HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

FRONTLINE MEDIA

Tourenplaner
für die Traildevils iOS App

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2017

Autor(en):	Roman Blum, Philipp Schilter
Betreuer:	Prof. Dr. Markus Stolze
Projektpartner:	Frontline Media GmbH, Rüti
Experte:	Thomas Kälin
Gegenleser:	Prof. Stefan F. Keller

ABSTRACT

AUSGANGSLAGE: Ein zentrales Produkt der Frontline Media GmbH ist die Traildevils Website, eine Online Community Site für Mountain Biker. Um die Attraktivität zu steigern, wurde bereits im Herbstsemester 2016/17 ein Prototyp als native iOS Applikation entwickelt, welche nun durch einen Tourenplaner erweitert werden soll. Ziel der aktuellen Arbeit ist die Validierung eines Interaktionskonzeptes für die mobile Tourenplanung mit Fokus auf intuitive Bedienung.

VORGEHEN UND TECHNOLOGIEN: In einer ersten Phase wurden Wireframes auf Papier gezeichnet und anschliessend mithilfe des Online Dienstes «Proto.io» zu einem klickbaren Prototyp umgewandelt. So konnten Ideen und Konzepte für die Umsetzung mit dem Projektpartner diskutiert und festgelegt werden. Zeitgleich wurden Experimente zur Minimierung weiterer Risiken gemacht, zum Beispiel das Verschieben von Wegpunkten oder die Verarbeitung einer Antwort des Routing Dienstes. Die erarbeiteten Konzepte wurden in einem zweiten Schritt implementiert und mit mehreren Personen der Zielgruppe validiert. Es stellte sich heraus, dass verschiedene Interaktionen mit der Karte und der Tour entweder nicht entdeckt wurden oder zu Problemen führten. Es zeigte sich zudem, dass der eingesetzte Geocoder teilweise zu wenig oder ungenaue Ergebnisse lieferte. Ein weiterer Usability Test sollte die Anpassungen wiederum validieren. Der zweite Test bestätigte, dass sowohl iOS als auch Android Benutzer eine Tour effizient und problemlos planen können.

ERGEBNIS: Am Ende dieser Arbeit kann ein intuitiver Tourenplaner mit validiertem Interaktionskonzept vorgewiesen werden. Weitere Funktionen zur Erweiterung der Tourenplanung konnten zudem implementiert werden, darunter die Anzeige von Zusatzinformationen, wie zum Beispiel ein Höhenprofil oder das Einfügen, Ändern, Verschieben und Löschen von Wegpunkten einer Tour. Als Nebenprodukt der Bachelorarbeit konnten zwei Swift Frameworks für GraphHopper entwickelt werden, welche nun auch auf der offiziellen Website verlinkt sind. Auf Basis der Tests hat der Auftraggeber bestätigt, dass mit dem aktuellen Tourenplaner nicht nur das Interaktionskonzept erfolgreich entwickelt und validiert werden konnte, sondern die Software eine Reife aufweist, dass die unveränderte Nutzung in der zukünftigen Traildevils App geplant ist. Da sich die Bachelorarbeit auf den Tourenplaner konzentriert hat, ist vor der Publikation der App noch Fleissarbeit notwendig. Insbesondere müssen Benutzeran- und abmeldung sowie Likes, Checkin oder Kommentare implementiert werden.

MANAGEMENT SUMMARY

AUSGANGSLAGE

Ein zentrales Produkt der Frontline Media GmbH ist die Traildevils Website, eine Online Community Site für Mountain Biker. Um die Attraktivität zu steigern, wurde bereits im Herbstsemester 2016/17 ein Prototyp als native iOS Applikation entwickelt, welche nun durch einen Tourenplaner erweitert werden soll. Ziel der aktuellen Arbeit ist die Validierung eines Interaktionskonzeptes für die mobile Tourenplanung mit Fokus auf intuitive Bedienung. Die Berechnung von optimalen Touren soll durch einen externen Dienst erledigt werden.

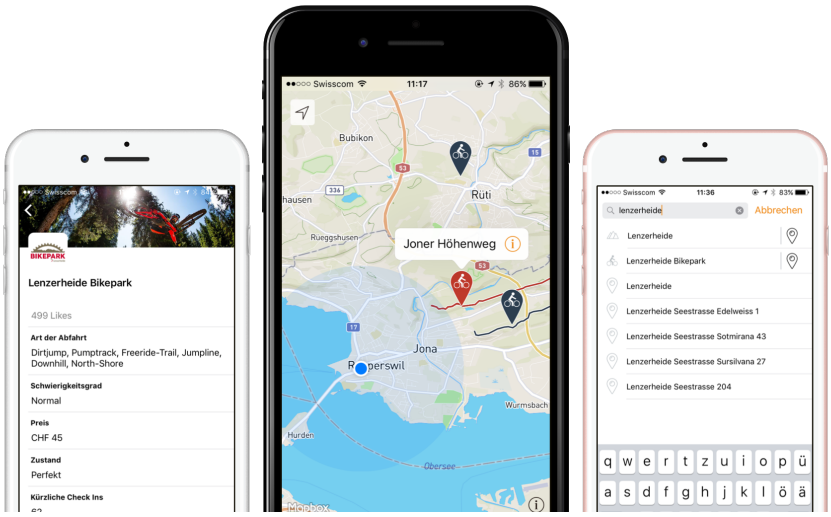


Abbildung 1: iOS Prototyp aus der Studienarbeit

VORGEHEN UND TECHNOLOGIEN

In einer ersten Phase wurden Wireframes auf Papier gezeichnet und anschliessend mithilfe des Online Dienstes «Proto.io» zu einem klickbaren Prototyp umgewandelt. So konnten Ideen und Konzepte für die Umsetzung mit dem Projektpartner diskutiert und festgelegt werden. Zeitgleich wurden Experimente zur Minimierung weiterer Risiken gemacht, zum Beispiel das Verschieben von Wegpunkten oder die Verarbeitung einer Antwort des Routing Dienstes und der anschliessenden Anzeige auf der Karte. Die erarbeiteten Konzepte wurden in einem zweiten Schritt implementiert und mit mehreren Personen der Zielgruppe validiert. Es stellte sich heraus, dass verschiedene

Interaktionen mit der Karte und der Tour entweder nicht entdeckt wurden oder zu Problemen führten. Es zeigte sich zudem, dass der eingesetzte Geocoder teilweise zu wenig oder ungenaue Ergebnisse lieferte. Ein weiterer Usability Test sollte diese gemachten Änderungen wiederum validieren. Der zweite Test bestätigte, dass sowohl iOS als auch Android Benutzer eine Tour effizient und problemlos planen können.

Abbildung 2: Usability Test mit einem Benutzer der Zielgruppe. Bild genutzt mit Einverständnis.

ERGEBNIS

Am Ende dieser Arbeit kann ein intuitiver Tourenplaner mit validiertem Interaktionskonzept vorgewiesen werden. Weitere Funktionen zur Erweiterung der Tourenplanung konnten zudem implementiert werden, darunter die Anzeige von Zusatzinformationen wie zum Beispiel ein Höhenprofil oder das Einfügen, Ändern, Verschieben und Löschen von Wegpunkten einer Tour. Als Nebenprodukt der Bachelorarbeit konnten zwei Swift Frameworks für GraphHopper entwickelt werden, welche nun auch auf der offiziellen Website verlinkt sind. Auf Basis der Tests hat der Auftraggeber bestätigt, dass mit dem aktuellen Tourenplaner nicht nur das Interaktionskonzept erfolgreich entwickelt und validiert werden konnte, sondern die Software eine Reife aufweist, dass die unveränderte Nutzung in der zukünftigen Traildevils App geplant ist. Da sich die Bachelorarbeit auf den Tourenplaner konzentriert hat, ist vor der Publikation der App noch Fleiss-

arbeit notwendig. Insbesondere müssen Benutzeran- und abmeldung sowie Likes, Checkin oder Kommentare implementiert werden.

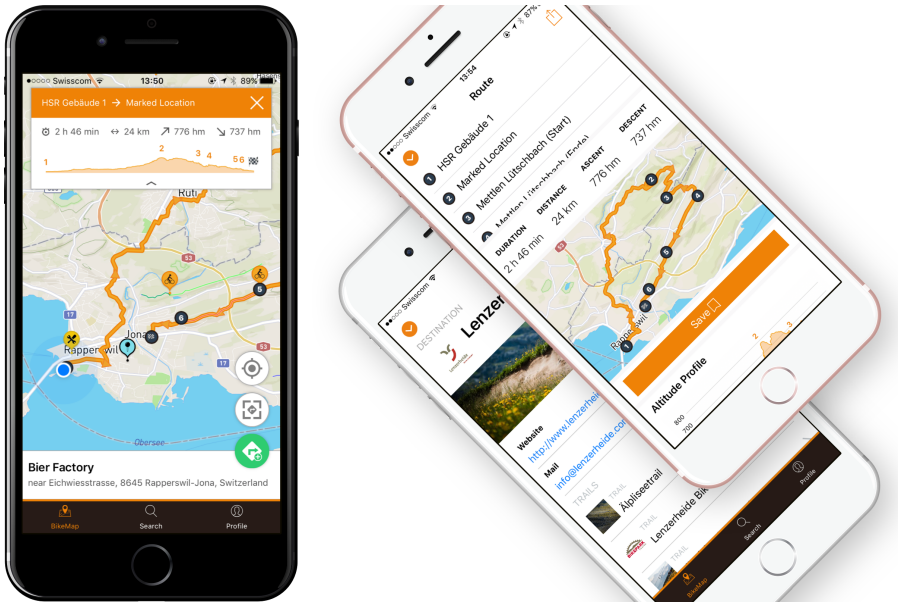


Abbildung 3: Screenshots des Tourenplaners

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

AKRONYME

API Application Programming Interface

ASQ After-Scenario Questionnaire

GPS Global Positioning System

HCID Human Computer Interaction Design

HSR Hochschule für Technik, Rapperswil

POI Point of Interest

UI User Interface

UX User Experience

SDK Software Development Kit

VIPER View-Interactor-Presenter-Entity-Router

MVC Model-View-Controller

MVVM Model-View-ViewModel

IFS Institut für Software

WLAN Wireless Local Area Network

WPF Windows Presentation Foundation

GLOSSAR

CSharp Typsichere, objektorientierte Programmiersprache

Callout Zeigt dem Benutzer Informationen über einen selektierten POI auf der Karte an

Core Data Persistenz Framework basierend auf SQLite von Apple

Dealer Bikeshop von Traildevils

Destination Touristische Region mit mehreren Trails von Traildevils

Geocoder Wandelt Adressen in geografische Koordinaten

Elevation Höhe über dem Meeresspiegel

Panning Verschieben der Karte

Placemark Ein vom Benutzer gesetzte Koordinate auf der Karte

Realm Open Source Datenbank mit Fokus auf Mobilität und Synchronisation

Reorder Control Steuerungselement für das Verschieben einer Zelle in einer Table View

SCRUM SCRUM ist die am weitesten verbreitete agile Methode im Projektmanagement

Tiles Quadratische Ausschnitte der Weltkarte

Track Abfahrt von Traildevils

Trail Startpunkt einer Abfahrt von Traildevils

Traildevils API Application Programming Interface des Projektpartners

Zooming Verstellen der Zoom-Stufe der Karte

Teil I

TECHNISCHER BERICHT

1 EINLEITUNG

1.1 VISION

Das Smartphone als täglicher Begleiter eignet sich nebst der Kommunikation auch hervorragend zur Navigation. Software-Giganten wie Apple, Google und Microsoft stellen Apps zur Planung von Routen und Navigation kostenlos zur Verfügung. Routenberechnung, Sprachausgabe und ein laufend aktualisiertes Verkehrsinfosystem gehören zum Standard-Funktionsumfang.

Die Traildevils iOS App soll mit der Funktion der Routenplanung erweitert werden. Als Grundlage dienen hierzu die Kartenelemente, hauptsächlich Trails, welche auf einfache Art und Weise zu einer Route zusammengestellt werden können.

Begriffe: Tour, Strecke und Route

In den nachfolgenden Kapiteln werden die Begriffe *Tour*, *Strecke* und *Route* verwendet. Die Begriffe sind gleichgestellt und bedeuten eine Planung von einem Punkt *A* nach einem Punkt *B*.

Der englische Begriff *Route*, auf Deutsch *Strecke* genannt, wurde im Source Code verwendet. In den Usability Tests hat sich herausgestellt, dass die Testpersonen auch oft den Begriff *Tour* verwendet haben.

Begriff: Kartenelemente

Kartenelement beschreiben von Traildevils stammende, geografische Daten. Dabei werden die englischen Ausdrücke verwendet, sprich *Trails* (Startpunkte von Abfahrten), *Tracks* (Abfahrten), *Dealers* (Shops) und *Destinations* (Destinationen).

Begriff: Benutzer

Aus Gründen der besseren Lesbarkeit wird in der Arbeit nur die männliche Form des Begriffs *Benutzer* verwendet. Gemeint ist stets sowohl die weibliche als auch die männliche Form.

1.2 HERAUSFORDERUNGEN

Die Benutzerfreundlichkeit und ein grossartiges Benutzererlebnis sind zentrale Ziele dieser Arbeit. Durch ein iteratives Vorgehen sollen die folgenden Fragestellungen mit Prototypen und Benutzertests beantwortet werden.

- Wie können Benutzer ohne Probleme eine Routenplanung starten?
- Wie wird dem Benutzer deutlich gemacht, dass er sich im Planungsmodus befindet?
- Wie kann der Benutzer den Planungsmodus verlassen?
- Wie kann der Benutzer neue Wegpunkte hinzufügen?
- Wie kann der Benutzer Wegpunkte verschieben?

1.3 VORGÄNGIGE STUDIENARBEIT

Die Bachelorarbeit baut auf dem Stand der Studienarbeit vom Herbstsemester 2016 auf. Die dort entwickelte App beinhaltet grundlegende Karteninteraktionen, wie zum Beispiel das Selektieren von Kartenelementen, das Anzeigen von Detailinformationen zu einem Kartenelement sowie das Suchen nach Traildevils Kartenelementen und öffentlichen Sehenswürdigkeiten, sogenannte Point of Interests (POIs).

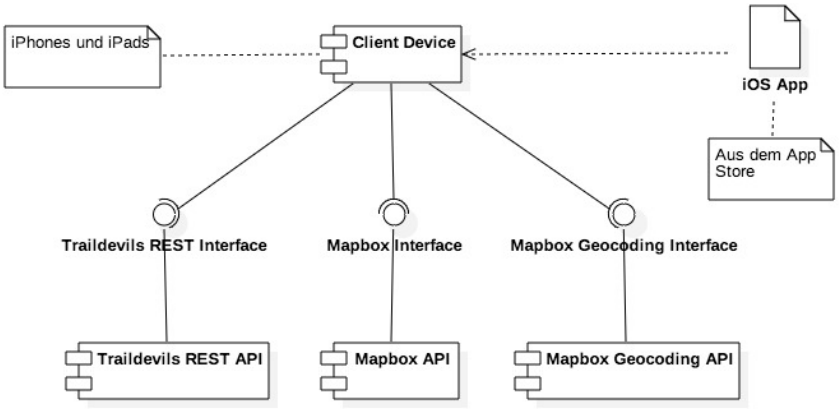


Abbildung 4: Ausgangslage des Systems nach der Studienarbeit [1]

In Abbildung 4 sind die zentralen Komponenten der Architektur vor dem Beginn der Bachelorarbeit dargestellt. Es ist ersichtlich, dass die Applikation drei externe Schnittstellen anspricht.

- Traildevils REST API: Kartenelemente wie zum Beispiel Trails, Tracks, Destination und Dealers
- Mapbox API: Kartendaten (Tiles)
- Mapbox Geocoding API: Suchresultate zu öffentlichem Kartenmaterial

Die aktuelle Systemübersicht am Ende dieser Bachelorarbeit kann dem Kapitel UMSETZUNG entnommen werden.

2 ANALYSE

Der Grundstein für die Traildevils iOS App wurde in der Semesterarbeit im Herbstsemester 2016/17 gelegt. Die stabile Ausgangslage erlaubte es, neue Funktionalität effizient zu integrieren. Gemäss der Aufgabenstellung soll die native iOS Traildevils App mit einem Tourenplaner erweitert werden.

2.1 ANFORDERUNGSSPEZIFIKATION

Der Tourenplaner erlaubt es, dem Benutzer im Restaurant, in öffentlichen Verkehrsmitteln oder von zuhause aus eine Strecke zu planen. Diese Strecke ist eine Zusammensetzung von bestehenden POIs, wie zum Beispiel Trails, Tracks, Dealers oder auch Bergseilbahnen. Zusätzlich soll es die Möglichkeit geben, eigene Global Positioning System (GPS)-Daten durch einen Import in die Planung einzuspeisen.

Die geplanten Strecken sind auf einfache Art und Weise editierbar. Wegpunkte sind austauschbar und es kann aus bestehenden POIs ausgewählt werden.

Die geplante Strecke soll durch einen externen Dienst berechnet werden. Es werden dazu die benötigten Koordinaten über ein Application Programming Interface (API) an den externen Dienst übermittelt. Anschliessend erhält der Benutzer eine komplett geplante Strecke. Diese liefert Zusatzinformationen wie zum Beispiel Höhenmeter, Distanz und Dauer, welche dem Benutzer angezeigt werden.

Geplante Strecken werden auf dem Gerät des Benutzers lokal gespeichert. So kann der Benutzer zu einem späteren Zeitpunkt auf die Strecke zurückgreifen.

2.1.1 *Use Cases*

Nachfolgend werden die Use Cases in den einzelnen Kapiteln im *Brief* Format erläutert. Die Use Cases wurden zusammen mit dem Projektpartner erhoben und von ihm abgenommen.

2.1.1.1 *UC01: Strecke planen mit Kartenelementen*

Eine Strecke besteht aus 2..N Wegpunkten. Der Benutzer kann bei der Planung seiner Strecke auf bereits existierende Kartenelemente zurückgreifen. Dazu gehören Trails, Tracks und Dealers. Über ein Suchfeld kann er die Kartenelmente seiner Strecke hinzufügen oder selektiert sie direkt auf der Karte.

Fügt ein Benutzer einen Track der Planung hinzu, wird der Start- und Endpunkt des Tracks der Strecke hinzugefügt.

Die Strecke soll über eine API eines Drittanbieters berechnet werden. Dabei werden die einzelnen Koordinaten der Strecke dem Planungsdienst übergeben.

2.1.1.2 *UC02: Strecke planen mit POIs*

Bei der Planung stehen auch POIs zur Auswahl, wie zum Beispiel Bergseilbahnen oder Restaurants.

Entweder kann der Benutzer über die in der App verfügbare Suche nach POIs suchen oder setzt ein Placemark, welches über den Geocoding Dienst in eine Adresse mit Koordinaten umgewandelt wird.

Die Koordinaten eines POI werden ebenfalls dem Planungsdienst eines Drittanbieters für die Berechnung übergeben.

2.1.1.3 *UC03: Strecke planen mit eigenem GPS Tracks*

Der Benutzer kann einen eigenen GPS Track im `.gpx` Format importieren und für die Planung einer Strecke verwenden.

Da die Koordinaten eines GPS Tracks bekannt sind, ist für diese Teilstrecke keine Berechnung des Planungsdiensts notwendig.

2.1.1.4 *UC04: Detailansicht zu einer geplanten Strecke abrufen*

Sobald sich der Benutzer im Planungsmodus befindet, kann er Detailinformationen in einer separaten Ansicht aufrufen. In diesen Detailinformationen sind die Wegpunkte ersichtlich. Zusatzinformationen über Distanz, Dauer, Aufstieg und Abstieg können ebenfalls entnommen werden. Eine Karte liefert erneut eine Übersicht über die geplante Strecke. Ein Höhenprofil mit den eingezeichneten Wegpunkten zeigt dem Benutzer visuell den Auf- und Abstieg.

Ein Track des Application Programming Interface des Projektpartners (Traildevils API) verfügt in der Regel über Daten zur Höhe über dem Meeresspiegel (Elevation). Diese wird für die Darstellung des Höhenprofils benötigt. Enthält ein Track keine Informationen zur Elevation, wird diese nicht berücksichtigt.

Liefert der Planungsdienst Alternativen zu einer Strecke, werden diese nicht berücksichtigt.

2.1.1.5 *UC05: Geplante Strecke editieren*

Der Benutzer ist in der Lage, die geplante Strecke zu editieren. Einzelne Wegpunkte können hinzugefügt, gelöscht oder verschoben werden.

Diese Interaktionen werden auf der Karte und in der Detailansicht unterstützt.

2.1.1.6 *UC06: Geplante Strecke lokal speichern*

Damit der Benutzer zu einem späteren Zeitpunkt die geplante Strecken erneut aufrufen kann, wird sie lokal auf seinem Gerät gespeichert.

Wird die Applikation gelöscht, gehen die gespeicherten Strecken verloren.

2.1.1.7 *UC07: Geplante Strecke offline verfügbar machen*

Es wird davon ausgegangen, dass die Strecke im Vorfeld geplant wird. Bei der Abfahrt ist der Benutzer jedoch oft in der Natur und hat je nach Abonnement oder Signalstärke keine mobilen Daten zur Verfügung.

Der Benutzer soll in der Lage sein, eine geplante Strecke offline verfügbar zu machen. Dies betrifft die Daten zur Strecke und das umliegende Kartenmaterial von Mapbox. Auch ohne mobile Daten kann der Benutzer so die Informationen zu einer geplanten Strecke jederzeit konsumieren.

2.1.1.8 *UC08: Geplante Strecke als GPS Track exportieren*

Der Benutzer kann eine geplante Strecke im `.gpx` Format exportieren. So steht die Datei anderen Apps zur Weiterverarbeitung zur Verfügung.

Beispielsweise kann die Datei in die *Dropbox* oder *Google Drive* App exportiert werden.

2.1.2 *Nicht funktionale Anforderungen*

2.1.2.1 *Usability*

Ein grossartiges Benutzererlebnis steht im Mittelpunkt. Das während der Arbeit entwickelte Interaktionskonzept soll es iOS Benutzern ermöglichen, die App zu bedienen.

Durch die Apple Human Interface Guidelines [2] und die Verwendung von User Interface (UI) Patterns erkennt der Benutzer Muster der vertrauten iOS-Umgebung wieder. So wird ein flüssiger Umgang mit der App einfacher.

Die Erfolgsrate ist eine binäre Metrik, welche misst, ob eine Testperson ein Szenario erfolgreich abschliessen kann oder nicht.

Gemäss einem Blog Post von Sauro beträgt die Durchschnitts-Erfolgsrate 78% [3]. Dieser Wert wird für das Messen dieser nicht funktionalen Anforderung verwendet.

ABNAHMETEST: 78% Erfolgsrate in Usability Test

2.1.2.2 *Performance*

Die App muss innerhalb von drei Sekunden gestartet sein und auf Benutzerinteraktionen reagieren. Zudem soll das Interagieren mit einer Route flüssig sein.

ABNAHMETEST: Profiling

2.1.2.3 *Testability*

Die zukünftige Weiterentwicklung und der Erfolg bei der Community ist sowohl im Interesse des Projektpartners, als auch der beiden Autoren. Die Testbarkeit ist deshalb ein wichtiges Kriterium, um dies zu gewährleisten. Gut testbarer Code verbessert gleichzeitig auch die Wartbarkeit. Die Testbarkeit soll durch ein geeignetes Mass an Abstraktion und Dependency Injection erreicht werden.

ABNAHMETEST: 70% Code Coverage

2.1.2.4 *Maintainability*

Die Applikation wird mit einem Developer-Manual ausgeliefert. Dieses Manual beinhaltet die nötigen Installationsschritte und Informationen zur Weiterentwicklung. Ein macOS Gerät ist für die Weiterentwicklung notwendig.

Durch die Entwickler-Dokumentation ist der Projektpartner in der Lage, das Projekt selbständig weiter zu entwickeln. Mit einem Manual werden die nötigen Schritte für das Setup erläutert. Durch die Nutzung von Code Style Guides, Linter und Patterns wird die Wartbarkeit zusätzlich gestärkt und sichergestellt.

Weitere Massnahmen sind dem Kapitel 9 zu entnehmen.

ABNAHMETEST: Developer-Manual, Code Review und SwiftLint

2.1.2.5 *Availability*

Der Einsatz von verschiedenen Diensten birgt auch die Gefahr der Abhängigkeit. Das Kartenmaterial wird von Mapbox verwendet, die Routing API von GraphHopper. Die Verfügbarkeit liegt deshalb in den Händen der Online Dienste. Viel wichtiger ist es, wie auf den Ausfall eines Dienstes innerhalb der App reagiert wird. Eine Meldung, dass der Dienst im Moment nicht zur Verfügung steht, ist notwendig.

ABNAHMETEST: Manueller Test ohne Netzwerk, Manueller Test mit schlechter Verbindung zum Internet

2.1.2.6 *Accessibility*

Gemäss dem Schweizerischen Zentralverein für das Blindenwesen [4] leben in der Schweiz 320'000 sehbehinderte und blinde Personen, was

ungefähr 3% der Schweizer Bevölkerung ausmacht. Die Anzahl sehbehinderter Traildevils Benutzer ist nicht bekannt, trotzdem dürfen diese beeinträchtigten Mitmenschen nicht vernachlässigt werden.

ABNAHMETEST: Test mit Sim Daltonism

2.1.3 *Einschränkungen*

Im Moment wird die Umsetzung ausschliesslich für die iOS Plattform entwickelt. Es ist jedoch nicht ausgeschlossen, dass zu einem späteren Zeitpunkt Applikationen für Android und Windows Geräte umgesetzt werden.

2.1.3.1 *Devices*

Durch die Universal-Applikation wird ein einheitliches Interface ausgeliefert. Trotz möglichem Download auf ein iPad, wird die Ansicht für Tablets nicht speziell optimiert. Im Rahmen dieser Bachelorarbeit wird garantiert, dass die Software auf den folgenden Geräten stabil läuft: iPhone 5, iPhone 5S*, iPhone 6*, iPhone 6 Plus*, iPhone 6S, iPhone 6S Plus*, iPhone 7, iPhone 7 Plus*, iPad mini 2, iPad mini 3*, iPad mini 4*, iPad Air 2*, iPad Pro.

* = nur im Xcode Simulator getestet, nicht auf physikalischen Geräten

2.1.3.2 *iOS Version*

Die Traildevils iOS App wird von allen Geräten unterstützt, welche mindestens iOS Version 10 installiert haben.

2.1.3.3 *Mapbox Offline Daten*

Durch den Einsatz von Mapbox ist das Abspeichern für Offline Daten auf 6'000 Quadratische Ausschnitte der Weltkarte (Tiles) pro User begrenzt [5]. Die Grösse entspricht etwa einer Destination. Beim Download der Daten gibt ein Indikator Auskunft über die bereits geladenen Daten.

2.1.3.4 *GraphHopper Directions API*

Die GraphHopper Directions API ist in unterschiedliche Schnittstellen aufgeteilt. Für die Arbeit sind lediglich die Routing und Geocoding Schnittstellen relevant.

Über einen API-Schlüssel stellt GraphHopper jedem Benutzer ein tägliches Kontingent in Form von Credits zur Verfügung. Jeder Request kostet Credits. Anschliessend eine kleine Übersicht, welcher Dienst wie viele Credits verbraucht [6].

ROUTING API: Ein Request von A nach B kostet ein Credit.

GEOCODING API: Ein Request kostet ein Credit. Dabei wird der Geocoder von GraphHopper verwendet. Über die Schnittstelle ist es auch möglich Resultate, von einem alternativen Anbieter (Nominatim [7] und OpenCage Data [8]) zu beziehen. Diese Requests kosten dann 15 Credits.

Im Rahmen der Bachelorarbeit wurde das *Free* Abo von GraphHopper [9] verwendet. Das *Free* Abo umfasst ein tägliches Kontingent von 500 Credits pro Abo, welches während der Entwicklung nie ausgeschöpft wurden.

Mit einem Release im App Store ist ein Wechsel auf ein höheres Abo unumgänglich. Beispielsweise können mit dem *Standard* Abo (109€ pro Monat) bis zu 15'000 Routen pro Tag berechnet werden [9]. Es ist schwer einzuschätzen, wie wieviele Credits notwendig sind. Gemäss Projektpartner hat Traildevils monatlich 118'000 Besucher, davon sind ca. 60% iOS Benutzer.

2.2 INSPIRATION

Dank den vom Projektpartner und Entwicklungsteam definierten Use Cases ist das gewünschte Ziel klar. Anhand der Use Cases wurde ein Kriterienkatalog definiert, welcher die Lösungen der Konkurrenten analysiert.

Der Schwerpunkt dieser Arbeit liegt auf der Planung einer Route. Dabei wurden Lösungen, welche sich bereits auf dem Markt befinden, nur im Bereich dieses Schwerpunkts analysiert.

Die Analyse soll zudem helfen, den richtigen Weg zu weisen und die Herausforderungen im Bereich UI/UX erfolgreich zu meistern.

2.2.1 *Kriterien*

Die Kriterien werden anhand der Use Cases definiert.

- UC01: Strecke planen mit Kartenelementen
- UC02: Strecke planen mit POIs
- UC03: Strecke planen mit eigenem GPS Tracks
- UC04: Detailansicht zu einer geplanten Strecke abrufen
- UC05: Geplante Strecke editieren
- UC06: Geplante Strecke lokal speichern
- UC07: Geplante Strecke offline verfügbar machen
- UC08: Geplante Strecke als GPS Track exportieren

2.2.2 *Quellen*

2.2.2.1 *Komoot*

Komoot [10] mit Sitz in Potsdam, Deutschland, [11] bezeichnet sich als *führende Outdoor-App* in Europa. Seit der Veröffentlichung im Jahr 2010 begleitet Komoot bereits drei Millinonen Outdoor-Nutzer. Neben Tourenvorschlägen für Wanderer, Fahrradfahrer und Mountainbiker, sind auch Tourenplaner mit Sprachnavigation und hochwertige topografische Offline Karten im Einsatz.

2.2.2.2 *Outdooractive*

Outdooractive [12] mit Hauptsitz in Immenstadt, Deutschland, hat noch drei weitere Niederlassungen in Österreich, Schweiz und Italien [13]. Sie bieten 150'000 Touren an, welche interaktiv auf der Karte bedienbar sind. Auch bei Outdooractive steht der Tourenplaner mit Navigationsmöglichkeit im Mittelpunkt. Die Routen können beliebig angepasst werden, indem Wegpunkte hinzugefügt, verschoben oder gelöscht werden können. Für die Touren werden Zusatzinformationen wie Distanz, Aufstieg, Abstieg, Zeit und Wetter angezeigt und ein Höhenprofil ist vorhanden. Kartenmaterial kann für die Offline-Nutzung lokal auf dem Gerät gespeichert werden. Auch das Aufzeichnen eigener Routen ist in der App möglich.

2.2.2.3 *Google Maps*

Google Maps ist der Kartendienst von Google und bietet dem Anwender die Möglichkeit, das Kartenmaterial von Google zu konsumieren. Neben dem Auswählen, Vergrössern und Verkleinern eines Kartenausschnitts, dem Wechseln des Kartentyps auf Satellitenbilder oder Terrain, kann sogar ein Tisch in einem Restaurant reserviert oder Sehenswürdigkeiten in einer 3D Ansicht angesehen werden [14].

Der Kartendienst bietet noch viele weitere Features: Es lassen sich Routen mit mehreren Wegpunkten zusammenstellen. Beim Planen einer Route stehen dem Benutzer Alternativrouten zur Verfügung. Diese differenzieren sich in Dauer oder Länge.

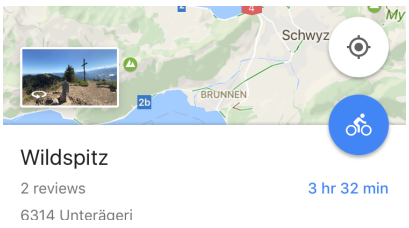
2.2.3 *Auswertung*

2.2.3.1 *Start der Streckenplanung*

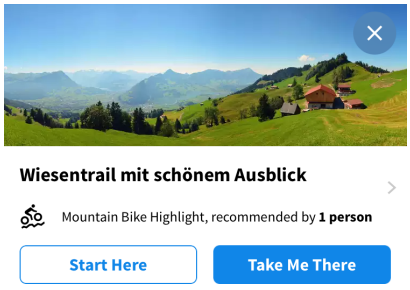
Bei Komoot und Google Maps wird die Streckenplanung über einen zentralen Button gesteuert. Startet der Benutzer die Planung ohne Auswahl eines POIs, wird er auf eine Maske weitergeleitet, in welcher die aktuelle Position als Startpunkt festgelegt ist. Der Benutzer kann in dieser Maske sein gewünschtes Ziel eingeben oder auf über die Karte selektieren.

Speziell bei Komoot ist, dass bei der Auswahl eines POIs die Möglichkeit geboten wird, vom selektierten POI zu starten, oder eine Strecke dorthin zu planen.

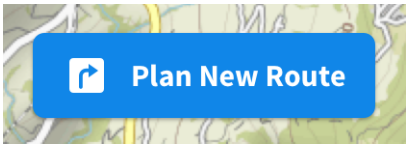
Beim Start der Planung einer Strecke leitet Outdooractive zum Planungsmodus weiter, in welchem nur die Karte sichtbar ist. Es wird gleich auf erweiterte Gesten aufmerksam gemacht, welche für die Interaktion mit dem Planungsassistenten notwendig sind. Beispielsweise wird ein neuer Wegpunkt durch langes Drücken erstellt.



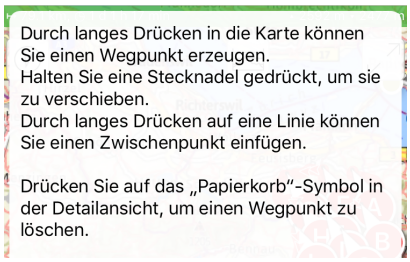
(a) Google Maps: POI mit Button für Streckenplanung



(b) Komoot: Auswahl bei selektiertem POI



(c) Komoot: Button für Streckenplanung

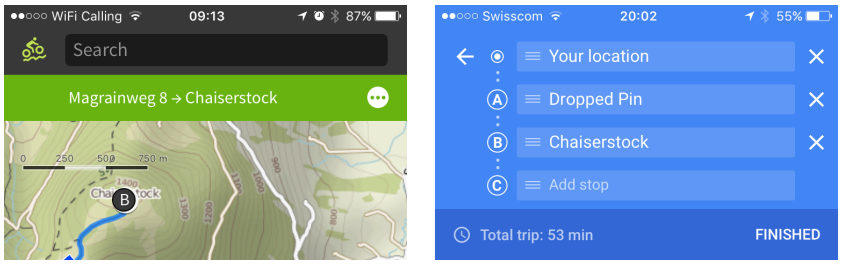


(d) Outdooractive: Anleitung für erweiterte Gesten

Abbildung 5: Startmöglichkeiten für Streckenplanung von Komoot, Google Maps und Outdooractive

2.2.3.2 *Übersicht der Strecke*

Die Übersicht zeigt auf einen Blick die gesetzten Wegpunkte für die Strecke von A nach B. Komoot zeigt im Verlgeich zu Google Maps nur den Start- und Endpunkt an. Google Maps für jeden einzelnen Wegpunkt auf. Komoot und Google zeigen die Übersicht zu den Wegpunkten direkt auf der Karte während der Planung. Outdooractive führt die Liste zu den Wegpunkten in einer Detailansicht auf.



(a) Komoot: Start und Endpunkt (b) Google Maps: Jeden Wegpunkt

Abbildung 6: Übersicht der Strecke von Komoot und Google Maps

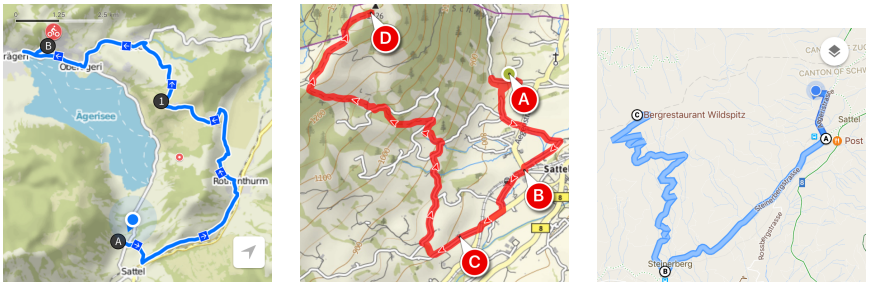
2.2.3.3 *Strecke auf der Karte*

Während der Planung wird die Strecke kontinuierlich mit den gesetzten Wegpunkten erweitert. Komoot nutzt für die Markierung von Start- und Endpunkt die Buchstaben «A» und «B» und dazwischen wird von 1 beginnend nummeriert.

Bei Outdooractive wird ausschliesslich mit Buchstaben gearbeitet. Sind alle Buchstaben vom Alphabet aufgebraucht, beginnt es wieder bei «A».

Google Maps verwendet für die Markierung des Startpunktes ein Icon und Buchstaben für die folgenden Wegpunkte.

Komoot und Outdooractive zeigen dem Benutzer mit Pfeilen auf der Strecke an, in welche Richtung geplant wurde.



(a) Komoot (b) Outdooractive (c) Google Maps

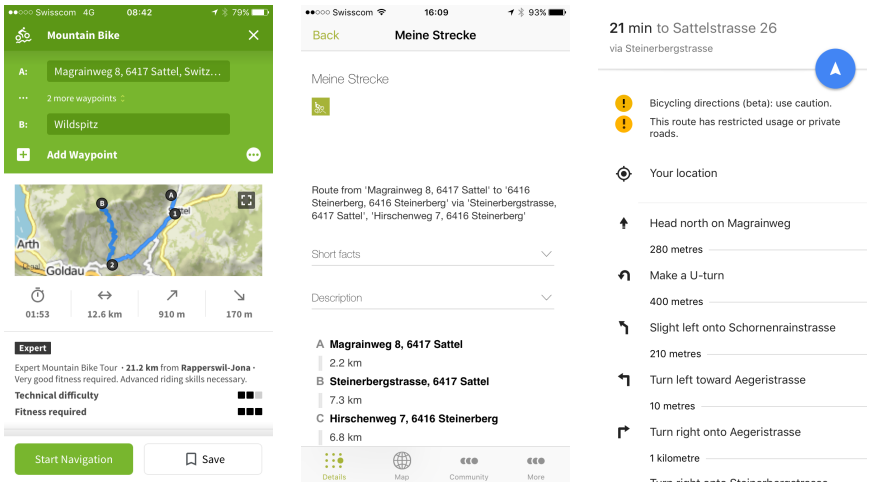
Abbildung 7: Strecke auf der Karte von Komoot, Outdooractive und Google Maps

2.2.3.4 *Details zur Strecke*

Alle untersuchten Anwendungen bieten einen separaten Screen für die Details einer Strecke an. Komoot weist den Screen mit den meisten Informationen auf. Es ist eine Liste mit allen gesetzten Wegpunkten verfügbar. Es können weitere Wegpunkte hinzugefügt, verschoben oder gelöscht werden.

Outdooractive weist ebenfalls eine Zusammenfassung der gesetzten Wegpunkte auf. Im Vergleich zu Komoot können hier keine Wegpunkte editiert werden.

Google Maps zeigt für die Details einer Strecke lediglich die Navigation an.



(a) Komoot (b) Outdooractive (c) Google Maps

Abbildung 8: Details zur Strecke von Komoot, Outdooractive und Google Maps

2.2.3.5 *Wegpunkt verschieben*

Outdooractive ist die einzig Anwendung, welche dem Benutzer das Verschieben eines Wegpunktes ermöglicht. Jeder gesetzte Wegpunkt besitzt eine Stecknadel. Um einen Wegpunkt zu verschieben, kann mit einer Stecknadel den Wegpunkt verschoben werden.

VORTEIL: Der Benutzer sieht während dem Verschieben den Wegpunkt.

NACHTEIL: Je nach Zoomstufe nehmen die Stecknadeln zu viel Platz in Anspruch, was die Ansicht unübersichtlich macht.

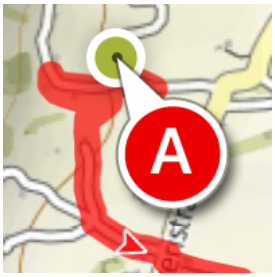


Abbildung 9: Outdooractive: Markierung eines Wegpunktes für das Verschieben

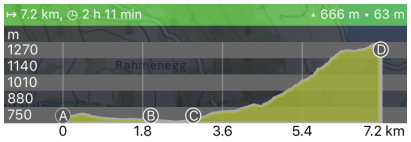
2.2.3.6 *Zusatzinformationen zur Strecke*

Komoot zeigt während der Planung auf der Karte Zusatzinformationen (Distanz, Dauer, Aufstieg und Abstieg) an, welche beim Hinzufügen von Wegpunkten direkt aktualisiert werden.

Outdooractive bietet beim Planen auf der Karte neben Distanz, Dauer, Aufstieg und Abstieg auch ein Höhenprofil an. Alle gesetzten Wegpunkte sind auf dem Höhenprofil ersichtlich.

Google Maps zeigt lediglich die Dauer der geplanten Strecke.

(a) Komoot: Zusammenfassung der geplanten Strecke



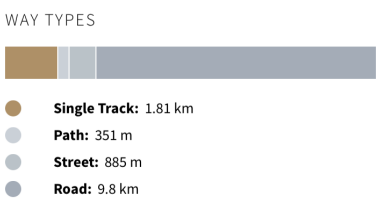
(b) Outdooractive: Höhenprofil direkt auf Karte

(c) Google Maps: Dauer der geplanten Strecke

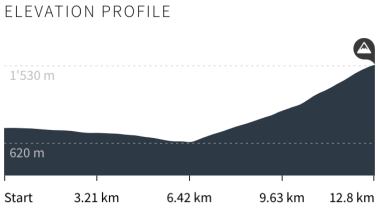
Abbildung 10: Zusatzinformationen zur Strecke von Komoot, Outdooractive und Google Maps

In den Details werden noch mehr Informationen angezeigt; Komoot liefert eine Auswertung über die Wegtypen und die Oberfläche der Strecke. Sie zeigt, wie viele Meter davon zum Beispiel Tracks, Waldwege oder Asphalt sind. Ein Höhenprofil mit Legenden zur X- und Y-Achse liefert Auskunft über den niedrigsten und höchsten Punkt der Strecke. Eine Durchschnittsgeschwindigkeit ist ebenfalls zu entnehmen.

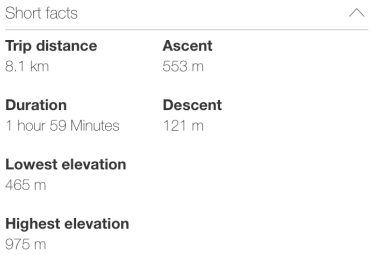
Outdooractive liefert ebenfalls Fakten zur Distanz, Dauer, Aufstieg und Abstieg. Auch ein Höhenprofil mit Legenden zur X- und Y-Achse wird dargestellt. Im Gegensatz zu Komoot ist das Höhenprofil detaillierter abgestuft.



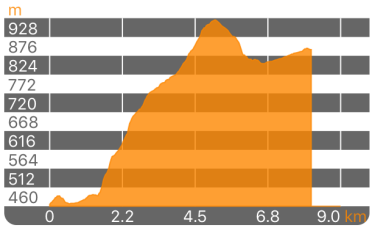
(a) Komoot: Informationen zur Oberfläche



(b) Komoot: Höhenprofil



(c) Outdooractive: Fakten zur Strecke



(d) Outdooractive: Höhenprofil

Abbildung 11: Zusatzinformationen zur Strecke in Details von Komoot und Outdooractive

2.2.4 *Schlussfolgerung*

Durch die Analyse der Konkurrenten konnten viele Eindrücke gesammelt werden. Anschliessend zeigt eine Übersicht, wie gut die einzelnen Kriterien von den verglichenen Anwendungen gelöst wurden. Es wurde ein Indikator zwischen eins und drei vergeben, wobei eins *schlecht* und drei *gut* bedeutet. Null bedeutet, dass die Funktion nicht vorhanden war.

USE CASE	KOMOOT	OUTDOORACTIVE	GOOGLE MAPS
UC01	3	0	3
UC02	3	0	3
UC03	0	0	0
UC04	3	1	1
UC05	2	2	2
UC06	3	3	0
UC07	3	0*	0**
UC08	0	2	0

Tabelle 1: Übersicht welche Anwendungen welchen Use Case abdeckt.

* = In der Premium Version verfügbar, ** = Es kann nur ein Kartenausschnitt offline verfügbar gemacht werden

2.3 PROTOTYP

Ein Prototyp soll helfen, das entwickelte Interaktionskonzept zu validieren.

Aus dem Human Computer Interaction Design (HCID) Modul der Hochschule für Technik, Rapperswil (HSR) ist bekannt, dass sich Prototypen in fünf Dimensionen unterscheiden lassen [15]:

- Visuelle Ausarbeitung und Detaillierungsgrad
- Breite
- Tiefe
- Interaktivität
- Daten im Prototyp

In einem iterativen Prozess wurde ein Prototyp entwickelt. Wie die oben genannten Dimensionen berücksichtigt wurden, sind den folgenden Unterkapiteln zu entnehmen.

2.3.1 *Phase 1: Handskizzen*

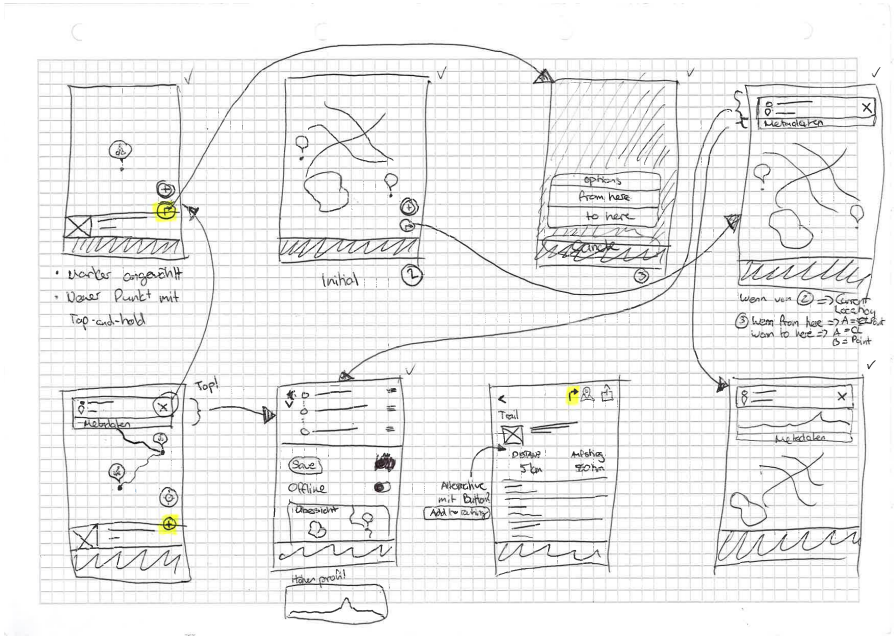


Abbildung 12: Erste Screens in Form von Handskizzen

Aus der Analyse der Quellen wurden die besten Ideen mitgenommen, ausgearbeitet und mit ersten Handskizzen festgehalten. Die Skizzen visualisieren die Vision und *Breite* des Prototyps.

In einer Besprechung mit dem Projektpartner wurden die Ideen diskutiert und abgenommen.

2.3.1.1 *Planung starten*

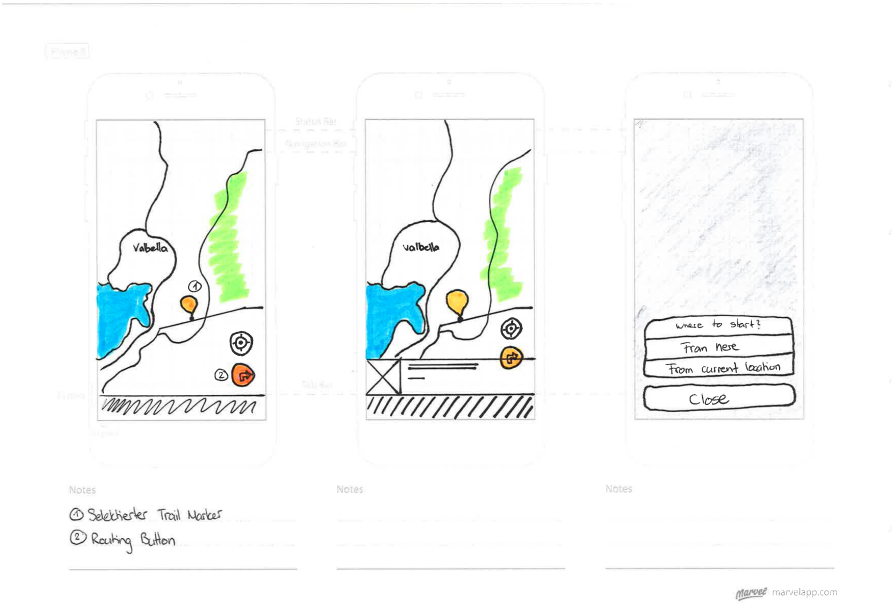


Abbildung 13: Screens zum Starten einer Tourenplanung

SCREEN 1: Im ersten Screen ist die BikeMap mit einem Routing Button ersichtlich. Nach Betätigung des Buttons, wird der Planungsmodus aktiviert. Da kein Kartenelement selektiert ist, wird der Planungsmodus mit der aktuellen Position vom Benutzer als Startpunkt gesetzt.

SCREEN 2: Der Benutzer hat bereits ein Kartenelement selektiert. Das Callout wird oberhalb der Tab Navigation ersichtlich. Auch hier steht dem Benutzer ein zentraler Routing Button zur Verfügung, mit welchem die Planung gestartet werden kann.

SCREEN 3: Dieser Screen erscheint, wenn der Benutzer aus dem *zweiten Screen* den Planungsmodus startet. Da zu diesem Zeitpunkt nicht genau gesagt werden kann, von wo der Benutzer gerne seine Route planen möchte, erscheint ein Auswahl, welches den Benutzer danach fragt.

Zur Auswahl stehen dem Benutzer zwei Optionen:

1. Von der aktuellen Position des Benutzers zum ausgewählten Kartenelement planen
2. Vom ausgewählten Kartenelement Planung starten

2.3.1.2 *Details ansehen*

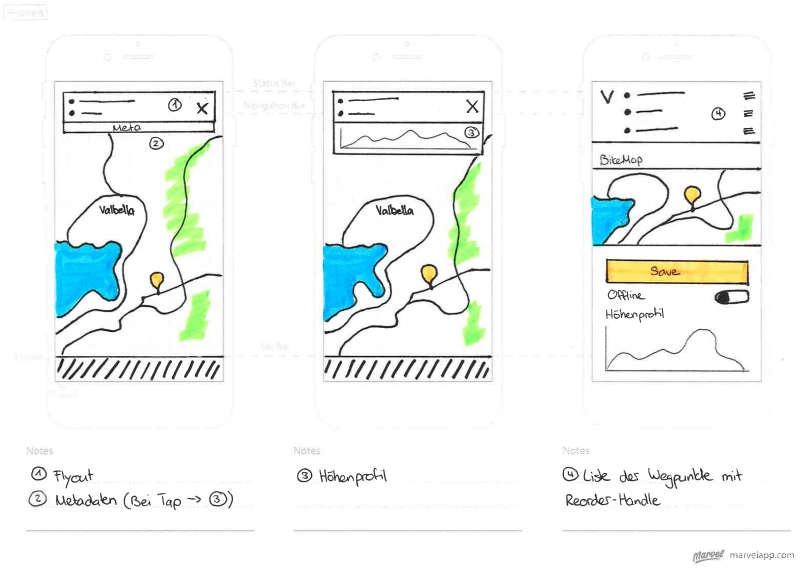


Abbildung 14: Screens zum Anschauen der Zusatzinformationen

SCREEN 1: Befindet sich der Benutzer im Planungsmodus, erscheint im oberen Screen das sogenannte *Flyout*. Im *Flyout* sind Start- und Endpunkt ersichtlich. Unterhalb von diesem Element, werden Zusatzinformationen zur Dauer, Distanz, Auf- und Abstieg aufgeführt. Mit dem Kreuz rechts im *Flyout* wird der Planungsmodus beendet.

SCREEN 2: Mit einem Tap auf die Zusatzinformationen wird das Höhenprofil dargestellt.

SCREEN 3: Mit einem Tap auf das *Flyout* gelangt der Benutzer zu den Details der geplanten Strecke. Dort wird jeder einzelne Wegpunkt der Strecke dargestellt. Diese können mit dem gewohnten Steuerungselement für das Verschieben einer Zelle in einer Table View (Reorder Control) verschoben werden. Oben links lassen sich die Details mit dem Pfeil nach unten wieder schliessen. Die geplante Strecke wird auf der Karte visualisiert.

Mit einem zusätzlichen Button lässt sich die Strecke lokal auf dem Gerät speichern. Es wird dem Benutzer zusätzlich die Möglichkeit geboten, auch das dazugehörige Kartenmaterial offline zu speichern.

Zum Abschluss des Screens wird das Höhenprofil der Strecke in einer grösseren Ansicht aufgeführt.

2.3.1.3 *Strecke bearbeiten*

Abbildung 15: Screens zum Bearbeiten einer Tour

SCREEN 1: Um neue Wegpunkte der Strecke hinzuzufügen, wählt der Benutzer ein Kartenelement zum Beispiel Trail aus und fügt diesen mit dem Plus-Button unten rechts der Strecke hinzu. Anschliessend wird die Strecke mit dem neuen Wegpunkt erweitert.

SCREEN 2: Möchte der Benutzer einen Wegpunkt wieder entfernen, betätigt er den Minus-Button.

SCREEN 3: Der Benutzer ist auch in der Lage ein Kartenelement, zum Beispiel Trail oder Dealer in seinen Details der Strecke hinzuzufügen. Dies geschieht über ein weiteres Icon in der *Navigation Bar*.

2.3.2 *Phase 2: Proto.io*

In einer zweiten Phase wurden die Handskizzen digitalisiert und mit Proto.io zu einem klickbaren Prototypen erweitert. Durch diesen Prototyp konnten statische Screens zum Leben erweckt werden. Interaktionen mit Elementen, wie zum Beispiel mit dem *Routing Button*, dem *Flyout* oder das Hinzufügen eines Startpunkt einer Abfahrt von Traildevilss (Trails) zur Route, wurden ermöglicht. Mit diesen Funktionalitäten konnten *Tiefe* und *Interaktivität des Prototyps* demonstriert werden.

Das interne Testing mit dem Betreuer und dem Projektpartner ermöglichte es, die Screens weiter zu optimieren.

Damit ein Usability Test überhaupt durchgeführt werden konnte, mussten Interaktionen mit der Karte, wie zum Beispiel Zooming und Panning, für den Benutzer ermöglicht werden. Mit diesem Prototypen konnte dies nicht erreicht werden, weil lediglich statische Bilder angezeigt wurden. Deshalb wurde in einer dritten Phase ein In-App Prototyp erstellt.

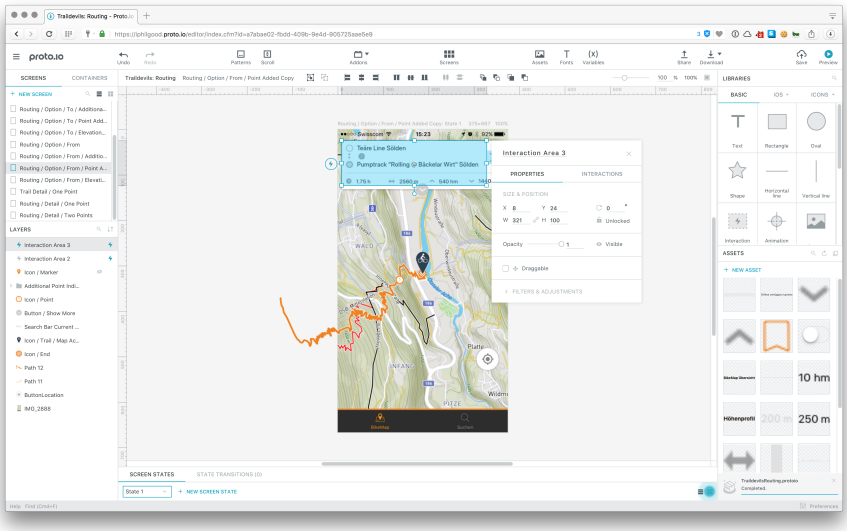


Abbildung 16: Benutzeroberfläche von Proto.io

2.3.3 *Phase 3: In-App*

Aufbauend auf der iOS App aus der Studienarbeit wurde innerhalb von zwei Wochen die Basisfunktionalität implementiert. Die optimierten Screens aus dem Proto.io Prototyp dienten als Grundlage.

Durch die direkte Implementation in der iOS App konnten von den bereits existierenden Kartenelementen von Traildevils Gebrauch gemacht werden. Dieser In-App Prototyp ermöglichte die Interaktionen mit der Karte. Zudem konnten so zum Beispiel Trails der Planung hinzugefügt werden.

Die *visuelle Ausarbeitung und der Detaillierungsgrad* waren dadurch höher und die *Daten im Prototyp* waren ebenfalls vorhanden. Der entstandene In-App Prototyp wies somit alle Dimensionen eines Prototyps auf.

Um das Interaktionskonzept in einem Usability Test zu validieren, war ein solcher Prototyp zwingend notwendig.

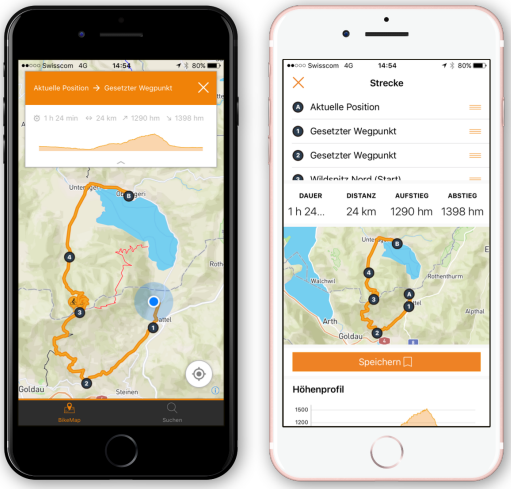


Abbildung 17: Screens aus dem In-App Prototyp

2.4 USABILITY TESTS

Mit zwei Usability Tests soll das entwickelte Interaktionskonzept validiert und sowohl mit iOS als auch Android Anwendern getestet werden. Weil eine zukünftige Android Applikation nicht ausgeschlossen ist, soll das Interaktionskonzept für beide Fraktionen validiert werden. Die Interaktionen, welche für das Planen einer Route notwendig sind, sind unabhängig vom Betriebssystem. Wird in Zukunft eine Android App entwickelt, kann das Interaktionskonzept einfach übernommen werden.

2.4.1 *Erster Test*

Im ersten Usability Test soll überprüft werden, ob die Benutzer in der Lage sind, Interaktionen mit dem Tourenplaner zu tätigen. Die Interaktionen beinhalten das Starten und Beenden des Planungsmodus, das Hinzufügen, Löschen und Verschieben von Wegpunkten aber auch das Entnehmen von Zusatzinformationen zum Beispiel Dauer, Distanz, Auf- und Abstieg.

2.4.1.1 *Rollen*

Im Tourenplaner der Traildevils iOS gibt es keine verschiedene Rollen. Somit wurde nur die Rolle als *Mountainbiker* geprüft.

2.4.1.2 *Testform*

Die bestehenden Traildevils iOS App wurde mit dem Tourenplaner erweitert. So konnte für den Usability Test direkt ein In-App Prototyp entwickelt werden. Die Testpersonen konnten auf die bestehenden Kartenelemente zurückgreifen.

2.4.1.3 *Metriken*

Als Metriken wurden die Zeit pro Szenario und die Erfolgsrate aus Sicht der Instruktoren gemessen.

2.4.1.4 *Testszenarien*

Die Szenarien zum ersten Test sollen die Testpersonen auf Herz und Nieren prüfen. Diese Testform brachte die Personen, wie auch den entwickelten Tourenplaner, an den Anschlag. Genau so konnten positive wie auch negative Eindrücke wahrgenommen werden.

2.4.1.5 *Testpersonen*

Für den Usability Test wurden acht Testpersonen rekrutiert. Davon sind sechs Testpersonen begeisterte Mountainbiker und damit Teil

der Zielgruppe. Eine Person liebt es zu wandern und ist oft in der Natur. Vier Testpersonen sind im Alter zwischen 23 und 32 Jahren und eine Person liegt mit 48 Jahren etwas über dem Altersdurchschnitt der Community. Fünf Testpersonen sind iOS Benutzer und drei sind Android Benutzer.

2.4.1.6 *Testvorbereitungen*

Der Usability Test fand am 13. April 2017 (KW15) im Büro der Frontline Media GmbH statt. Dabei wurde ein iPhone 6S und ein iPhone 7 zur Verfügung gestellt, auf welchen die Szenarien durchgearbeitet wurden. Der Screen der Testperson wird durch die integrierte Software von Lookback aufgezeichnet. Das Mikrofon zeichnet zusätzlich die Audiosignale auf und die Frontkamera des iPhones filmt die Testperson während des Tests.

2.4.1.7 *Prototyp testfähig machen*

Für den Usability Test soll der In-App Prototyp verwendet werden. Die App wurde lediglich mit dem Lookback Software Development Kit (SDK) erweitert, was den Selbsttest ermöglichte. So konnten jegliche Interaktionen der Testperson mit der App festgehalten werden.

2.4.1.8 *Durchführung der Tests*

Eine Veröffentlichung von Nielsen trägt den Namen *First Rule of Usability? Don't Listen to Users*:

> *To design the best UX, pay attention to what users do, not what they say. Self-reported claims are unreliable, as are user speculations about future behavior. Users do not know what they want.* – Nielsen [16]

Genau deshalb wurde während dem Test grossen Wert auf die Beobachtungen gelegt. Zur Dokumentation des Tests wurde das *Rainbow Spreadsheet* [17] verwendet. Kollaborativ wurden Beobachtungen, Bemerkungen oder Probleme der Testperson in das Spreadsheet aufgenommen.

Jeder Testperson wird während dem Test eine Farbe zugewiesen. Trifft eine Beobachtung auf mehrere Testpersonen zu, bekommt die Spalte der Testperson die Farbe zugewiesen.

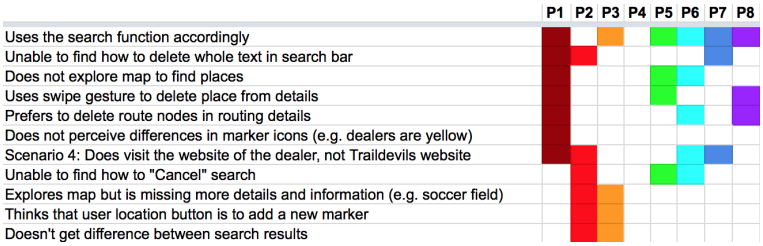


Abbildung 18: Auszug des verwendeten *Rainbow Spreadsheets*

2.4.1.9 *Auswertung der Testergebnisse*

Die Beobachtungen gaben aufschlussreiche Einblicke in die Bedienung der Traildevils iOS App. Dank dem eingesetzten *Rainbow Spreadsheet* konnten nach jeder Session Verbesserungsvorschläge mit dem Probanden ermittelt werden. Die Hauptprobleme sind in der folgenden Auflistung ersichtlich:

GEOCODER: Der initial eingesetzte Geocoder von Mapbox lieferte vor allem beim Suchen nach Bergen nicht die gewünschten Resultate. Der Geocoder von GraphHopper soll dies ändern.

INTERAKTIVE POIS Leider ist es nicht möglich, direkt mit den Kartenelementen von Mapbox, zum Beispiel Talstationen oder Bushaltestellen, zu interagieren. Das führte dazu, dass Testpersonen repetitiv auf den Bildschirm tappten. Der Vorgang wurde verbessert, indem mit einmaligem Tap auf der Karte ein Placemark gesetzt wird.

HILFE FÜR ERWEITERTE GESTEN: Der Test hat gezeigt, dass die Probanden leider keine Möglichkeit gefunden haben, die Route auf der Karte zu editieren. Sei dies mit einem Tap auf einen Wegpunkt um ihn zu löschen oder mit Tap-and-Hold um einen Wegpunkt zu verschieben. Eine Hilfestellung beim ersten Routing soll den Benutzern helfen, diese Interaktionen zu entdecken.

FARBEN DER TRACKS: Wenn bestehende Tracks der Route hinzugefügt werden, sind sie anschliessend nicht einfach zu erkennen. Um dieses Problem zu beheben, sollen Tracks innerhalb der Route ihre Farbe behalten.

PFEILE AUF DER ROUTE: Die Route zeigt mit Pfeilen dem Benutzer die Richtung an. Die Pfeile waren zu klein, damit Sie für den Benutzer erkennbar waren. Die Pfeile sollen grösser werden.

ICON ZUM SCHLIESSEN: Das Schliessen des Planungsmodus und der Details zur Planung passiert über ein Kreuz. Die Benutzer waren verwirrt, dass für unterschiedliche Aktionen die gleichen Icons verwendet wurden. Um dieses Problem zu beheben, werden die Details nicht mit einem Kreuz geschlossen, sondern mit einem Pfeil, der nach unten zeigt.

Der erste Usability Test hat gezeigt, dass die Testpersonen mit dem Planen einer Route keine Probleme hatten. Die einzelnen Interaktionen mit Wegpunkten und der Route sind noch zu wenig klar.

Die durchschnittliche Erfolgsrate der Testpersonen beträgt *64.5%*. Dies liegt unter dem genannten Schnitt von 78%. Vor allem die ersten beiden Szenarien hatten den Testpersonen Schwierigkeiten gemacht.

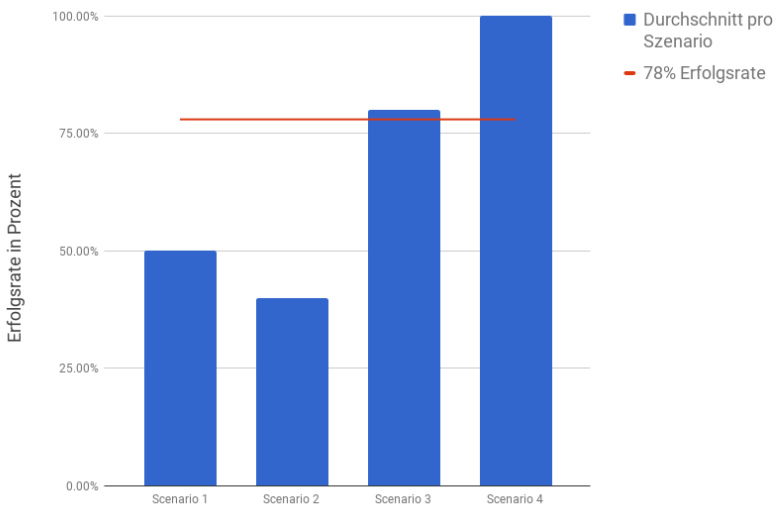


Abbildung 19: Durchschnittliche Erfolgsrate pro Szenario des ersten Usability Tests

2.4.2 *Zweiter Test*

Die Vorgehensweise im ersten Test hat sich bewährt. Nach den oben erwähnten Anpassungen wurde der Tourenplaner mit weiterer Funktionalität ausgestattet. Dazu gehört das Speichern und Umbenennen einer Route, das Offline verfügbar machen des Kartenmaterials und das Importieren sowie Exportieren einer Route.

Die Szenarien fielen in diesem Test etwas kürzer aus, da primär Interaktionen validiert wurden.

Für den zweiten Test wurden vier iOS und vier Android Benutzer eingeladen. Alle im Alter zwischen 24 und 32 Jahren. Die Tests fanden hauptsächlich an der HSR statt. Teilweise wurden Tests auch bei den jeweiligen Testpersonen zuhause durchgeführt.

2.4.2.1 *Auswertung der Testergebnisse*

HILFESTELLUNG: Für die erweiterten Gesten (Tap und Tap-and-Hold) wurde eine Hilfestellung entwickelt, welche beim ersten Routing dem Benutzer als Unterstützung dienen soll. Es hat sich gezeigt, dass die Benutzer die Hilfe zwar wahrnehmen, aber sich zu einem späteren Zeitpunkt nicht mehr daran erinnern konnten. Ein erneuter Hinweis der Entwickler führte dazu, dass die Wegpunkte anschliessend getapped und verschoben wurden.

Es wurde klar, dass auch die Interaktion mit der Route selber erläutert werden muss.

Eine ausführliche Einführung der Gesten soll es dem Benutzer ermöglichen, die erweiterten Gesten auf der Karte erfolgreich anzuwenden.

SPEICHERN: Beim Speichern der Route wurde auf die aktuelle Netzwerkverbindung Rücksicht genommen. Befindet sich der Benutzer in einem Wireless Local Area Network (WLAN), wird beim Speichern der Route automatisch der Download für das Kartenmaterial aktiviert. Es erscheint eine Fortschrittsanzeige, welche Auskunft über den aktuellen Download liefert. Vielen Testpersonen war nicht genau bewusst, was heruntergeladen wird. Mit kritischem Hinterfragen wurde dann klar, dass neben der Route auch das dazugehörige Kartenmaterial heruntergeladen wird. Hat der Benutzer keine WLAN Verbindung, passiert der Download nicht automatisch.

Um den Download zu verdeutlichen, müssen die dazugehörigen Labels verbessert werden. Auch die Fortschrittsanzeige muss für eine deutlichere Wahrnehmung prominenter werden.

NAME DER ROUTE: Beim Speichern der Route wird automatisch der Name *Route* vergeben. Bei mehrfachem Speichern führt dies zu einer Verwirrung, weil mehrere Routen den Namen *Route* tragen, wenn sie nicht umbenannt werden. Um dies zu verhindern, soll beim Speichern dem Benutzer die Möglichkeit geboten werden, seine Route zu benennen. Zusätzlich wird ein generischer Namen mit Start- und Endpunkt vergeben, wie zum Beispiel «Von Rapperswil nach Hinwil».

IMPORT UND EXPORT: Der Import und Export hat sehr gut funktioniert. Den iOS Benutzer war bewusst, dass dies über die Sharing Funktion von iOS passieren muss. Die Android Benutzer wollten hingegen direkt auf das Dateisystem vom Testgerät greifen. Dies ist auch berechtigt, denn unter Android ist ein Zugriff darauf möglich. Aufgrund der App-Sandbox unter iOS ist dies aber nicht möglich. Mit Erforschen der übrigen Möglichkeiten auf dem Screen haben es jedoch auch die Android Benutzer erfolgreich gemeistert.

Die durchschnittliche Erfolgsrate der Testpersonen beträgt *74.3%*. Dies liegt leider immer noch unter dem genannten Schnitt von 78%. Auch hier machten die ersten beiden Szenarien Schwierigkeiten.

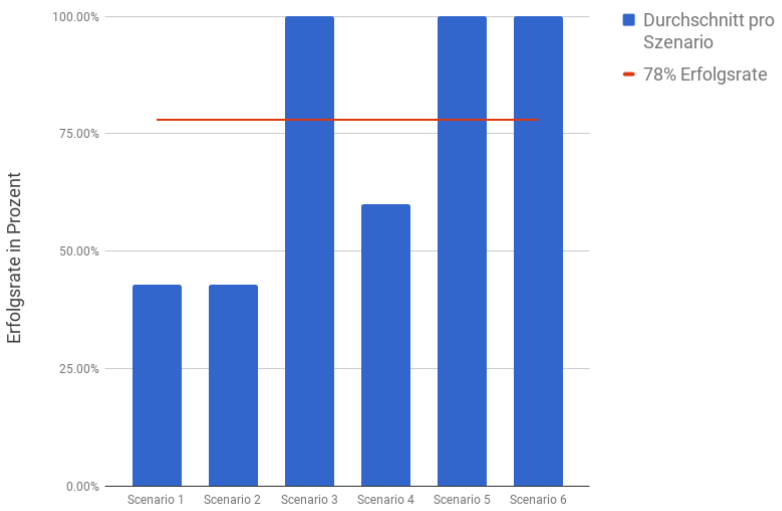


Abbildung 20: Durchschnittliche Erfolgsrate pro Szenario des zweiten Usability Tests

2.4.2.2 *Epilog: Metriken*

Wie im Unterunterabschnitt 2.4.1.3 erwähnt, wurden Metriken zur Erfolgsrate und aufgewendeten Zeit zum Lösen eines Szenarios gemessen. Eine Selbsteinschätzung zur Befriedigung hätte interessante Vergleiche zeigen können. Der Verzicht auf das Sammeln dieser Metrik war ein Fehler.

In einer erneuten Recherche nach den Usability Tests wurde ein interessanter Artikel von Lewis entdeckt. Für eine psychometrische Evaluation beschäftigte sich der Autor mit dem Sammeln von Metriken zur Befriedigung [18]. Interessant an der Evaluation war der dafür entwickelte After-Scenario Questionnaire (ASQ). Dieser Fragebogen bestand aus drei Fragen, welche nach jedem Szenario von der Testperson beantwortet werden musste.

Alle drei Fragen erschienen in Form einer Skala von eins bis sieben, wobei eins *strongly agree* und sieben *strongly disagree* bedeutete. Auch eine Option *not applicabale* wurde dem Benutzer angeboten.

Anschliessend eine Übersicht der Fragen:

FRAGE 1: Overall, I am satisfied with the ease of completing the tasks in this scenario.

FRAGE 2: Overall, I am satisfied with the amount of time it took to complete the tasks in this scenario.

FRAGE 3: Overall, I am satisfied with the support information (online help, messages, documentation) when completing the task.

Somit wurde die Befriedigung, aufgewendete Zeit und die Hilfestellung als Usability Charakteristiken ermittelt.

Ausserdem erwähnte Sauro und Dumas in einem Bericht [19], dass das unmittelbare Beantworten der Fragen nach dem Praktizieren des Szenarios die Validität erhöht und diagnostische Informationen über Usability Issues gesammelt werden können.

Lewis ermutigt alle diese Art von Fragebogen für einen Usability Test zu verwenden. Aus diesem Grund werden die Test-Skripte für einen zukünftigen Test angepasst.

2.5 EVALUATION PLANUNGSDIENST

Die Strecke soll über einen externen Planungsdienst berechnet werden. Dafür gilt es deinen Dienst zu evaluieren. Mit einem Kriterienkatalog werden drei der meist verbreitetsten Planungsdienste verglichen. Jedes Kriterium wird mit einem Indikator von eins bis drei bewertet, wobei eins der schlechteste und drei der beste Wert ist. Zum Schluss werden die Punkte addiert und der Service mit der höchsten Punktzahl wird für die Berechnung der Route verwendet.

Untersucht werden die Dienste von *Mapbox* [20], *GraphHopper* [21] und *OpenRouteService* [22].

2.5.1 *Kriterien*

Um die Projektanforderungen zu erfüllen, wurde der folgende Kriterienkatalog erarbeitet. Die einzelnen Kriterien werden in den nächsten Unterkapiteln erläutert.

2.5.1.1 *Biking Modus*

Der Planungsdienst soll einen Modus anbieten, bei dem Strecken für Mountainbiker berücksichtigt werden. Dazu gehören Wald-, Wander- und Feldwege. Ein Dienst, welcher nur mit Verkehrsstrassen Routen berechnet ist zwecklos.

MAPBOX: Es wird ein *Cycling* Modus angeboten [23].

GRAPHHOPPER: Hier werden mehrere Modi angeboten. Es wird zwischen *Bike*, *Mountain Bike* und *Racing Bike* unterschieden [24].

OPENROUTESERVICE: Auch dieser Dienst bietet mehrere Modi an. Neben *Cycling Regular*, *Cycling Road*, *Cycling Safe*, *Cycling Mountain* und *Cycling Tour* ist es möglich, den Modus *Cycling Electric* anzugeben [25].

2.5.1.2 *Zusatzinformationen*

Die geplante Strecke soll Zusatzinformationen über Dauer, Distanz, Auf- und Abstieg enthalten. Mit den Werten Auf- und Abstieg kann ein Höhenprofil generiert werden.

MAPBOX: Über ein zusätzliches Surface API [26] können Informationen über Elevation und Oberfläche abgerufen werden. Das API befindet sich noch in *private Beta*.

GRAPHHOPPER: Der Dienst bietet eine Option an, welche beim Absenden vom Request mitgegeben werden kann. Anschliessend sind die Daten zur Elevation in der Antwort vom Dienst enthalten [24].

OPENROUTESERVICE: Auch hier kann ein eine Option aktiviert werden, welche einem Request mitgegeben werden kann. Anschliessend sind die Daten zur Elevation in der Antwort vom Dienst enthalten [25].

2.5.1.3 *Kosten/Lizenzen*

Viele der Planungsdienste verlangen ab einer gewissen Schwelle Geld für die Verwendung der Schnittstelle. Maximale Anzahl von Requests pro Minute an den Service, Kosten und Lizenzen werden ebenfalls berücksichtigt.

MAPBOX: Im *Free* Abo können Routen mit bis zu 25 Wegpunkten pro Request berechnet werden. Die Anzahl Requests ist auf 60 pro Minute begrenzt. Es steht auch ein *Enterprise* Abo zur verfügung, in welchem man mehr als 25 Wegpunkte pro Request berechnen kann. Die Anzahl Requests kann je nach Bedürfnis angepasst werden. [27]

GRAPHHOPPER: Die *Routing API* ist Teil der *Directions API*, welche im *Free* Abo genutzt werden kann. GraphHopper stellt den Verwendern der API ein tägliches Kontingent zur Verfügung. Ein Request an die *Routing API* kostet ein Credit. Neben dem *Free* Abo ist ein *Basic*, *Standard*, *Premium* oder ein komplett anpassbares *Custom* verfügbar.

OPENROUTESERVICE: Im *Free* Abo sind 500 Requests pro Tag imbegriffen. Pro Minute sind nicht mehr als 40 Requests möglich. Ein kommerzielles Abo wird nicht angeboten.

2.5.1.4 *Native API*

Der Planungsdienst soll ein natives API anbieten, über welches die Schnittstelle angesprochen werden kann.

MAPBOX: Ein natives API wird zur Verfügung gestellt [28].

GRAPHHOPPER: Dieser Dienst verfügt über kein natives API.

OPENROUTESERVICE: Dieser Dienst verfügt über kein natives API.

2.5.2 *Vergleich*

Anschliessend die Evaluationstabelle. Es wurde ein Indikator zwischen eins und drei vergeben, wobei eins *schlecht* und drei *gut* bedeutet. Null bedeutet, dass die Funktion nicht vorhanden war.

KRITERIUM	MAPBOX	GRAPHHOPPER	OPENROUTESERVICE
Biking Modus	1	2	3
Zusatzinformationen	2	3	3
Kosten / Lizenzen	3	3	1
Native API	3	0	0
Total	9	8	7

Tabelle 2: Evaluation des Planungsdienstes

Mapbox weist die höchste Zahl auf. Die *Directions API* wird somit als Planungsdienst verwendet. Entscheidend war das vorhandene native API.

3 UMSETZUNG

Die Implementation des Routing-Modus wurde bereits früh in Angriff genommen, um so schnell wie möglich einen funktionsfähigen In-App Prototypen zu haben. Denn nur mit einem solchen Prototypen konnten auch realitätsnahe Usability Tests durchgeführt werden.

3.1 SYSTEMÜBERSICHT

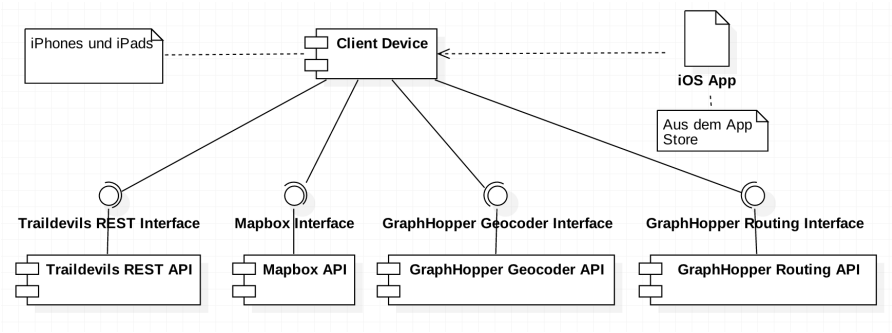


Abbildung 21: Systemübersicht am Ende der Arbeit

Wie in der Abbildung 21 ersichtlich, haben sich die Schnittstellen gegenüber der Ausgangslage in Abbildung 4 verändert. Der Geocoder von Mapbox wurde mit dem von GraphHopper ausgetauscht und neu hinzugekommen ist die Routing API Schnittstelle von GraphHopper. Beide Schnittstellen werden über die in der Bachelorarbeit erstellten Frameworks *GraphHopperGeocoder* und *GraphHopperRouting* angesprochen.

3.1.1 *Planungsdienst und Geocoder*

Warum der Planungsdienst und Geocoder von *Mapbox* nicht mehr in der Systemübersicht ersichtlich sind, ist dem nächsten Unterkapitel zu entnehmen.

3.1.1.1 *Planungsdienst*

Die Evaluation im Abschnitt 2.5 zeigte, dass die *Directions API* von *Mapbox* verwendet werden sollte. Während den internen Tests hat sich herausgestellt, dass die berechneten Routen des Dienstes nicht konform für Mountain Biker waren. Es wurden vielfach Strassen anstelle von Wander- und Feldwegen bevorzugt.

Aus diesem Grund wurde für den Planungsdienst von *GraphHopper* ein Framework entwickelt, um die Kommunikation mit dem API zu ermöglichen. Der neue Dienst bietet explizit einen *Mountain Bike* Modus an.

3.1.1.2 *Geocoder*

Aufgrund der Auswertung des ersten Usability Tests im Unterunterabschnitt 2.4.1.9 musste auch der Geocoder ausgewechselt werden. Der Geocoder von *Mapbox* lieferte vor allem bei der Suche nach Bergen keine Resultate. Anhand der eingesetzten Szenarien im ersten Usability Test wurde der Geocoder von *GraphHopper* geprüft. Alle Suchresultate, welche von den Szenarien verlangt wurden, konnten gefunden werden.

Deshalb wurde auch für die Geocoder API von *GraphHopper* ein Framework entwickelt. Nähere Details können dem Kapitel Frameworks entnommen werden.

3.2 ARCHITEKTUR

Auf dem Gebiet der Architektur ist bekannt, dass Menschen zuerst Gebäude gestalten und danach die Gebäude den Menschen gestalten.

Dieser Ansatz gilt auch für die Architektur im Software Engineering. Es ist wichtig, Code so zu gestalten, dass jedes Teil einer Software leicht identifizierbar ist, einen bestimmten und offensichtlichen Zweck hat und mit anderen Teilen in einer logischen Weise zusammenpassen. *Das* ist Softwarearchitektur.

Um das entwickelte Interaktionskonzept zu validieren war ein In-App Prototyp notwendig. Dies bedeutete, dass die erhobene Funktionalität aus den Anforderungen in die iOS App implementiert werden musste.

Mit Überzeugung starteten die Autoren mit der Entwicklung, bis sie auf eine Problematik stiessen. Um welche Problematik es sich hielt, ist den nächsten Unterkapiteln zu entnehmen.

3.2.1 *Prolog: VIPER*

Die in der Studienarbeit eingesetzte View-Interactor-Presenter-Entity-Router (VIPER)-Architektur war ein neuer Ansatz, um eine iOS Applikation aufzubauen. Jede dieser einzelnen Teile hat eine bestimmte Aufgabe, sei es das Anzeigen von Daten (View), Interaktionen entgegennehmen (Interactor) und weiterleiten (an den Presenter), usw. Alle Teile zusammen bilden eine Komponente, welche wiederum Teil von weiteren Komponenten einer App sind.

Zwischen dem Ende der Studienarbeit und dem Beginn der Bachelorarbeit wurden weitere Features implementiert und Fehler be-

hoben. Schnell kristallisierte sich heraus, dass der Einsatz der VIPER-Architektur nicht nur seine Vorteile hat, sondern das Umsetzen von weiteren, kleinen Features eine *Dateischlacht* zur Folge hatte. Für jedes kleine Feature musste eine Komponente erstellt werden, welche in die oben genannten Teile aufgeteilt ist.

3.2.2 *MVC aka Model-View-Controller*

Obschon die VIPER-Architektur seine Daseinsberechtigung hat und speziell in umfangreichen iOS Apps eingesetzt wird [29], wurde sie nach kurzer Besprechung mit dem Auftraggeber vor dem Beginn der Bachelorarbeit verworfen und durch die von Apple empfohlene Model-View-Controller (MVC)-Architektur ersetzt. Mit der Umstellung zu MVC konnten über 3'000 Zeilen Code eingespart werden.

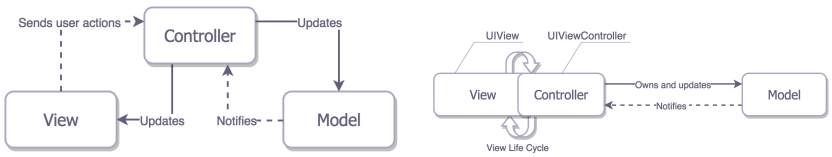
Der Umbau der Architektur vor der Bachelorarbeit konnte erstaunlich schnell und einfach gemacht werden. Die in der Studienarbeit thematisierten Probleme betreffend *Massive*-View Controller waren bekannt, waren aber aufgrund des geringen Funktionsumfangs der App *noch* nicht ersichtlich.

3.2.3 *MVC aka Massive-View-Controller*

Im Jahr 1979 stellte Reenskaug, damals Student der Universität Oslo, MVC vor - eine allgemeine Lösung für das Problem, wie Benutzer die Kontrolle über die Informationen, die sie auf einem Gerät sehen, behalten können. Das ursprüngliche kleine Dokument [30] wurde in kurzer Zeit extrem populär und schließlich kamen viele Firmen und Einzelpersonen mit ihrem eigenen Verständnis und Implementierungen von MVC, die nicht unbedingt die ursprüngliche Bedeutung eines Models, einer View und eines Controllers bewahrten.

Apple übernahm die Architektur und ermutigt Entwickler [31], diese auch einzusetzen. Das Problem an MVC beginnt allerdings schon mit einer Hauptkomponente von iOS Applikationen: dem UIView-Controller.

A view controller is tightly bound to the views it manages and takes part in the responder chain used to handle events. [...] A view controller is the sole owner of its view and any subviews it creates. It is responsible for creating those views and for relinquishing ownership of them at the appropriate times such as when the view controller itself is released. – Apple UIView-Controller API Referenz [32]



(a) Apple MVC: Vorstellung (b) Apple MVC: Realität

Abbildung 22: Apple's Vorstellung von MVC entspricht meistens nicht der Realität [33]

In Abbildung 22b ist deutlich ersichtlich, dass der UIViewController sowohl die Rolle der View als auch die Rolle des Controllers übernimmt. Daraus folgt eine schwerere Wartbarkeit und die Testbarkeit leidet sehr darunter.

Die in der Traildevils App fortlaufende Weiterentwicklung und Implementierung des Routing-Modus brachte immer mehr Funktionen mit sich und dadurch erhielt der ViewController der BikeMap laufend mehr Verantwortung über diverse Funktionen.

- Touch-Interaktionen auf der Karte entgegennehmen und verarbeiten
- Geplante Route auf der Karte darstellen
- Interaktionen auf der Route entgegennehmen und verarbeiten
- Callout darstellen und Touch-Interaktionen entgegennehmen und verarbeiten
- Flyout darstellen und Touch-Interaktionen entgegennehmen und verarbeiten
- Instanzierung und Anzeige von weiteren ViewControllern, zum Beispiel Details zu Kartenelementen oder der geplanten Route

Die Anzahl Zeilen erhöhte sich bis 1'000 Zeilen im BikeMapViewController und schnell wurde klar: Die Traildevils iOS App leidet unter dem *Massive*-View-Controller Problem.

Abbildung 23: Die *Idee* des Massive-View-Controllers startete mit einem einfachen Tweet von Colin Campbell, Entwickler bei Facebook

Zur Behebung des Problem wurden verschiedene Lösungsansätze [34] [35] [36] versucht, doch Nichts verhalf zum gewünschten Erfolg. Es musste eine andere Lösung her.

3.2.4 *MVVM: A New Hope*

Wartbarkeit und Testbarkeit sind zwei wichtige nicht-funktionale Anforderungen für die Weiterentwicklung während und nach der Arbeit. Zum gegebenen Zeitpunkt war die App allerdings weder einfach wartbar noch richtig testbar.

Bereits die vorgängige Studienarbeit setzte sich mit der Frage nach der *besten* Architektur für iOS Apps auseinander [1] und wie die meisten Fragen im Software Engineering lässt sich auch diese Frage beantworten mit: *It depends.* In die engere Auswahl kamen Redux und Model-View-ViewModel (MVVM). VIPER wurde aufgrund den genannten Probleme in Unterabschnitt 3.2.1 bei der Wahl bewusst ausgeschlossen. Nach kurzer Recherche über die beiden Patterns im Internet fiel die Entscheidung am Ende auf MVVM. Der Grund war die höhere Anzahl an Ressourcen im Internet (Beispiel Code und Tutorials) und die bereits vorhandene Erfahrung der beiden Autoren in anderen Programmiersprachen (mit Windows Presentation Foundation (WPF) und Typsichere, objektorientierte Programmiersprache (CSharp)).

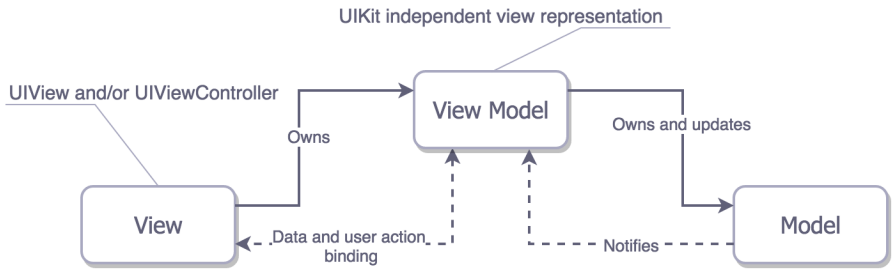


Abbildung 24: MVVM im Überblick [33]

Das MVVM Pattern besteht aus drei Hauptkomponenten: View, Models und View Models. Data Binding und Interaktionen zwischen den einzelnen Komponenten werden durch Observables gelöst. Wie in Abbildung 24 ersichtlich, unterscheidet sich MVVM in Swift nicht anders wie in anderen Programmiersprachen.

Erwähnenswert ist allerdings, dass in MVVM der UIViewController und die einzelnen UIViews als einheitliche Komponente betrachtet werden. Neu hinzugekommen ist das View Model, welches als Mediator zwischen View und Model dient. Wichtig ist, dass das View Model keine UI-relevanten Frameworks wie zum Beispiel UIKit importiert, da nur so die Trennung der Verantwortlichkeiten gewährleistet wird.

Ebenfalls in Abbildung 24 ersichtlich sind die einzelnen Verantwortungen und Referenzen von View, View Model und Model. Die View

hat Kenntnis vom View Model und bindet sich auf Eigenschaften des View Models. Das View Model wiederum besitzt einen internen State und ist verantwortlich für das Abrufen und die Aufbereitung von Models.

Das nachfolgende Beispiel zeigt, wie MVVM in der Traildevils iOS App ansatzweise umgesetzt wurde.

```
struct SearchResult {
  let title: String
}
```

Listing 1: MVVM Beispiel: Das Model

Zuerst wird ein Model (class oder struct) definiert. Models sollten wenn möglich keine Logik implementiert haben.

```
class SearchViewModel {
  let store: LocalStore

  init(store: LocalStore) {
    self.store = store
  }

  ...
}
```

Listing 2: MVVM Beispiel: Das View Model - Constructor Injection

Dem View Model werden via Constructor Injection alle notwendigen Abhängigkeiten in Form von Protocols übergeben. Der Vorteil ist, dass durch die Verwendung von Protocols bei Unit Tests auch Mocks übergeben werden können.

```
class SearchViewModel {
  ...

  var results = Observable<[SearchResult]>([])

  ...
}
```

Listing 3: MVVM Beispiel: Das View Model - Observable

Zusätzlich werden im View Model *Observables* definiert. *Observables* bieten die Möglichkeit, sich auf Änderungen zu registrierten, um bei einer allfälligen Änderung des Wertes benachrichtigt zu werden.

```
class SearchViewModel {
  ...

  func search(text: String) {
    let newResults = store.search(text: text)
    results.value = newResults
  }
}
```

Listing 4: MVVM Beispiel: Das View Model - Suchfunktion

Zum Schluss werden Funktionen definiert, die später von der View aufgerufen werden können. Die Funktion *search(text:)* in MVVM Beispiel: Das View Model - Suchfunktion durchsucht einen Store nach dem übergebenen Text und setzt anschliessend den Wert des *Observables*.

```
protocol LocalStore {
  func search(text: String) -> [SearchResult]
}
```

Listing 5: MVVM Beispiel: LocalStore Protocol

In MVVM Beispiel: Das View Model - Constructor Injection wird ein Protocol vom Typ LocalStore übergeben. Protocols sind vergleichbar mit Interfaces in Java.

```
class CoreDataStore: LocalStore {
  func search(text: String) -> [SearchResult] {
    let dbResults = ... // Query the database for the search text
    return dbResults
  }
}
```

Listing 6: MVVM Beispiel: LocalStore Implementation

Das LocalStore Protocol wird von einer konkreten Klasse implementiert und greift in diesem Fall auf den lokalen Core Data Store zu.

```
class SearchViewController: UIViewController {
  let viewModel: SearchViewModel

  init(viewModel: SearchViewModel) {
    self.viewModel = viewModel
  }

  ...
}
```

Listing 7: MVVM Beispiel: Die View - Constructor Injection

Der View wird schlussendlich das View Model via Constructor Injection übergeben.

```
class SearchViewController: UIViewController {
  ...

  override func viewDidLoad() {
    super.viewDidLoad()
    viewModel.results.bind { newResults
      // Populate data in view e.g. in UITableView
    }
  }

  ...
}
```

Listing 8: MVVM Beispiel: Die View - Binding

Sobald die View in den Speicher geladen wurde, registriert sie sich auf Änderungen von Eigenschaften des View Models.

```
class SearchViewController: UIViewController {
  ...

  func searchButtonTapped() {
    let searchText = searchBar.text
    viewModel.search(text: searchText)
  }
}
```

Listing 9: MVVM Beispiel: Die View - Aufruf der Suchfunktion im View Model

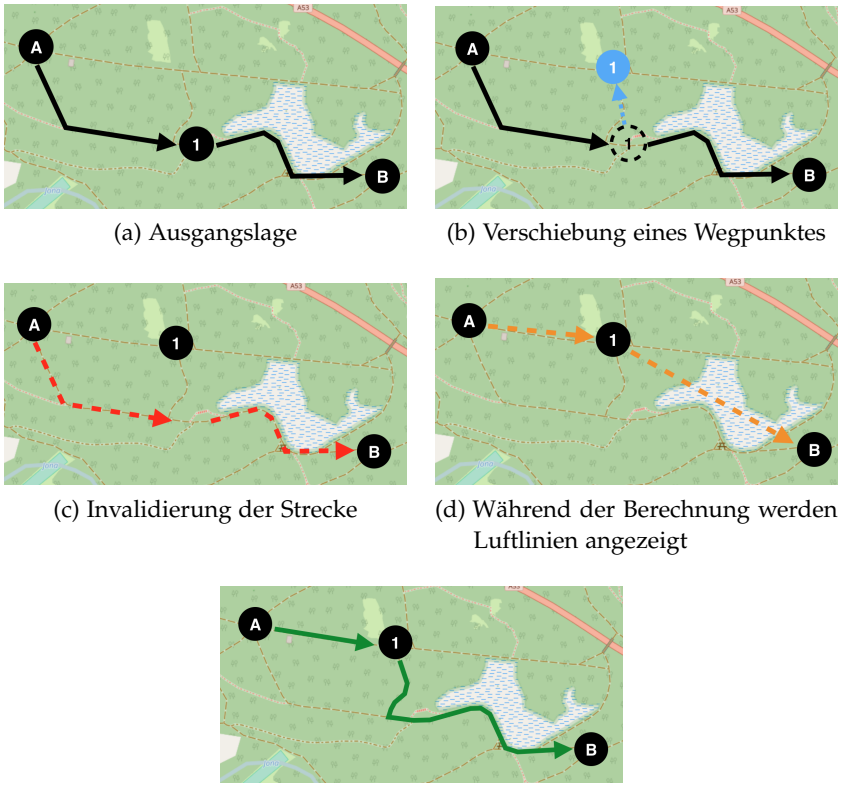
Sobald der Benutzer gesucht hat, wird der Suchtext an das View Model weitergeleitet. Die View wird über die neuen Suchresultate

nach Änderung des *Observables* benachrichtigt, siehe MVVM Beispiel: Das View Model - Suchfunktion.

Die Wartbarkeit ist durch das MVVM-Pattern gewährleistet, da die Verantwortungen klar identifiziert und aufgeteilt werden können. Um auch die Testbarkeit zu gewährleisten, wird in der Traildevils iOS App stark auf Dependency Injection gesetzt, welches im Kapitel UMSETZUNG genauer thematisiert wird.

3.3 ROUTING

Das Hauptziel dieser Arbeit ist die Validierung eines Interaktionskonzeptes für einen Tourenplaner. Damit auch Touren intuitiv geplant werden können, wurde intern ein *Route Prozessor* implementiert, der die Verwaltung der Wegpunkte übernimmt und auf Interaktionen des Benutzers reagiert. Die Funktionsweise wird nachfolgend näher erläutert.



(a) Ausgangslage

(b) Verschiebung eines Wegpunktes

(c) Invalidierung der Strecke

(d) Während der Berechnung werden Luftlinien angezeigt

(e) Die neuen Strecken wurden berechnet

Abbildung 25: Funktionsweise des Routings in der Traildevils iOS App

Als Ausgangslage dient die in Abbildung 25a bereits berechnete Strecke von A nach B mit einem Zwischenstopp 1. Durch halten und schieben können auf der Karte Wegpunkte verschoben werden. Der Zwischenstopp 1 wird wie in Abbildung 25b gezeigt verschoben und

erhält eine neue Koordinate. Bei dieser Aktion wird im *Route Prozessor* ein Ereignis ausgelöst, welches dazu führt, dass die in Abbildung 25c gezeigten Strecken von A nach 1 und von 1 nach B nicht mehr gültig sind. Der *Route Prozessor* sendet im Hintergrund zwei Anfragen an den Routing Dienst von GraphHopper, um die beiden neuen Strecken berechnen zu lassen. Während dieses Vorgangs werden dem Benutzer die Luftlinien der fehlenden Strecken angezeigt (Abbildung 25d). Sobald GraphHopper die Route berechnet und eine Antwort an die Traildevils iOS App zurückgesendet hat, wird das UI benachrichtigt, um die neuen Strecken auf der Karte anzuzeigen (Abbildung 25e).

3.4 INTERAKTIONS-DESIGN

Dieses Kapitel beschreibt wichtige Designkomponenten des Tourenplaners. Es wird zudem das abschliessende Design nach Überarbeitung und Validierung durch Nutzertest gezeigt.

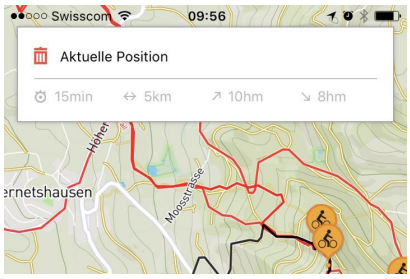
3.4.1 *Komponenten*

3.4.1.1 *Flyout*

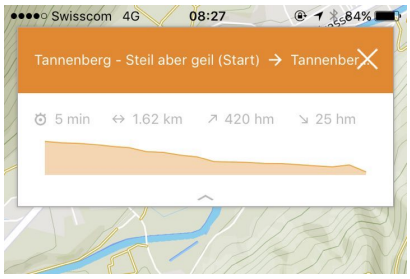
Das *Flyout* dient dem Benutzer als Unterstützung während der Planung. Mit einem Blick sind Startpunkt, Endpunkt und Zusatzinformationen ersichtlich. Das Höhenprofil kann auf Wunsch ein- oder ausgeblendet werden.

(a) Geschlossen mit Indikator für Zwischenpunkte

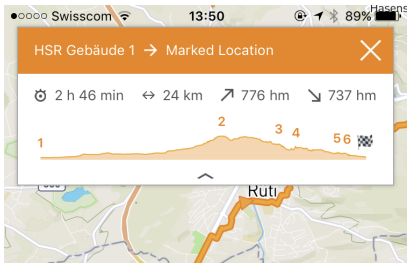
(b) Geöffnet mit Höhenprofil



(c) Mülleimer links als Analogie zum Schliessen vom Planungsmodus



(d) Kreuz rechts zum Schliessen vom Planungsmodus



(e) Platzoptimierte Darstellung des Flyouts mit Darstellung der Wegpunkten im Höhenprofil

Abbildung 26: Unterschiedliche Versionen des Flyouts

Von Anfang an war klar, dass das *Flyout* mit wenig Platz auf dem Bildschirm auskommen muss. Die Abbildung 26a zeigt eine Darstellung, in welcher Start- und Endpunkt untereinander dargestellt wird. Dazwischen ist ein Indikator, welcher Auskunft über die Anzahl dazwischen gesetzten Wegpunkten gibt. Es wurde schnell klar, dass bei dieser Darstellung noch weiter an Platz gespart werden kann.

Zur Zeit des ersten Usability Tests konnten der Planungsmodus und die Details zur Planung mit einem Kreuz geschlossen werden. Dies führte zur Verwirrung der Testpersonen. Die Auswertung im Unterunterabschnitt 2.4.1.9 hat gezeigt, dass die unterschiedlichen Aktionen unterschiedliche Icons benötigen. Deshalb zeigt die Abbildung 26c einen Mülleimer.

Folglich wurde das *Flyout* so angepasst, dass Start- und Endpunkt auf einer Zeile sind. So konnte deutlich an Platz in der Vertikalen gespart werden, wie die Abbildung 26e zeigt.

3.4.1.2 *Routing Button*

Der *Routing Button* ermöglicht dem Benutzer den zentralen Einstieg in den Planungsmodus. Während der Planung nimmt er unterschiedliche Stati ein.

PLANUNG STARTEN: Initialer Einstiegspunkt um den Planungsmodus zu starten.

WEGPUNKT HINZUFÜGEN: Ist ein Kartenelement oder Placemark selektiert, kann dies der Planung hinzugefügt werden.

WEGPUNKT ENTFERNEN: Ist ein Wegpunkt selektiert, kann dieser von der Planung entfernt werden.

(a) Erste Version des Routing Buttons mit Start, Hinzufügen und Löschen

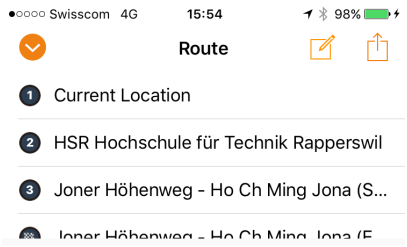
(b) Zweite Version des Routing Buttons mit Start, Hinzufügen und Löschen

Abbildung 27: Unterschiedliche Designs für den Routing Button

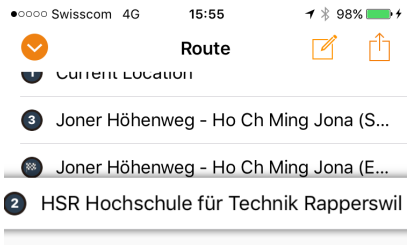
Die Abbildung 27a zeigt die verschiedenen Stati des Routing Buttons in einer ersten Phase. Die Auswertung des Usability Tests im Unterunterabschnitt 2.4.1.9 haben gezeigt, dass die Benutzer die Veränderung des Status auf den ersten Blick nicht wahrnehmen. Deshalb wurde der Button so geändert, dass er je nach Status die Farbe ändert. Die Abbildung 27b zeigt die gemachten Anpassungen. Auch das Routing Icon wurde für erhöhten Wiedererkennungswert beibehalten.

3.4.1.3 *Liste der Wegpunkte in Details*

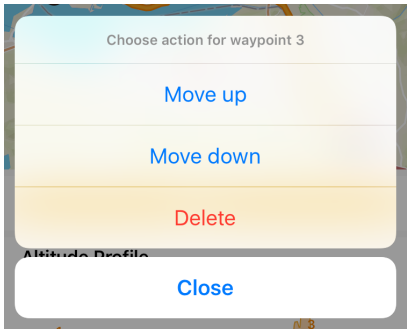
Der Benutzer kann in den Details der Planung mit Wegpunkten interagieren. Er kann sie entweder verschieben oder löschen.



(a) Liste der Wegpunkte



(b) Verschieben eines Wegpunktes in der Liste



(c) Action Sheet bei einem Tap auf einen Wegpunkt

Abbildung 28: List der Wegpunkte in den Details zur Planung

In der Abbildung 28a ist die Ausgangslage der Liste ersichtlich. Hält der Benutzer seinen Finger auf einem Wegpunkt, ist er in der Lage, einen Wegpunkt zu verschieben. Die Abbildung 28b zeigt den ausgewählten Wegpunkt in einer vergrösserten Ansicht. So ist dem Benutzer klar, welchen Wegpunkt er gerade am verschieben ist.

Mit einem Tap auf einen Wegpunkt erreicht der Benutzer die Ansicht in Abbildung 28c. Er kann nun auswählen, ob er den Wegpunkt gerne nach oben oder nach unten verschieben oder löschen möchte.

3.4.1.4 *Magnifier*

Der *Magnifier* unterstützt den Benutzer während der Feinjustierung eines Wegpunktes auf der Karte.

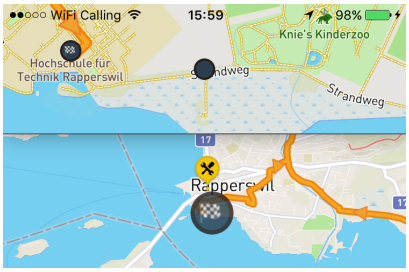


Abbildung 29: Magnifier Ansicht während des Verschieben eines Wegpunktes auf der Karte

Hält der Benutzer seinen Finger auf einem Wegpunkt, erscheint der *Magnifier*, wie es in der Abbildung 29 ersichtlich ist. Im Screen erscheint oben eine Vergrösserung des Kartenausschnitts unter dem Finger vom Benutzer. Eine Feinjustierung kann so vorgenommen werden.

3.5 FRAMEWORKS

Um Frameworks herunterzuladen und in das Projekt einzubinden, wird CocoaPods, ein Open Source Package Manager verwendet. Nähere Details zur Funktionsweise und Verwendung von CocoaPods können aus der vorgängigen Studienarbeit entnommen werden [37].

Nachfolgend eine Liste der eingesetzten Frameworks im Traildevils Projekt (GitHub Stars, Version und Lizenz sind Stand 8. Juni 2017).

3.5.1 *Charts*

Stars: 14'631 | Eingesetzte Version: 3.0.2 | Lizenz: Apache-2.0

Beautiful charts for iOS/tvOS/OSX! The Apple side of the cross-platform MPAndroidChart.

BESCHREIBUNG: Mit Charts können, wie es der Name schon sagt, Diagramme gezeichnet werden.

GRUND FÜR DEN EINSATZ: Wird für die Anzeige des Höhenprofils verwendet, hat die meisten Stars auf GitHub und wird aktiv weiterentwickelt.

3.5.2 *GraphHopperRouting*

Stars: 1 | Eingesetzte Version: 1.0.2 | Lizenz: MIT

The GraphHopper Routing API wrapped in an easy-to-use Swift framework.

BESCHREIBUNG: Da GraphHopper kein natives SDK für Swift hatte, wurde während der Bachelorarbeit ein Framework von Philipp Schilter und Roman Blum entwickelt und veröffentlicht. Das Framework wird auch offiziell auf der GraphHopper-Website erwähnt [38].

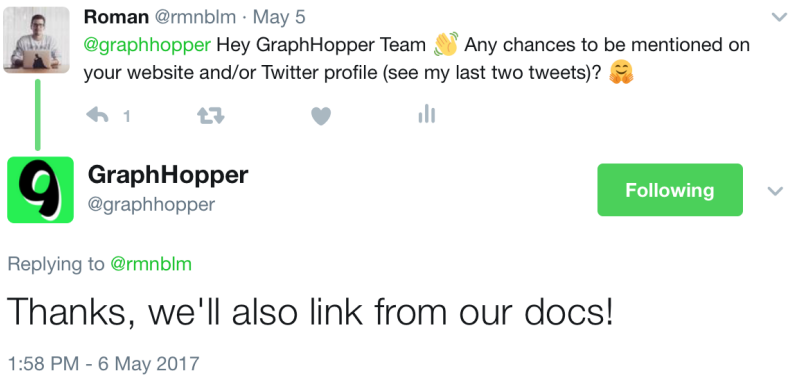


Abbildung 30: GraphHopper Tweet

GRUND FÜR DEN EINSATZ: Eigenentwicklung um die GraphHopper Routing API auf eine *Swifty* Art und Weise anzusprechen.

3.5.3 *GraphHopperGeocoder*

Stars: 1 | Eingesetzte Version: 1.0.2 | Lizenz: MIT

The GraphHopper Geocoding API wrapped in an easy-to-use Swift framework.

BESCHREIBUNG: Da GraphHopper kein natives SDK für die Geocoding API in Swift hatte, wurde während der Bachelorarbeit ein Framework von Philipp Schilter und Roman Blum entwickelt und veröffentlicht. Das Framework wird auch offiziell auf der GraphHopper-Website erwähnt [38].

GRUND FÜR DEN EINSATZ: Eigenentwicklung um die GraphHopper Geocoder API auf eine *Swifty* Art und Weise anzusprechen.

3.5.4 *Kingfisher*

Stars: 8'620 | Eingesetzte Version: 3.10.1 | Lizenz: MIT

A lightweight, pure-Swift library for downloading and caching images from the web.

BESCHREIBUNG: Kingfisher wird als Offline Cache für Bilder eingesetzt. Die Bilder werden im Cache anhand der URL (oder einem anderen Identifier) identifiziert, diese können auch invalidiert werden, damit beim nächsten Zugriff auf die URL das neue Bild aus dem Internet heruntergeladen wird. In der Traildevils App werden zum Beispiel bei einem Force-Refresh (oder bei Überschreitung des Ablauf-datums) sowohl die Detailinformationen im Cache von Realm, als auch die Bilder im Cache von Kingfisher invalidiert.

GRUND FÜR DEN EINSATZ: Hat sich bereits in der Studienarbeit bewährt.

3.5.5 *Lightbox*

Stars: 99 | Eingesetzte Version: 1.0.0 | Lizenz: MIT

A convenient and easy to use image viewer for your iOS app

BESCHREIBUNG: Lightbox ist ein Framework zur Vollbildanzeige von Bildern und Videos. Es besitzt Twitter-ähnliche Gesten wie zum Beispiel ein Swipe nach oben oder unten um die Vollbildanzeige zu verlassen. Bilder werden im Hintergrund asynchron heruntergeladen.

GRUND FÜR DEN EINSATZ: Ist zwar wenig bekannt aber bietet genau den Funktionsumfang, der gewünscht ist (und nicht mehr), wird für die Vollbildanzeige verwendet, wenn ein Bild in der Detailansicht berührt wird (in Zukunft auch für die Anzeige von Galerien von Bikern). Zusätzlich wurden während der Arbeit zwei Pull Requests erstellt [39] [40].

3.5.6 *Mapbox*

Stars: 1'917 | Eingesetzte Version: 3.5.4 | Lizenz: BSD

A library based on Mapbox GL Native for embedding interactive map views with scalable, customizable vector maps into Cocoa Touch applications on iOS 7.0 and above using Objective-C, Swift, or Interface Builder.

BESCHREIBUNG: Nebst Google Maps und Apple Maps ist Mapbox ein weitere, grosse Mapping Plattform zur Anzeige von Karten.

GRUND FÜR DEN EINSATZ: Vorgabe des Projektpartners.

3.5.7 *Nimble*

Stars: 2'153 | Eingesetzte Version: 7.0.1 | Lizenz: Apache-2.0

A Matcher Framework for Swift and Objective-C

BESCHREIBUNG: Nimble wird als Teil von Quick mitinstalliert. Tests lassen sich dadurch aussagekräftiger schreiben.

```
expect(1 + 1).to(equal(2))
expect(1.2).to(beCloseTo(1.1, within: 0.1))
expect(3) > 2
expect("seahorse").to(contain("sea"))
expect(["Atlantic", "Pacific"]).toNot(contain("Mississippi"))
expect(ocean.isClean).toEventually(beTruthy())
```

Listing 10: Beispiel Nimble Framework

GRUND FÜR DEN EINSATZ: Einfacheres und deskriptiveres Testing, wird mit Quick automatisch installiert.

3.5.8 *Quick*

Stars: 6'485 | Eingesetzte Version: 1.1.0 | Lizenz: Apache-2.0

The Swift (and Objective-C) testing framework.

BESCHREIBUNG: Quick bezeichnet sich selber als *behavior-driven development framework* und wird zusammen mit Nimble installiert.

```
describe("the spring semester 2017") {
  describe("this Bachelor thesis") {
    context("before hand-in") {
      expect(students.mood).to(be(.stressed))
    }
    context("after hand-in") {
      expect(students.mood).to(be(.relieved))
    }
  }
}
```

Listing 11: Beispiel Quick Framework

GRUND FÜR DEN EINSATZ: Einfaches und deskriptiveres Testing

3.5.9 *Raclette*

Stars: 6 | Eingesetzte Version: 1.1.3 | Lizenz: MIT

Let's get cheesy with Raclette, a UITableView extension to add rows and sections on-the-fly.

BESCHREIBUNG: Raclette ist ein Framework zur Erweiterung der UITableView. Das Framework wird für die einfache Manipulation der Tabelle verwendet. Es wurde von Roman Blum während der Studienarbeit entwickelt und unter der MIT Lizenz im CocoaPods-Katalog veröffentlicht

GRUND FÜR DEN EINSATZ: Hat sich bereits in der Studienarbeit bewährt, wird in Views mit statischem TableView Inhalt verwendet (zum Beispiel SettingsViewController)

3.5.10 *SnapKit*

Stars: 9'900 | Eingesetzte Version: 3.2.0 | Lizenz: MIT

A Swift Autolayout DSL for iOS & OS X

BESCHREIBUNG: In Xcode können User-Interfaces entweder aus einer Mischung von Storyboards (Grafischer UI Designer) und Programmcode oder reinem Programmcode realisiert werden. Über die Vor- und Nachteile von Storyboards lässt sich streiten ([41]). Alle View-Komponenten werden im Code erstellt und im UI platziert. Um das Auto Layout (Margins, Paddings, Breite, Höhe, usw.) der View-Komponenten im Programmcode zu erleichtern, wird SnapKit eingesetzt.

GRUND FÜR DEN EINSATZ: Hat sich bereits in der Studienarbeit bewährt, wird praktisch in jedem ViewController für das Layouting der Views verwendet.

3.5.11 *SwiftyJSON*

Stars: 14'625 | Eingesetzte Version: 3.1.4 | Lizenz: MIT

The better way to deal with JSON data in Swift

BESCHREIBUNG: Swift enthält im Foundation Framework bereits die Klasse JSONSerialization, mit welcher sich JSON-Daten parsen lassen. Das Parsen mit dieser Klasse erwies sich aber als sehr umständlich und unübersichtlich, weshalb mit SwiftyJSON ein vereinfachter JSON Parser eingesetzt wird.

```
{
  "users": [
    {
      "name": "Roman",
      "age": 24
    },
    {
      "name": "Fippu",
      "age": 26
    }
  ]
}
```

Listing 12: JSON Vorlage für SwiftyJSON Beispiel

```
let json = JSON(url: URL(string: "https://traildevils.ch/users"))
let name = json["users"][0]["name"].stringValue // name: Roman
let age = json["users"][0]["age"].stringValue // age: 24
```

Listing 13: Beispiel SwiftyJSON

GRUND FÜR DEN EINSATZ: Hat sich bereits in der Studienarbeit bewährt, wird hauptsächlich für das Parsen der Traildevils API Antworten verwendet.

3.5.12 *SwiftyXML*

Stars: 20 | Eingesetzte Version: 1.4.0 | Lizenz: MIT

The most swifty way to deal with XML data in swift 3.

BESCHREIBUNG: Wie auch JSON lässt sich XML mit dem Foundation Framework in Swift parsen, dies allerdings auf noch umständlichere Art und Weise wie mit JSON. Mit SwiftyXML kann im gleichen Stil auf Elemente und Attribute in einer XML-Datei zugegriffen werden wie mit SwiftyJSON.

```
<?xml version="1.0" encoding="UTF-8"?>
<users>
  <user name="Roman" age="24" />
  <user name="Fippu" age="26" />
</users>
```

Listing 14: XML Vorlage für SwiftyXML Beispiel

```
let xml = XML(url: URL(string: "https://traildevils.ch/users"))
let name = xml["users"][0]["@name"].string // name: Roman
let age = xml["users"][0]["@age"].string // age: 24
```

Listing 15: Beispiel SwiftyXML

GRUND FÜR DEN EINSATZ: Einfacheres Handling von XML Dateien, wird für das Parsen der GPX-Dateien verwendet.

3.6 DELIVERABLE

In diesem Kapitel geht es um den aktuellen Stand der abgegebenen iOS App. Wo nicht anders erwähnt, betreffen die Aussagen zu den funktionalen und nicht-funktionalen Anforderungen die App mit der MVC-Architektur. Zum Zeitpunkt der Abgabe trug die App die Versionsnummer 1.0 und Build-Nummer 26. Gegen Ende der Realisierungsphase wurde dieser Stand auch ein letztes Mal über TestFlight den Testbenutzern zur Verfügung gestellt. Eine Aufwandschätzung für Optimierungen und Verbesserungen befinden sich im Anhang.

3.6.1 *Status Funktionale Anforderungen*

Im Rahmen dieser Bachelorarbeit konnten alle acht Use Cases umgesetzt werden. Namentlich sind dies:

- UC01: Strecke planen mit Kartenelementen: Der Benutzer kann den Routing-Modus starten und Kartenelemente zur Route hinzufügen. Eine Route von A nach B wird asynchron an GraphHopper gesendet und die berechnete Route wird anschliessend auf der Karte angezeigt. Das Hinzufügen eines bestehenden Tracks hat keine Berechnung des Online Dienstes zur Folge.
- UC02: Strecke planen mit POIs: Der Benutzer kann POIs zu seiner Route hinzufügen, indem er einen beliebigen Ort auf der Karte antapped. Die Adresse wird asynchron vom Geocoder ermittelt.
- UC03: Strecke planen mit eigenem GPS Tracks: Der Benutzer kann GPS Tracks mit dem Format .gpx in die App importieren und auf der Karte anzeigen lassen.
- UC04: Detailansicht zu einer geplanten Strecke abrufen: Dem Benutzer werden Details zu seiner geplanten Strecke angezeigt. Sowohl im Flyout als auch in den Details zur Route werden Distanz, Dauer, Aufstieg, Abstieg und ein Höhenprofil angezeigt.
- UC05: Geplante Strecke editieren: Der Benutzer kann Wegpunkte seiner geplanten Strecke umsortieren, verschieben oder löschen.
- UC06: Geplante Strecke lokal speichern: Der Benutzer kann seine geplante Strecke abspeichern. Er kann der Route auch einen Namen geben.
- UC07: Geplante Strecke offline verfügbar machen: Der Benutzer kann das gesamte Kartenmaterial, welches seine Route umfasst, herunterladen.

- UC08: Geplante Strecke als GPS Track exportieren: Der Benutzer kann eine geplante Route als GPS Track im Format .gpx in andere Apps exportieren (zum Beispiel via Mail, WhatsApp, Telegram, usw.)

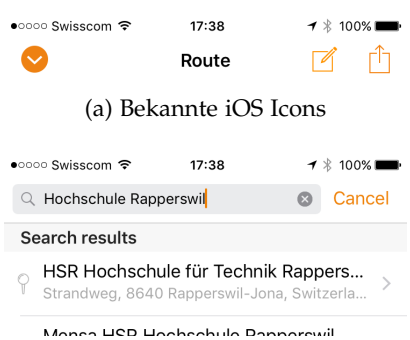
Alle Use Cases wurden mit manuellen Tests plus zwei Usability Tests validiert.

3.6.2 *Status Nicht-funktionale Anforderungen*

3.6.2.1 *Usability*

Allgemein wurden während der Implementierung so gut wie möglich die Apple Human Interface Guidelines [2] befolgt und viele iOS Standard-Komponenten verwendet.

Aus den Usability Tests war ersichtlich, dass das Interaktionskonzept sowohl für iOS als auch Android Benutzer erfolgreich validiert werden konnte.

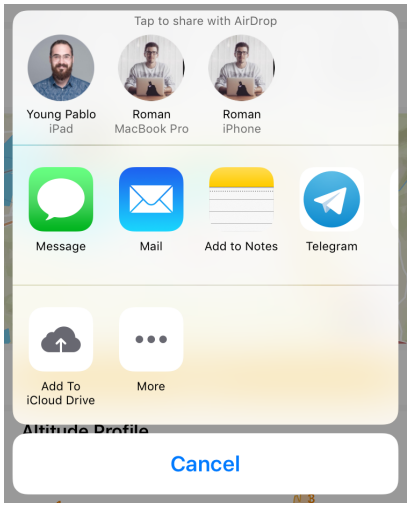


(a) Bekannte iOS Icons

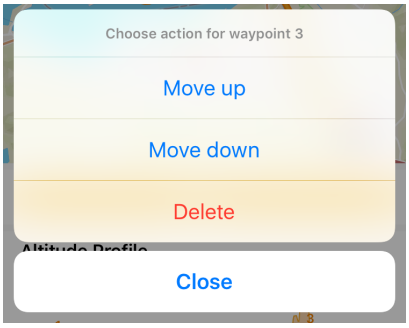
(b) UINavigationController

(c) UISearchBar

(d) UITabBarController



(e) UIActivityViewController



(f) UIAlertController

Abbildung 31: Traildevils iOS Komponenten

Mithilfe der durchgeführten Usability Tests konnten extrem viele Erkenntnisse über die Zielgruppe gewonnen und die Traildevils iOS App dadurch iterativ weiter entwickelt werden. Die Benutzerfreundlichkeit der App widerspiegelt sich in den Schlusstests und lässt nur wenig Wünsche offen. Optimierungen, die nach dem zweiten Usability Test noch notwendig sind, können dem Unterunterabschnitt 2.4.2.1 entnommen werden.

3.6.2.2 *Performance*

Die Performance konnte im Gegensatz zur App in der Studienarbeit massiv verbessert werden. Das Starten der App auf einem iPhone 7 dauert weniger als eine Sekunde. Erreicht wurde dies durch zwei Massnahmen:

- Die Umstellung von Realm zu Core Data löste den nicht benötigten Sync-Overhead von Open Source Datenbank mit Fokus auf Mobilität und Synchronisation (Realm). Mit Persistenz Framework basierend auf SQLite von Apple (Core Data) werden hauseigene Mittel von Apple verwendet, die nur einen Zweck erfüllt: Daten verwalten.
- Es wurde festgestellt, dass GeoJSON Dateien schneller von Mapbox geladen und angezeigt werden, als ein Array von Feature-Objekten. Aufgrund dieser Feststellung werden die von der Traildevils API gelieferten GeoJSON Dateien nicht nur geparst und in der Datenbank abgelegt (für Indexierung, Suche, usw.), sondern auch im Dokumentordner (Documents directory) des Apps abgelegt. Beim Start der App wird Mapbox die lokale URL der Datei übergeben und die Kartenelemente werden geladen.

3.6.2.3 *Testability*

Aufgrund den beschriebenen Problemen in Unterabschnitt 3.2.3 kann für die MVC-Version keine Testbarkeit gewährleistet werden. Im Vergleich zur MVC-Version ist die angefangene MVVM-Version deutlich besser testbar. Durch Dependency Injection und den Einsatz von Protocols lassen sich in alle Klassen Mocks injecten und einfach testen.

```
protocol ForwardGeocoder {
  func search(text: String, completion: ([GeocodedPlacemark]) -> Void)
}

protocol LocalStore {
  func search(text: String, completion: ([PointFeature]) -> Void)
}

class SearchViewModel {
  private let store: LocalStore
  private let forwardGeocoder: ForwardGeocoder

  init(store: LocalStore, forwardGeocoder: ForwardGeocoder) {
    self.store = store
    self.forwardGeocoder = forwardGeocoder
  }

  func search(text: String) {
    store.search(text: text, ...)
    fordwardGeocoder.search(text: text, ...)
  }
}
```

Listing 16: Beispiel Dependency Injection in der Traildevils App

Der momentane Stand der MVVM-Version hat eine Code-Coverage von 44%. Durch den modularen Aufbau kann dieser Prozentsatz aber deutlich erhöht werden.

3.6.2.4 *Maintainability*

Das GitHub Repository enthält eine ausführliche Installationsanleitung mit allen wesentlichen Schritten zur Einrichtung der Entwicklungsumgebung, um so problemlos an der Applikation weiter arbeiten zu können. Die klar strukturierte Hierarchie des Xcode-Projektes, die einheitliche Namensgebung von Klassen, Funktionen und Variablen macht den Einstieg und die Weiterentwicklung erheblich leichter. Zusätzlich wurde ein Code Review des MVVM Codes mit Toni Suter (M. Sc. FHO in Engineering, Swift-Experte und Mitarbeiter des Institut für Software (IFS)) durchgeführt und die vorgeschlagenen Änderungen wurden teilweise bereits integriert.

3.6.2.5 *Availability*

Die Verfügbarkeit der Karte beziehungsweise einzelnen Tiles ist in der App durch *Ambient Caching* [42] und den Offline Download von Routen gewährleistet. Kartenelemente sind offline verfügbar, bereits an-

gesehene Detailinformationen werden in der lokalen Datenbank mit Ablaufdatum zwischengespeichert. Der Geocoder und das Routing sind nur verfügbar, wenn der Benutzer online ist.

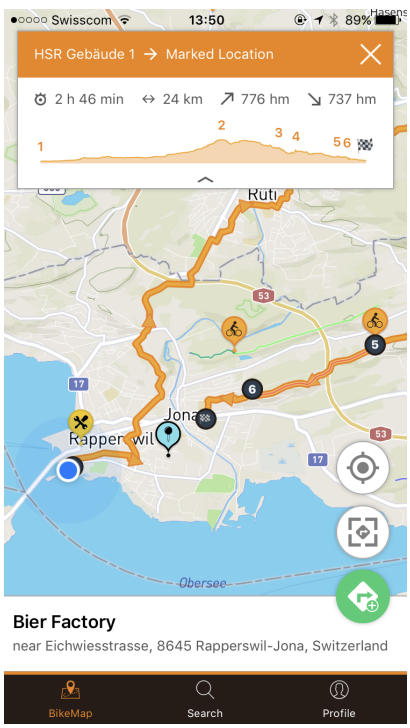
Beim Testen konnten keinerlei Abstürze bei schlechter Internetverbindung festgestellt werden, allerdings kam es vereinzelt zu Unterbrüchen beim Laden der Tiles oder beim Routing.

3.6.2.6 *Accessibility*

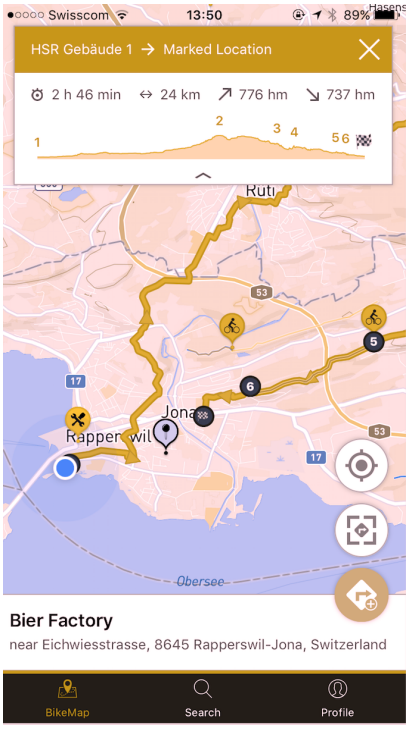
In Unterunterabschnitt 2.1.2.6 wurde erwähnt, dass die App auch für sehbehinderte Menschen gut bedienbar sein soll. Leider kann die App diese handicapierten Benutzer leider noch nicht vollständig befriedigen, wie nachfolgende Tests zeigen.

Die Tests wurden mit Sim Daltonism [43], ein Color Blindness Simulator für iOS und macOS, durchgeführt. Mit diesem Simulator lassen sich unterschiedliche Typen von Farbenblindheit visualisieren.

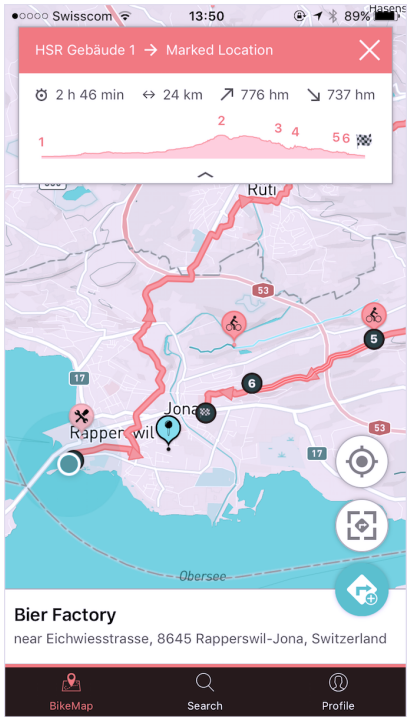
In den nachfolgenden Bildern lässt sich deutlich sehen, dass vor allem die Marker-Farben von Dealer und Trail sich schwer unterscheiden lassen. Trotzdessen ist eine Unterscheidung durch die Icon-Sprache gegeben.



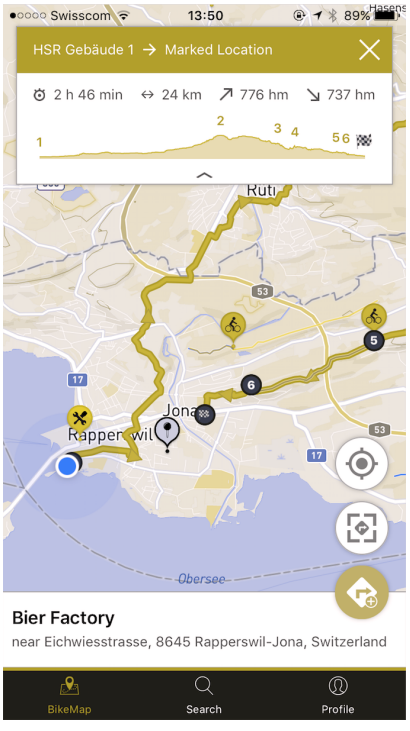
(a) Originale Ansicht



(b) Deuteranopia



(c) Tritanopia



(d) Protanopia

Abbildung 32: Traildevils unter verschiedenen Arten von Farbenblindheit

3.6.3 *Codestatistik*

3.6.3.1 *Lines of Code*

Mit dem Tool *cloc* [44] wurden die *Lines of Code* ausgewertet. Dabei wurden die beiden Architekturen MVC und MVVM ausgewertet.

PROJEKT	SWIFT FILES	BLANK	COMMENT	CODE
MVC	118	1'607	857	7'326
MVVM*	203	1'442	61	7'946

Tabelle 3: Lines of Code der beiden Architekturen

* = Aktueller Stand der MVVM Architektur

3.6.3.2 *Commits and PRs*

Anhand des Repository auf GitHub wurden manuell Werte zu Commits, Pull Requests und Releases gesammelt. *Insgesamt* bedeutet den Stand seit der Studienarbeit und. *Bachelorarbeit* zeigt die Zahlen seit dem Beginn der Arbeit im Frühlingssemester 2017.

ZEITSPANNE	COMMITS	PULL REQUESTS	RELEASES
Bachelorarbeit	753	32	23
Insgesamt	1'131	62	26

Tabelle 4: Übersicht aller Commits, Pull Requests und Releases

3.6.3.3 *Cyclomatic Complexity*

Der in Abschnitt 9.2 erwähnte Linter überprüft nicht nur Richtlinien des Codestyles, sondern überprüft den Code unter anderem auch auf Länge der Parameter von Funktionen, Zeilenlänge von Klassen und Dateien oder die zyklomatische Komplexität.

Der Standardwert für die zyklomatische Komplexität beträgt 10 - wie es auch offiziell von McCabe [45] empfohlen wird - und wurde bei der Implementation sowohl in MVC als auch in MVVM nie überschritten.

3.6.4 *Abnahme*

In einer gemeinsamen Besprechung wurden Funktionale und Nichtfunktionalen Anforderungen mit einem Abnahmeprotokoll nach HERMES [46] vom Projektpartner abgenommen.

4 ZUSAMMENFASSUNG

4.1 ERGEBNISSE

Nebst dem in der Studienarbeit erarbeitetem Funktionsumfang [47] konnten folgende Funktionen implementiert werden:

- Planung einer Tour mit Kartenelementen und POIs
- Anzeige von Zusatzinformationen zur aktuell geplanten Strecke
- Hinzufügen von zusätzlichen Streckenpunkten als verschiebbarer Zwischenstopp oder zusätzlicher Streckenpunkt am Anfang oder Ende der Strecke
- Hinzufügen eines Bike-Trails (Anfang, Endpunkt und Strecke dazwischen) zu einer geplanten Strecke
- Verschieben von Streckenpunkten auf der Karte
- Ändern der Reihenfolge der abgefahrenen Streckenpunkte
- Löschen von Streckenpunkten
- Offline Verfügbarkeit der Karte einer geplanten Route

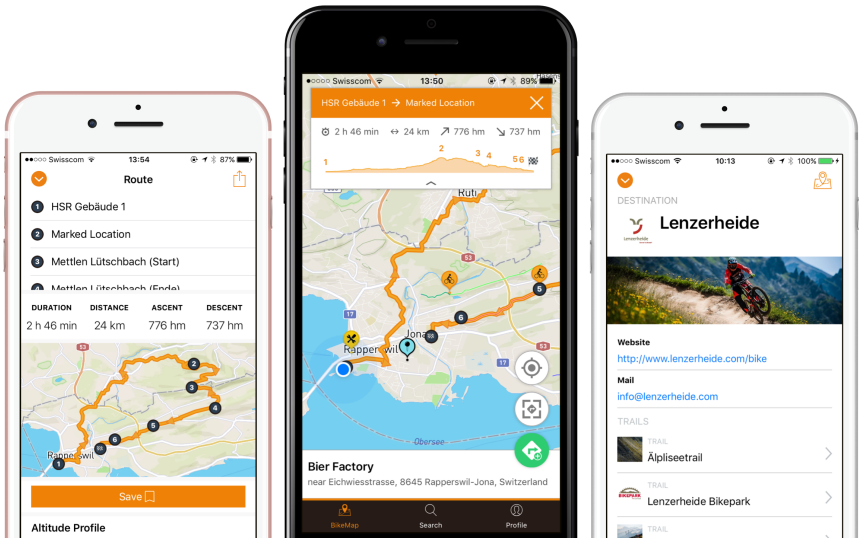


Abbildung 33: Traildevils iOS Mockups: Screenshots aus der finalen Applikation

y

Zusätzlich zur Arbeit konnten zwei Frameworks für die native Unterstützung von GraphHopper Geocoder [48] und Routing [49] erstellt und auf CocoaPods veröffentlicht werden.

4.2 AUSBLICK

Mit der erfolgreichen Implementierung des Routing Modus, welcher durch Usability Tests validiert wurde, steht der Weiterentwicklung der Traildevils iOS App Nichts mehr im Wege.

Der nächste Schritt aus der Sicht der Autoren ist zuerst das Abschliessen der Umsetzung auf MVVM und das gleichzeitige Einarbeiten der Verbesserungsvorschläge aus dem zweiten Usability Test. Da Traildevils von der Community lebt, steht die Umsetzung der Benutzeranmeldung und Benutzerregistrierung an zweiter Stelle. So können Funktionen freigeschaltet werden, welche einen angemeldeten Benutzer voraussetzen, wie zum Beispiel ein Trail Check-In, einen Zustand melden, ein Kartenelement bewerten und kommentieren oder etwa einen Rider anzeigen. Zudem kann ein Aktivitätenstream implementiert werden, welcher bereits auf der Webseite im Einsatz ist. Je nach Aufenthalt eines Users, können so auch basierend auf dessen Standort, Events in Form von Push Notifications versendet werden.

- Umstellung zu MVVM abschliessen
- Allgemein Benutzerfreundlichkeit verbessern (zum Beispiel Meldung bei schlechter Internetverbindung)
- Statusanzeige für das Importieren einer Strecke
- Swipe Gesten Support in Route Details
- Reorder Control anzeigen
- Offline Download nicht automatisch starten
- Überarbeitung der Fortschrittsanzeige beim Offline Download
- Generischen Namen für die geplante Route beim Speichern
- Onboarding: Konzept und Umsetzung
- Accessibility verbessern (zum Beispiel Marker noch nicht ausreichend unterscheidbar)
- Weitere Features gemäss Backlog

Teil II

PROJEKTDOKUMENTATION

5 PROJEKTMANAGEMENT

5.1 ZEITLICHE PLANUNG

Das Projekt wurde im Rahmen der Bachelorarbeit durchgeführt. Die bestehende Traildevils iOS App wurde während 16 Wochen weiterentwickelt. Der Abgabetermin für die Arbeit war der 16. Juni 2017 um 12:00 Uhr.

Der Aufwand pro Teammitglied betrug insgesamt 360 Stunden. Pro Woche waren dies im Schnitt 22.5 Stunden Arbeitsaufwand.

5.2 ZEITLICHE AUSWERTUNG

5.2.1 *Teammitglied*

Die Teammitglieder trafen sich jeweils montags, donnerstags und freitags an der HSR um an der Bachelorarbeit zu arbeiten. So wurde sichergestellt, dass pro Woche ausreichend Zeit investiert wurde.

KALENDERWOCHE	ROMAN BLUM	PHILIPP SCHILTER	TOTAL
KW08	15	24	39
KW09	22	22	44
KW10	26.5	29.5	56
KW11	44	18.75	62.75
KW12	19	21	40
KW13	33	28	61
KW14	32.5	30.25	62.75
KW15	46	42.25	88.25
KW16	22	26	48
KW17	20	23	43
KW18	29	32	61
KW19	52	36	88
KW20	38	32	70
KW21	25	32	57
KW22	23	23	46
KW23	22	22	44
KW24	27	27	54
Total	496	468.75	964.75

Tabelle 5: Zeitliche Auswertung pro Teammitglied in Stunden

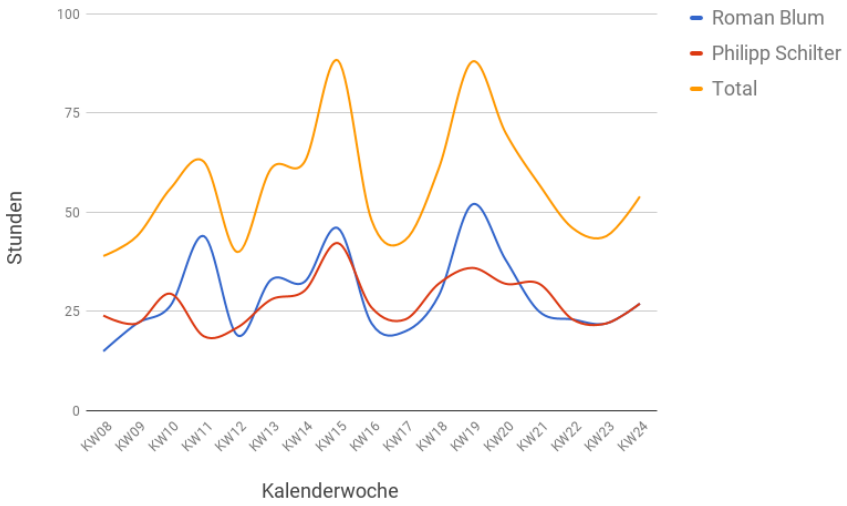


Abbildung 34: Grafische Auswertung pro Teammitglied in Stunden

5.2.1.1 *Auswertung*

In der Tabelle 5 ist ersichtlich, dass die Stunden pro Woche sehr variieren. Anhand der Aufwände in den übrigen Module wurde je nach Möglichkeit mehr oder weniger Zeit investiert. Am Ende der Arbeit sind die Teammitglieder zusammen auf 964.75h gekommen, was 244.75h über der eingeplanten Zeit liegt. Roman Blum hat 136h und Philipp Schilter hat 108.75h mehr als die erforderlichen 360h investiert. Dies ist auf die hohe Motivation der Teammitglieder zurückzuführen.

In der Kalenderwoche 15 fand das Usability Testing statt. Die Vor- und Nacharbeiten, wie auch die Durchführung haben die wöchentliche Summe erhöht.

5.2.2 *Repositories*

Die Bachelorarbeit wurde in unterschiedliche Repositories auf GitHub unterteilt. In den Mapbox Examples wurde für die Vorstudie getüftelt. Die iOS App und Dokumentation erhielten jeweils ein eigenes Repository.

Für die Auswertung wurden die Mapbox Examples und die iOS App als *Code* zusammengeführt.

REPOSITORY	ROMAN BLUM	PHILIPP SCHILTER	TOTAL
Code	370.5 (58.05%)	267.75 (41.95%)	638.25
Dokumentation	134.25 (41.12%)	192.25 (58.88%)	326.5
Total	504.75 (52.32%)	460 (47.68%)	964.75

Tabelle 6: Zeitliche Auswertung pro Repository in Stunden und Prozent

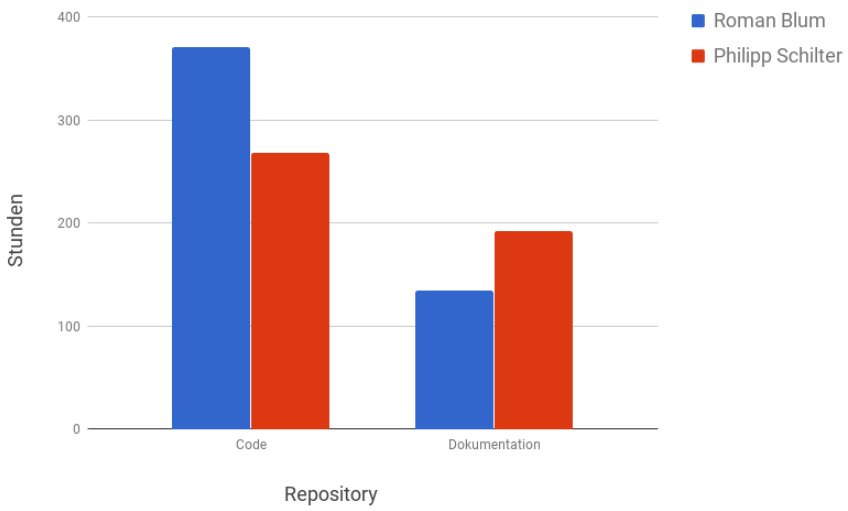


Abbildung 35: Grafische Auswertung pro Repository

5.2.2.1 *Auswertung*

Aus der Tabelle 6 ist zu entnehmen, dass die iOS App (Code) mehr Zeit in Anspruch nahm. Zudem ist der prozentuale Anteil eines Teammitglieds pro Projekt sichtbar.

5.3 PHASEN

Das Projekt wurde grob in zwei Phasen geteilt.

5.3.1 *Analysephase*

In der Analysephase wurden die Anforderungen des Projektpartners erhoben. Anschliessend wurden die Machbarkeit mit Hilfe von Experimenten überprüft. Mit einem Prototyp und einem Usability Test konnten die Anforderungen und Interaktionskonzept zum Schluss der Phase validiert werden. Risiken wurden in der Analysephase minimiert oder im besten Fall eliminiert.

Das Ende der Analysephase und der Start der Entwicklungsphase hatten sich etwas überschnitten, aufgrund der Entwicklung des In-App Prototyps.

5.3.2 *Enwicklungsphase*

In der Entwicklungsphase startete das klassische Software Projekt. Die erhobenen Anforderungen wurden in Features und kleinere Tasks zerlegt. Diese Tasks wurden vom Projektpartner priorisiert und von den Entwicklern geschätzt.

5.4 SPRINTS

Während der Entwicklungsphase wurde die agile Vorgehensweise nach SCRUM ist die am weitesten verbreitete agile Methode im Projektmanagement (SCRUM) eingesetzt. Gemäss dem priorisierten Backlog des Projektpartners planten die Entwickler während dem Sprint Planning die Tasks ein und schätzten diese.

Die Tasks wurden so grob und so granular definiert, dass sie von einem einzelnen Entwickler erledigt werden konnten. Wurde ein Task abgeschlossen, wurde ein Pull Request erstellt und das andere Teammitglied für ein Review benachrichtigt. Entsprach der Code den Qualitätsansprüchen, wurde er in die bestehende Code-Basis integriert.

Die Sprints dauerten zwei Wochen. Dies bot genügend Zeit um ein Artefakt zu generieren, welches anschliessend dem Projektpartner und dem Betreuer zum Testen übergeben wurde. Es kam auch vor, dass mehrere Releases während eines Sprints veröffentlicht wurden, um noch schneller an Feedback für die weitere Entwicklung zu gelangen.

Der Projektpartner priorisierte die Tasks bis Montagmorgen vor dem Sprint Planning. Das Review des vergangenen und die Planung des kommenden Sprints fand alle zwei Wochen jeweils am Montag um 13:10 Uhr in der Cafeteria statt. So hatten die Entwickler genügend Zeit, die Tasks bis zum Mittag zu schätzen und zu planen. Nicht erledigte Tasks wurden wieder in den Backlog verschoben.

6 MILESTONES

MS01: PROJEKTPLAN UND ERSTE EXPERIMENTE ZUR ROUTENPLANUNG

GEPLANT: 06.03.2017

ERREICHT: Zum Termin

INHALT:

- Review Projektplan
- Review erster Experimente zur Routenplanung

MS02: PROTOTYP FÜR ERSTES, INTERNES USER TESTING

GEPLANT: 20.03.2017

ERREICHT: Zum Termin

INHALT:

- Proto.io Prototyp

MS03: FINALISIERTER IN-APP PROTOTYP UND VORBEREITUNGEN FÜR USER TESTING

GEPLANT: 03.04.2017

ERREICHT: 05.04.2017

INHALT:

- Erster In-App Prototyp
- Testkonzept für Usability Test
- UC01: Strecke planen mit Kartenelementen
- UC02: Strecke planen mit POIs
- UC05: Geplante Strecke editieren

Bemerkungen

Nach dem Update von Xcode auf die Version 8.3 gab es Probleme mit dem Interface Builder. Views konnten nicht mehr angepasst werden, was zu Verzögerungen führte. Auch das Feedback vom Projektpartner war umfangreicher als gedacht und verzögerte das gesetzte Ziel.

MS04: USER TESTING ABGESCHLOSSEN

GEPLANT: 18.04.2017

ERREICHT: Zum Termin

INHALT:

- In-App Prototyp abgeschlossen
- User Testing durchgeführt
- UC04: Detailansicht zu einer geplanten Strecke abrufen

Bemerkung

Durch das Feedback der Testpersonen hat sich herauskristallisiert, dass das Bedürfnis für das Exportieren einer geplanten Strecke hoch ist. Gemeinsam mit dem Projektpartner wurde entschieden, dass der Export einer Route als neuer Use Case UC08: Geplante Strecke als GPS Track exportieren aufgenommen wird.

MS05: VERBESSERUNGSVORSCHLÄGE, ROUTE SPEICHERN/LADEN/OFFLINE

GEPLANT: 15.05.2017

ERREICHT: 19.05.2017

INHALT:

- Verbesserungsvorschläge aus dem Usability Test implementiert

Bemerkung

Die Verbesserungsvorschläge und die Entwicklung der GraphHopper Frameworks nahmen zu viel Zeit in Anspruch, weshalb folgende Use Cases in den nächsten Sprint verschoben werden mussten:

- UC03: Strecke planen mit eigenem GPS Tracks
- UC06: Geplante Strecke lokal speichern
- UC07: Geplante Strecke offline verfügbar machen
- UC08: Geplante Strecke als GPS Track exportieren

MS06: ROUTE IMPORTIEREN/EXPORTIEREN UND ZWEITER USABILITY TEST

GEPLANT: 29.05.2017

ERREICHT: zum Termin

INHALT:

- UC03: Strecke planen mit eigenem GPS Tracks
- UC06: Geplante Strecke lokal speichern
- UC07: Geplante Strecke offline verfügbar machen
- UC08: Geplante Strecke als GPS Track exportieren

MS07: APP POLISHING

GEPLANT: 05.06.2017

ERREICHT: 14.06.2017

INHALT:

- Kleinere Bugfixes
- Kleinere Optimierungen

Bemerkung

Die Vorbereitung und Durchführung des zweiten Usability Test verzögerte das Polishing.

MS08: ABGABE DOKUMENTATION

GEPLANT: 16.06.2017

ERREICHT: 16.06.2017

INHALT:

- Abgabe der Dokumentation

7 RISIKOMANAGEMENT

Zum Start des Sprints werden neue Risiken ermittelt und bestehende erneut bewertet. So kann über die Gesamtdauer des Projekts der Risikoverlauf nachvollzogen werden.

7.1 RISIKEN

Die folgende Risikotabelle veranschaulicht alle Risiken und dessen Gewicht. Der Indikator für das Gewicht liegt zwischen eins und fünf, wobei eins die kleinste Gewichtung und fünf die grösste Gewichtung ist.

ID	BESCHREIBUNG	GEWICHT
R01	Berechnung der Routen und Verarbeitung des Resultats ist zu komplex	3
R02	Editieren der geplanten Strecke genügt nicht den Anforderungen	5
R03	Komplexität des UI/UX wird unterschätzt	4

Tabelle 7: Risikotabelle

Wird die Gewichtung mit der Eintrittswahrscheinlichkeit von 0% - 100% multipliziert, entsteht als Produkt der tatsächliche Risikowert (0 bis maximal 500).

Auswertung

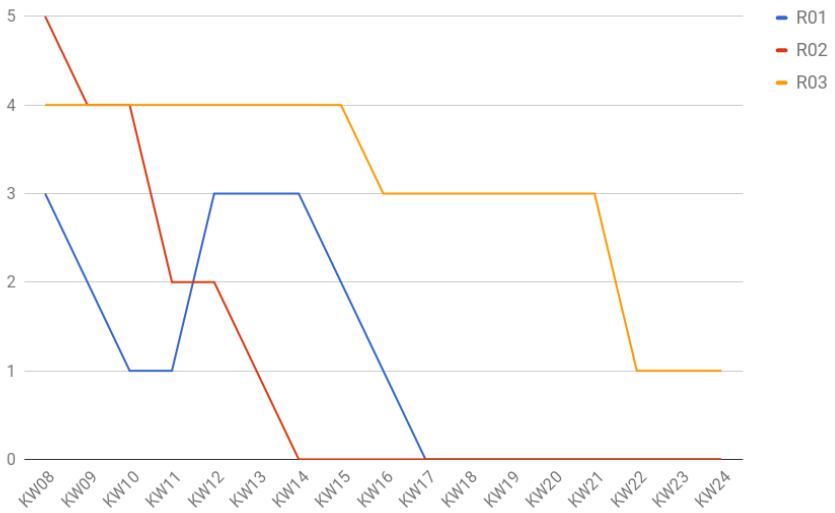


Abbildung 36: Verlauf der Risiken über die Arbeit

Aus Abbildung 36 ist zu entnehmen, dass die zu Beginn evaluierten Risiken über die Zeit minimiert oder ganz eliminiert werden konnten. Zusätzliche Risiken sind während der Arbeit nicht dazugekommen. Das Risiko *R01* musste nach der Minimierung in KW12 wieder erhöht werden, da der Mapbox Routing Dienst nicht den Anforderungen entsprochen hatte (Abschnitt 2.5). Das Risiko *R02* konnte dank der Experimentierphase in den ersten Wochen eliminiert werden, da die Umsetzung der Funktionen für das Editieren der geplanten Strecke in einem Protoyp schneller implementiert werden konnte als gedacht. Das Risiko *R03* konnte über den Verlauf der Arbeit nach jedem Usability Test minimiert, jedoch nicht ganz eliminiert werden.

Umgang mit Risiken

Durch die Analysephase sollen Risiken früh erkannt, minimiert und im besten Fall eliminiert werden. So kann einen effizienten Fortschritt in der Entwicklungsphase garantiert werden. Ebenfalls wird grossen Wert auf den Prototypen aus der Analysephase gelegt. Dieser Prototyp validiert die Anforderungen und stellt die Machbarkeit sicher.

8 ENTWICKLUNGSUMGEBUNG

Für die Entwicklung der Traildevils iOS App werden verschiedene Tools und Konzepte eingesetzt, die nachfolgend näher erläutert werden.

Beide Autoren verwenden für die Umsetzung ein MacBook Pro Retina mit macOS Sierra. Als Testgeräte kamen nebst den in Xcode verfügbaren Simulatoren auch physische Geräte zum Einsatz, darunter ein iPhone 5, iPhone 6s, ein iPhone 7 und ein iPad Mini 2. Auf den physischen Geräten wurde grundsätzlich immer mit der neusten iOS Version (ab 10.x.x) getestet.

8.1 KOSTEN

Für die Umsetzung der iOS App fallen lediglich die Kosten für die Apple Developer Lizenz an, welche zwingend für das Herausgeben von Apps im App Store ist. Mit einem Preis von $100 pro Jahr fällt dieser Preis aber relativ niedrig aus. Jegliche Services wie zum Beispiel iCloud, Game Center, Apple Pay, Home Kit, Push Notifications, usw. können dann kostenlos genutzt und in der App verwendet werden.

8.2 HARDWARE

Für die Entwicklung der nativen iOS Applikation wird jeweils das persönliche MacBook verwendet. Getestet wird es auf dem mitgelieferten iOS Simulator von Xcode und den privaten iPhones und iPads.

8.3 SOFTWARE & DIENSTE

8.3.1 *Xcode*

Xcode ist die integrierte Entwicklungsumgebung von Apple für die Entwicklung von Apps für iOS, macOS, watchOS und tvOS. Der IDE ist proprietär und deshalb nur für macOS verfügbar, somit werden zwangsweise auch Apple Geräte zur Umsetzung vorausgesetzt. Die Nutzung von Xcode ist kostenlos und kann gratis im Mac App Store heruntergeladen werden.

8.3.2 *Swift*

Swift ist eine sehr junge (Erscheinungsjahr 2014), von vielen anderen Sprachen beeinflusste Programmiersprache zur Entwicklung von Apps. Sie löst die Sprache Objective-C als Hauptprogrammiersprache von Apple ab und ist seit Dezember 2015 Open Source. Zur Zeit ist Swift 3.1 die aktuellste Version. Swift 4 kann mit Xcode 9 Beta bereits verwendet werden.

8.3.3 *TestFlight*

Die App wird von Zeit zu Zeit in einer privaten Gruppe released. Um die App vorläufig zu testen, bietet Apple TestFlight an. Mit TestFlight können Apps, bevor sie im App Store erscheinen, von internen und externen Benutzern aus einem *zweiten App Store* heruntergeladen und getestet werden.

8.3.4 *GitHub*

GitHub wird in erster Linie als zentrales Code Repository verwendet. Zusätzlich werden die Milestones und Issues auf der Platform erfasst. Mittlerweile besitzen die Repositories sogar einen Projekt-Tab, in welchem ein individualisierbares Kanban Board ersichtlich ist.

8.3.5 *Everhour*

Um die aufgewendete Zeit für die Issues zu messen, wurde Everhour verwendet. Ergänzend zu der Planung auf GitHub können Zeiteinheiten den einzelnen Tasks hinterlegt werden. Das Tool generiert anhand der erfassten Zeiten aussagekräftige Reports für die Retrospektive.

8.3.6 *Travis CI*

Continuous Integration wird mit Hilfe des Services von Travis CI eingesetzt. So wird zum Beispiel beim Einreichen eines Pull Requests dessen Gültigkeit geprüft. Eine Webhook aktiviert die Instanz von Travis CI, builded die neue Software und lässt die Tests laufen.

8.3.7 *Sketch*

Um erste Ideen nach den Handskizzen zu visualisieren, wurde Sketch verwendet. Sketch ist ein proprietärer Vektorgrafik-Editor für macOS. Durch den Einsatz in früheren Studienprojekten konnten in kurzer Zeit Screens entworfen werden. Auch Icons und Marker für Kartenelemente wurden mit dem Editor entworfen.

8.3.8 *Proto.io*

Proto.io bietet die Möglichkeit, sehr schnell Prototypen zu erstellen. Durch die entworfenen Screens in Sketch konnten diese über ein Plugin direkt in Proto.io importiert werden.

Anschliessend wurden Flächen über die Screens gelegt, welche es dem Benutzer ermöglichten, mit der App zu interagieren.

8.3.9 *Lookback*

Lookback liefert einfache Möglichkeiten, kollaborative User Tests durchzuführen. Durch die Integration ihres SDKs in der Traildevils iOS App, werden jegliche Interaktionen mit dem Testgerät aufgezeichnet.

Lookback bietet drei Verschiedene Testarten an.

LIVE TESTS: Die Testperson und deren Interaktionen werden live beobachtet.

SELBSTTEST: Die Testperson erhält einen Link, mit welchem direkt die zu testende App geöffnet wird und anschliessend die Interaktionen der Testperson aufgezeichnet werden.

IN-PERSON TESTING: Durch die macOS App von Lookback wird die Möglichkeit geboten den Screen des iPhones über das MacBook aufzuzeichnen.

Die Aufzeichnungen der Testpersonen werden anschliessend auf der Lookback Platform gesammelt. Diese können diskutiert und einzelne Highlights der Aufzeichnungen können extrahiert werden. So war es möglich, nicht nur die Probleme zu diskutieren, sondern auch erfolgreiche Interaktionen festzuhalten.

9 QUALITÄTSMASSNAHMEN

9.1 ENTWICKLUNG

Code Qualität und eine stets stabile Applikation besitzen hohen Stellenwert. Um den Ansprüchen gerecht zu werden, sind folgende Massnahmen notwendig.

9.2 SWIFTLINT

SwiftLint ist ein Linter, der vor jeder Kompilierung eine statische Code-Analyse durchführt und basierend auf einem Regelsatz Warnungen oder Fehler ausgibt.

Basierend auf dem Swift Style Guide von GitHub [50] wird mithilfe des Tools SwiftLint [51] bei jedem Build geprüft, ob die definierten Code Richtlinien eingehalten wurden. Dabei werden die standardmässigen Einstellungen von SwiftLint übernommen. Um Einstellungen anzupassen, kann eine `.swiftlint.yml`-Datei im Root-Verzeichnis angelegt und Ausnahmen definiert werden.

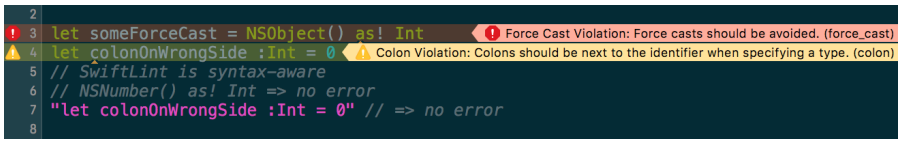


Abbildung 37: SwiftLint Integration in Xcode

Traildevils übernimmt alle Standardeinstellungen für das iOS Projekt, deaktiviert aber das linten in sowohl den Test-Projekten als auch dem Pods-Ordner.

9.3 GIT BRANCHING

Der `master` Branch reflektiert zu jedem Zeitpunkt den aktuellen produktiven Status. Parallel wird während der Entwicklung in einem `development` Branch gearbeitet, von dem dann sogenannte Feature-Git-Branches abgezweigt werden.

Arbeitspakete werden in Form von Feature-Git-Branches umgesetzt. So erhaltet zum Beispiel ein Arbeitspaket *Trails suchen* den Branch-Namen `features/search_trails`. Die Tasks beinhalten dann mehrere Commits. Mit dieser Konstellation werden die Arbeitspakete abgearbeitet. Bei den Commits wird darauf geachtet, dass die sieben Regeln von Chris Beams [52] berücksichtigt werden.

Falls dennoch Hotfixes notwendig sind, zweigen diese direkt vom `master` Branch ab und werden nach dem Fix in den Branch zurück gemerged. Anschliessend wird der Fix auch in den `development` Branch gemerged.

Als Erläuterung dient das Git Branching Model im Anhang B.

9.4 CONTINUOUS INTEGRATION

Nach dem Erstellen des Pull-Requests wird per Webhook automatisch einen Build der Applikation auf Travis CI erstellt. Anschliessend wird die Test Suite ausgeführt. Schlägt der automatische Test fehl, wird der Code an den Entwickler zurück gegeben und muss repariert werden.

9.5 CODE REVIEWS

Ist ein Pull-Request bereit für die Integration, wird das Teammitglied benachrichtigt um den geschriebenen Code zu verifizieren. So wird auch sichergestellt, dass beide Entwickler auf dem gleichen Stand sind und den Überblick über die Code Basis nicht verlieren.

9.6 TESTING

Unit-Tests werden, wo sinnvoll, laufend während der Entwicklung geschrieben. Swift bietet die Möglichkeit gleich im Xcode IDE Testklassen zu erstellen. Jeder Task wird begleitet von den nötigen Tests. So wird die Funktionalität und Performance sichergestellt.

Tests werden mithilfe von Travis CI automatisch bei jedem Push und Pull Request durchgeführt. Das Repository wird in der neuesten Umgebung, sprich iPhone 7 Simulator mit iOS 10.

9.7 CODE STYLE GUIDE

Damit die Konsistenz im Code gewährleistet wird, greifen die Entwickler auf den Code Style Guide von Ray Wenderlich zurück [50].

Grund für die Wahl des Style Guides von Ray Wenderlich ist der vorhandene Support für die Programmiersprache Swift in der Version 3. Durch Einhaltung des Style Guides wird der Code einfacher und schneller zu verstehen sein.

Zudem besagt der Style Guide *Spaces* anstelle von *Tabs* zu verwenden, was die Entwickler sympathischer und angenehmer finden.

9.8 DESIGN

Die Traildevils iOS App haltet sich prinzipiell an die von Apple vorgeschriebenen iOS Human Interface Guidelines [2]. Selbstverständlich werden gewisse Farbakzente mit den beiden *Traildevils-Farben* Orange und Braun gesetzt.

Teil III

APPENDIX

A

AUFGABENSTELLUNG

Aufgabenstellung Bachelorarbeit
Abteilung I, FS 2017
Roman Blum, Philipp Schilter

Traildevils iOS App Streckenplaner

1. Betreuer & Praxispartner

Betreuer dieser Arbeit ist
Prof. Dr. Markus Stolze mstolze@hsr.ch

Co-Referent für diese Arbeit ist
Stefan Keller

Externer Experte für diese Arbeit ist
Thomas Kälin

Praxispartner für dieser Arbeit ist
Mischa Trecco
Frontline Media GmbH
Rosenbergstrasse 9
8630 Rüti

2. Ausgangslage

Ein zentrales Produkt der Frontline Media GmbH ist der Traildevils-Website, eine Online Community Site für Mountain-Bikers. Im HS 2016/17 hatten Roman Blum und Philipp Schilter in ihrer Studienarbeit einen Prototyp einer iOS App für die Traildevils Nutzer entwickelt. In ihrer Arbeit hatten die beiden Studierenden zeigen können, dass sich (aufbauend auf dem Mapbox Kartenservice) eine App entwickeln lässt, die durch die flüssige Bedienbarkeit eine hohe Attraktivität aufweist. Der Fokus dieses Prototyps war die Darstellung von "Points of Interest" (POI: Trails, Bike-Shops etc.) auf der Karte, sowie das flüssige Zoomen und Verschieben auf der Karte - und der Wechsel zwischen POIs und Kartenansicht. Die Daten zu POIs wurden vom Traildevils Server geladen. Weiterhin wurde ein Such-Service integriert.

In der Arbeit explizit ausgeschlossen wurden Funktionen wie Login, Personenprofile und Rating, sowie der ganze Themenbereich zur Streckenplanung.

3. Ziele der Arbeit

In der aktuellen Bachelorarbeit soll nun die Funktionalität der iOS App dahingehend erweitert werden, dass Nutzer in der Lage sind, auf ihrem iOS Gerät im mobilen Kontext einfach und schnell eine Strecke zu planen. Dabei soll zuerst eine Streckenplanung zwischen zwei Punkten ermöglicht werden.

Hauptziel der Arbeit die Entwicklung eines validierten Interaktionskonzeptes für die mobile Streckenplanung mit der Traildevils iOS App. Das Interaktionskonzept soll die folgenden Eigenschaften der App sicherstellen:

- Erfolgreiche Streckenplanung durch eine breitgefächerte Gruppe von Nutzern inklusive Erstnutzer der App ist gewährleistet, d.h. die für die Streckenplanung bereitgestellten App-Funktionen zeichnen sich durch gute "discoverablility" aus, bauen für die die wichtigen Szenarien logisch aufeinander auf, sind vorhersehbar, konsistent und fehlertolerant und schränken Nutzer nicht in der Reihenfolge der Nutzung ein
- Effiziente Nutzung durch erfahrene Nutzer
- Nutzung erfreut Erstnutzer und erfahrene Nutzer
- Reguläre Nutzung (Navigation auf Karte, finden von POI, Suche etc.) wird durch die zusätzlichen Funktionen nicht behindert

Zusätzliche Funktionen zur Erweiterung der Streckenplanung können durch die Studierenden mit dem Auftraggeber

vereinbart werden. Zum aktuellen Zeitpunkt wurde schon eine Reihe möglicher Funktionen und Eigenschaften im Bereich Streckenplanung durch die Studierenden identifiziert:

- Anzeige von Zusatzinformationen zur aktuell geplanten Strecke
- Hinzufügen von zusätzlichen Streckenpunkten als verschiebbarer Zwischenstopp oder zusätzlicher Streckenpunkt am Anfang oder Ende der Strecke
- Hinzufügen eines Bike-Trails (Anfang, Endpunkt und Strecke dazwischen) zu einer geplanten Strecke
- Verschieben von Streckenpunkten auf der Karte
- Ändern der Reihenfolge der abgefahrenen Streckenpunkte
- Löschen von Streckenpunkten

Funktionen, welche als Teil des Interaktionskonzeptes definiert werden, müssen die gleichen UX-Eigenschaften aufweisen wie die Grundfunktionen.

Um das Interaktionskonzept mit Nutzern optimal testen zu können, ist es notwendig, dass die zentralen Funktionen direkt in der App implementiert werden.

Neben der Entwicklung des Interaktionskonzeptes sollen auch die softwaretechnischen Aspekte in der Bachelorarbeit Beachtung finden. Es ist durch die Studierenden mit dem Auftraggeber zu vereinbaren, welche Priorität die die Stabilität und Wartbarkeit des entwickelten Codes hat.

4. Dokumentation

Für den Praxispartner ist eine angemessene Dokumentation der Arbeitsresultate zu verfassen. Umfang und Form dieser Dokumentation ist im Rahmen der Anforderungsanalyse festzulegen. Folgende Elemente sollten enthalten sein: Dokumentation zur Einrichtung der Entwicklungsumgebung, Kompilierung und Testen der App sowie Instruktion zum Publizieren der App, so dass Übernahme und Weiterentwicklung des Projekts einfach möglich ist.

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen. Die Dokumentation (inkl. Sourcecode) ist vollständig entsprechend den Instruktionen des Studiengangs abzugeben. Zudem ist ein Download-Link für Prof. Stolze und weitere Exemplare nach Absprache mit dem Co-Referenten und dem Experten bereitzustellen.

Zudem ist eine kurze Projektresultatdokumentation im öffentlichen Wiki von Prof. Dr. M. Stolze zu erstellen.

5. Weitere Regeln und Termine

Weitere Kriterien zur Bewertung der Arbeit und der Dokumentation sind in dem separat verteilten Bewertungsraster (Bewertungsraster-2017-FS-BA-Blum+Schilter-TrailDevilsAppFS17) festgehalten. Das Bewertungsraster kann in Absprache zwischen Betreuer und Studierenden im Rahmen der geltenden Regeln der Abteilung Informatik adaptiert werden.

Im Weiteren gelten die allgemeinen Regeln zu Bachelor und Studienarbeiten
„Abläufe und Regelungen Studien- und Bachelorarbeiten im Studiengang Informatik“ (HSR Intranet)
https://www.hsr.ch/Ablaeufe-und-Regelungen-Studie.7479.0.html

Der Terminplan ist hier ersichtlich (HSR Intranet)
https://www.hsr.ch/Termine-Bachelor-und-Studiena.5142.0.html

6. Rechte

Die resultierende Software und Dokumentation darf von den Studenten, vom Auftraggeber und der HSR genutzt und erweitert werden. Eine Veräusserung an Dritte erfordert die Zustimmung des Praxispartners. Der Source-Code darf ohne die Einwilligung der Studierenden und des Praxispartners nicht veröffentlicht werden. Eine Veröffentlichung in Ausschnitten (z.B. Blogposts und öffentliche Dokumentation der Bachelorarbeit) durch die Studierenden ist erlaubt. Es wird durch alle Parteien sichergestellt, dass im Source-Code und im Impressum der App die originale Urheberschaft durch die HSR Studenten weiterhin sichtbar bleibt.
Der Bericht der Bachelorarbeit (ohne geheime Anhänge) wird von der HSR im E-Prints Respository der HSR (eprints.hsr.ch) elektronisch veröffentlich. Titel, Abstract und Praxispartner der Arbeit dürfen von der HSR und Studierenden schon während der Arbeit kommuniziert werden.

7. Beurteilung

Eine erfolgreiche Bachelorarbeit zählt 12 ECTS-Punkte pro Studierenden. Für 1 ECTS Punkt ist eine Arbeitsleistung von ca. 25 bis 30 Stunden budgetiert. Entsprechend sollten ca. 350h Arbeit für die Bachelorarbeit aufgewendet werden. Dies entspricht ungefähr 25h pro Woche (auf 14 Wochen) und damit ca. 3 Tage Arbeit pro Woche pro Student.

Für die Beurteilung ist der HSR-Betreuer verantwortlich. Die Bewertung der Arbeit erfolgt entsprechend der verteilten Kriterienliste.

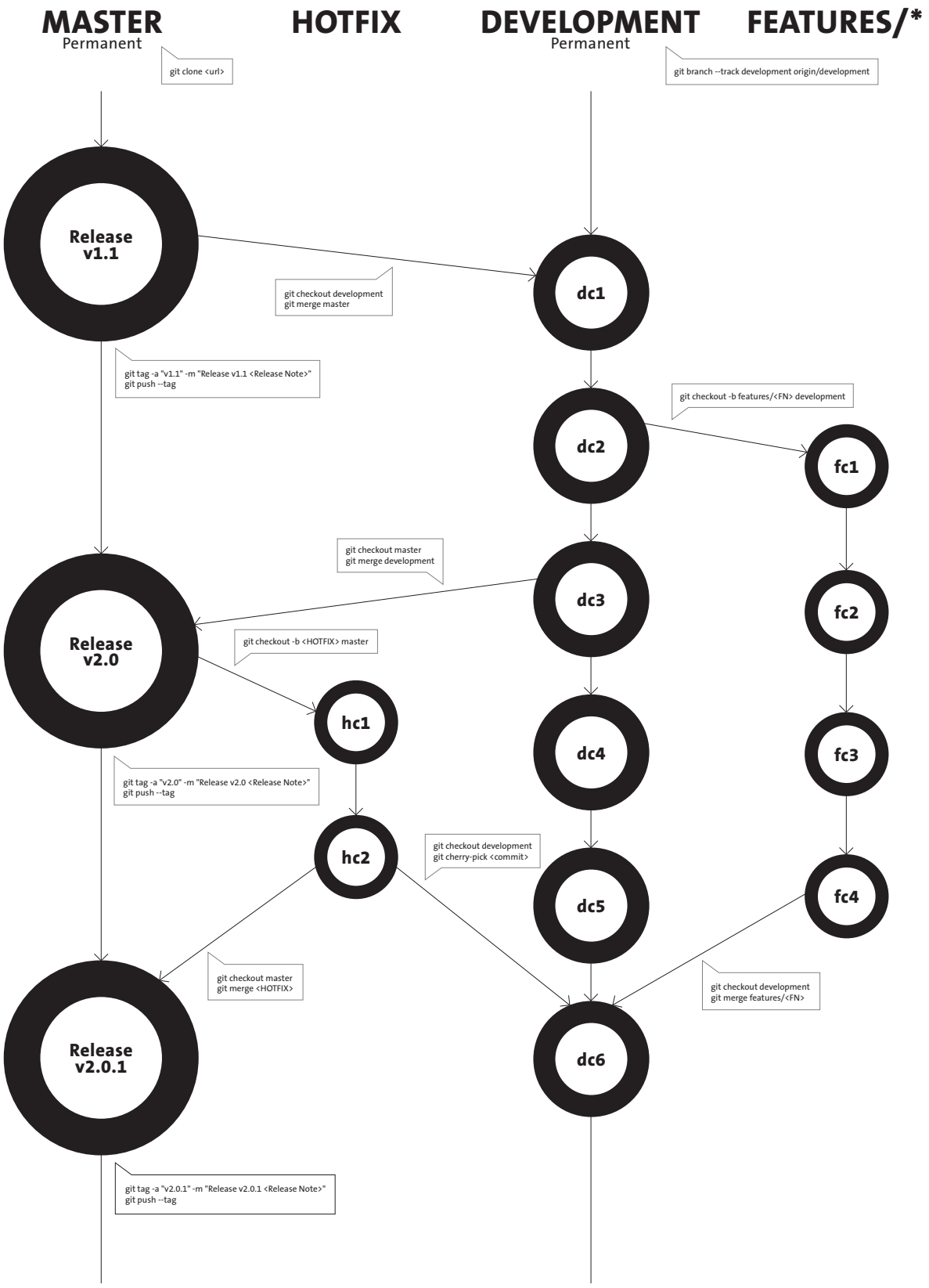
Diese definitive Aufgabenstellung wurde am 18.4.2017 beschlossen.

Rapperswil, 18.4.2017
Prof. Dr. Markus Stolze, Institut für Software, Hochschule für Technik Rapperswil

B

GIT BRANCHING MODEL

GIT BRANCHING MODEL



LITERATUR

[1] Philipp Schilter Roman Blum. *Studienarbeit Traildevils iOS App, Architektur*. 2016. URL: https://eprints.hsr.ch/553/ (besucht am 09. 06. 2017).

[2] Apple. *Apple iOS Human Interface Guidelines*. 2017. URL: https://developer.apple.com/ios/human-interface-guidelines/ (besucht am 03. 03. 2017).

[3] Jeff Sauro. *WHAT IS A GOOD TASK-COMPLETION RATE?* 2011. URL: https://measuringu.com/task-completion/ (besucht am 13. 06. 2017).

[4] SZB. *Schweizerischer Zentralverein für das Blindenwesen*. 2017. URL: http://www.szb.ch/szb/ (besucht am 09. 06. 2017).

[5] Mapbox. *Offline maps with Mapbox Mobile*. 2017. URL: https://www.mapbox.com/help/mobile-offline/#tile-ceiling--limits (besucht am 27. 05. 2017).

[6] GraphHopper. *FAQ - GraphHopper Directions API Documentation*. 2017. URL: https://graphhopper.com/api/1/docs/FAQ/ (besucht am 27. 05. 2017).

[7] OpenStreetMap. *Nominatim - OpenStreetMap Wiki*. 2017. URL: https://wiki.openstreetmap.org/wiki/Nominatim (besucht am 27. 05. 2017).

[8] OpenCage Data. *OpenCage Geocoder - Easy, Open, Worldwide, Affordable Geocoding*. 2017. URL: https://geocoder.opencagedata.com/ (besucht am 27. 05. 2017).

[9] GraphHopper. *Preise - GraphHopper Directions API*. 2017. URL: https://www.graphhopper.com/de/preise/ (besucht am 27. 05. 2017).

[10] Tobias Hallerman. *Presse*. 2017. URL: https://www.komoot.de/press (besucht am 28. 03. 2017).

[11] komoot GmbH. *imprint*. 2017. URL: https://www.komoot.de/imprint (besucht am 28. 03. 2017).

[12] Outdooractive GmbH & Co. KG. *Das ist Outdooractive*. 2017. URL: https://www.outdooractive.com/de/ueber-uns.html (besucht am 28. 03. 2017).

[13] Outdooractive GmbH & Co. KG. *Impressum*. 2017. URL: https://www.outdooractive.com/de/impressum.html (besucht am 28. 03. 2017).

[14] Google. *About - Google Maps*. 2017. URL: https://www.google.com/maps/about/ (besucht am 04. 05. 2017).

[15] Dr. Sybille Peuker. *Lektion 4 - Konzeption & Prototyping*. 2016.

[16] Jakob Nielsen. *First Rule of Usability? Don't Listen to Users*. 2001. URL: https://www.nngroup.com/articles/first-rule-of-usability-dont-listen-to-users/ (besucht am 13. 06. 2017).

[17] Tomer Sharon. *The Rainbow Spreadsheet: A Collaborative Lean UX Research Tool*. 2013. URL: https://www.smashingmagazine.com/2013/04/rainbow-spreadsheet-collaborative-ux-research-tool/ (besucht am 07. 04. 2017).

[18] James R. Lewis. "Psychometric Evaluation of an After-scenario Questionnaire for Computer Usability Studies: The ASQ". In: *SIGCHI Bull.* 23.1 (Jan. 1991), S. 78–81. ISSN: 0736-6906. DOI: 10.1145/122672.122692. URL: http://doi.acm.org/10.1145/122672.122692.

[19] Jeff Sauro und Joseph S. Dumas. "Comparison of Three One-question, Post-task Usability Questionnaires". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '09. Boston, MA, USA: ACM, 2009, S. 1599–1608. ISBN: 978-1-60558-246-7. DOI: 10.1145/1518701.1518946. URL: http://doi.acm.org/10.1145/1518701.1518946.

[20] Mapbox. *Mapbox Directions API*. 2017. URL: https://www.mapbox.com/directions/ (besucht am 13. 06. 2017).

[21] GraphHopper. *Routing API - GraphHopper Directions API Documentation*. 2017. URL: https://graphhopper.com/api/1/docs/routing/ (besucht am 13. 06. 2017).

[22] OpenRouteService. *OpenRouteService*. 2017. URL: https://www.openrouteservice.org (besucht am 13. 06. 2017).

[23] Mapbox. *Mapbox Directions API Documentation*. 2017. URL: https://www.mapbox.com/api-documentation/#directions (besucht am 13. 06. 2017).

[24] GraphHopper. *FAQ - GraphHopper Directions API Documentation*. 2017. URL: https://graphhopper.com/api/1/docs/routing/#parameters (besucht am 13. 06. 2017).

[25] OpenRouteService. *Build, Collaborate & Integrate APIs | SwaggerHub*. URL: https://app.swaggerhub.com/apis/OpenRouteService/ors-api/4.0#operations,get-/directions,default (besucht am 13. 06. 2017).

[26] Mapbox. *Surface API | Mapbox*. 2017. URL: https://www.mapbox.com/developers/api/surface/ (besucht am 13. 06. 2017).

[27] Mapbox. *Directions API Pricing*. 2017. URL: https://www.mapbox.com/directions/#pricing (besucht am 13. 06. 2017).

[28] Mapbox. *MapboxDirections*. 2017. URL: https://github.com/mapbox/MapboxDirections.swift (besucht am 13. 06. 2017).

[29] Uber. *Swift with a hundred engineers*. 2017. URL: https://www.skilled.io/u/swiftsummit/swift-with-a-hundred-engineers (besucht am 04. 05. 2017).

[30] Trygve Reenskaug. *Models - Views - Controllers*. 1979. URL: http://folk.uio.no/trygver/2007/MVC_Originals.pdf (besucht am 08. 06. 2017).

[31] Apple. *Model-View-Controller*. URL: https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html (besucht am 08. 06. 2017).

[32] Apple. *UIViewController - UIKit*. 2017. URL: https://developer.apple.com/documentation/uikit/uiviewcontroller (besucht am 08. 06. 2017).

[33] Bohdan Orlov. *iOS Architecture Patterns*. URL: 2017 (besucht am 08. 06. 2017).

[34] Soroush Khanlou. *Massive View Controller*. 2015. URL: http://khanlou.com/2015/12/massive-view-controller/ (besucht am 08. 06. 2017).

[35] Jeremiah Gage. *Refactoring a Massive View Controller in Swift*. 2016. URL: https://vimeo.com/182489008 (besucht am 08. 06. 2017).

[36] Rui Peres. *Model-View-Controller (MVC) in iOS: A Modern Approach*. 2016. URL: https://www.raywenderlich.com/132662/mvc-in-ios-a-modern-approach (besucht am 08. 06. 2017).

[37] Philipp Schilter Roman Blum. *Studienarbeit Traildevils iOS App, Frameworks*. 2016. URL: https://eprints.hsr.ch/553/ (besucht am 09. 06. 2017).

[38] GraphHopper. *API Clients and Examples*. 2017. URL: https://graphhopper.com/api/1/docs/#api-clients-and-examples (besucht am 08. 06. 2017).

[39] Roman Blum. *Fix half-black screen after dismissal*. 2017. URL: https://github.com/hyperoslo/Lightbox/pull/103 (besucht am 09. 06. 2017).

[40] Roman Blum. *Fix half-black screen after dismissal*. 2017. URL: https://github.com/hyperoslo/Lightbox/pull/103 (besucht am 09. 06. 2017).

[41] Roadfire Software. *Should you use Storyboards on more complex apps?* 2015. URL: http://roadfiresoftware.com/2015/03/the-pros-and-cons-of-using-storyboards/ (besucht am 09. 06. 2017).

[42] Mapbox. *Ambient Caching*. 2017. URL: https://www.mapbox.com/help/mobile-offline/#ambient-caching (besucht am 09. 06. 2017).

[43] Sim Daltonism. *The color blindness simulator.* 2017. URL: https://michelf.ca/projects/sim-daltonism/ (besucht am 09. 06. 2017).

[44] Al Danial. *cloc counts blank lines, comment lines, and physical lines of source code in many programming languages.* 2017. URL: https://github.com/AlDanial/cloc (besucht am 13. 06. 2017).

[45] T. J. Watson A. H. & McCabe. *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*. Gaithersburg, Maryland, USA, Sep. 1996. URL: http://www.mccabe.com/pdf/mccabe-nist235r.pdf.

[46] Schweizerische Eidgenossenschaft. *Ergebnis: Abnahmeprotokoll*. 2017. URL: http://www.hermes.admin.ch/szenarien/szenario_50_Alles/scenario/de/ergebnis_abnahmeprotokoll.html (besucht am 14. 06. 2017).

[47] Philipp Schilter Roman Blum. *Studienarbeit Traildevils iOS App, Funktionsumfang*. 2016. URL: https://eprints.hsr.ch/553/ (besucht am 09. 06. 2017).

[48] Philipp Schilter Roman Blum. *GraphHopperGeocoder - The GraphHopper Geocoding API wrapped in an easy-to-use Swift framework.* 2017. URL: https://github.com/rmnblm/GraphHopperGeocoder (besucht am 09. 06. 2017).

[49] Philipp Schilter Roman Blum. *GraphHopperRouting - The GraphHopper Routing API wrapped in an easy-to-use Swift framework.* 2017. URL: https://github.com/rmnblm/GraphHopperRouting (besucht am 09. 06. 2017).

[50] raywenderlich.com. *The official Swift style guide for raywenderlich.com*. 2017. URL: https://github.com/raywenderlich/swift-style-guide (besucht am 09. 06. 2017).

[51] Realm. *SwiftLint: A tool to enforce Swift style and conventions.* 2017. URL: https://github.com/realm/SwiftLint (besucht am 03. 03. 2017).

[52] Chris Beams. *How to Write a Git Commit Message*. 2017. URL: http://chris.beams.io/posts/git-commit/ (besucht am 03. 03. 2017).

EIGENSTÄNDIGKEITSERKLÄRUNG

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde.
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.
- dass wir keine durch Copyright geschützten Materialien (zum Beispiel Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Rapperswil, 16. Juni 2017

Roman Blum

Philipp Schilter

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography *"The Elements of Typographic Style"*. `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/