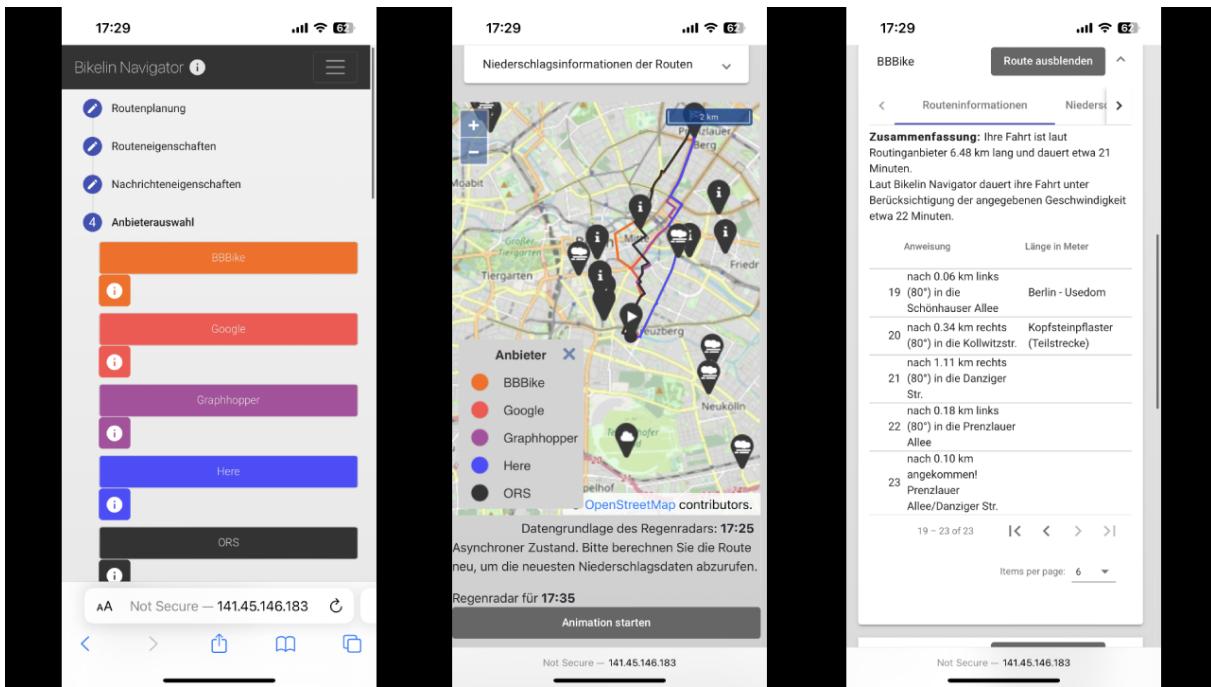


Abbildung 3.2: Benutzer:innenoberfläche der Web Anwendung Bikelin Navigator auf dem Smartphone

Die App soll den Benutzer:innen ermöglichen, von Unterwegs Informationen über Routen von dem Handy abzurufen. Besonders hilfreich ist eine live Navigationshilfe. Diese soll den Nutzer:innen während des Fahrradfahrens anzeigen, wo sie sich momentan auf der Karte befinden, wie die Route weiter verläuft und was die nächste Navigationsanweisung ist. Des Weiteren sollte die App eine übersichtliche Benutzer:innenoberfläche besitzen. Die Nutzer:innen sollten während der Fahrt so wenig wie möglich mit der App interagieren müssen. Wesentliche Informationen für die Navigation sollten zum richtigen Zeitpunkt auf dem Bildschirm ohne zusätzliche Interaktion angezeigt werden. Des Weiteren sollten angelegte Routen gespeichert werden können, um sie zu einem späteren Zeitpunkt wiederzuverwenden. Die folgenden Kapitel dienen zum Finden und Definieren der funktionalen und nicht funktionalen Anforderungen.

### 3.3 Marktanalyse

Bei einer Marktanalyse werden wichtige Merkmale eines potenziellen Marktes beobachtet. Ihr Ziel kann unter anderem, die Erforschung von Zielgruppen oder die Vervollständigung eines Businessplans sein.[36] Der Fokus dieser Analyse liegt auf das Identifizieren von sinnvollen Anforderungen bestehender Anwendungen, welche sich für die Bikelin Navigator App eignen. Des Weiteren wird sich auf zwei wesentliche Prozesse beschränkt.

- **Anlegen einer Route:** Dieser Prozess beschreibt den Vorgang, in dem Benutzer:innen eine neue Route anlegen. Wesentliche Aktionen sind die Angaben von Start und End Position. Darüber hinaus wird die Darstellung der angelegten Route mit inbegriffen.
- **Live Navigationshilfe:** Dieser Prozess beschreibt alle Vorgänge, die während der Fahrt stattfinden und die Wegfindung erleichtern.

Die Webanwendung des Bikelin Navigators benutzt für die Berechnung der Routen Google Maps, BBBike, ORS, Graphhopper und Here. Zwei dieser fünf Anbieter bieten ebenfalls eine App an und werden für die Marktanalyse verwendet. Zusätzlich wird Komoot, eine renommierte Navigations-App speziell für Wanderer:innen, Radfahrer:innen und Mountainbiker:innen, in die Analyse mit einbezogen.

### 3.3.1 Google Maps

#### Anlegen einer Route

Der Vorgang vom Anlegen einer Route inklusive ihrer Darstellung mit Google Maps wird in Abbildung 3.3 durch Screenshots festgehalten. Der erste Bildschirm zeigt den Startzustand der App. Neben vielen weiteren Optionen und Funktionen ist im oberen Abschnitt ein großes Eingabefeld, mit der Beschriftung „Suche hier“. Durch Klicken des Eingabefelds gelangt man zu Bildschirm zwei. Dort lässt sich ein beliebiger Ort in die Suchleiste eingegeben, oder zwischen gespeicherten Orten, letzten Eingaben und weiteren Vorschlägen wählen. Anschließend gelangt man zu Bildschirm drei. Dargestellt wird ein Ausschnitt der Karte mit dem gewählten Ort, einer Übersicht mit Informationen und mehrere Knöpfe mit verschiedenen Optionen. Es kann entweder durch Drücken auf „Start“ direkt die Navigation beginnend von der eigenen Position gestartet werden oder mit dem Knopf „Wegbeschreibung“ zum Bildschirm vier navigiert werden. Dort ist eine Übersicht der Route zusehen. Als Start ist automatisch die aktuelle Position der Benutzer:innen eingestellt, kann aber durch einen Klick auf die Suchleiste angepasst werden. Das Ziel der Route ist der Ort, welcher in den vorherigen Bildschirmen gewählt wurde. Darunter ist erneut ein Kartenausschnitt zu sehen. Dieser zeigt die Route zwischen Start und End Punkt. Der untere Teil des Bildschirms zeigt eine ausfahrbare Fläche, welche im letzten Bildschirm zusehen ist. Sie zeigt Informationen über die Strecke und einen Knopf zum Starten der Navigation. Unter anderem werden Daten über die Beschaffenheit der Straßen, die einzelnen Navigationsschritte der Route sowie Steigung und Gefälle angezeigt.

Die Analyse ergibt wertvolle Erkenntnisse. Es fällt auf, dass der Vorgang mit Google Maps relativ viele Schritte beinhaltet. Zum Beispiel werden beim Anlegen einer Route mit einer

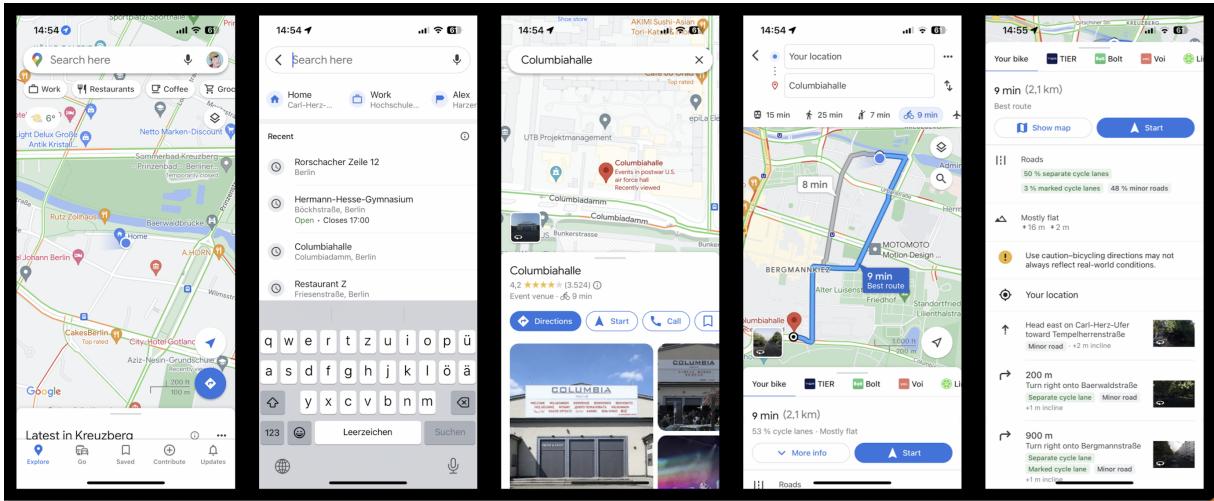


Abbildung 3.3: Benutzer:innenoberfläche von Google Maps beim Anlegen einer Route

Startposition, welche nicht der eigene Standort ist, insgesamt sechs Bildschirme verwendet. Dies lässt sich darauf zurückführen, dass Google Maps keine reine Navigationsapp ist, sondern auch viele weitere Funktionen beinhaltet.[37] Für die Bikelin Navigator App kann und sollte dieser Prozess deutlich simpler und übersichtlicher gestaltet werden. Das Darstellen einer Route (Bildschirm vier und fünf) ist jedoch sehr übersichtlich und bietet viele nützliche Informationen über die angelegte Route. Die Bildschirme werden als Vorlage für die Anforderung der Darstellung einer Route dienen. Eine optionale Anforderung für die Bikelin Navigator App ist, die Möglichkeit der Verwendung des eigenen Standorts bei der Startpunkt Eingabe. Dies kann beim Anlegen der Route Zeit sparen, wenn vom aktuellen Standort aus gestartet werden möchte. Des Weiteren gibt es die Möglichkeit, dass die Benutzer:innen die Adresse ihres aktuellen Standortes nicht kennen und somit trotzdem von dort aus starten können. Jedoch soll den Benutzer:innen hier die Option gelassen werden, ob sie selbst eine Startadresse angeben oder ihren aktuellen Standort nutzen möchten.

## Live Navigationshilfe

Die *Live Navigationshilfe* von Google Maps wird in Abbildung 3.4 durch drei verschiedenen Screenshots zusammengefasst. Zu Beginn der Fahrt zoomt und fokussiert die Karte automatisch auf den Standort der Benutzer:innen. Sobald dieser Standort sich ändert, bewegt sich die Kamera mit. Oben ist eine Leiste mit der aktuellen Navigationsanweisung. Unten im Bildschirm ist eine ausfahrbare Fläche, welche die verbleibende Dauer, Entfernung und geschätzte Ankunftszeit anzeigt. Beim Ausfahren wird eine Liste mit allen Navigationsweisung dargestellt. Des Weiteren kann der Kartenausschnitt durch Zoomen oder Scrollen beliebig angepasst werden.

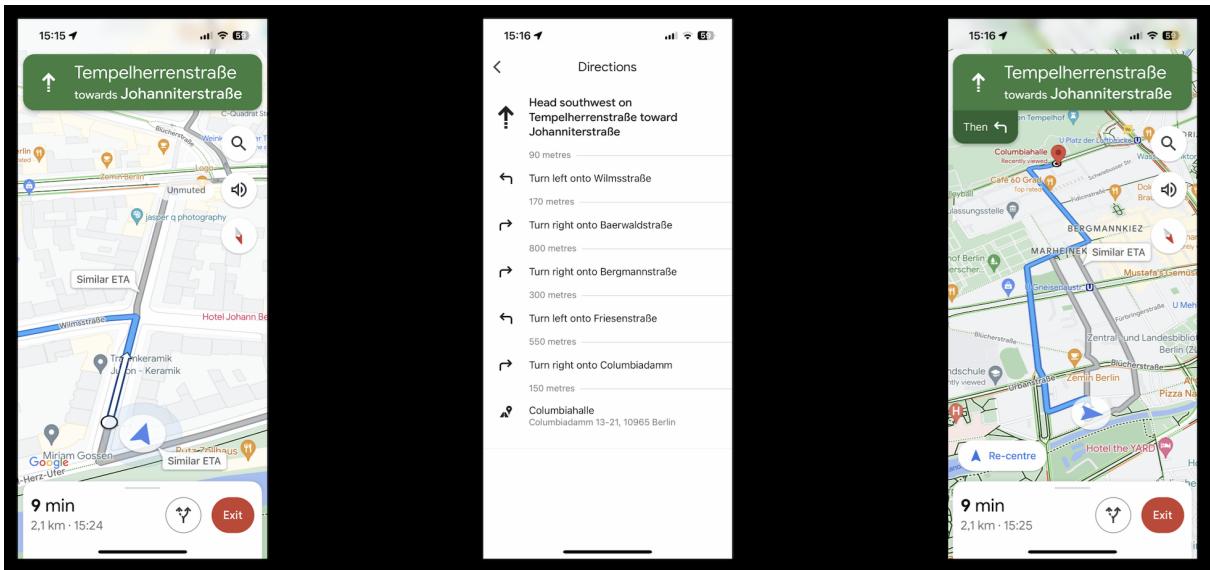


Abbildung 3.4: Benutzer:innenoberfläche von Google Maps während der Fahrt

Besonders nützlich und hilfreich ist die automatische Fokussierung auf den Standort der Nutzer:innen und die Anzeige mit der nächsten Weganweisung. Durch die Anwendung dieser Technologien wird es den Benutzer:innen ermöglicht, ihre Hände während der Fahrt freizuhalten, indem die Notwendigkeit einer ständigen Justierung der Karte vermieden wird. Beide Funktionen sind sinnvolle Anforderungen für die Bikelin Navigator App.

### 3.3.2 Komoot

#### Anlegen einer Route

Das Anlegen einer Route bei Komoot wird in Abbildung 3.5 durch fünf verschiedenen Screenshots zusammengefasst. Die App besitzt eine untere Navigationsleiste. Um zum ersten Bildschirm zu gelangen, muss dort „Plan“ ausgewählt werden. Dieser Screen ähnelt dem Startbildschirm von Google Maps und bietet keine neuen Erkenntnisse. Durch Tippen auf „Neue Route planen“ erfolgt die Weiterleitung zu Bildschirm zwei. Hier können neben anderen Einstellungen Start und Zielpunkt angegeben werden. Bildschirm drei, vier und fünf beinhalten eine Übersicht der Route. Zuerst ist wie schon bei Google ein Kartenausschnitt mit der Route zusehen. Unterhalb dieser befindet sich eine Leiste mit weiteren Informationen und einem Button zum Starten der Navigation. Durch Tippen auf die Leiste öffnet sich ein Bildschirm, der auf dem Layout von Screen zwei basiert. Zu sehen sind weitere, ausführliche Informationen über die gesamte Strecke. Unter anderem Straßenarten, Schwierigkeitslevel und Bodenprofil.

In diesem Beispiel ist besonders Bildschirm zwei relevant für die Bikelin Navigator App. Es ist ein einfacher Weg, das Anlegen der Route auf einen Bildschirm zu beschränken und

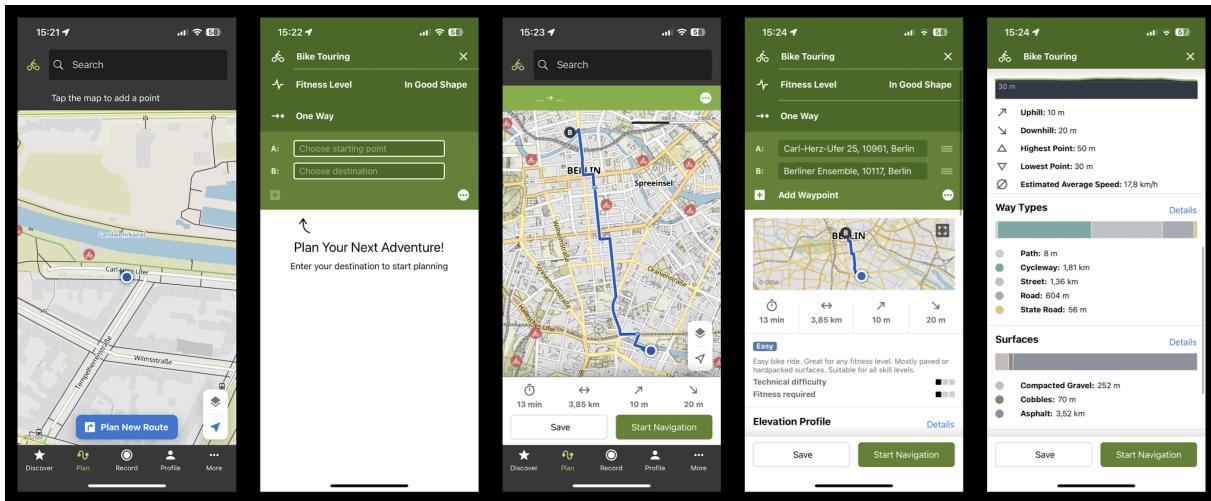


Abbildung 3.5: Benutzer:innenoberfläche von Komoot beim Anlegen einer Route

wird als Vorlage für diese Anforderung dienen. Ebenso die ausführlichen Informationen über die Route sind nützlich. Jedoch ist es eher unwahrscheinlich, die Vielfalt und Menge der Daten in der gegebenen Zeit auch in der App zu implementieren.

### Live Navigationshilfe

Die Live Navigationshilfe von Komoot und Google ähneln sich sehr. Abbildung 3.6 beinhaltet die 3 relevanten Bildschirme von Komoot für dieses Feature. Beim Starten der Navigation wird die Kamera auf den Standort der Benutzer:innen fixiert und verfolgt diesen von da an. Oben im Bildschirm wird die aktuelle Weganweisung angezeigt, sowie weitere Informationen über die Route. Durch Zoomen oder Scrollen kann der Kartenausschnitt beliebig angepasst werden.

Obwohl die Analyse von Komoot bei der Navigationshilfe keine Erkenntnisse über neue Anforderung gebracht hat, konnten durch sie die gesammelten Anforderungen der Analyse von Google Maps nochmal bestätigt werden. Wichtig ist es, dass der Kartenausschnitt während der Fahrt immer die aktuelle Position der Nutzer:innen behält. Dazu sollte die aktuelle Weganweisung sowie generelle Informationen zu Strecke angezeigt werden.

### 3.3.3 Ergebnis

Analysen von weiteren Anwendungen wie, Apple Maps, Bikemap oder Bbybike haben zu keinen neuen Erkenntnissen geführt. Viele Apps bieten den vollen Funktionsumfang nur gegen die Bezahlung einer Gebühr. Auch sind nur wenige der Routenplaner Anbieter auf das Fahrrad als Transportmittel spezialisiert. Apple Maps zum Beispiel unterstützt zum Zeitpunkt der Analyse (Januar 2023) die Navigation mit dem Fahrrad fast ausschließlich

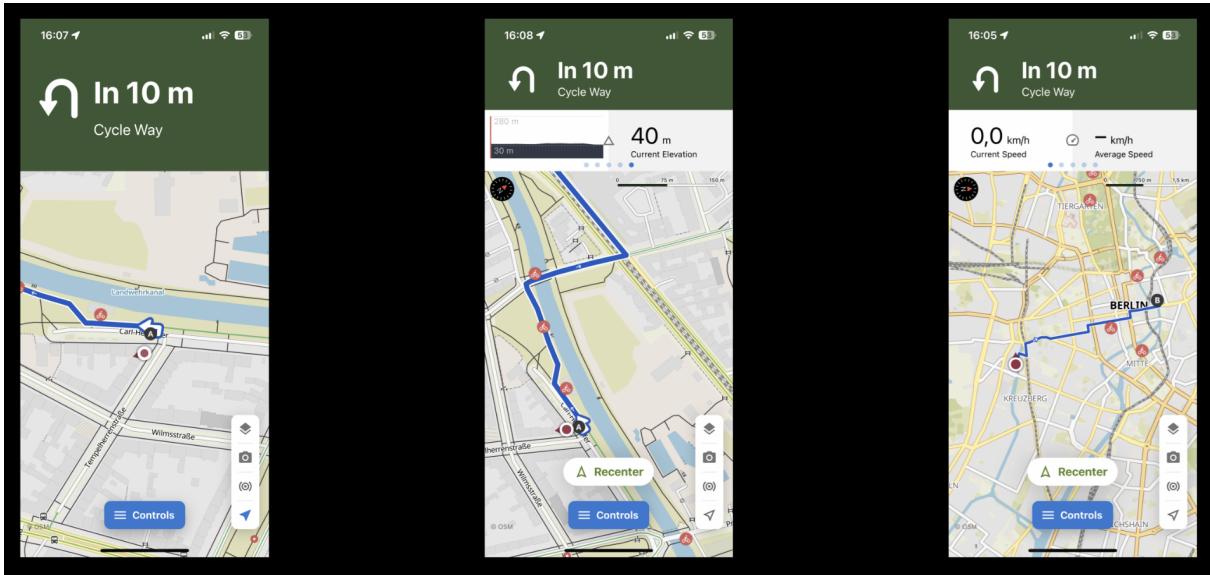


Abbildung 3.6: Benutzer:innenoberfläche von Komoot während der Fahrt

in den Vereinigten Staaten.[38]

Die Analyse hat wertvolle Erkenntnisse für die Umsetzung der Bikelin Navigator App gebracht. Bei der Navigationshilfe gab es viele einheitliche Merkmale, die als Vorlage dienen werden. Vor allem das Verhalten der Karte während der Fahrt soll essenzieller Teil der App werden. Ebenso die Navigationsanweisungen sowie in reduzierter Form, nützliche Informationen über die Strecke. Die Analyse vom Anlegen einer Route hat gezeigt, dass viele der bestehenden Anwendungen mehr als nur eine Routenplanung anbieten. Dadurch werden Prozesse, die sonst einen relativ simplen Verlauf haben, deutlich komplexer als nötig. Da die Bikelin Navigator App, zunächst jedoch nur ein klares Ziel verfolgen soll, kann der Prozess vom Anlegen und Darstellen einer Route optimiert werden. Bildschirm zwei von Abbildung 3.5 dient hier als grobe Vorlage.

### 3.4 Funktionale Anforderungen

Aus dem definierten Soll-Zustand und der Marktanalyse von anderen Routenplaner Anbietern lassen sich funktionale Anforderungen für die Bikelin Navigator App aufstellen. Laut Ian Sommerville, beschreiben funktionale Anforderungen, was das System tun sollte. Als Benutzer:innenanforderungen ausgedrückt, sollten funktionale Anforderungen in natürlicher Sprache geschrieben werden, damit sie für alle Interessengruppen verständlich sind. Des Weiteren sollten sie detailliert die Systemfunktionen, Ein- und Ausgaben sowie Ausnahmen beschreiben.[39, S.124]

Name	Beschreibung	Priorisierung
Anlegen einer Route	Die Benutzer:innen sollen eine Start- und Endposition angeben können, um eine neue Route anzulegen.	Essenziell
Darstellen einer Route	Den Benutzer:innen soll eine angelegte Route auf der Karte visualisiert werden, um sich einen Überblick über sie zu verschaffen.	Essenziell
Speichern einer Route	Die Benutzer:innen sollen eine generierte Route speichern können, um sie zu einem späteren Zeitpunkt erneut abzurufen.	Essenziell
Anzeigen von gespeicherten Routen (lokal)	Den Benutzer:innen soll in der App lokal gespeicherte Routen in einer Liste angezeigt werden.	Essenziell
Navigation zwischen Screens	Die Benutzer:innen sollen zwischen den verschiedenen Screens hin und her wechseln können.	Essenziell
Anzeigen der eigenen Position	Den Benutzer:innen soll während dem Verwenden der Karte, immer die eigene Position angezeigt werden, damit sie sich besser orientieren können.	Essenziell
Anzeigen von Navigationsanweisungen	Den Benutzer:innen sollen beim Absolvieren der Strecke Navigationsanweisung angezeigt werden.	Essenziell
Automatische Standort Verfolgung	Den Benutzer:innen soll beim Absolvieren der Strecke, die eigene Position immer als Karten Mittelpunkt angezeigt werden. Dadurch sollen unnötige Interaktionen mit der App beim Fahren mit dem Fahrrad vermieden werden.	Essenziell
Benutzer:innen Login	Die Benutzer:innen sollen sich mit ihren Login Credentials in der App anmelden können.	Bedingt Notwendig
Anzeigen von gespeicherten Routen (Account)	Den Benutzer:innen sollen die in der Webanwendung gespeicherten Routen in einer Liste angezeigt werden.	Bedingt Notwendig
Wahl des Routenplaners	Die Benutzer:innen sollen zwischen verschiedenen Routenplaner Anbietern wählen können.	Bedingt Notwendig
Darstellung der Niederschlagsdaten (Radar)	Den Benutzer:innen sollen beim Anzeigen einer Route, die Option eines Regenradars geboten werden.	Optional
Eigener Standort als Startpunkt	Die Benutzer:innen sollen beim Setzen des Startpunktes einer Route die Wahl zwischen einer individuellen Adresse und dem aktuellen eigenen Standort haben.	Optional

Tabelle 3.1: Funktionale Anforderung der mobilen App des Bikelin Navigators

## 3.5 Nicht funktionale Anforderungen

Nicht funktionalen Anforderungen spezifizieren in der Regel die Eigenschaften des Gesamtsystems. Sie sind oft relevanter als einzelne funktionale Systemanforderungen. Die Benutzer:innen des Systems finden gewöhnlich Wege, um eine Funktion zu umgehen, die ihre Bedürfnisse nicht voll und ganz erfüllt. Wenn jedoch eine nicht funktionale Anforderung nicht erfüllt wird, kann dadurch das gesamte System unbrauchbar werden.[39, S.126] Abbildung 3.7 zeigt in einem Qualitätsmodell die verschiedenen nicht funktionalen Anforderungen für ein Softwareprodukt. Bei der Entwicklung dieser Anwendung soll der Fokus besonders auf der Benutzbarkeit und Wartbarkeit liegen. Die anderen dargestellten Anforderungen sollten nicht vernachlässigt werden, erhalten jedoch keine besondere Aufmerksamkeit.

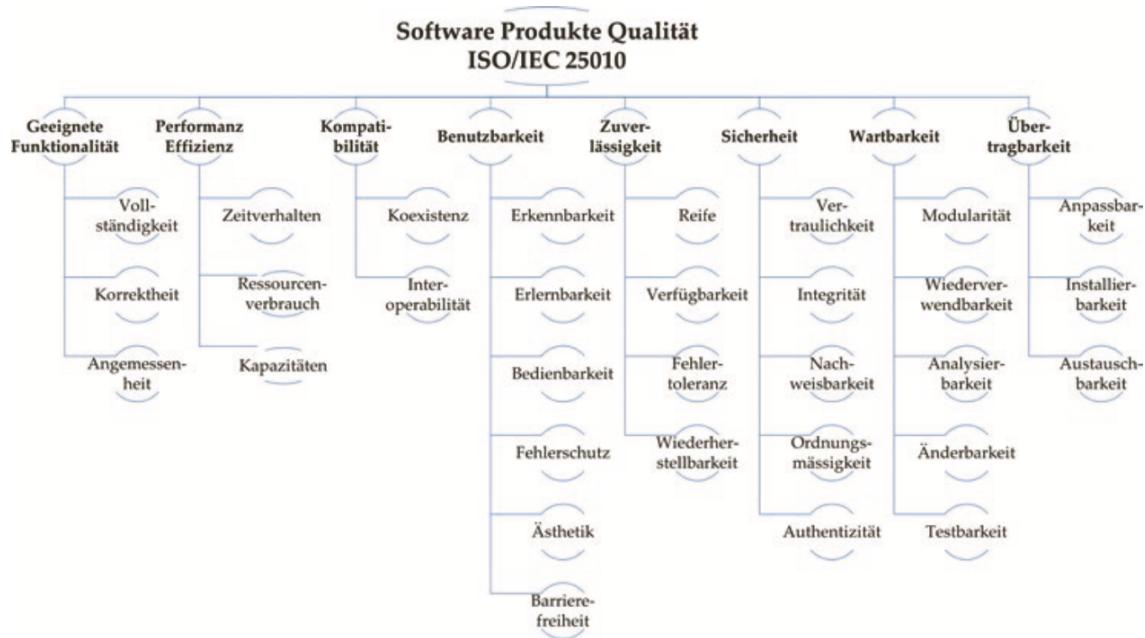


Abbildung 3.7: Qualitätsmodell für äussere und innere Qualität eines Softwareprodukts nach ISO/IEC 25010 [40, S.106]

## Benutzbarkeit

Das Qualitätsmerkmal Benutzbarkeit fokussiert sich auf die Mensch-Maschine-Schnittstelle. [40, S.108-109] Da die App hauptsächlich von unterwegs und während dem Fahren verwendet wird, ist es besonders wichtig, dass die Anwendung übersichtlich und leicht zu bedienen ist. Vor allem während der Fahrt sollte die Benutzeroberfläche so designet sein, dass möglichst wenig mit ihr interagiert werden muss, um zu verhindern, dass durch Ablenkung Unfälle provoziert werden.

## Wartbarkeit

Wartbarkeit ist ein wichtiges Qualitätsmerkmal für Softwaresysteme. Wartbarkeit wird unter anderem durch einen gut durchschaubaren modularen Aufbau, optimale Wiederverwendbarkeit, effiziente Änderbarkeit, sowie einfache Testbarkeit erreicht.[40, S.111]

Für die Bikelin Navigator App ist diese nicht funktionale Anforderung von besonderer Bedeutung, da die Bachelorarbeit das Ziel hat, die erste Version einer App zu entwickeln, welche von anderen Student:innen in weiteren Semestern und Abschlussarbeiten ausgebaut werden kann.

Zwischen den Entwickler:innen wird höchstwahrscheinlich in den seltensten Fällen eine direkte Kommunikation stattfinden. Somit sollte der Code schon im frühen Stadium mo-

dular und leicht erweiterbar sein. Dies erleichtert die Wiederaufnahme des Projektes und weitere Entwicklungsschritte in kommenden Semestern. Um einen schnellen Einstieg in das Projekt zu ermöglichen, wird auch der Dokumentation eine hohe Priorität zugesprochen.

# 4 Konzeption

## 4.1 Scrum

Um eine strukturelle und professionelle Implementierungsphase zu ermöglichen, wird sich an der Entwicklungsmethode Scrum orientiert. Mit Scrum wird das zu entwickelnde System über Produkteigenschaften definiert, die in einem *Backlog* in priorisierter Reihenfolge festgelegt sind. Die eigentliche Entwicklung des Systems ist in sogenannte Sprints unterteilt, die in der Regel zwischen 1 und 4 Wochen dauern. Innerhalb eines Sprints werden alle Anforderungen umgesetzt, die zuvor aus dem *Backlog* ausgewählt wurden.[41] Da es kein Entwickler:innenteam mit verschiedenen Rollen gibt und aufgrund der zeitlichen Limitation werden lediglich einzelne Elemente von Scrum übernommen. Die in Kapitel 3.4 definierten Anforderungen dienen hier als *Backlog*.

## 4.2 Zeitplan

Abbildung 4.1 zeigt den Zeitplan der gesamten Bachelorarbeit. Für die Implementationsphase sind insgesamt 4 Sprints, mit einer Sprintdauer von einer Woche vorgesehen.

Woche	1	2	3	4	5	6	7	8	9	10
	Recherche		Schreiben				Schreiben			
1. Recherche										
2. Schreibphase										
Grundlagen										
Analyse										
Konzeption										
Implementierung										
Fazit										
3. Implementierung				Sprint 1	Sprint 2	Sprint 3	Sprint 4			
4. Korrektur									Korrektur	

Abbildung 4.1: Zeitplan der Bachelorarbeit

## 4.3 Wireframes

Ein Wireframe ist ein Hilfsmittel für die Konzeption von Benutzer:innenoberflächen. Beim sogenannten „wireframing“ werden in technischer, reduzierter Darstellungsweise die

Elemente und Inhalte einer Anwendung geplant. Konkrete Designentscheidungen treten beim Wireframe völlig in den Hintergrund.[42] Sie können sicherstellen, dass die App den festgelegten Zielen entspricht, indem die Funktionsweise, die Platzierung und die Vorteile der Funktionen gezeigt werden. Des Weiteren können Wireframes Benutzer:innenfreundlichkeit, Navigation und Funktionsplatzierung aus einem objektiven Blickwinkel betrachten. Sie helfen dabei, Fehler in der Architektur oder in Funktionen auszubessern und zeigen wie gut der Ablauf aus der Sicht der Nutzer:innen funktioniert.[43]

Die in Abbildung 4.2 dargestellten Wireframes wurden auf Basis der Ergebnisse von Kapitel 3.4 und 3.5 entwickelt. Der erste Entwurf zeigt den Screen zum Anlegen einer Route. Er wurde so einfach und übersichtlich wie möglich gestaltet. Für Startposition und Ziel gibt es jeweils ein Eingabefeld mit Überschrift und eine Kartenvorschau. Die Vorschau soll bei erfolgreicher Angabe eines Ortes einen passenden Kartenausschnitt anzeigen. Unterhalb dieser Karte befindet sich ein Button zum Bestätigen, welcher zum nächsten Bildschirm navigiert und eine Navigationsleiste. Das zweite Wireframe ist für die Übersicht aller gespeicherten Routen. Er besteht aus einer Liste und der Navigationsleiste. Die beiden Wireframes drei und vier zeigen den Kartenbildschirm vor und während der Navigation. In der Mitte wird die Karte angezeigt. Darüber befindet sich eine Übersicht über Start, Ziel, Dauer und Entfernung sowie ein Button zum Starten der Navigation. Während der Fahrt, wird im unteren Bereich eine Liste, mit den Weganweisungen angezeigt.

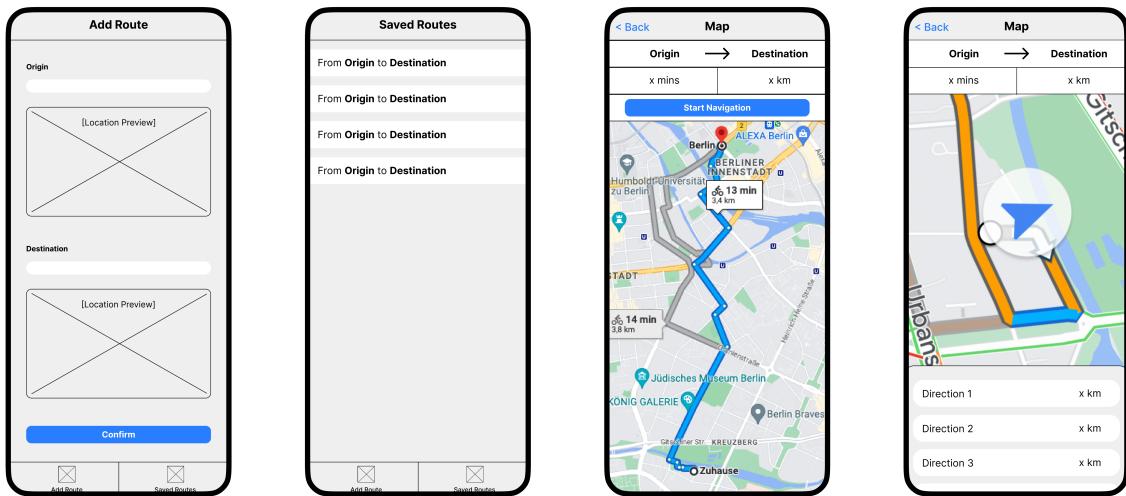


Abbildung 4.2: Wireframes für die Bikelin Navigator App

# 5 Implementation

Der folgende Abschnitt dokumentiert und beschreibt die implementierten Anforderungen. Die Kapitel sind in die vorher definierten Sprints gegliedert.

## 5.1 Sprint 1

### 5.1.1 Projekt Struktur

React Native basiert auf dem Konzept von Komponenten. Eine Komponente ist ein kleiner, wiederverwendbarer Teil einer Benutzer:innenoberfläche, der bestimmte Funktionalitäten bereitstellt. Jede Komponente kann aus anderen Komponenten zusammengesetzt werden, um komplexere Benutzer:innenoberflächen zu erstellen. Dieses Modell ermöglicht es Entwickler:innen, ihre Anwendungen in kleine, überschaubare Teile zu gliedern und jeden Teil unabhängig voneinander zu entwickeln und zu testen. Außerdem kann jede Komponente leicht wiederverwendet werden, was Zeit und Ressourcen spart.[44] Um diesem Konzept und der genannten Wartbarkeit aus Kapitel 3.5 gerecht zu werden, ist auch die Projektstruktur der Bikelin Navigator App feingliedrig und in Komponenten aufgeteilt. Die „App.js“ Datei ist die Hauptkomponente, von der aus die Anwendung gestartet wird. Alle weiteren Komponenten sind den passenden Ordnern untergeordnet. Im „Screens“ Ordner befinden sich für jeden der 3 Bildschirme, „Add Route“, „Map“ und „Saved Routes“ eine eigene Wurzelkomponente. Im „Components“ Ordner liegen alle restlichen Komponenten. Diese sind entweder einem spezifischen Bildschirm zugeordnet oder universelle UI Komponenten. Der „Store“ verwaltet den Zustand der Anwendung und ermöglicht es, dass Änderungen zentral verwaltet werden. Dies erleichtert es, den Zustand einer Anwendung zu überwachen und zu ändern, insbesondere in größeren Anwendungen mit mehreren Komponenten.[45]

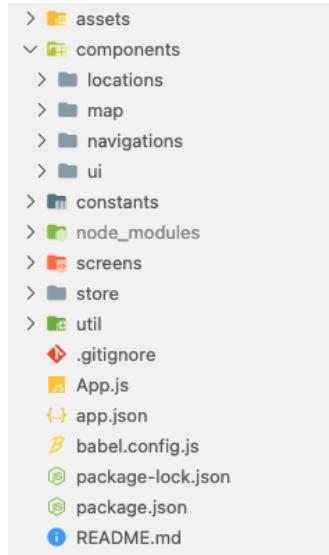


Abbildung 5.1: Projekt  
Struktur

## 5.1.2 Navigation

Für die Navigation zwischen den verschiedenen Screens wird die Open-Source Bibliothek „react-navigation“ verwendet. Eine häufig verbreitete Art der Navigation in mobilen Anwendungen ist die Registerkarten-Navigation. Dabei kann es sich um Registerkarten am unteren Rand des Bildschirms (Bottom-Tab) oder am oberen Rand unterhalb der Kopfzeile (oder sogar anstelle einer Kopfzeile) handeln.[46] In der App wird eine Kombination der Bottom-Tab Navigation und Stack Navigation verwendet. Bei der Stack Navigation, wird jeder neue Bildschirm über dem Vorherigen platziert. Beim zurück Navigieren werden die benutzten Bildschirme dann wie ein Stapel abgearbeitet.[47] Die Bottom-Tab Navigation wird für das Erstellen einer neuen Route sowie der Übersicht aller gespeicherten Routen benutzt. Zwischen diesen beiden Bildschirmen kann in der unteren Leiste hin und her gewechselt werden. Die Stack Navigation wird für das Öffnen des Kartenbildschirm benutzt. Wird eine neue Route angelegt oder auf eine bestehende geklickt, legt sich der Kartenbildschirm über den aktuellen Screen.

## 5.1.3 Anlegen einer Route

Die Benutzer:innenoberfläche orientiert sich stark an den in Kapitel 4.3 gezeigten Wireframes. Sie besteht aus einer wiederverwendbaren *Location*-Komponente und einem Button zum Berechnen der Route. Die Komponente beinhaltet ein Texteingabefeld mit Überschrift und einer Fläche, welche entweder als Platzhalter dient oder ein Bild des ausgewählten Ortes auf der Karte anzeigt. Sie benötigt das Attribut „type“, welches die Komponente an die jeweilige Variante anpasst. Die Komponente wird einmal für die Startposition und einmal für den Zielort verwendet. Die Eingabe in das Textfeld wird durch die Enter Taste der Tastatur bestätigt. Der erste Screenshot auf Abbildung 5.2 zeigt die fertig implementierte Benutzer:innenoberfläche. Die Komponente für die Startposition ist in dem Beispiel bereits ausgefüllt und der Zielort ist leer. Das Bild für die Kartenvorschau wird von der *Maps Static API* von Google bereitgestellt. Mit der API können Google Maps-Bilder in Anwendungen eingebettet werden, ohne dass Javascript oder ein dynamisches Laden der Seite erforderlich ist. Die Karte wird anhand von Koordinaten über eine Standard-HTTP-Anfrage erstellt.[48] Sobald beide Komponenten erfolgreich ausgefüllt worden sind, kann über den „Calculate Route“ Button zum Kartenbildschirm navigiert werden. Dabei werden die eingegebenen Orte als Parameter übergeben, damit sie anschließend verarbeitet werden können.

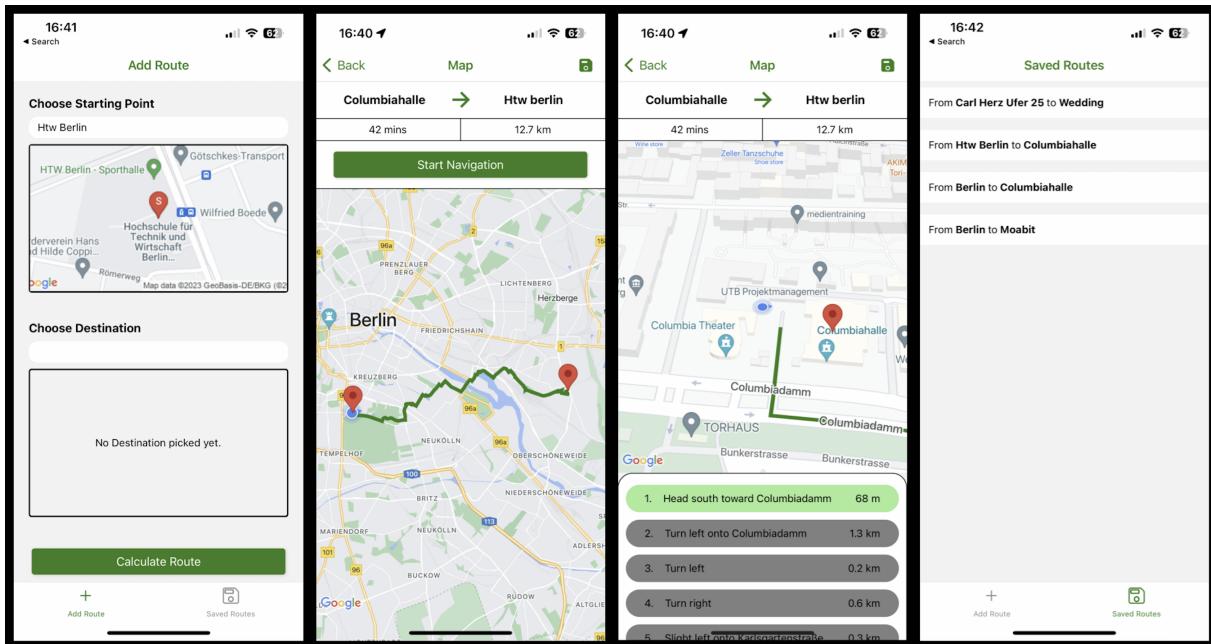


Abbildung 5.2: Benutzer:innenoberfläche der App

## 5.2 Sprint 2

### 5.2.1 Karte

Für das Darstellen einer Route oder der Navigationshilfe wird eine Karte innerhalb der App gebraucht. Die „react-native-maps“ Bibliothek beinhaltet die *MapView* Komponente, die es ermöglicht interaktive Karten in Anwendungen darzustellen. Mit Ihr können Marker auf der Karte platziert, Linien und Formen gezeichnet und der Kartentyp (Straßenkarte, Satellitenbild, Hybrid) geändert werden. Des Weiteren können auch interaktive Funktionen wie das Zoomen und Scrollen der Karte eingebunden werden.[49]

Die *MapView* Komponente der Bikelin Navigator App benutzt folgende Konfigurationen. Über das Attribut *provider* wird der Kartenanbieter festgelegt. Per Standard wird für iOS das *MapKit* von Apple und für Android Google Maps verwendet. Für ein einheitlicheres Aussehen ist in der App der Anbieter jedoch auf Google gesetzt. Somit wird für beide Plattformen Google Maps als Kartenanbieter verwendet.

Damit die Position der Nutzer:innen auf der Karte dargestellt werden kann, muss das Attribut *showUserLocation* auf „true“ gesetzt werden.

Dem *onLayout* Attribut wird die Methode *onLayoutHandler* übergeben, welche beim Auslösen des Events die Karten so ausrichtet, dass sowohl der Start auch als das Ziel gut sichtbar ist. Hier wird als Hilfsmittel die vor implementierte Methode *fitToMarkers* verwendet.

Auch dem Event *onUserLocationChange* kann eine Methode übergeben werden. Diese wird immer dann ausgelöst, wenn sich die Position vom mobilen Endgerät verändert. Diese Funktionalität wurde in Sprint 3 implementiert und wird in Kapitel 5.3.3 dokumentiert.

## 5.2.2 Marker

Um auf der Karte Positionen, wie Start und Ziel visuell zu markieren, eignen sich sogenannte *Pins* oder *Marker*. Auch das bietet die „react-native-maps“ Bibliothek. Der Marker der Bibliothek ist ein UI-Element, das auf einer Karte platziert wird, um einen bestimmten Ort oder eine bestimmte Position anzuzeigen. Es kann mit verschiedenen Optionen konfiguriert werden, einschließlich der Position auf der Karte, des Icons, der Farbe, des Titels und der Beschriftung. Die Position wird durch das Attribut *coordinate*, in Form eines Javascript Objektes mit Längen- und Breitengrad gesetzt.[50]

## 5.2.3 Context API

Die *Context API* ist ein integriertes Feature von React. Sie wurde entwickelt, um Entwickler:innen eine einfache Möglichkeit zu bieten, Daten und Funktionen zu teilen, die über den normalen Parent-Child-Lifecycle hinausgehen.[45] Einer der Vorteile der *Context API* ist, dass dadurch das sogenannte Props-Drilling verhindert werden kann. Props-Drilling, ist ein Szenario in React, bei dem Daten oder Funktionen über mehrere Komponenten weitergegeben werden müssen, um zu einer tiefer gelegenen Komponente zu gelangen. Code dieser Art ist schwierig zu verwalten sowie zur Fehlerbehebung ungeeignet und erzeugt Redundanz in den betroffenen Komponenten.[51]

Die *Context API* übernimmt in der App das Zustandsmanagement. Konkret werden dort die angelegten Routen gespeichert, sowie Funktionen zu ihrer Verwaltung definiert. Es ist wichtig zu beachten, dass dies nicht dazu gedacht ist, um Daten dauerhaft zu speichern. Stattdessen sollte es nur für die Verwaltung von Daten und Funktionen während einer Sitzung der Anwendung verwendet werden. Für dauerhafte Datenspeicherung kann zu einem späteren Zeitpunkt das Bikelin Navigator Backend hinzugezogen werden.

## 5.2.4 Speichern von Routen

Um schnell auf Routen zwischen bestimmten Punkten zuzugreifen, sollte es möglich sein, sie zu speichern. Für das Speichern wird die *Context API* aus Kapitel 5.2.3 verwendet. Damit Nutzer:innen eine Route speichern können, wird in der oberen Leiste des Kar-

tenbildschirms ein Button eingefügt. Über die React Methode `useContext` kann in der Komponente, dann auf den Kontext für die gespeicherten Routen direkt zugegriffen werden. Anschließend kann über die Methode `addRoute` hinzugefügt werden.

## 5.3 Sprint 3

### 5.3.1 Routen Berechnung

Zum Berechnen der Routen wird die React Native Bibliothek *react-native-maps-directions* verwendet. Sie ermöglicht es Routen zwischen Orten auf einer Karte zu berechnen und anzuzeigen und basiert auf der *react-native-maps* Bibliothek, welche für die Komponenten *MapView* und *Marker* aus Kapitel 5.2.1 und 5.2.2 zuständig ist. Für die Routenberechnung verwendet die Bibliothek die Google Maps API.

Ähnlich wie bei den Markern, gibt es für die Route eine eigene Komponente namens *MapViewDirections*, welche dem *MapView* als Unterelement abgehängt werden kann. Über die beiden Attribute *origin* und *destination* können die Start- und Zielpunkte auf der Karte festlegt werden, um die schnellste oder kürzeste Route zwischen diesen Punkten zu berechnen. Das Attribut *mode* bestimmt, ob die Strecke für das Auto, Fahrrad oder für zu Fuß sein soll. Für unsere Anwendung verwenden wir den Fahrradmodus.[52]

### 5.3.2 Darstellen einer Route

Die Benutzer:innenoberfläche (Abbildung 5.2, Screenshot 2) des Kartenbildschirms orientiert sich sehr stark an den angefertigten Wireframes aus Kapitel 4.3. Im Mittelpunkt ist die vom *MapView* generierte Karte. Dort wird mit zwei *Markern* Start und Ziel markiert. Die generierte Route wird durch eine grüne Linie zwischen den beiden Punkten dargestellt. Der Kartenausschnitt kann durch Zoomen oder Wischen beliebig angepasst werden. Im oberen Teil des Bildschirms werden die Namen von Start und Ziel angezeigt. Des Weiteren wird die geschätzte Dauer sowie die Länge der Strecke angegeben. Darunter befindet sich ein Button zum Starten der Navigation. In der Kopfzeile kann zum vorherigen Bildschirm zurück navigiert, oder die angezeigte Route gespeichert werden.

### 5.3.3 Kamerafokus

Das Feature des Kamerafokus sorgt für eine sichere und komfortable Benutzung der Anwendung während der Fahrt. Die konkrete Anforderung lautet, dass nach Starten der

Navigation der Kartenausschnitt von der Routen-Übersicht nah an die Position der Nutzer:innen zoomen soll, und von da an stets ihre Position verfolgt.

Für die Umsetzung wird die Methode *onUserLocationChange* der Komponente *MapView* eingesetzt. Diese Methode wird jedes Mal ausgeführt, wenn die Benutzer:innen ihren Standort ändern oder sich bewegen. Zu Beginn der Methode wird überprüft, ob die Navigation bereits gestartet wurde. Falls nicht, wird die Methode sofort beendet. Wenn die Navigation aktiv ist, werden zunächst aus dem „Location“ Objekt, welches von der Methode übergeben wird, die neuen Koordinaten isoliert. Diese werden dann an die Methode *animateCameraToUser* übergeben. Dort wird über eine Referenz auf den *MapView* mit der Methode *animateCamera* die neue Position des Kartenausschnitts bestimmt. Für das Setzen des Ausschnitts sind folgende Attribute relevant.

*Center* bestimmt den Mittelpunkt des Kartenausschnitts und erwartet ein Javascript Objekt mit Breiten- und Längengrad.

*Heading* bestimmt die Ausrichtung der Kamera in Grad bezogen auf die Nordrichtung. Das bedeutet, wenn *Heading* 0 Grad beträgt, die Kamera nach Norden ausgerichtet ist und wenn *Heading* 90 Grad beträgt, die Kamera nach Osten zeigt. Somit wird die Kamera immer in die Richtung ausgerichtet, in der das mobile Endgerät gerade zeigt, bzw. in die die Nutzer:innen fahren.

Ebenfalls relevant ist das *pitch* Attribut. Das Neigen der Karte kann bei der Navigation hilfreich sein, um den Benutzer:innen einen besseren Überblick über die Umgebung zu geben, Hindernisse früh zu erkennen und den Verlauf von Straßen und Wegen besser nachzuvollziehen.[53]

## 5.4 Sprint 4

### 5.4.1 Anzeigen von gespeicherten Routen

Für das Anzeigen von gespeicherten Routen wird ein neuer Bildschirm angelegt. Dieser ist über die untere Navigationsleiste neben dem Anlegen einer Route erreichbar. Die gespeicherten Routen sollen in einer Liste mit Start und Ziel angezeigt werden. Implementiert wird diese List mit der *FlatList* Komponente. Sie ist eine Kernkomponente von React Native, die für das Rendern von scrollbaren Listen mit vielen Elementen verwendet wird. Die *FlatList* ist eine performante Alternative zur *ScrollView* Komponente, da sie nur diejenigen Elemente rendert, die sichtbar sind und nicht alle Elemente, die sich in der Liste befinden. Sowohl die Liste als auch die Listenelemente sind in separaten Komponenten implementiert, um spätere Änderungen oder Erweiterungen zu erleichtern. Der fertig

umgesetzte Bildschirm ist in Abbildung 5.2 auf Screenshot 4 zu sehen.[54]

### 5.4.2 Live Navigation

Das Feature der *Live Navigation* besteht aus zwei Teilen. Zum einen, das Anzeigen aller Weganweisungen der Route in einer Liste. Zum anderen das farbliche Hervorheben der aktuellen Weganweisung. Der fertig implementierte Bildschirm ist auf Abbildung 5.2 in Screenshot drei zusehen.

Das Anzeigen der Anweisungen ist, wie in dem Wireframe aus Kapitel 4.3 dargestellt, als Liste im unteren Teil des Bildschirms implementiert. Dafür wird wie im vorherigen Kapitel die Komponente *FlatList* verwendet. Die Komponente *MapViewDirections* aus Kapitel 5.3.1 besitzt die Methode *onReady*, welche ausgeführt wird, sobald die Route fertig berechnet wurde und zur Verfügung steht. Neben vielen weiteren Informationen werden die einzelnen Navigationsanweisungen in der Methode bereitgestellt. Sobald die Navigation gestartet wird, erscheint im unteren Bereich des Bildschirms eine scrollbare Liste mit den Anweisungen.

Das farbliche Hervorheben der aktuellen Weganweisung, dient zur besseren Orientierung während der Fahrt und für eine angenehmere Benutzer:innenerfahrung. Beim Starten der Navigation wird automatisch das erste Element aus der Liste hervorgehoben. In der Methode *onUserLocationChange* der *MapView* Komponente wird neben dem Kamerafokus ebenfalls bei gestarteter Navigation die Methode *calcDistanceToNextWaypoint* ausgeführt. Diese berechnet die Distanz zwischen dem aktuellen Standort und dem nächsten Wegpunkt und gibt diese zurück. Sollte die Entfernung kleiner als die festgelegte Konstante sein, wird der nächste Wegpunkt aus der Liste farblich markiert.

# 6 Tests

Das Testen der App stellte zu Beginn eine Herausforderung dar. Die Anwendungen besitzt kaum Funktionen oder Komponenten, die sich sinnvoll isoliert mit Unit Tests testen lassen. Auch UI Tests wären wenig Zielführend aufgrund der überschaubaren Benutzer:innenoberfläche. Da die Benutzbarkeit eine von den beiden Haupt- nicht funktionalen Anforderungen aus Kapitel 3.5 ist, fiel die Entscheidung schließlich auf die Durchführung von Benutzer:innen Tests, um das Feedback von echten Benutzer:innen zu erhalten und sicherzustellen, dass die App auch in der Praxis funktioniert. Benutzer:innen Tests sind ein wertvolles Instrument, um Probleme aufzudecken, die in anderen Tests möglicherweise nicht entdeckt wurden. Sie können auch dazu beitragen, das Verständnis für die Perspektive der Benutzer:innen zu verbessern und die Benutzer:innenfreundlichkeit der App zu optimieren.[55]

Für einen systematischen und strukturierten Ansatz werden die Benutzer:innen Tests in Form von 3 Aufgaben durchgeführt. Die Aufgaben sollen dafür sorgen, dass während der Tests immer dieselben Ausgangsbedingungen herrschen. Folgende Aufgaben werden den Testpersonen nacheinander gestellt:

1. Lege eine neue Route an, lese die Gesamtdauer und Entfernung vor und speicher die Route anschließend.
2. Navigiere zur Übersicht der gespeicherten Routen und rufe deine gerade gespeicherte Route wieder auf.
3. Starte die Navigation und fahr die Route mit einem Fahrrad ab.

Die Resultate der ersten beiden Aufgaben können direkt beim Ausführen dokumentiert werden. Die dritte Aufgabe wird von der Testperson zuerst ausgeführt und anschließend von der Erfahrung berichtet.

## **Testperson 1**

Das Anlegen der Route verlief bei Testperson 1 ohne Probleme. Es wurden zwei Orte eingegeben und anschließend zum nächsten Bildschirm navigiert. Auch das Ablesen von Dauer und Entfernung stellte keine Herausforderung dar. Der Button zum Speichern konnte ohne Probleme identifiziert werden. Es herrschte jedoch nach Betätigen des Buttons kurz Verwirrung, ob das Speichern funktioniert hat.

Aufgabe 2 konnte von der Testperson problemlos abgeschlossen werden.

Die Fahrt konnte von der Testperson erfolgreich abgeschlossen werden. Das einzige Feedback war, dass in manchen Momenten die nächste Weganweisung zu spät markiert wurde.

## **Testperson 2**

Beim Anlegen der Route, hat Testperson 2 versucht durch Zoomen des Kartenausschnitts, sich einen besseren Überblick zu verschaffen. Jedoch handelt es sich hierbei nur um ein Bild und keine benutzbare Karte. Anschließend wurde zum nächsten Bildschirm navigiert und erfolgreich Dauer und Entfernung abgelesen. Beim Speichern der Route wurde der Button allerdings zweimal gedrückt, da beim ersten Mal die Testperson sich nicht sicher war, ob es funktioniert hat.

Aufgabe 2 konnte von Testperson 2 ebenfalls problemlos abgeschlossen werden. Durch das doppelte Speichern wurde die Route zweimal angezeigt und sich für eine von beiden entschieden.

Auch die Fahrt stellte für Testperson 2 keine Probleme. Ihr Feedback war, dass die Navigationshilfe wie gewünscht funktioniert hat.

## **Testperson 3**

Aufgabe 1 und 2 wurde von Testperson 3 ohne Probleme oder Fehler absolviert.

Im Feedback aus Aufgabe 3 wurde gewünscht, dass sich die angezeigte Dauer und Entfernung während der Fahrt an die verbleibende Strecke anpasst. Des Weiteren wurde genau wie von Testperson 1 angemerkt, dass manchmal die nächste Weganweisung zu spät markiert wurde. Ein weiterer Punkt war, dass die Liste der Weganweisungen zwar immer die aktuelle Anweisung markiert, jedoch nicht automatisch weiter scrollt.

## Resultate

Die Benutzer:innentests haben einige aufschlussreiche Ergebnisse liefern können. Bei Aufgabe 1 war die größte Erkenntnis, dass beim Speichern einer Route zu wenig visuelles Feedback existiert. Zwei der drei Testpersonen schienen Probleme damit zu haben. Zwar gibt es beim Drücken des Buttons ein visuelles Feedback in Form von einer kurzen Animation, welche die Deckkraft des Symbols verändert, jedoch scheint dies noch ausbaufähig zu sein. Eine Lösung dafür könnte ein Popup sein, welches das Speichern bestätigt und nach kurzer Zeit von allein verschwindet.

Beim zurück Navigieren und dem Finden der gespeicherten Routen in Form von Aufgabe 2 hatte keine der Testpersonen Probleme.

Aus der dritten Aufgabe resultierten die meisten Erkenntnisse darüber, die Anwendung anzupassen oder zu erweitern. Einer der Erweiterungen wäre, die Dauer und Distanz der verbleibenden Strecke während der Navigation anzuzeigen, anstatt durchgehen die gesamte Dauer und Distanz. Dass die aktuelle Weganweisung nicht nur farblich markiert wird, sondern zugleich auch immer im Mittelpunkt dargestellt wird, ist ebenfalls eine sinnvolle Erweiterung. Das Problem, dass Weganweisungen teilweise zu spät markiert worden sind, wurde direkt im Anschluss im Code angepasst.

# 7 Fazit und Ausblick

## Fazit

Im Rahmen dieser Arbeit wurde eine App für die Webanwendung *Bikelin Navigator* geplant und implementiert. Dabei wurden in den ersten Schritten zunächst die Grundlagen geklärt und ein geeigneter Techstack definiert. Die Recherche hat ergeben, dass für die Entwicklung der App, React Native als optimales Framework dient. Mit React Native kann die App in kurzer Zeit für iOS und Android entwickelt werden. Im Weiteren Verlauf der Arbeit wurde eine intensive Analyse durchgeführt. Dabei wurde zunächst die bestehende Anwendung beleuchtet, um passende Anforderungen für die zu entwickelnde App zu definieren. Des Weiteren wurde eine Marktanalyse von ähnlichen mobilen Anwendungen durchgeführt, um Stärken und Schwächen festzustellen, die bei der eigenen Anwendung von Relevanz sind. Aus dieser Analyse resultierte eine konkrete Auflistung aller funktionalen Anforderungen sowie einer Priorisierung von nicht funktionalen Anforderungen. Weitere Planung in Form von einer Entwicklungsmethode sowie die Entwicklung von Wireframes wurde im Konzeptions-Teil durchgeführt. Das anschließende Implementierungs-Kapitel beschreibt und dokumentiert die gesamte Anwendung. Dabei sind die einzelnen umgesetzten Anforderungen in vier Sprints unterteilt. Das Ergebnis ist die erste benutzbare Version der Bikelin Navigator App. Sie ermöglicht das Anlegen, Speichern und Darstellen von Routen sowie eine Live Navigationshilfe.

## Ausblick

Bei der Planung und Implementierung wurde von Beginn darauf geachtet, dass die Anwendung leicht wartbar und erweiterbar ist. Folgende Erweiterungen würden sich für weitere Versionen anbieten: Eine Verbindung zwischen bestehendem Backend und App Frontend kann hergestellt werden. Dazu gehört das Einrichten von Möglichkeiten zum Anmelden und Registrieren in der App. Angelegte Routen aus dem Backend sollten im Frontend sichtbar sein. Des Weiteren wäre es sinnvoll weitere Daten aus dem Backend wie Luftqualität oder den Regenradar als optionales Feature in die App zu integrieren. Neben Google

Maps als Routenplaner könnten, wie auch schon in der Webanwendung weitere Anbieter zur Berechnung von Routen hinzugefügt werden. Ein weiteres Feature, was besonders für mehr Sicherheit der Routen sorgen würde, ist das Umgehen von Gefahrenpunkten bei der Erstellung einer Route. Auch die Ergebnisse der Tests aus Kapitel 6, die noch nicht umgesetzt werden konnten, können noch implementiert werden. Zum Beispiel das visuelle Feedback des Speicherns einer Route oder die verbleibende Dauer bzw. Distanz der Strecke während der Fahrt. Abschließen lässt sich feststellen, dass es viele sinnvolle Möglichkeiten gibt, die App für den Bikelin Navigator in weiteren Projekten zu erweitern.

# Abbildungsverzeichnis

2.1	Von Softwareentwickler:innen weltweit verwendete plattformübergreifende mobile Frameworks in 2021 [22] . . . . .	6
3.1	Benutzer:innenoberfläche der Web Anwendung Bikelin Navigator . . . . .	10
3.2	Benutzer:innenoberfläche der Web Anwendung Bikelin Navigator auf dem Smartphone . . . . .	11
3.3	Benutzer:innenoberfläche von Google Maps beim Anlegen einer Route . . . . .	13
3.4	Benutzer:innenoberfläche von Google Maps während der Fahrt . . . . .	14
3.5	Benutzer:innenoberfläche von Komoot beim Anlegen einer Route . . . . .	15
3.6	Benutzer:innenoberfläche von Komoot während der Fahrt . . . . .	16
3.7	Qualitätsmodell für äussere und innere Qualität eines Softwareprodukts nach ISO/IEC 25010 [40, S.106] . . . . .	18
4.1	Zeitplan der Bachelorarbeit . . . . .	20
4.2	Wireframes für die Bikelin Navigator App . . . . .	21
5.1	Projekt Struktur . . . . .	22
5.2	Benutzer:innenoberfläche der App . . . . .	24

# Tabellenverzeichnis

3.1	Funktionale Anforderung der mobilen App des Bikelin Navigators . . . . .	17
-----	--	----

# Literatur- und Quellenverzeichnis

- [1] „Radverkehr.“ Umweltbundesamt. Publisher: Umweltbundesamt. (22. Okt. 2012), Adresse: <https://www.umweltbundesamt.de/themen/verkehr-laerm/nachhaltige-mobilitaet/radverkehr> (besucht am 05.01.2023).
- [2] C. Latz, „Neue Unfall-Karte vorgestellt: Wo Radfahren in Berlin am gefährlichsten ist.“ *Der Tagesspiegel Online*, 14. Apr. 2021, ISSN: 1865-2263. Adresse: <https://www.tagesspiegel.de/berlin/wo-radfahren-in-berlin-am-gefaehrlichsten-ist-5395402.html> (besucht am 05.01.2023).
- [3] „Smartphone subscriptions worldwide 2027.“ Statista. (22. Aug. 2022), Adresse: <https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/> (besucht am 28.12.2022).
- [4] A. Turner. „How many people have smartphones worldwide (dec 2022).“ Section: Research. (10. Juli 2018), Adresse: <https://www.bankmycell.com/blog/how-many-phones-are-in-the-world> (besucht am 28.12.2022).
- [5] „Global: Mobile app downloads by segment 2017-2025.“ Statista. (15. Sep. 2021), Adresse: <https://www.statista.com/forecasts/1262881/mobile-app-download-worldwide-by-segment> (besucht am 28.12.2022).
- [6] „App - worldwide | statista market forecast.“ Statista. (), Adresse: <https://www.statista.com/outlook/dmo/app/worldwide> (besucht am 28.12.2022).
- [7] M. Poczwarcowski. „What are the risks of choosing wrong technology for your web app?“ (14. Apr. 2020), Adresse: <https://www.netguru.com/blog/choosing-wrong-technology-risks> (besucht am 09.01.2023).
- [8] „Native App entwickeln – Ein Überblick | itPortal24.“ (), Adresse: <https://www.itportal24.de/ratgeber/native-app-entwickeln> (besucht am 28.12.2022).
- [9] S. Augsten. „Was ist eine Cross-Platform App?“ (7. Feb. 2020), Adresse: <https://www.dev-insider.de/was-ist-eine-cross-platform-app-a-898699/> (besucht am 28.12.2022).
- [10] M. Strizic. „How native apps influence user experience | DECODE.“ (5. Jan. 2022), Adresse: <https://decode.agency/article/native-apps-ux/> (besucht am 30.12.2022).

- [11] A. Manchanda. „The ultimate guide to cross platform app development frameworks in 2023,“ Insights - Web and Mobile Development Services and Solutions. (4. Nov. 2022), Adresse: <https://www.netsolutions.com/insights/cross-platform-app-frameworks-in-2019/> (besucht am 10.01.2023).
- [12] „Native vs. cross platform apps | microsoft power apps,“ Native, hybrid, or cross-platform apps? (), Adresse: <https://powerapps.microsoft.com/de-de/native-vs-cross-platform-apps/> (besucht am 28.12.2022).
- [13] V. Makhov. „Mobile app development: Best practices [guide 2022],“ DOIT Software. (3. Okt. 2021), Adresse: <https://doit.software/blog/mobile-app-development> (besucht am 10.01.2023).
- [14] J. Solanki. „Native vs. cross platform: Decoding best choice for your apps,“ Simform - Product Engineering Company. (22. Sep. 2022), Adresse: <https://www.simform.com/blog/native-vs-cross-platform-development/> (besucht am 28.12.2022).
- [15] A. Marchuk. „Native vs cross-platform development: Pros & cons revealed.“ (), Adresse: <https://www.uptech.team/blog/native-vs-cross-platform-app-development> (besucht am 28.12.2022).
- [16] J. Pokorska. „Cross platform app development - facts and myths,“ Appstronauts. (4. Aug. 2020), Adresse: <https://appstronauts.co/blog/cross-platform-app-development-facts-and-myths/> (besucht am 30.12.2022).
- [17] „Mobile app testing: Definition, why it is important, how to do it,“ BrowserStack. (), Adresse: <https://browserstack.wordpress.com/mobile-app-testing/> (besucht am 30.12.2022).
- [18] V. Vahromovs. „Native vs. cross-platform app development: Which should you choose?“ (22. Okt. 2021), Adresse: <https://builtin.com/software-engineering-perspectives/native-vs-cross-platform-app-development> (besucht am 30.12.2022).
- [19] S. Singh. „Native vs hybrid vs cross platform - what to choose in 2023?“ Insights - Web and Mobile Development Services and Solutions. (19. Nov. 2022), Adresse: <https://www.netsolutions.com/insights/native-vs-hybrid-vs-cross-platform/> (besucht am 28.12.2022).
- [20] „Essential factors and benefits of UX in mobile app development.“ (), Adresse: <https://neetable.com/blog/essential-factors-of-ux-design-in-app-development> (besucht am 30.12.2022).
- [21] R. Piluta. „React native vs. flutter: Which one to choose – NIX united,“ NIX United – Custom Software Development Company in US. (30. Jan. 2020), Adresse: <https://nix-united.com/blog/flutter-vs-react-native/> (besucht am 09.01.2023).

- [22] „Cross-platform mobile frameworks used by global developers 2021,“ Statista. (), Adresse: <https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/> (besucht am 07.01.2023).
- [23] B. Eisenman, *Learning React Native: building mobile applications with JavaScript*, First edition. Beijing: O'Reilly, 2015, 254 S., OCLC: ocn908375566, ISBN: 978-1-4919-2900-1.
- [24] B. Dragomir. „React native: Under the hood,“ Medium. (7. Juli 2021), Adresse: <https://betterprogramming.pub/react-native-under-the-hood-281df5f548f> (besucht am 07.01.2023).
- [25] „What is Flutter and Its Advantages?“ (), Adresse: <https://adservio.fr/post/what-is-flutter-and-what-are-its-advantages> (besucht am 07.01.2023).
- [26] „Skia documentation,“ Skia. (), Adresse: <https://skia.org/docs/> (besucht am 07.01.2023).
- [27] R. Gehrer. „How flutter works under the hood and why it is game-changing,“ Medium. (18. März 2020), Adresse: <https://medium.com/@ralfgehrer/how-flutter-works-under-the-hood-and-why-it-is-game-changing-2335954a5bfc> (besucht am 07.01.2023).
- [28] A. R. Robin. „How Flutter and Dart Work behind the scene ?“ (30. Juli 2019), Adresse: <https://www.linkedin.com/pulse/how-flutter-dart-work-behind-scene-abdur-rahman-robin> (besucht am 07.01.2023).
- [29] A. Rogatiuk. „React native vs flutter: Which one is better for 2023?“ Fireart Studio. (7. Apr. 2020), Adresse: <https://fireart.studio/blog/flutter-vs-react-native-what-app-developers-should-know-about-cross-platform-mobile-development/> (besucht am 09.01.2023).
- [30] M. Joshi. „Flutter vs react native: A comparison,“ BrowserStack. (11. Nov. 2022), Adresse: <https://browserstack.wengine.com/guide/flutter-vs-react-native/> (besucht am 09.01.2023).
- [31] „Most used languages among software developers globally 2022,“ Statista. (), Adresse: <https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/> (besucht am 09.01.2023).
- [32] „Common questions,“ Expo Documentation. (), Adresse: <https://docs.expo.dev/introduction/faq> (besucht am 17.02.2023).
- [33] „Was ist eine Mikrodienstarchitektur?“ Google Cloud. (), Adresse: <https://cloud.google.com/learn/what-is-microservices-architecture?hl=de> (besucht am 26.01.2023).
- [34] „Keycloak.“ (), Adresse: <https://www.keycloak.org/> (besucht am 26.01.2023).

- [35] „Angular - What is Angular?“ (), Adresse: <https://angular.io/guide/what-is-angular> (besucht am 26.01.2023).
- [36] „Die Marktanalyse – ein Beispiel | Gründerplattform.“ (), Adresse: <https://gruenderplattform.de/businessplan/marktanalyse> (besucht am 30.01.2023).
- [37] „Über Google Maps.“ (), Adresse: <https://www.google.de/intl/de/maps/about/#/> (besucht am 31.01.2023).
- [38] „iOS and iPadOS - feature availability.“ Apple. (), Adresse: <https://www.apple.com/ios/feature-availability/#maps-cycling> (besucht am 02.02.2023).
- [39] I. Sommerville, *Software Engineering* (Informatik), 10., aktualisierte Auflage. Hallbergmoos: Pearson, 2018, 896 S., ISBN: 978-3-86894-344-3.
- [40] H. Tremp, *Agile objektorientierte Anforderungsanalyse: Planen – Ermitteln – Analyseren – Modellieren – Dokumentieren – Prüfen* (erfolgreich studieren). Wiesbaden: Springer Vieweg, 2022, 251 S., ISBN: 978-3-658-37194-4 978-3-658-37193-7.
- [41] D. M. Siepermann. „Definition: Scrum,“ <https://wirtschaftslexikon.gabler.de/definition/scrum-53462>. Publisher: Springer Fachmedien Wiesbaden GmbH Section: economy. (), Adresse: <https://wirtschaftslexikon.gabler.de/definition/scrum-53462> (besucht am 11.02.2023).
- [42] „Was ist ein Wireframe?“ kulturbanause®. (), Adresse: <https://kulturbanause.de/faq/wireframe/> (besucht am 10.02.2023).
- [43] „Was sind Website-Wireframes?“ Lucidchart. (), Adresse: <https://www.lucidchart.com/pages/de/was-sind-website-wireframes> (besucht am 11.02.2023).
- [44] „Components and props – react.“ (), Adresse: <https://reactjs.org/docs/components-and-props.html> (besucht am 04.02.2023).
- [45] „Context – react.“ (), Adresse: <https://reactjs.org/docs/context.html> (besucht am 11.02.2023).
- [46] „Tab navigation | react navigation.“ (), Adresse: <https://reactnavigation.org/docs/tab-based-navigation/> (besucht am 11.02.2023).
- [47] „Stack navigator | react navigation.“ (), Adresse: <https://reactnavigation.org/docs/stack-navigator/> (besucht am 11.02.2023).
- [48] „Google maps platform documentation | maps static API.“ Google Developers. (), Adresse: <https://developers.google.com/maps/documentation/maps-static> (besucht am 16.02.2023).
- [49] „GitHub - react-native-maps/react-native-maps: React Native Mapview component for iOS + Android.“ (), Adresse: <https://github.com/react-native-maps/react-native-maps> (besucht am 16.02.2023).
- [50] *react-native-maps*, original-date: 2015-12-29T19:54:20Z, 16. Feb. 2023. Adresse: <https://github.com/react-native-maps/react-native-maps/blob/>

aeb8edac1ed054de2112403e697c78e41d51dd7b/docs/marker.md (besucht am 16.02.2023).

- [51] „What is prop drilling in react?“ Educative: Interactive Courses for Software Developers. (), Adresse: <https://www.educative.io/answers/what-is-prop-drilling-in-react> (besucht am 11.02.2023).
- [52] Bramus! *react-native-maps-directions*, original-date: 2017-11-19T20:27:42Z, 7. Feb. 2023. Adresse: <https://github.com/bramus/react-native-maps-directions> (besucht am 16.02.2023).
- [53] „react-native-maps/mapview.md at master · react-native-maps/react-native-maps · GitHub.“ (), Adresse: <https://github.com/react-native-maps/react-native-maps/blob/master/docs/mapview.md> (besucht am 16.02.2023).
- [54] „FlatList · react native.“ (12. Jan. 2023), Adresse: <https://reactnative.dev/docs/flatlist> (besucht am 16.02.2023).
- [55] „The importance of user testing: What every business should know,“ Marvel Blog. (26. Juli 2019), Adresse: <https://marvelapp.com/blog/importance-user-testing/> (besucht am 18.02.2023).

## **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