HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

FRONTLINE
MEDIA

Traildevils iOS App

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2016

Autoren:	Roman Blum, Philipp Schilter
Betreuer:	Prof. Dr. Markus Stolze
Projektpartner:	Frontline Media GmbH, Rüti

Abstract

Ausgangslage: Ein zentrales Produkt der Frontline Media GmbH ist die Traildevils Website, eine Online Community Site für Mountain Biker. Der Auftraggeber möchte die Attraktivität durch eine native iOS Applikation steigern. Zur Darstellung von Kartendaten soll die Mapping Plattform Mapbox verwendet werden.

Vorgehen/Technologien: In einer ersten Phase wurde aufgrund fehlender Erfahrung mit dem iOS Mapbox SDK ein Prototyp erstellt. Anfänglich identifizierte Probleme konnten gegen Ende aus dem Weg geräumt werden. Es konnte gezeigt werden, dass sich die zentralen, funktionalen und nichtfunktionalen Anforderungen mit Mapbox auf iOS Geräten umsetzen lassen. Für einzelne Anforderungen war es nötig, Workarounds zu definieren, bei anderen wurde Mapbox durch Features erweitert.

Der zweite Teil beschäftigt sich mit der konkreten Implementierung. Mit der Programmiersprache Swift und der Software-Architektur VIPER wurden viele der gewünschten Features in die App implementiert und für den Projektpartner zum Testen über TestFlight zur Verfügung gestellt.

Ergebnis: Im Rahmen der Studienarbeit zeigte sich, dass das SDK für iOS weit weniger fortgeschritten ist als jenes in JavaScript. Zum Vorteil von uns durchlief die eingesetzte Version 3.4.0 mehrere Alpha- und Beta-Releases, wodurch wöchentlich neue Features dazukamen. Ein fehlendes, aber doch zwingend notwendiges Feature (POI Icons für Style-Layers), konnte von uns noch während der Experimentierphase im Kern des Frameworks implementiert werden. Unser Code wird auch Bestandteil des nächsten, offiziellen Releases sein.

Am Ende dieser Arbeit kann eine iOS App vorgewiesen werden, deren zentrale Eigenschaft die schnelle und flüssige Reaktion auf Benutzerinteraktionen mit der Karte ist. Details zu Kartenelementen werden aus dem Offline Cache oder von der Webschnittstelle geladen und angezeigt. Eine schnelle Suchfunktion für Traildevils- und öffentliche Kartendaten wurde ebenfalls implementiert. Als Nebenprodukt der Implementierung der responsiven Detail-Ansicht für Kartenelemente wurde eine generelle iOS-GUI Komponente entwickelt und unter dem Namen “Raclette” auf GitHub veröffentlicht.

Der Grundstein für eine erfolgreiche Weiterentwicklung und Veröffentlichung im App Store konnte durch den Abschluss dieser Arbeit gelegt werden. Bis aber die erste Version der Community zur Verfügung gestellt werden kann, fehlt es noch an Fleissarbeit. Insbesondere müssen noch 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. Die Traildevils Website ist schon heute mit dem Webbrowser von Mobilgeräten aus zugreifbar. Der Auftraggeber möchte die Attraktivität der Community durch eine native iOS Applikation steigern. Zur Darstellung von Kartendaten soll die Mapping Plattform Mapbox verwendet werden.

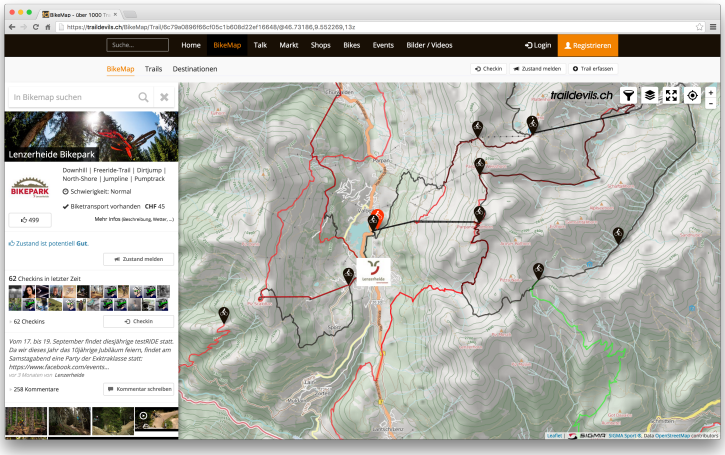


Abbildung 1: Traildevils Website mit BikeMap

Vorgehen und Technologien

In einer ersten Phase wurde aufgrund fehlender Erfahrung mit dem iOS SDK ein Prototyp erstellt. Die gemachten Experimente wurden zusätzlich in einem GitHub Repository dokumentiert. Dies hatte einerseits den Vorteil, dass bei Problemen mit Mapbox in Issues darauf verwiesen werden konnte, andererseits konnten dadurch wichtige Erkenntnisse und Best-Practices der Community zur Verfügung gestellt werden. Anfänglich identifizierte Probleme mit Mapbox konnten gegen Ende aus dem Weg geräumt werden.

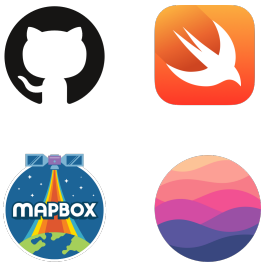


Abbildung 2: Zentrale Komponenten der App

Es konnte gezeigt werden, dass sich die zentralen, funktionalen und nicht-funktionalen Anforderungen mit Mapbox auf iOS Geräten umsetzen lassen. Für einzelne Anforderungen war es nötig, Workarounds zu definieren, bei anderen wurde Mapbox durch Features erweitert.

Der zweite Teil beschäftigt sich mit der konkreten Implementierung. Mit der Programmiersprache Swift und der Software-Architektur VIPER wurden viele der gewünschten Features in die App implementiert und für den Projektpartner zum Testen über TestFlight zur Verfügung gestellt. VIPER verfolgt einen anderen Ansatz als bekanntere Architekturen wie MVC oder MVVM. In VIPER ist jede View bzw. jeder Use Case ein Modul bestehend aus mehreren Komponenten. Jede Komponente erfüllt dabei immer nur eine kleine Aufgabe einer gesamten Logik. Der Vorteil dabei ist, dass jedes Modul unabhängig und entkoppelt vorliegt und jede einzelne Komponente seperat besser getestet werden kann.

Ergebnisse

Im Rahmen der Studienarbeit zeigte sich, dass das SDK für iOS weit weniger fortgeschritten ist als jenes in JavaScript. Zum Vorteil von uns durchlief die eingesetzte Version 3.4.0 mehrere Alpha- und Beta-Releases, wodurch wöchentlich neue Features dazukamen. Ein fehlendes, aber doch zwingend

notwendiges Feature (POI Icons für Style-Layers), konnte von uns noch während der Experimentierphase im Kern des Frameworks implementiert werden. Unser Code wird auch Bestandteil des nächsten, offiziellen Releases sein.

Am Ende dieser Arbeit kann eine iOS App vorgewiesen werden, deren zentrale Eigenschaft die schnelle und flüssige Reaktion auf Benutzerinteraktionen mit der Karte ist. Details zu Kartenelementen werden aus dem Offline Cache oder von der Webschnittstelle geladen und angezeigt. Eine schnelle Suchfunktion für Traildevils- und öffentliche Kartendaten wurde ebenfalls implementiert. Als Nebenprodukt der Implementierung der responsiven Detail-Ansicht für Kartenelemente wurde eine generelle iOS-GUI Komponente entwickelt und unter dem Namen “Raclette” auf GitHub veröffentlicht.

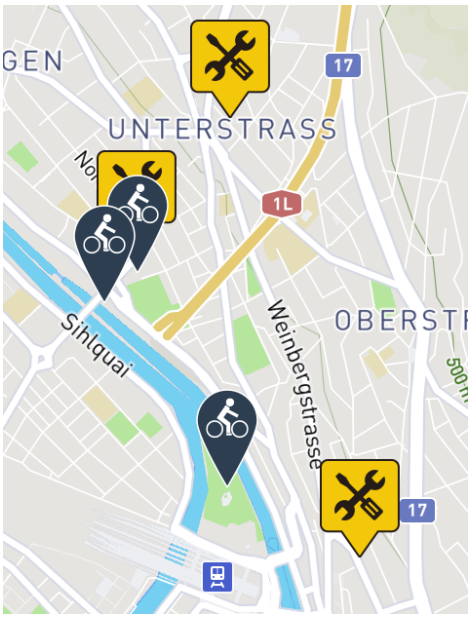


Abbildung 3: Vor der SA noch keine Selbstverständlichkeit: POI Icons für Style-Layers

Ausblick

Der Grundstein für eine erfolgreiche Weiterentwicklung und Veröffentlichung im App Store konnte durch den Abschluss dieser Arbeit gelegt werden. Der nächsten Schritt wäre aus der Sicht der Autoren die Umsetzung der Benutzeranmeldung und Benutzerregistrierung.

Abbildung 4: Screenshots aus der finalen Applikation

So können Funktionen freigeschaltet werden, welche einen eingeloggten Benutzer voraussetzen, wie z.B. ein Trail Check-In, einen Zustand melden, ein Kartenelement bewerten und kommentieren, oder etwa einen Rider anzeigen. Zudem kann ein Aktivitätenstream gebaut werden, welcher bereits im Einsatz ist auf der Webseite.

Inhaltsverzeichnis

Teil I

Technischer Bericht

Kapitel 1

Einleitung

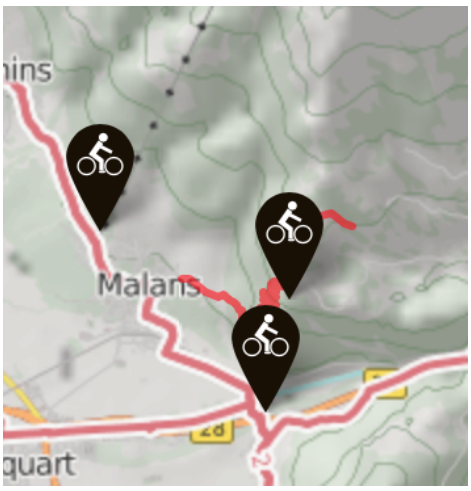
1.1 Vision

Der Fokus dieser Studienarbeit liegt darin, die aufgetretenen Probleme der geschriebenen Bachelorarbeit “Traildevils App” von Mirco Stässle und Robert Vogt zu lösen. Es hat sich gezeigt, dass die Implementation mit React Native zwar eine einfache Methode ist, um Cross-Platform-Apps zu entwickeln, aber aufgrund der noch fehlenden nativen Unterstützung und des hohen Technologiestacks haben sich einige Hürden und Probleme während der Nutzung gezeigt.

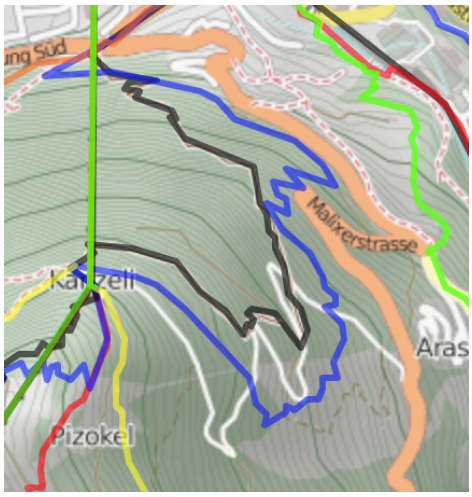
Das Ziel am Ende dieser Arbeit ist ein Prototyp, der alle technischen und Nutzer-bezogenenen Risiken eliminiert.

Hinweis: In den nachfolgenden Kapiteln wird häufig der Begriff “Kartenelement” verwendet. Damit sind von Traildevils stammende, geografische Daten gemeint. Dabei werden die englischen Ausdrücke verwendet, sprich Trails (Startpunkte von Abfahrten), Tracks (Abfahrten), Dealers (Shops) und Destinationen.

a: Eine Destination



b: Mehrere Trails



c: Tracks in unterschiedlichen Farben

d: Ein Dealer

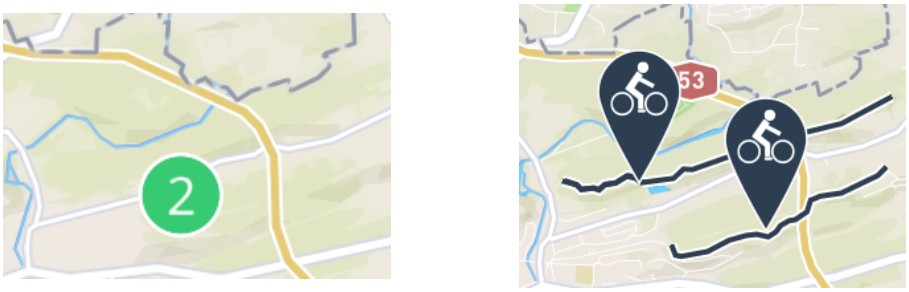
Abbildung 1.1: Kartenelemente von Traildevils

1.2 Herausforderungen

Es besteht nun die Herausforderung, herauszufinden, ob die aufgetretenen Problemen der vorangehenden Bachelorarbeit mit einer nativen Implementation eliminiert werden können. Nachfolgenden werden diese Herausforderungen, die es zu bewältigen gibt, kurz beschrieben.

1.2.1 Clustering

Je nach Zoom-Stufe ist es schwierig alle gewünschten Kartenelemente anzuzeigen. Das Clustering beschreibt das Gruppieren von Point-of-Interest Markern (POIMs), um den Benutzer nicht mit Informationen zu überfluten. Eine Gruppe von POIMs wird zusammengefasst zu einem Cluster Marker, welcher standardmässig die Anzahl darunter liegender POIMs anzeigt. Beim Zoom-In unterteilen sich die grossen Gruppen in kleinere Gruppen. Dieser Vorgang wird bis zu einem definierten Zoomelevel wiederholt und die POI Marker sichtbar werden.



a: Cluster Marker

b: POI Marker

Abbildung 1.2: Ab einem bestimmten Zoomlevel werden Cluster Marker zu POI Marker

1.2.2 Semantic Zooming

Semantic Zoom ist eine Form von Details auf Anfrage, welche dem Benutzer abhängig von der Zoom-Stufe eine unterschiedliche Menge an Details betrachten lässt. [1]

1.2.3 Panning

Panning beschreibt das flüssige Nachladen von Markers beim Verschieben der Karte. Bei der Auswahl eines neuen Kartenausschnitts sollen die neuen POIs performant auf dem Screen erscheinen.

1.2.4 Marker Selection Feedback

Die Berührung eines Markers soll dessen Status auf “selektiert” setzen und dies auch visuell ersichtlich machen. Eine Selektion soll auch wieder rückgängig gemacht werden.

1.2.5 Filtering

Die Herausforderung beim Filtering ist, Kartendaten zur Laufzeit anhand von Attributen ein- oder auszublenden.

1.2.6 Path Selection

Tracks sollen wie die Marker einen selektierten und nicht-selektierten Status haben. Bei der Auswahl eines Tracks soll sich dieser visuell zu den nicht-selektierten Tracks unterscheiden. Bei der Selektion eines Tracks soll auch dessen Startpunkt (Trail) selektiert werden.

Kapitel 2

Einführung in Mapbox

Abbildung 2.1: Mapbox Kartenplattform: Dem Styling sind keine Grenzen gesetzt (Quelle: Mapbox Presskit)

Mapbox ist eine auf OpenStreetMap basierende Kartenplattform, mit der sich Karten für Print, Web und mobile Anwendungen erstellen und integrieren lassen. Die Stärke von Mapbox besteht in den zahlreichen Möglichkeiten, die Karte persönlich gestalten zu können: Nach der Registrierung auf Mapbox können über eine grafische Oberfläche die einzelnen Layer der Karte individualisiert werden. Die Farben von Strassen, Gebieten, Gewässern,

Ländern und allen anderen Objekten der Karte lassen sich frei definieren. Mapbox bietet Frameworks für zahlreiche Plattformen, darunter auch iOS.

Um die nachfolgenden Experimente besser zu verstehen, werden hier kurz wichtige Konzepte von Mapbox erläutert.

2.1 GeoJSON Format

GeoJSON ist ein Dateiformat zur Codierung geografischer Datenstrukturen. Es wurde im Jahre 2008 spezifiziert [2] und ist seit August 2016 mit dem Standard RFC 7946 [3] offiziell anerkannt.

Das Format beschreibt die geometrischen Typen: `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString` und `MultiPolygon`. Geometrische Objekte mit zusätzlichen Eigenschaften werden als `Feature` Objekte bezeichnet.

Das folgende Beispiel zeigt den Aufbau einer GeoJSON Datei.

```
1  {
2    "type": "Feature",
3    "geometry": {
4      "type": "Point",
5      "coordinates": [125.6, 10.1]
6    },
7    "properties": {
8      "name": "Dinagat Islands"
9    }
10 }
```

2.2 Die Karte: MGLMapView

Das Kernelement von Mapbox ist die Klasse `MGLMapView` mit dem dazugehörigen Delegate `MGLMapViewDelegate`. Diese Klasse dient als Eintrittspunkt für interaktive Anpassungen der Karte. Es können GeoJSON Daten hinzuge-

fügt, Style Layers definiert oder Ansichtsänderungen vorgenommen werden.

2.3 Auf Events registrieren: MGLMapViewDelegate

Um zu verstehen, wozu der `MGLMapViewDelegate` ist, muss zuerst erläutert werden, wozu Delegates dienen. In vielen iOS-Frameworks kommt das Delegate-Pattern zum Einsatz. Dabei benachrichtigt ein Objekt ein konfigurierbares delegate-Objekt über aufgetretene Ereignisse. Dieses Pattern ist vor allem nützlich für das Zusammenspiel zwischen `View` und `ViewController`. Der `ViewController` registriert sich dabei als delegate bei der `View` und wird dann über Ereignisse informiert.

Analaog verhält es sich unter Mapbox: Zuerst wird eine Instanz von `MGLMapView` erstellt. Der verantwortliche `ViewController` wird dann als delegate der Instanz zugewiesen. Dadurch wird er über Ereignisse von Mapbox wie zum Beispiel über Karteninteraktionen informiert. Ein Beispiel wäre die Änderung der aktuellen GPS-Position des Benutzers auf der Karte. Bei einer Änderungen wird der `ViewController` über die Funktion `public func mapView(_ mapView: MGLMapView, didUpdate userLocation: MGLUserLocation?)` darüber informiert.

```
1 let mapView = MGLMapView()
2 mapView.delegate = self
3 self.addSubview(mapView)
```

2.4 Annotationen

Mapbox trennt das Anzeigen von Daten in Annotationen und Style Layers. Annotationen dienen zur Selektion eines **einzelnen** POIs, also von einem Punkt (`MGLPoint`), einer Linie (`MGLPolyline`) oder einer Fläche (`MGLPolygon`). All diese Klassen basieren auf der Basis-Klasse `MGLAnnotation`. Aus der Dokumentation:

> The `MGLAnnotation` protocol is used to provide annotation-

> related information to a map view. To use this protocol, you adopt it in any custom objects that store or represent annotation data. Each object then serves as the source of information about a single map annotation and provides critical information, such as the annotation's location on the map. Annotation objects do not provide the visual representation of the annotation but typically coordinate (in conjunction with the map view's delegate) the creation of an appropriate objects to handle the display. - Source [4]

2.5 Style Layers

Im Unterschied zu einer Annotation dienen Style Layers zum **Beschreiben** von Feature Collections. Diese Style Layers werden wiederum unterteilt in verschiedene "Darstellungsarten". Zum Beispiel zeigt `MGLCircleStyleLayer` jede Koordinate eines Features mit `type": Point"` als einen farbigen Punkt auf der Karte an. Natürlich können Farbe, Transparenz und diverse andere Eigenschaften auf jedem Style Layer seperat angegeben werden.

Abbildung 2.2: Unterteilung von Mapbox in verschiedene Ebenen

Die wichtigste Eigenschaft (Property) einer Style Layer Instanz ist das Prädikat (`predicate`), welches vom Typ `NSPredicate` ist. Aufgrund einer *logischen* Bedingung können dann gewisse Daten angezeigt werden, wenn das

Prädikat `true` zurückgibt.

Um zum Beispiel **nur** einen Punkt aus einer GeoJSON Feature Collection auf der Karte anzeigen zu lassen, würde ein Style Layer mit folgendem Prädikat erstellt und der Karte hinzugefügt werden:

```
1 styleLayer.predicate = NSPredicate(format: "name == '
      ↪ Dinagat Islands'")
```

2.6 To Parse or Not To Parse

Die ersten Experimente mit Mapbox zeigten, dass Datenquellen, normalerweise `.geojson`-Daten, unter Mapbox in zwei verschiedenen Arten zur Karte hinzugefügt werden. Beide Arten bieten zum jetzigen Stand von Mapbox-iOS-SDK Vor- und Nachteile, da gewisse Funktionen bei der einen Variante vorhanden, bei der anderen aber fehlend sind (und umgekehrt).

Variante 1: Das Parsen selber in die Hand nehmen

Bei der ersten Variante werden die JSON-Daten nicht von Mapbox geparst. Dabei wird über alle Elemente iteriert und jedes Element der Karte als Annotation manuell hinzugefügt. Für das Parsen kann beispielsweise ein Framework wie SwiftyJSON [5] verwendet werden. Nachfolgendes Beispiel zeigt das Laden einer GeoJSON-Datei und das anschliessende Parsen der einzelnen Array-Werte.

```
1 let jsonPathUrl = Bundle.main.url(forResource: "mcdonalds",
      ↪  withExtension: "geojson")
2 let jsonData = try Data(contentsOf: jsonPathUrl!)
3 let json = JSON(data: jsonData)
4
5 for feature in json["features"].arrayValue {
6     if feature["geometry"]["type"].stringValue == "Point" {
7         let coordinates = feature["geometry"]["coordinates"
      ↪ ].arrayValue
8         let point = MGLPointAnnotation()
9         point.coordinate = CLLocationCoordinate2D(latitude:
```

```
        ↪  coordinates[1].doubleValue, longitude:
        ↪ coordinates[0].doubleValue)
10          mapView.addAnnotation(point)
11      }
12  }
```

Unsere Analyse hat gezeigt, dass die Performance leidet, wenn ein extrem grosses GeoJSON geparst und Punkt-Annotationen erstellt werden müssen (bis zu 5 Sekunden Wartezeit bei einem JSON mit mehr als 5000 Elementen auf einem iPhone 7).

Vorteile:

- Jedes Feature des GeoJSON kann als seperate Punkt-Annotation zur Karte hinzugefügt werden. Dies hat den Vorteil, dass zum jetzigen Zeitpunkt (3.4.0-alpha.4) Punkt-Annotationen besser *customized* werden (was auch wichtig für die spätere Umsetzung von Traildevils ist, um zum Beispiel bei Destinationen ein "Berg-Icon" anzuzeigen).

Nachteile:

- Massiver Performance-Verlust bei einer hohen Menge an Daten
- Clustering kann nicht verwendet und müsste ebenfalls manuell implementiert werden

Variante 2: Die Arbeit Mapbox überlassen

Mapbox bietet mit der Klasse `MGLGeoJSONSource` die Möglichkeit, im Konstruktor eine URL zu einer `.geojson`-Datei zu geben. Das Parsen wird somit von Mapbox übernommen. Basierend auf der Source können nun Style Layers definiert werden.

Vorteile:

- Die Aufgabe des Parsen wird Mapbox überlassen und es wird kein zusätzliches Framework benötigt

- Hohe Performance
- Daten können aufgrund von Style Layers und mithilfe von Prädikaten ein- und ausgeblendet werden

Nachteile:

- Minimale Anpassungsmöglichkeiten von Style Layers für die darzustellenden POIs (nur Farben, Borders, Opacity, usw.)

2.7 Warten oder selber implementieren?

Beide Varianten aus vorherigem Kapitel sind nicht wirklich zufriedenstellend. Die erste Variante bietet bessere Anpassungsmöglichkeiten der POI-Markers, die zweite Möglichkeit bietet höhere Performance und vorallem Clustering.

Hinweis: Die nach dem Hashtag # folgende Nummer in den Klammern ist die eindeutig referenzierbare Nummer eines Issues auf GitHub. Um mehr Details zu erhalten, kann diese in der URL `https://github.com/mapbox/mapbox-gl-native/issues/<ID ohne Hastag>` ersetzt werden, z.B. `https://github.com/mapbox/mapbox-gl-native/issues/5727`

Dies kann mit der aktuellsten Alpha-Version nicht umgesetzt werden. Der Issue #6436 dokumentierte diese Problematik, in dem mit dem Mapbox-Entwickler **1ec5** eine Alternativlösung bzw. ein Workaround gesucht wurde.

Deshalb wurde beschlossen, die API von Mapbox zu erweitern und zu hoffen, dass die Änderungen den Weg in das Framework und in den nächsten, offiziellen Release findet. Der Pull Request mit der ID #6637 und dem Titel "[ios, macos] possibility to set custom images to style" dokumentiert die komplette Implementierung dieses Features. Am 12. Oktober 2016 wurde der Pull Requst in den master-Branch aufgenommen. Das Feature ist nun seit Version 3.4.0-beta.1 Bestandteil von Mapbox-iOS-SDK und wird Teil des offiziellen Releases sein.

2.7.1 Umsetzung

Drei Wochen vor der Implementierung dieses Features wurde das Feature zur Bearbeitung des Sprite Atlas im Mapbox-GL-Core durch den PR #6375 implementiert. Der Pull Request führte lediglich folgende Schnittstelle zum Core ein:

```
1 // Auszug aus dem Mapbox Core, Datei include/mbgl/map/map.
      ↪ hpp
2  void addImage(const std::string&, std::unique_ptr<const
      ↪ SpriteImage>);
3  void removeImage(const std::string&);
```

Dank dieser Implementierung konnte nun die Funktionalität auf SDK-Level für macOS und iOS umgesetzt werden. Dazu wurden auf dem Header-File `/platform/darwin/src/MGLStyle.h` die folgenden Methoden definiert:

```
1  /**
2  Adds or overrides an image used by the 'styles layers.
3  To use an image in a style layer, give it a unique name
       ↪ using this method,
4  then set the `iconImage` property of an `
       ↪ MGLSymbolStyleLayer` object to that name.
5   @param image The image for the name.
6   @param name The name of the image to set to the style.
7   */
8  - (void)setImage:(MGLImage *)image forName:(NSString *)name
       ↪ ;
9
10 /**
11  Removes a name and its associated image from the style.
12  @param name The name of the image to remove.
13  */
14 - (void)removeImageForName:(NSString /*)name;
```

Durch die Unterstützung von iOS und macOS mit dem gleichen Framework, musste bei der Anpassung des Header-Files darauf geachtet werden, dass zwischen den beiden Ziel-Plattformen unterschieden wird. Denn macOS verwendet für Bilder den Typ `NSImage` und iOS den Typ `UIImage`. Die beiden Typen unterschieden sich nicht nur durch ihren Namen, sondern auch durch eine unterschiedliche API.

Hierzu wurde zuerst ein Typ in der Datei `/platform/darwin/src/MGLTypes.h` definiert:

```
#if TARGET_OS_IPHONE
@class UIImage;
#define MGLImage UIImage
#else
@class NSImage;
#define MGLImage NSImage
#endif
```

In der Implementierungs-Datei `/platform/darwin/src/MGLStyle.mm` wurde dann zuerst wieder zwischen iOS-SDK und macOS-SDK unterschieden:

```
#if TARGET_OS_IPHONE
    #import "UIImage+MGLAdditions.h"
#else
    #import "NSImage+MGLAdditions.h"
#endif
```

Die Implementierung der in der Header-Datei definierten Methoden nutzte dann die im PR #6375 hinzugefügten Methoden zur Bearbeitung des Sprite Atlas.

```
- (void)setImage:(MGLImage *)image forName:(NSString *)name
{
    NSAssert(image, @"image is null");
    NSAssert(name, @"name is null");

    self.mapView.mbglMap->addImage([name UTF8String], image
      ↪ .mgl_spriteImage);
}

- (void)removeImageForName:(NSString *)name
{
    NSAssert(name, @"name is null");

    self.mapView.mbglMap->removeImage([name UTF8String]);
}
```

Die oben verwendete Eigenschaft `image.mgl_spriteImage` musste zudem auch plattform-abhängig für macOS und iOS implementiert werden. Weitere

Details zur Implementierung finden sich im Pull Request #6637.

2.7.2 Resultat

Aufgrund der eigenen, für die Umsetzung von Traildevils iOS zwingend notwendigen Umsetzung dieser Funktion kann nun ein Bild (z.B. das "Berg-Icon" für Destinationen) **zur Laufzeit** zum Sprite Atlas hinzugefügt und in einem Style Layer verwendet werden.

```
1 mapView.style().setImage(UIImage(named: "DestinationIcon")
      ↪ !, forName: "destination")
2 ...
3 destinations.iconImage = MGLStyleValue.init(rawValue: "
      ↪ destination")
4 ...
```

Abbildung 2.3: Eigene Bilder oder Icons in Style Layers

Kapitel 3

Analyse

Die Bachelorarbeit aus dem FS16 hat gezeigt, dass die Cross Platform Library von Mapbox noch nicht für den produktiven Einsatz bereit ist. Zum Zeitpunkt dieser Arbeit wird die Library immer noch als *experimental* bezeichnet.

Gemäss der Aufgabenstellung soll nun eine native iOS Traildevils App entwickelt werden.

- Vorteil: Die Implementation genügt den nicht-funktionalen Anforderungen betreffend Performance
- Nachteil: Eine separate Android App müsste zusätzlich entwickelt werden

3.1 Anforderungsspezifikation

Zusätzlich zur bestehenden Traildevils Website soll eine native Umsetzung für iOS Geräte gemacht werden. Es soll unterwegs oder zuhause auf dem Sofa nach Trails, Tracks, Dealers und Destinations gesucht werden können. Zusätzlich wird mit POI-Markern von Trails, Dealers und Destinations auf der Karte interagiert, um die gewünschten Detailinformationen einsehen zu können. Kartenausschnitte sollen offline abgespeichert werden können, um

sie anschliessend bei schlechtem Empfang in der Bergwelt oder gar keinem mobilen Netz trotzdem verfügbar zu machen.

Bei den folgenden Anforderungen handelt es sich um Projektanforderungen. Die Anforderungen wurden zusammen mit dem Projektpartner erarbeitet und von ihm abgenommen. Die für die Basis-App relevanten Produktanforderungen wurden iterativ während dem Sprint-Planning spezifiziert. Dabei wurden die Wünsche vom Projektpartner berücksichtigt und dementsprechend erfasst. Einzelne Details zu den Produktanforderungen im Kapitel 8 zu entnehmen.

3.1.1 Benutzer Charakteristik

Das Produkt ist auf Personen ausgerichtet, die passioniert mit ihrem Mountain Bike unterwegs sind. Natürlich soll die App auch den Personen einen Einstieg bieten, welche sich ein neues Hobby im Bereich Mountainbiking suchen.

Für Design und Test der App orientieren wir uns an dem folgenden Personenprofil:

- **Name:** Johann Aschwanden
- **Alter:** 24 Jahre
- **Wohnort:** Rapperswil
- **Tätigkeit:** Informatiker
- **Zivilstand:** Ledig

3.1.1.1 Lebenssituation

Johann wohnt mit seinen zwei Mitbewohner in einer WG in der Altstadt von Rapperswil.

3.1.1.2 Geräte

Johann zählt die folgenden Geräte zu seinem Inventar:

- iPhone 6S
- iPad mini 3
- MacBook Pro Early 2013

3.1.1.3 Technische Fähigkeiten

Durch seinen Beruf gerät Johann täglich in Kontakt mit technischen Geräten. Er versteht die Funktionsweise und die verwendeten Technologien vieler Geräte.

3.1.1.4 Bezug zu Traildevils

Mountainbiking gilt für Johann als Ausgleich zum Alltag. Er teilt sein Hobby mit seinen Mitbewohnern und sie shredden praktisch jedes Wochenende zusammen. Auf der Traildevils Plattform tauscht er sich mit anderen Ridern aus und ist selber ein aktives Community-Mitglied.

3.1.1.5 Hobbies

Nebst dem Mountainbiking zählt Johann das Kochen zu seinen Freizeitaktivitäten. Nach einer erfolgreichen Mountain Bike-Tour mit seinen Freunden verwöhnt er ihnen und seiner Familie gerne mal den Gaumen.

3.1.1.6 Szenario

Am Freitag Abend nach dem ersten Bier, welches das Wochenende einläutet, sehnt sich Johann nach spannenden Trails mit seinen Freunden. Er hat von seinen Community-Kollegen über die neue Traildevils iOS App erfahren.

Nach dem gemeinsamen Abendessen mit seinen WG-Kollegen startet Johann den App Store auf seinem iPhone und lädt die Traildevils iOS App auf sein Gerät. Anschliessend loggt er sich mit seinem Account auf der Platform ein. Die Jungs besprechen die Destination gemeinsam und einigen sich auf einen Ausflug nach Lenzerheide im wunderschönen Bündnerland. Johann zoomt auf der interaktiven Karte, um die einzelnen Trails genauer unter die Lupe zu nehmen. Schlussendlich einigen sich die Jungs auf den "Lenzerheide Bikepark" und möchten dort den morgigen Tag verbringen. Johann tippt auf den Trail und gelangt zu den Detailinformationen. Dort sind Zustand, Likes und andere Metadaten des Trails ersichtlich. Erfreut nehmen die Kollegen noch ein Bier aus dem Kühlschrank und sehnen sich nach der morgigen Reise.

3.1.2 Einschränkungen

Im Moment wird ausschliesslich eine native iOS App ausgeliefert. Es ist jedoch nicht ausgeschlossen, dass in einem zweiten Schritt Applikationen für Android und Windows Geräte umgesetzt werden.

3.1.2.1 Devices

Als Benutzerschnittstelle dient das jeweilige Apple Device. Egal ob iPhone 6S Plus oder das neue iPhone 7. 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 Semesterarbeit wird garantiert, dass die Software auf den folgenden Geräten stabil läuft.

iPhone:

- iPhone 4S
- iPhone 5 *
- iPhone 5S *
- iPhone 6 *
- iPhone 6 Plus*

- iPhone 6S
- iPhone 6S Plus *
- iPhone 7
- iPhone 7 Plus *

iPad:

- iPad mini 2
- iPad mini 3 *
- iPad mini 4 *
- iPad Air
- iPad Air 2 *
- iPad Pro

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

3.1.2.2 iOS

Die Traildevils iOS App wird von allen Geräten unterstützt, welche mindestens iOS Version 9.3.5 installiert haben. Ältere Versionen wurden zwar getestet, werden aber nicht offiziell unterstützt und es besteht keine Garantie auf Fehlerfreiheit.

3.1.2.3 Traildevils API

Für den Datenaustausch wurde die Traildevils API neu spezifiziert. Jegliche Daten zu Trails, Tracks, Dealers, Destinations und Benutzern werden über diese Schnittstelle ausgetauscht. Während der Entwicklung befindet sich ein selbstentwickleter Node.js Mock-Server im Einsatz.

3.1.2.4 Mapbox API

Das Kartenmaterial wird ausschliesslich von Mapbox bezogen.

3.1.2.5 Offline Daten

Durch den Einsatz von Mapbox ist das Abspeichern für Offline Daten auf 6000 Kacheln (einzelne Ausschnitte der Weltkarte, auch Tiles genannt) pro User beschränkt. Dies entspricht etwa der Grösse einer Destination. Beim Download der Daten gibt ein Indikator Übersicht über die bereits geladenen Daten.

3.1.2.6 Geocoding

Für den Use Case "Geodaten suchen" wird das Geocoding von Mapbox [6] verwendet. Das Geocoding ist auf 600 Requests pro Minute [7] beschränkt und somit im Free-Modell genutzt. Wird dieses Limit überschritten, muss ein Enterprise-Modell berücksichtigt werden.

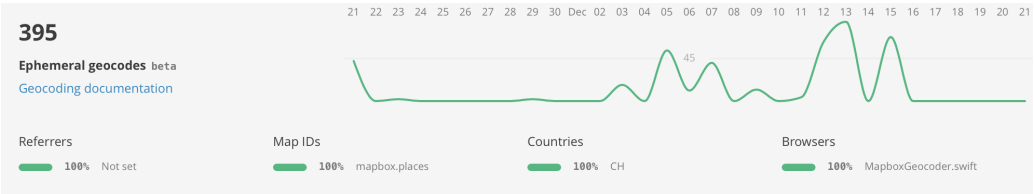


Abbildung 3.1: Visualisierung der Geocoding Requests im Dashboard von Mapbox

Es ist schwer zu sagen, wieviele parallele Benutzer den Geocoding Service im Free Modell nutzen können. Angenommen ein Benutzer sucht mindestens jede Minute nach einem Begriff und die Durchschnittslänge des Suchbegriffes beträgt fünf Zeichen, so werden fünf Requests ausgelöst und es können 120 Benutzer pro Minute den Service gleichzeitig nutzen.

Es ist aber zu berücksichtigen, dass Suchresultate auf dem Gerät zwischengespeichert werden und somit die erneute Suche nach demselben Begriff keinen Request auslöst. Zusätzlich kann diese Anzahl paralleler Benutzer durch eine Verzögerung des Requests minimiert (zum Beispiel wird der Suchbegriff erst nach 3 Sekunden abgsendet, nachdem der Benutzer aufgehört hat zu schreiben).

3.1.3 Use Cases

Die Use Cases werden in den einzelnen Kapiteln im “Brief” Format beschreiben.

3.1.3.1 Kartenelemente selektieren

Der Benutzer kann mit einem Touch ein interaktives Kartenelement selektieren. Anschliessend werden dem Benutzer in einem Callout ein paar wenige Informationen zum selektierten Kartenelement angezeigt. Ein weiterer Touch auf das Callout öffnet die Detailansicht des Kartenelementes.

Der Inhalt des Callouts gilt es noch zu definieren.

3.1.3.2 Kartenelemente suchen

Der Benutzer kann nach den grundlegenden Kartenelementen suchen. Dem Benutzer werden die bereits getätigten Suchanfragen angezeigt.

3.1.3.3 Geodaten suchen

Nebst den grundlegenden Kartenelementen sind auch Ortschaften, Gemeinden, Regionen, Kantone oder ganze Länder suchbar. Dabei wird auf das Geocoding von Mapbox zugegriffen. Sucht der Benutzer nach einem Ort, erscheint dieser zusätzlich unter den Suchresultaten. Bei einer Selektion eines solchen Elementes wird die Bike Map bei diesem Element zentriert dargestellt und die umliegenden Kartenelemente sind ersichtlich. Da die Suche nach einem Ort, Bereich oder Region der Karte getätigt wurde, ist das Zoomlevel bei 10 von maximal 22 Zoomlevels definiert. So sind noch umliegende Cluster-Marker ersichtlich, damit der Benutzer die Wahl hat, ob er gewisse Cluster-Marker mit einem Zoom-In auflösen möchte, um an die Details von Trails, Tracks oder Dealers zu gelangen, oder er verringert das Zoomlevel, um sich einen Überblick der Location zu verschaffen.

3.1.3.4 Detailinformationen zu einem Kartenelement anzeigen

Der Benutzer kann sich detaillierte Informationen von Trails, Dealers oder Destinations anzeigen lassen.

3.1.3.5 Trail Check-in

Der Benutzer kann ein Check-in bei einem Trail tätigen. Dabei ist dies auf der Detailansicht eines Trails und über eine Schnell-Aktion innerhalb der Navigation (Tab Bar) möglich.

Dem Benutzer sollen umliegende Trails anhand seiner aktuellen Position vorgeschlagen werden.

Nach erfolgreichem Check-in wird der Benutzer auch gleich nach dem Zustand gefragt. Falls er sich dafür entscheidet, wird der Use Case “Zustand melden” gestartet.

3.1.3.6 Zustand melden

Der Benutzer kann entweder über die Detailansicht eines Trails oder über die generelle Aktion in der Navigation dessen Zustand melden.

Wird die Aktion über die Navigation ausgelöst, sieht der Benutzer vorgeschlagene Trails anhand seiner aktuellen Position oder wählt aus seinen zuletzt verwendeten Trails aus. Zudem hat der Benutzer auch die Möglichkeit, nach einem spezifischen Trail zu suchen.

3.1.3.7 Kartenausschnitt offline speichern

Der Benutzer hat die Möglichkeit einen Kartenausschnitt offline zu speichern. Über eine eigene Ansicht wird dem Benutzer einen Rahmen innerhalb des Viewport angezeigt, welcher die Selektion eines Kartebereiches veran-

schaulicht. Der Benutzer kann sich so für einen beliebigen Kartenausschnitt entscheiden. Dies soll wie die Lösung von Google Maps ablaufen.

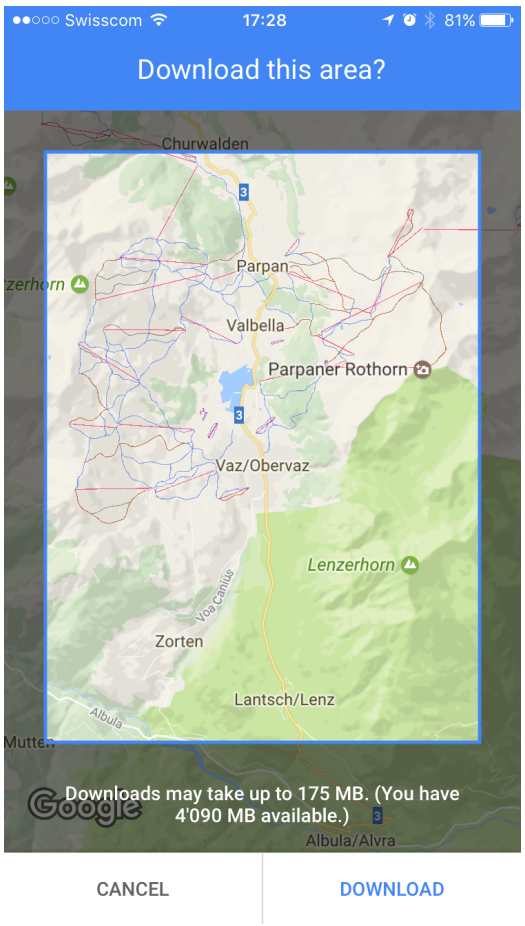


Abbildung 3.2: Selektion der Offline Area in Google Maps

Der Benutzer hat die Möglichkeit nur eine Region mit einem Namen zu speichern. Falls bereits eine offline Region existiert, hat der Benutzer die Möglichkeit diese zu löschen. Dies ist auch zwingend für das Speichern eines neuen Kartenausschnitts falls die maximale Anzahl an Tiles überschreitet ist.

Für diese Aktion ist ein Login notwendig.

3.1.3.8 Kartenelement auf WhatsApp teilen

Auf der Detailansicht eines Kartenelementes gibt es einen zusätzlichen Share-Button. Bei der Betätigung dieses Buttons wird WhatsApp geöffnet und es wird eine Nachricht verfasst mit dem Titel und der Web-URL zum Kartenelement. Anschliessend kann der Benutzer auf der Website die Detailansicht in der Traildevils Applikation öffnen.

WhatsApp ist mittlerweile in der Lage, Meta-Informationen einer Website anzuzeigen. So können auf der Website diese Meta-Tags erfasst werden, um z.B. ein Headerbild eines Trails anzuzeigen und dessen Titel.

Der Schritt über die Website wird bewusst gemacht, um den Android-Benutzern trotzdem eine Möglichkeit zu bieten, einen solchen Link aufzurufen.

Hier muss noch evaluiert werden, ob für Benutzer mit bereits installierter Traildevils App der Schritt über die Website wirklich notwendig ist. Instagram weist z.B. das gewünschte Verhalten vor.

3.1.3.9 Kartenelement bewerten und kommentieren

Der Benutzer kann ein Kartenelement auf der Detailansicht bewerten. Die Bewertung liegt zwischen einem und maximal fünf Sternen. Zudem kann der Benutzer einen zusätzlichen optionalen Kommentar abgeben.

3.1.3.10 Detailinformation zu einem Rider

Der Benutzer kann zusätzlich zu den bestehenden Kartenelementen nach einem Rider suchen. Dabei wird ihm auf einer Detailview die Informationen angezeigt.

3.1.3.11 Aktivitäten Stream anzeigen

Der Benutzer sieht Aktivitäten einzelner Kartenelemente in einem Aktivitäten Stream. Dieser Use Case hängt mit dem “Kartenelement folgen” zusammen. Im Stream werden Zustandsmeldungen zu einem Trail oder kürzliche Check-ins von “befreundeten” Ridern angeziegt.

Die Aktivitäten gilt es noch genau zu definieren.

3.1.3.12 Kartenelementen folgen

Der Benutzer ist in der Lage einzelnen Kartenelementen zu folgen, um Aktivitäten dieser Kartenelemente im Aktivitäten stream zu empfangen.

3.1.3.13 Benutzer anmelden

Der Benutzer ist in der Lage sich mit seinem Traildevils Login einzuloggen.

3.1.3.14 Benutzer registrieren

Der Benutzer kann sich mit seiner E-Mail-Adresse und Passwort für ein Traildevils Login registrieren. Dieses Aktivierung des Kontos ist aber nicht erforderlich für die Interaktion mit der Applikation. Trotzdem wird dem Benutzer eine E-Mail mit einem Aktivierungslink verschickt.

Nach der Registration ist der User direkt eingeloggt.

3.1.3.15 Bild hochladen

Der Benutzer kann auf der Detailansicht eines Trails ein Bild hochladen.

3.1.4 Nicht funktionale Anforderungen

3.1.4.1 Performance

Die App muss innerhalb von drei Sekunden gestartet sein und reagiert auf Benutzereingaben. Zudem soll die Karte während der Bedienung nicht “stocken” und flüssig reagieren.

Die Performance wird von Testpersonen mit einem Release über TestFlight Test Flight überprüft.

3.1.4.2 Security

Sensitive Benutzerdaten sollen stets verschlüsselt sein. Lokal, wie auch bei der Übermittlung und der Lagerung der Daten auf dem Server.

Die Security wird mit einem internen Code Review überprüft.

3.1.4.3 Wartbarkeit

Die Applikation wird mit einem dazugehörigen Manual ausgeliefert. Dieses Manual beinhaltet die nötigen Installationsschritte und Informationen zur Weiterentwicklung. Dadurch, dass die Applikation in Swift programmiert ist, ist Xcode und ein macOS-Gerät 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 Styleguides, Linter und Patterns wird die Wartbarkeit zusätzlich gestärkt und sichergestellt.

3.1.4.4 Verfügbarkeit

Die ausgelieferte Applikation liefert einen vollen Funktionsumfang bei konstanter Verbindung zum Internet. Durch die offline Synchronisation gelten die vollumfänglichen Funktionen auch für einen ausgewählten und lokal gespeicherten Kartenausschnitt.

Die unterstützten Geräte und Betriebssysteme werden in den Kapiteln 3.1.2.1 und 3.1.2.2 aufgelistet.

3.1.4.5 Usability

Durch die Apple Human Interface Guidelines [8] erkennt der Benutzer Muster der vertrauten iOS-Umgebung wieder. So wird ein problemloser Umgang mit der App gewährleistet.

Kapitel 4

Lösungskonzept

4.1 Systemübersicht

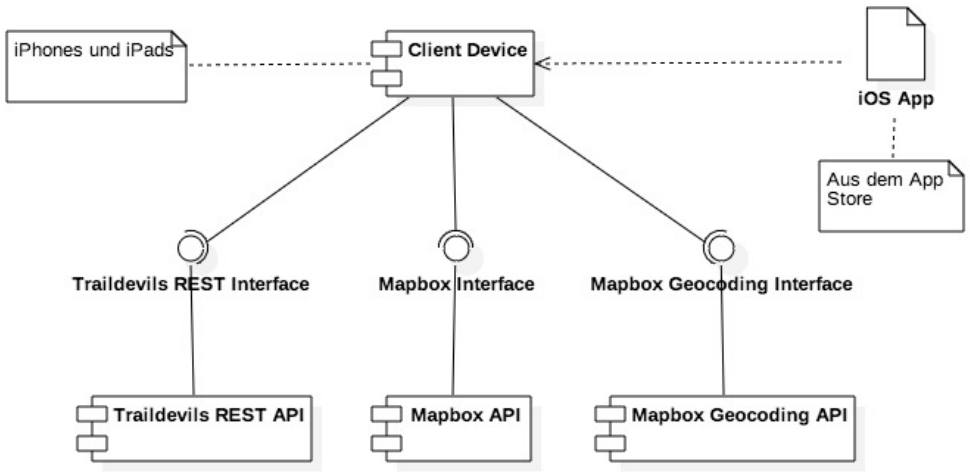


Abbildung 4.1: Deployment Diargramm

Aus dem Diagramm 4.1 sind die zentralen Komponenten der Architektur 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

4.2 Design Pattern

Bei der Wahl eines Software Design Patterns für die Implementierung kommen prinzipiell die drei bekannten MV(X) Architekturen in Frage:

- Model-View-Controller (MVC)
- Model-View-Presentation (MVP)
- Model-View-ViewModel (MVVM)

Diese drei Design Patterns lösen das gleiche Problem auf verschiedene Arten, sind im Kern aber immer das Selbe:

- **M**odels - verantwortlich für den Data Access Layer und Persistierung von Daten
- **V**iews - verantwortlich für den Presentation Layer (GUI), in der iOS Umgebung sind das alle Komponenten mit dem Präfix "UI"
- **C**ontroller/**P**resenter/**V**iew**M**odel - repräsentiert das mittlere Glied zwischen Models und Views, verantwortlich für das Aktualisieren von Models augrund von Benutzereingaben in der View oder umgekehrt

Apple selbst gibt keine Restriktion bei der Wahl vor, in der Dokumentation wird aber das MVC Design Pattern vorgeschlagen [9].

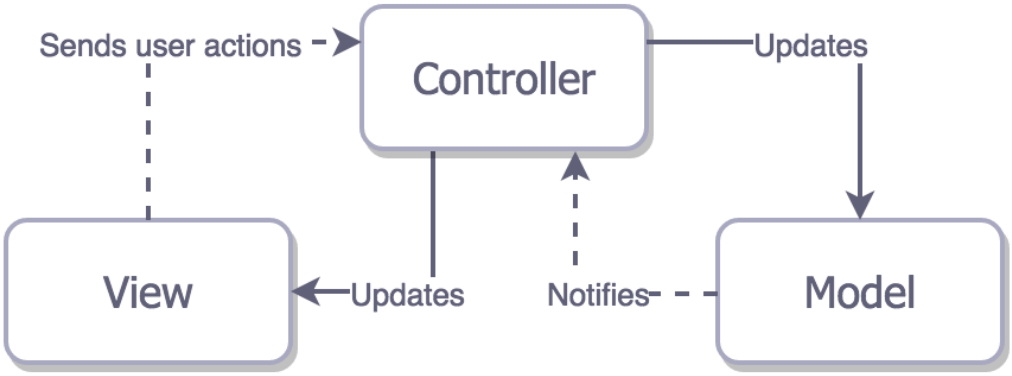


Abbildung 4.2: Klare Aufgabenteilung in Apple's MVC Pattern

Damit sich View und Model nicht kennen, dient der Controller als **Mediator** zwichen den beiden Komponenten. Die am wenigsten Wiederverwendbare Komponente ist offensichtlich der Controller, was in der Regel auch in Ordnung ist.

Dieses Pattern wird aber meistens schlecht umgesetzt und führt häufig zu Problemen.

4.2.1 "Massive View Controller"-Problem

Abbildung 4.3: Tweet über das MVC-Problem in iOS

In der Realität sieht das von Apple vorgeschlagene Design Pattern meist folgendermassen aus:

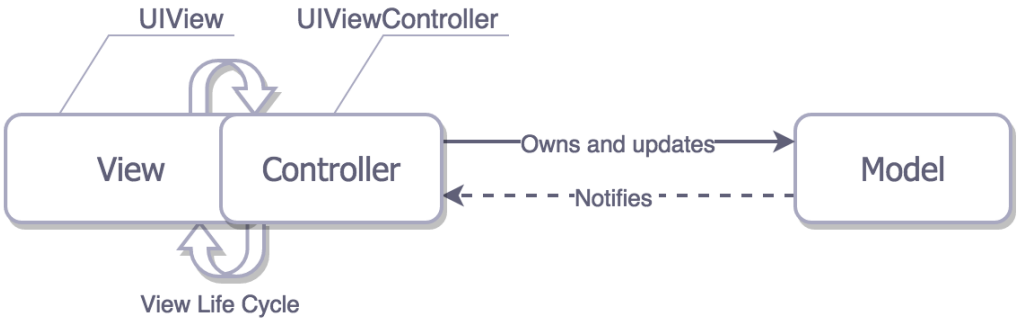


Abbildung 4.4: View und Controller bilden eine Einheit, das Model dient lediglich als Payload

Warum ist das so?

Der Grund dafür liegt nicht nur bei den Entwicklern, sondern auch bei Apple selbst. Entwickler tendieren dazu, den schnellsten Weg für eine Lösung zu implementieren und nehmen dafür diverse Nachteile in Kauf. Allen voran die Wiederverwendbarkeit und Testbarkeit von Klassen. Viele Tutorials im Internet aber auch Beispiel-Code in GitHub Repositories zeigen oft, wie Problem X gelöst werden kann und lösen dieses Problem mit möglichst geringem Aufwand, was bedeutet, dass Design Patterns missachtet werden. Dazu kommt auch noch, dass Apple die Entwickler dazu ermutigt, die meiste Arbeit im Controller zu erledigen. Der Controller ist nicht nur Delegate für verschiedene UI Komponenten, sondern ist gleichzeitig auch noch Datenquelle und muss auf Benutzeringaben reagieren.

> "[…] View controllers also coordinate their efforts with other controller objects—including other view controllers—and help manage your app's overall interface. […] A view controller is tightly bound to the views it manages and takes part in the responder chain used to handle events. […] - Source [10]"

```
1 var userCell = tableView.dequeueReusableCellWithIdentifier(
      ↪ "identifier") as UserCell
2 userCell.configureWithUser(user)
```

Oben gezeigtes Code Snippet lässt sich oft in verschiedenen GitHub Repositories, aber auch in Code Beispielen von Apple, wiederfinden. Die `userCell`

repräsentiert eine View (eine Zelle innerhalb einer Liste `UITableView`) und wird in der zweiten Zeile mit einem `user`-Model konfiguriert. Das MVC-Pattern wurde also eindeutig verletzt.

4.2.2 VIPER to the Rescue

Abbildung 4.5: Ein VIPER Modul ist in fünf verschiedene Komponenten aufgeteilt

VIPER ist ein Backronym für **V**iew, **I**nteractor, **P**resenter, **E**ntity und **R**outer. VIPER verfolgt den Ansatz, mithilfe des Single Responsibility Principle eine klare und modulare Struktur des iOS Projekts zu erreichen. Die Idee hinter dem Pattern ist die Isolation von Abhängigkeiten, um die Verantwortlichkeiten zwischen den Komponenten besser zu balancieren.

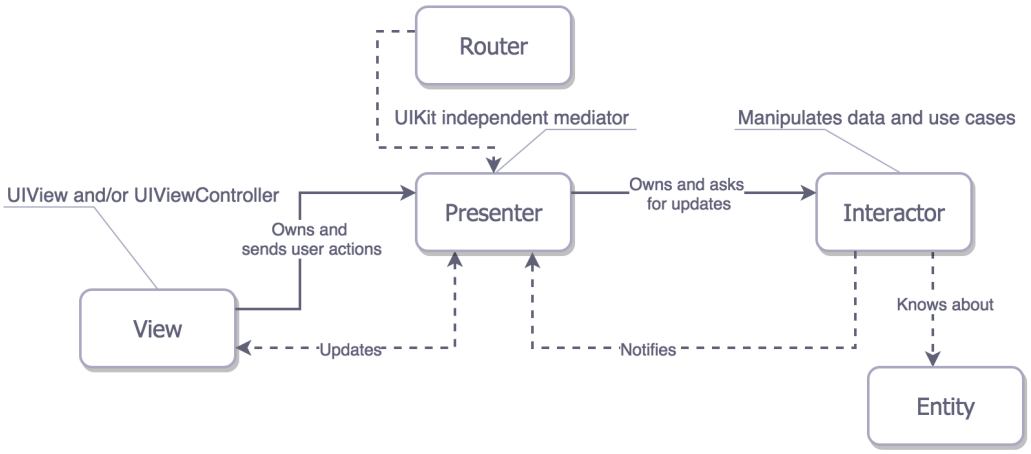


Abbildung 4.6: Jede Komponente eines Moduls ist nur für eine kleine Aufgabe verantwortlich

Das Diagramm 4.6 zeigt anschaulich die Aufteilung der Verantwortlichkeiten innerhalb eines Moduls in mehrere Komponenten. Jede Komponente stellt dazu Inputs und Outputs zur Kommunikation untereinander zur Verfügung. Grundsätzlich repräsentiert **ein Modul einen Screen oder eine User Story**. Wird von einem Screen zu einem anderen Screen navigiert, so wird über den Router von Modul X zum Router von Modul Y gewechselt. Entities werden dabei nicht zwischen Modulen geteilt, sondern sollten nur innerhalb des jeweiligen Moduls verwendet werden.

Die verschiedenen Komponenten in VIPER werden “gewired”, also zusammengeschweisst. Das beduetet, dass jede Komponente jeweils Referenzen auf die anderen Komponenten zugewiesen bekommt. Eine View besitzt zum Beispiel eine Referenz auf den Interactor. Ein Presenter besitzt eine Referenz auf die View und auf den Interactor.

4.2.2.1 View

Beschreibung:

Die View, meist ein `UIViewController`, zeigt stupide nur das an, was die Presenter-Komponente ihr mitteilt.

Aufgaben:

1. Dem Benutzer Informationen anzeigen
2. Benuterinteraktion detektieren

Beispiel:

Zu 1.: Der Presenter teilt der View mit, eine Fehlermeldung im UI anzuzeigen. Die View nimmt dies entgegen und zeigt eine `UIAlertView` mit der entsprechenden Meldung an.

```
1 // Aufruf im Presenter
2 view.showError(message: "Something went wrong")
```

Zu 2.: Der Benutzer hat gerade Benutzername und Passwort ind zwei Textboxen angegeben und drückt auf den Login-Button. Dieses Ereignis wird inklusive Benutzername und Passwort an den Presenter weitergeleitet.

```
1 // Aufruf in der View
2 presenter.didRequestLoginWith(username: "roman", password:
      ↪ "mypwd")
```

4.2.2.2 Presenter

Beschreibung:

Der Presenter teilt der View mit, was sie anzuzeigen hat und reagiert auf eingehende Ereignisse.

Aufgaben:

1. Teilt der View mit, was anzuzeigen ist
2. Verarbeitet Ereignisse und reagiert darauf entsprechend

Beispiel:

Zu 1.: Der Presenter hat eine Fehlermeldung vom Interactor in Form eines Objektes erhalten. Der Presenter leitet dieses Objekt nicht einfach an die View weiter, sondern nimmt sich die relevanten Informationen aus dem Objekt und formatiert dies zu einer ausdrucksvollen Nachricht.

Zu 2. (a): Auf das Ereignis, dass der Benutzer sich einloggen möchte, werden Benutzername und Passwort an den Interactor weitergegeben, welcher dann diese Anfrage weiterleitet.

```
1 // Aufruf im Presenter
2 interactor.loginWith(username: "roman", password: "mypwd")
```

Zu 2. (b): Der Presenter kann auch Ereignisse vom Interactor erhalten, zum Beispiel dann, wenn die Anmeldung mit den angegebenen Benutzerinformationen fehlgeschlagen ist.

```
1 // Aufruf im Interactor
2 presenter.loginFailedWith(error: "Something went wrong")
```

4.2.2.3 Interactor

Beschreibung:

Der Interactor weiss, wie auf Ereignisse vom Presenter reagiert und welche Business-Logik ausgeführt werden muss.

Aufgaben:

1. Business-Logik ausführen

Beispiel:

Zu 1.: Die Login Anfrage wird vom Interactor verarbeitet, eventuell auch validiert und dann ein Login durchführt. Dazu muss ein API Request gemacht werden. Wichtig hierbei ist, dass der Interactor selbst den API Request nicht durchführt, sondern bloss entscheidet, wie der Task ausgeführt werden muss, damit er erfolgreich ist.

4.2.2.4 (Data Manager)

Aufgaben:

1. Daten empfangen
2. Daten persistieren

Beschreibung:

Der Data Manager ist eigentlich keine offizielle Komponente von VIPER, wird aber trotzdem häufig in Kombination mit dem Interactor eingesetzt. Der Data Manager weiss, woher er benötigte Daten bekommt (z.B. aus lokalen Datenbank oder via API Request) und entscheidet auch, ob Daten persistiert werden müssen oder nicht (z.B. ein Login Token).

Beispiel:

Zu 1.: Der Interactor sucht zum Beispiel nach einem gewissen Datensatz mit ID “1337”. Der Data Manager prüft nun in der lokalen Datenbank nach, ob dieser Datensatz existiert und falls nicht, wird ein API Request erstellt, um den Datensatz server-seitig zu suchen und zu empfangen.

```
1 // Aufruf im Interactor
2 dataManager.fetchDataWith(id: 1337)
```

Zu 2.: Nach dem Empfang des Datensatzes mit der ID 1337 von der API entscheidet der Data Manager, dass dieser für zukünftige Anfragen in der lokalen Datenbank abgespeichert werden. Anschliessend wird der Datensatz dem Interactor übergeben.

```
1 // Aufruf im DataManager
2 interactor.found(data: someDataEntity)
```

4.2.2.5 Entity

Aufgaben:

- Daten repräsentieren (enthält Informationen)

Beschreibung:

Eine Entity dient als Payload und repräsentiert lediglich Daten, keine Funktionen. Ein Entity "verlässt" nie ein Modul, sondern wird nur innerhalb des jeweiligen Moduls zwischen den einzelnen Komponenten hin und her gereicht.

4.2.2.6 Router

Aufgaben:

1. Initialisiert alle Komponenten
2. Navigiert zwischen einzelnen Modulen/Screens

Beschreibung:

Der Router (oft auch Wireframe genannt) erstellt und setzt alle anderen VIPER Komponenten zusammen ("wiring") und ist verantwortlich für das Navigieren zwischen Modulen/Screens.

Beispiel:

Zu 1.: In VIPER besteht jedes Modul aus einer View, einem Presenter, einem Interactor, einem Data Manager und einem oder mehreren Entities. Zum Beispiel lässt sich der Use Case "Perform user login" mit einem Modul folgendermassen lösen:

- Eine **View** mit einem Benutzernamen und Passwort Feld sowie einem Button "Log in"
- Ein **Presenter** der die Login-Anfrage entgegennimmt, gleichzeitig der **View** mitteilt, einen Activity Indicator anzuzeigen
- Der **Interactor** nimmt die Login-Anfrage wiederum entgegen und weiss, welche Methode auf dem Data Manager auszuführen ist
- Der **Data Manager** führt einen API Request aus, um sich einzuloggen
- Bei erfolgreichem Einloggen teilt der **Data Manager** dem Interactor dies mit
- Der **Interactor** wiederum leitet dieses Ereignis an den **Presenter** weiter

- Der **Presenter** weiss nun, dass das Login erfolgreich war, teilt der **View** mit, den Activity Indicator auszublenden und eine Meldung anzuzeigen, dass das Login erfolgreich war. Gleichzeit teilt der **Presenter** dem **Router** mit, zu einem anderen Modul zu navigieren
- Der **Router** erstellt das neue Modul und übergibt die Verantwortung nun an den neuen Router des anderen Moduls

4.2.3 Vorteile von VIPER

Durch die Unterteilung von Aufgaben an verschiedene Komponenten ergeben sich viele verschiedene Vorteile.

- **Klare Struktur**: Module sind klar strukturiert und immer nach demselben Prinzip aufgebaut, dadurch lässt sich schnell feststellen, wo die Ursache für ein Problem sein könnte
- **Granulares Testen**: Einzelnen Komponenten lassen sich besser testen, das Erstellen von Mocks gestaltet sich einfacher
- **Collaboration-friendly**: Mehrere Entwickler können an verschiedenen Use Cases in unterschiedlichen Modulen arbeiten, ohne dass es Konflikte gibt
- **Single Responsibility Principle**: Jede Komponente erfüllt nur eine kleine Aufgabe und macht den Code übersichtlicher und lesbarer

4.2.4 Entscheid

Die Architektur VIPER war vor der Umsetzung unbekannt. Das Konzept hörte sich sehr interessant an, die Vorteile liegen klar auf der Hand. Hauptgrund für die Umsetzung mit VIPER war aber die Überzeugung, dass auch die Studienarbeit dazu da sein sollte, etwas zu lernen. Deshalb wurde entschieden, dieser Architektur eine Chance zu geben und diese in der Applikation nach bestem Wissen einzusetzen.

4.3 Traildevils API

Um die Trails, Tracks, Dealers und Destination auf der Karte anzuzeigen, braucht es eine Schnittstelle, welche die Daten für die Endgeräte zur Verfügung stellt. Mapbox benötigt, für das Anzeigen der Daten, zwingend das `.geojson` Format. Somit ist klar, wie die Collections der Kartendaten vom Webserver ausgeliefert werden müssen. Für die Detailviews wurde `.json` verwendet.

Die Umsetzung der API ist nicht Bestandteil dieser Arbeit. Jedoch wurde ein Konzept entwickelt, welches dem Projektpartner die notwendige Struktur zeigt.

Die Spezifikation dieser API wurde fortlaufend von den Teammitgliedern ergänzt und vom Auftraggeber kontrolliert und abgenommen.

Die detaillierte Spezifikation ist im Anhang zu finden.

4.3.1 Apiary

Apiary hat uns während dem Designen der API sehr unterstützt. Zum Zeitpunkt der Arbeit war es möglich auf zwei Arten APIs zu erstellen. Auf der einen Seite war die API Blueprint Syntax, welcher auf der vereinfachten Auszeichnungssprache Markdown basiert, und auf der anderen Seite war es die Swagger Syntax. Aus dem Grund, dass sich der Support für den Swagger Syntax noch in der Betaphase befand und die Autoren den einfachen Syntax von Markdown lieben, wurde die API mit dem API Blueprint Syntax realisiert.

Ein Hello World Beispiel mit dem API Blueprint Syntax sieht wie folgt aus:

```
# GET /message
+ Response 200 (text/plain)

        Hello World!
```

Die Plattform bietet einen Editor, in welchem die API spezifiziert wird. Binnen Minuten lässt sich ein API Mock erstellen. Die Spezifikation lässt sich im Editor validieren und testen. Alternativ kann man auch auf die Command Line Tools zurückgreifen, wenn lokales Arbeiten bevorzugt wird.

Kapitel 5

Umsetzung

Durch die vorgängigen Vorbereitungen, speziell die Experimente, konnte das Fundament für die Applikation schnell gelegt werden.

5.1 Vorstudie

In Zusammenarbeit mit dem Auftraggeber wurde für jedes Problem aus der vorangegangenen Bachelorarbeit ein Experiment ausgearbeitet, welches die Machbarkeit der Traildevils App unter Mapbox iOS SDK aufzeigen soll. Zum Erstellen und Testen der einzelnen Experimente wurde ein öffentliches GitHub Repository ([11]) erstellt, was den Vorteil hatte, dass während der Umsetzung der verschiedenen Experimente Bugs der Mapbox-Community berichtet und auf das öffentliche Repository verwiesen werden konnte.

Für die Experimente wurden folgende GeoJSON-Beispieldaten verwendet:

- `mcondalds.geojson` (Quelle: [12]): McDonald's Restaurants der USA (Punkt-Koordinaten)
- `amsterdam.geojson` (Quelle: [13]): Bereiche der Stadt Amsterdam (Polygon-Koordinaten)
- `portland.geojson` (Quelle: [14]): Eine Strecke durch die Stadt Portland (Linien-Koordinaten)

- `siliconvalley.geojson` (Quelle: [15]): Point-of-Interests wie z.B. Restaurants, Bars, Clubs, usw. im Silicon Valley (kategorisierte Punkt-Koordinaten)

In den folgenden Abschnitten werden die Experimente und die daraus entstandenen Erfahrungen näher erläutert werden. Die Experimente sollen helfen, Risiken aufgrund von aufgetretenen Problemen oder fehlenden Features frühzeitig zu erkennen.

5.1.1 Clustering

Das Clustering von Punkten ist für die Umsetzung der Traildevils App ein zentrales Element, denn die Anzeige von hunderten von Trails kann dazu führen, dass die Schweiz von den POI-Markern überdeckt wird.

Leaflet, eine JavaScript Bibliothek für interaktive Karten, unterstützt bereits seit 2013 das Marker Clustering. Hingegen untzerstützt Mapbox GL JS dieses Feature erst seit Juli 2016 ([16]).

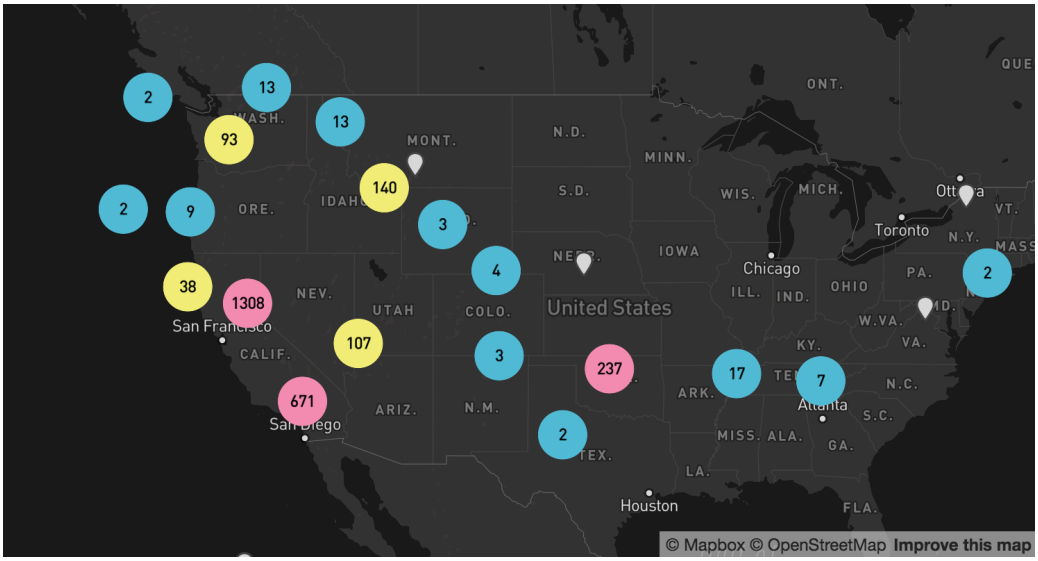


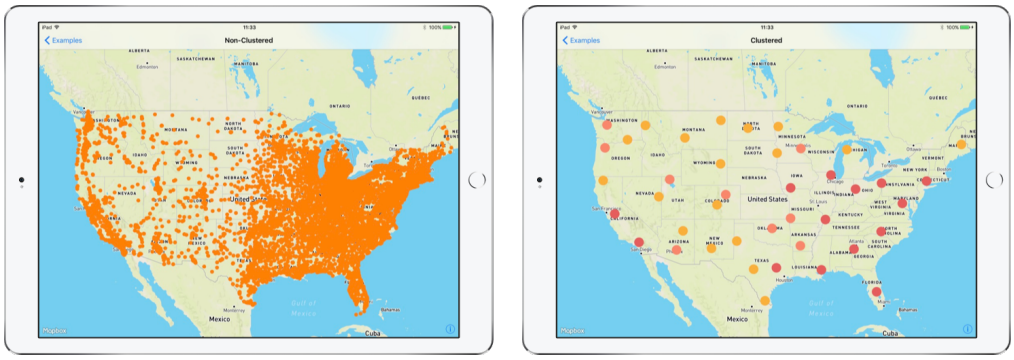
Abbildung 5.1: Clustering mit Mapbox GL JS

Zum Start dieser Semesterarbeit (19. September 2016) unterstützt Mapbox iOS SDK in der offiziellen Version 3.3.4 noch kein Clustering. Allerdings

wird seit Anfang August 2016 an der Version 3.4.0 gearbeitet, welche seit v3.4.0-alpha.1 die Unterstützung für Clustering beinhaltet ([17]).

Dem Konstruktor von MGLGeoJSONSource können Parameter zur Angabe von Clustering-Optionen mitgegeben werden. Die Optionen werden nachfolgend kurz erläutert:

- MGLGeoJSONClusterOption Diese Option aktiviert/deaktiviert das Clustering. Ist die Option auf `YES` gesetzt, so werden Punkte innerhalb des definierten Radius geclustert.
- MGLGeoJSONClusterRadiusOption Diese Option setzt den Radius, welche mehrere Punkte zu einem Cluster zusammenfasst.
- MGLGeoJSONClusterMaximumZoomLevelOption Diese Option legt fest, bis zu welchem Zoomlevel die Punkte geclustert werden sollen.



a: Ohne Clustering b: Mit Clustering

Abbildung 5.2: Variationen der Callouts

5.1.2 Semantic Zooming

Das Feature "Semantic Zooming" lässt sich elegant durch die Nutzung von Style Layers lösen. Wie wichtig es ist, diesen Ansatz zu wählen, wurde aber erst im zweiten Anlauf erkannt.

Ansatz 1 Der erste Ansatz war straight-forward: Es wurde eine Klasse erstellt, die alle nötigen Informationen über einen Style Layer beinhaltete.

```
1 class SemanticZoomingLayer {
```

```
    let title: String
    let color: UIColor
    let predicate: NSPredicate
    let minimumZoomLevel: Float
    var styleLayer: MGLStyleLayer?
}
```

Für jeden gewünschten Layer wurde dann ein Objekt instanziert, zum Beispiel:

```
SemanticZoomingLayer(
    title: "Cafes",
    color: UIColor.brown,
    predicate: NSPredicate(format: "amenity == 'cafe'"),
    minimumZoomLevel: 9.0
)
```

Die `styleLayer`-Eigenschaft der Klasse wurde dann nachträglich beim Erstellen der Karte gesetzt.

Bei jedem Aufruf der Methode `public func mapViewDidFinishRenderingMap(_ mapView: MGLMapView, fullyRendered: Bool)` des `MGLMapViewDelegate` wurde dann geprüft, ob sich der Zoom Level zu vorher geändert hat. Falls dies zutraf, wurden alle vorhandenen Style Layers neu validiert. Falls der Zoom Level niedriger als die Variable `minimumZoomLevel` war, wurde der Layer mit `mapView.style().remove(layer)` von der Map entfernt. Umgekehrt war es, wenn der Zoom Level höher als die Variable `minimumZoomLevel` war, dann wurde der Layer mit `mapView.style().add(layer)` zur Karte hinzugefügt.

Dieser Ansatz hört sich sehr kompliziert an und funktionierte auch nicht wirklich reibungslos, denn es traten immer wieder Fehler auf. Aufgrund dieser Erkenntnisse wurde der Issue #6478 bei Mapbox erfasst. Es wurde vermutet, dass dieses Problem auf die asynchrone Behandlung von Mapbox beim Hinzufügen und Entfernen von Layers zurückzuschliessen ist.

Ansatz 2 Aufgrund der gewonnenen Erfahrung nach einigen Tagen mit dem Framework konnte dass Experiment dann einfacher umgesetzt

werden. Denn Mapbox bietet bereits auf jedem Layer die beiden Eigenschaften `minimumZoomLevel` und `maximumZoomLevel`, welche den Layer **ab** `minimumZoomLevel` einblendet und **nach** `maximumZoomLevel` wieder ausblendet.

5.1.3 Panning

Damit das Panning-Experiment als erfolgreich markiert werden kann, muss Mapbox auch bei einer sehr grossen Menge an Daten noch eine gute Reaktionsfähigkeit auf Benutzereingaben haben. Nun stellt sich die Frage, wie eine solche Reaktionsfähigkeit überhaupt gemessen werden kann, denn: wie schnell ist eigentlich schnell genug? Handelt es sich hierbei rein um eine subjektive Wahrnehmung oder kann die Panning-Performance auch analysiert und ausgewertet werden?

5.1.3.1 "How Fast is Fast Enough"?

In der ACM Publikation "How Fast is Fast Enough? A Study of the Effects of Latency in Direct-Touch Pointing-Tasks" [18] wurde untersucht, welchen Einfluss die Latenzzeit auf die Ausführung einer bestimmten Aufgabe bei direkter Berührungeingabe hat. Die Publikation wurde für diese Arbeit zusammengefasst und übersetzt. Eine Lizenz für die Verwendung des Textes und der Bilder der Publikation wurde erworben und befindet sich im Anhang.

Zusammenfassung

Berührungsempfindliche Eingabeschnittstellen sind heutzutage allgegenwärtig, häufig in Form von Mobiltelefonen, Tablets und E-Readers. Obwohl die Technologie immer präziser wird, sind hohe Latenzzeiten in Endgeräten auch heute noch oft vorhanden. Es konnte festgestellt werden, dass eine höhere Latenzzeit, also die Verzögerung zwischen der Berührung/Bewegung eines Fingers auf einem Touchscreen und der darauffolgenden Reaktion, nachweislich von den Benutzern wahrgenommen wird.

Wichtig: Im Zusammenhang mit der Publikation bedeutet Performance die **Schnelligkeit und Genauigkeit** bei der Ausführung einer Aufgabe mit einem Touchscreen.

Konkret beschäftigte sich die Arbeit mit folgenden Fragestellungen:

1. Beeinflusst die Latenzzeit die Performance bei direkter Eingabe (z.B. Finger) auf gleiche Weise wie es bei indirekter Eingabe (z.B. Computermaus) der Fall ist?
2. Gibt es einen Grenzwert, bei dem eine schnellere Antwortzeit keinen weiteren Einfluss auf die Performance hat?
3. Auf welche Phasen einer direkten Benutzereingabe hat die Latenzzeit Einfluss und wie reagiert der Benutzer auf diese Latenzzeit?

Auf Basis dieser drei Fragen wurden zwei Experimente definiert und durchgeführt. Diese beiden Experimente werden nachfolgend näher erläutert.

Experiment 1: Pointing Performance

Im ersten Experiment, mussten die Partizipanten ein Rechteck aus einer Startposition in eine vorgegebenen Zielfläche bewegen (sogenannte Dragging Tasks basierend auf ISO 9241-9). Die Dragging Tasks unterscheideten sich in drei unabhängigen Variablen: *Latenz* der Bewegung des Rechtecks (1, 10, 25 und 50ms, künstlich zwischen den Frames eingefügt), *Breite* der Zielfläche (3, 4, und 5cm) und die *Distanz* zwischen der Startposition und der Zielfläche (3.5, 8.5 und 15cm).

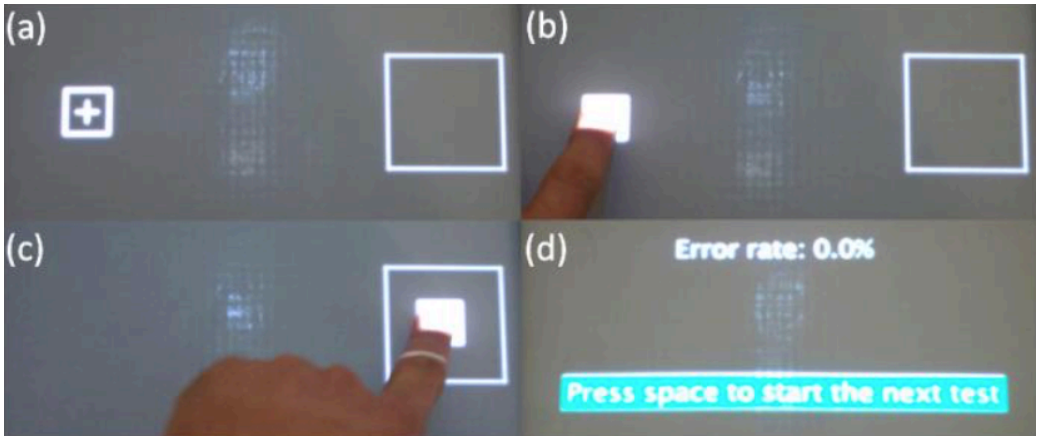


Abbildung 5.3: Der Dragging-Task des ersten Experiments

In Figure 5.3 wird der Ablauf des Experiments dargestellt. (a) vor der Berührung (b) nach der ersten Berührung (c) nachdem das Rechteck in der Zielfläche platziert wurde (d) nachdem der Benutzer den Finger wieder anhebte.

Für die Analyse und Auswertung des Experiments wurde ein Dragging Task in drei zeitlich aufeinanderfolgende Phasen aufgeteilt:

- **Phase 1 (Reaktionszeit)**: Die Zeit zwischen der Platzierung des Fingers an der Startposition und dem Wahrnehmen des visuellen Feedbacks
- **Phase 2 (Bewegungszeit)**: Die Zeit zwischen der Bewegung von der Startposition in Richtung Zielbereich
- **Phase 3 (Abschlusszeit)**: Die Zeit für die Feinjustierung innerhalb des Zielbereichs

Aufgrund der Testergebnisse konnten verschiedene Aussagen über die einzelnen Phasen des Tests getroffen werden.

- Zu Phase 1: Eine Änderung der Latenzzeit zwischen der ersten Berührung und dem visuellen Feedback hat einen signifikanten Einfluss auf den Zeitaufwand der Phase. Ein Unterschied von 1 und 10ms hatte allerdings nur marginalen Einfluss auf die Performance.
- Zu Phase 2: Interessanterweise hatte eine Erhöhung der Latenzzeit von 1 auf 10ms genauso wenig Einfluss wie die Erhöhung von 25 auf 50ms.

Zwischen Phase 2 und 3 konnten bei der Analyse des Protokolls verschiedene Strategien festgestellt werden: Die Partizipanten gingen je nach veränderter Variable (Latenz, Breite oder Distanz) unterschiedlich an die Aufgabe heran. Einige Partizipanten bewegten den Finger schnell und warteten je nach Länge der Latenzzeit in der Nähe des Zielbereichs auf das Rechteck, um anschliessend das Rechteck genau im Zielbereich platzieren zu können. Andere Partizipanten bewegten das Rechteck mit konstanter Geschwindigkeit von Startposition nach Zielbereich und mussten dadurch weniger Zeit für Feinjustierungen im Zielbereich investieren.

- Zu Phase 3: Bei der Feinjustierung in der letzten Phase konnten die grössten Differenzen zwischen den einzelnen Latenzzeiten festgestellt werden. Unterschiede zwischen 1 und 10ms waren wiederum nicht marginal, doch dann ab 25 oder 50ms wurden Performanceeinbussen deutlich ersichtlich. Dies war aufgrund der Tatsache, dass kleinere Positionsänderungen innerhalb des Zielbereichs durch eine erhöhte Latenzzeit schwerer war. Verglichen mit 10ms war die Durchführungszeit bei 50ms um 26% höher.

Experiment 2: Perception Responsiveness

Im zweiten Experiment wurden die Partizipanten dazu aufgefordert, den Bildschirm zu berühren (tappen). Bei der Berührung erschien nach einer gewissen (Latenz-)Zeit ein Rechteck, welches dann sogleich wieder verschwindete. Die Partizipanten mussten den Vorgang noch einmal wiederholen und beantworten, welches der beiden Rechtecke “schneller” war.

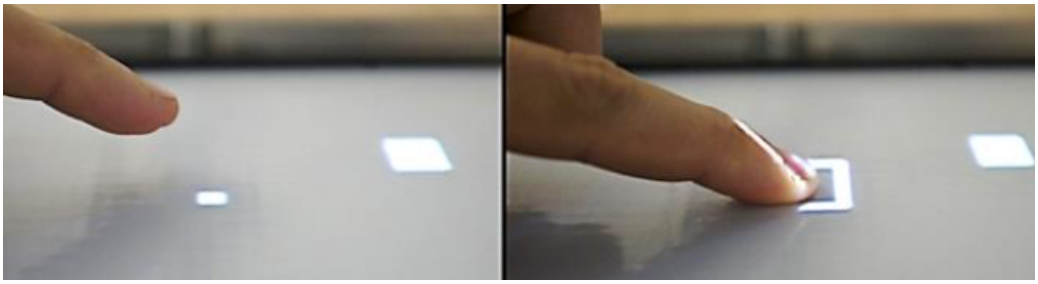


Abbildung 5.4: Der Pointing-Task des zweiten Experiments

In Figure 5.4 wird der Ablauf des Experiments dargestellt. Links: Bevor der Partizipant den Touchscreen berührt. Rechts: Nachdem er den Touchscreen berührt hat.

Die Testresultate zeigten, dass die Partizipanten nur in der Lage waren, einen Unterschied im Bereich von 20 und 100ms zu erkennen (Durchschnitt). Dieses Resultat ist insofern hilfreich um zu verstehen, warum in Phase 1 des ersten Experiments eine Erhöhung von 1 auf 10ms nur marginalen Einfluss auf die Performance hatte.

Fazit: Wie schnell ist denn nun schnell genug?

Eine Displaylatenzzeit von 20ms ist ausreichend, sodass die Allgemeinheit der Benutzer keine Leistungseinbussen wahrnimmt, solange die Benutzereingabe nur auf Tapping beschränkt ist. Bei keiner Beschränkung der Eingabemöglichkeiten, in unserem Fall z.B. das Dragging/Panning in Karten-Apps, hat bereits eine Latenzzeit höher als 2.38ms Einfluss auf die Wahrnehmung und Performance des Benutzers.

Zusammengefasst kann gesagt werden, dass eine höhere Latenzzeit einen grossen Einfluss auf die Performance haben kann. In der Umsetzung der Traildevils App muss deshalb besonders darauf geachtet werden, dass die Interaktion mit der Karte flüssig läuft. Bei Problemen müssen Performance-Optimierungen in Betracht gezogen werden, zum Beispiel durch Clustering oder Semantic Zooming.

In Bezug auf das Experiment eignete sich die Datenquelle `mcdonalds.geojson` sehr gut, hat es doch mehr als 14'000 McDonald's Restaurants in den USA und somit die gleiche Menge an Punkt-Koordinaten. Für das Experiment wurden keine aufwändigen Messungen gemacht, sondern es wurde nur subjektiv von den Studierenden, dem Auftraggeber und dem Betreuer beurteilt. Auf einem iPad Pro zeichneten sich mit dieser hohen Anzahl Punkte bereits hohe Latenzzeiten beim Panning ab.

Disclaimer zur Zusammenfassung und Übersetzung der ACM Publikation: "This translation is a derivative of ACM-copyrighted material. ACM did not prepare this translation and does not guarantee that it is an accurate copy of the originally published work. The original intellectual property contained in this work remains the property of ACM."

5.1.4 Marker Select Feedback

Sowohl das Erstellen und Hinzufügen als auch das Anpassen von einzelnen Punkten funktioniert mit Mapbox mühelos. Das Hinzufügen eines **einzelnen** Punktes funktioniert folgendermassen:

```
1 let point = MGLPointAnnotation()
2 point.coordinate = CLLocationCoordinate2D(latitude:
      ↪ 47.223272, longitude: 8.81734)
```

```
3 point.title = "University of Applied Sciences Rapperswil"
4 point.subtitle = "Where magic happens"
5 mapView.addAnnotation(point)
```

Nachdem der Punkt der Karte hinzugefügt wurde, werden die Delegate-Methoden von `MGLMapViewDelegate` aufgerufen. Erst innerhalb der aufgerufenen Delegate-Methoden kann ein Punkt weiter angepasst und gestylt werden. Die Methode `func mapView(_ mapView: MGLMapView, imageFor annotation: MGLAnnotation) -> MGLAnnotationImage?` gibt beispielsweise ein Bild für einen Punkt zurück. Welches Bild für welchen Punkt zurückgegeben werden soll, kann mithilfe von Informationen innerhalb der Annotation unterschieden werden.

Abbildung 5.5: Mapbox bietet für unsere Bedürfnisse genügend Möglichkeiten zur Anpassung von Punkten.

Möchte man sowohl den Marker als auch das Callout komplett überarbeiten, stehen zwei Basis-Klassen zur Verfügung. `MGLAnnotationView` und `MGLCalloutView` erlauben, den statisch sichtbaren Marker und das bei einem Touch erscheinende Callout anzupassen.

5.1.5 Filtering

Als Basis dient die Datenquelle `siliconvalley.geojson`. Ein Datensatz (POI) sieht folgendermassen aus:

```
"geometry": {
  "type": "Point",
  "coordinates": [-122.0583837, 37.4106419]
},
"type": "Feature",
"id": "node/1742329171",
"properties": {
  "amenity": "school",
  "id": "node/1742329171",
  "name": "Singularity University "
}
```

Für die Umsetzung wurden für alle gewünschten Typen Style Layers mit unterschiedlichen Prädikaten definiert, welche ein- oder ausgeblendet werden.

```
styleLayer.predicate = NSPredicate(format: "amenity == '
    ↪ school'")
```

Mit Switches vom Typ `UISwitch` wird die Sichtbarkeit des Layers geändert.

```
func filterStateChanged(sender: UISwitch) {
    styleLayer.isVisible = sender.isOn
}
```

5.1.6 Path Selection

Eine Linie ist eine Verbindung zwischen zwei Punkten. Eine Polyline ist eine Verbindung zwischen mehreren Linien. Das Hinzufügen von Linien oder Polylines funktioniert ähnlich wie bei Punkten. Es müssen bloss die einzelnen Koordinaten als `CLLocationCoordinate2D` Array Referenz dem Konstruktor mitgegeben werden.

```
var coordinates = [CLLocationCoordinate2D]()
```

```
2
3 for point in points {
4     coordinates.append(point.coordinate)
5 }
6
7 let line = MGLPolyline(coordinates: &coordinates, count:
      ↪ UInt(coordinates.count))
8 line.title = "Marathon-Strecke"
9 mapView.addAnnotation(line)
```

Es könnte eigentlich erwartet werden, dass auch eine `MGLPolyline` auf ein Touch-Feedback reagiert, dies war aber leider nicht der Fall (wird aber in Issue #2082 getracked). Es musste also ein Workaround entwickelt werden. Der Workaround nutzt den Vorteil, dass Punkte auf Touch-Feedback reagieren und auch individuell gestylt werden können. Die hinzugefügten Punkte sind alle unsichtbar und liegen im Vordergrund der Linie. Der Workaround funktionierte bestens mit wenig Linien (und somit wenigen Punkten). Es wurde dann aber schnell klar, dass es bei einer hohen Menge an Daten (bereits ab 100 Linien) Performance-Probleme gab.

Idee des Workarounds

1. Der erste Punkt (Startpunkt) wird aus dem Array ausgelesen und der Karte hinzugefügt
2. Der nächste Punkt wird ausgelesen und der Karte hinzugefügt
3. Überschreitet die Länge zwischen dem vorherigen Punkt und dem aktuellen Punkt einen Schwellwert (z.B. 200m), so werden rekursiv zwischen den beiden Punkten weitere Punkte der Karte hinzugefügt
4. Wieder zu 2.

Umsetzung des Workarounds

```
1 var coordinates = [CLLocationCoordinate2D]()
2 for (index, _) in coordinates.enumerated() {
3     if index == 0 { continue } // skip first
4     split(coordinates[index - 1], coordinates[index])
5 }
```

```
6
7  func split(_ from: CLLocationCoordinate2D, _ to:
       ↪ CLLocationCoordinate2D) {
8      if distance(from, to) > 200 { // THRESHOLD is 200m
9          let middle = mid(from, to)
10         add(coordinate: middle)
11         split(from, middle)
12         split(middle, to)
13     }
14
15     add(coordinate: from)
16     add(coordinate: to)
17 }
18
19 func distance(_ from: CLLocationCoordinate2D, _ to:
       ↪ CLLocationCoordinate2D) -> Double {
20     let fromLoc = CLLocation(latitude: from.latitude,
       ↪ longitude: from.longitude)
21     let toLoc = CLLocation(latitude: to.latitude, longitude
       ↪ : to.longitude)
22     return fromLoc.distance(from: toLoc)
23 }
24
25 func add(coordinate: CLLocationCoordinate2D) {
26   let point = MGLPointAnnotation()
27   point.coordinate = coordinate
28   point.title = "Marker selected"
29   point.subtitle = "\(coordinate.latitude) / \(coordinate.
         ↪ longitude)"
30   mapView.addAnnotation(point)
31 }
```

Lösung

Der vorhergehende Workaround wurde verworfen, doch mithilfe des erfassten Issues #6505 und der Antwort des Mapbox-Entwicklers Minh Nguyen konnte dann eine relativ simple Lösung gefunden und umgesetzt werden. Durch

das Hinzufügen eines `UITapGestureRecognizer` auf die `MGLMapView` kann ausgelesen werden, was sich unterhalb eines Touches auf der Karte befindet.

> UITapGestureRecognizer is a concrete subclass of UIGestureRecognizer that looks for single or multiple taps. For the gesture to be recognized, the specified number of fingers must tap the view a specified number of times.

Nachdem ein oder mehrere Touches registriert wurden, kann die exakte Location innerhalb dieser View ausgelesen und so bestimmt werden, was für Elemente (Features) sich an eben dieser Position befinden.

```
1 func handleTap(recognizer: UITapGestureRecognizer) {
2   let location = recognizer.location(in: mapView)
3   for feature in mapView.visibleFeatures(in:
        ↪ touchableSquare) {
4     print(feature)
5     print(feature.attributes)
6   }
7 }
```

Es besteht zudem die Möglichkeit, beim Aufruf von `mapView.visibleFeatures` die Auswahl an Style Layers einzuschränken.

```
1 for feature in mapView.visibleFeatures(in: touchableSquare,
      ↪  styleLayerIdentifiers: ["destinations", "trails",
      ↪  "tracks", "dealers"]) {
2
3 }
```

Der Touch auf das Gebäude 1 der Hochschule Rapperswil ergibt zum Beispiel folgenden Output in der Konsole

```
1 <MGLPolygonFeature: 0x1702b8cc0; count = 24; bounds = { sw
      ↪ = {47.2, 8.8}, ne = {47.2, 8.8}}>
2 ["type": university, "class": school]
3 <MGLPointFeature: 0x17266d840; title = (null); subtitle = (
      ↪ null); coordinate = 47.223341, 8.817389>
```

```
4 ["ref": , "name": HSR Gebäude 1, "name_fr": HSR Gebäude 1,
      ↪ "name_es": HSR Gebäude 1, "name_en": HSR Gebäude
      ↪ 1, "name_de": HSR Gebäude 1, "maki": marker, "
      ↪ localrank": 2, "name_zh": HSR Gebäude 1, "type":
      ↪ University, "scalerank": 3, "name_ru": HSR Gebäude
      ↪  1]
```

5.1.7 Fazit

Während dem Umsetzen der Experimente wurden die Grenzen und Möglichkeiten von Mapbox schnell erreicht. Der zum Zeitpunkt der Experimente aktuelle Stand von Mapbox iOS SDK 3.3.4 war nicht ausreichend, um alle Experimente zufriedenstellend umzusetzen. Glücklicherweise konnten die Experimente mit der Test-Version 3.4.0 durchgeführt werden. Diese Version erlebte während dem Experimentieren mehrere Alpha- und Beta-Versionssprünge und entwickelte sich stetig weiter. Die bereits umgesetzten Experimente mussten deswegen mehrmals umgeschrieben werden. Version 3.4.0 konzentriert sich stark auf das iOS Runtime Styling und somit auf Kernfunktionen, die von der Traildevils App benötigt werden. Auch die Stabilität des Mapbox hat sich laufend verbessert und Abstürze, die auf Mapbox zurückzuführen sind, konnten seit der zuletzt veröffentlichten Version 3.4.0-beta.5 nicht mehr beobachtet werden. Es steht im Moment noch kein Termin für einen Stable-Release von 3.4.0 fest, allerdings neigt sich der Milestone “ios-v3.4.0” gemäss GitHub dem Ende zu.

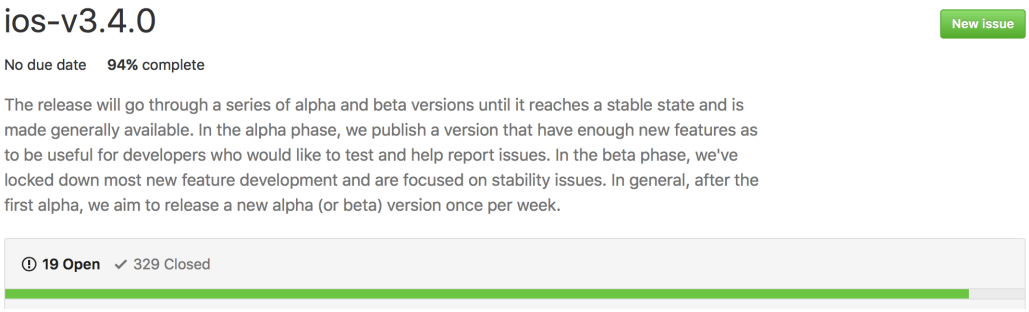


Abbildung 5.6: Der nächste Mapbox iOS Release steht in den Startlöchern

5.2 Szenarien

Dieses Kapitel dient zur Festhaltung aller erreichten Resultate während der Vorstudie. Mit einem Prototyp wurden die möglichen Funktionen von Mapbox aufgezeigt. In einem nächsten Schritt wird mit realen Daten gearbeitet, um am so nah wie möglich an das fertige Produkt zu kommen. Anhand von definierten Szenarien wird die Machbarkeit erneut geprüft, um die Risiken mit einem realitätsnahen Beispiel zu minimieren.

5.2.1 Szenario 1: Zoom in and show data

Startpunkt dieses Szenarios ist die Sicht auf die ganze Schweiz. Das Clustering ist aktiv und gruppiert naheliegende Trails, um dem Benutzer zu visualisieren, wo sich die Startpunkte ungefähr befinden.

Mit einer "Zoom-In" Interaktion, formieren sich die Clustergruppen neu bis zu einer Zoomstufe, in der Wanderwege und Höhenlinien sichtbar werden. Jetzt wird umgeschaltet. Tracks und deren Startpunkte werden sichtbar.

Sichtbare Trails oder Tracks kann der Benutzer selektieren. Bei der Selektion hebt sich der Trail oder Track visuell von den anderen ab. Er besitzt also einen Aktiv-Status.

5.2.1.1 Clustering

Durch die vorgängigen Experimente konnte das Clustering sehr schnell implementiert werden. Zusätzlich wurde durch eine Prädikatenlogik die Farbe und die Grösse der Cluster-Punkte festgelegt.

Cluster-Punkte unterscheiden sich anhand drei verschiedenen Attributen:

- Farbe des Punktes
- Durchmesser des Punktes
- Label für die Anzahl POI-Marker innerhalb eines Clusters

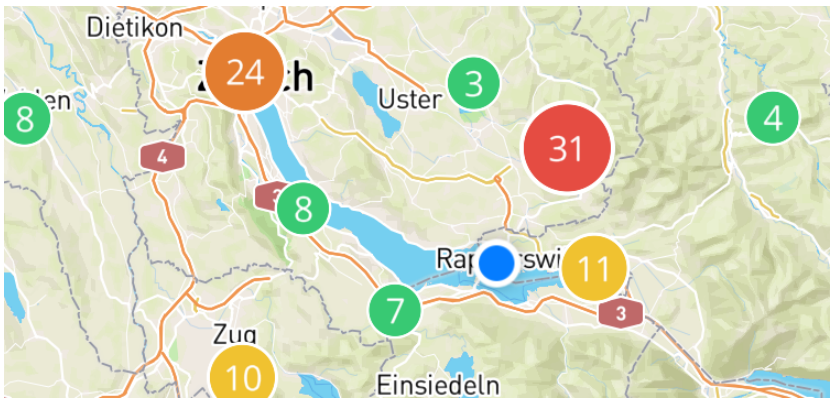


Abbildung 5.7: Visuelle Unterschiede der Cluster-Punkte

Sie soll dem Benutzer helfen, grosse und kleine Cluster schnell zu identifizieren. So weiss er, dass ein grosser und roter Cluster-Punkt tendenziell mehr Trails beinhaltet, als ein kleiner und gelber. Durch die unterschiedliche Visualisierung eignet sich das Clustering auch für Personen mit Farbenblindheit.

5.2.1.2 Darstellen von Trails und Tracks

Es gilt zu evaluieren, welche Zoomstufe sich am besten für das Anzeigen der Trails und Tracks eignet. Durch das Verwenden der Outdoor-Map von Mapbox werden uns Wanderwege und Höhenlinien angezeigt. Je nach Zoomstufe erscheinen mehr oder weniger Informationen.

Die Zoomstufe 12 hat sich als sehr geeignet erwiesen, da diese die relevanten Informationen anzeigt. Erste Höhenlinien sind sichtbar und es kann auf einen Blick das Höhenprofil des Trails ermittelt werden. Zudem werden erste Wanderwege angezeigt. Die Zoomstufe ist ausreichend, um den Überblick zu bewahren.

5.2.1.3 Selektierung von Trails und Tracks

Bereits in den Experimenten ist aufgefallen, dass die Auswahl eines Punktes, wie auch die Auswahl einer Polyline mit Schwierigkeiten verbunden war. Konkret wird die Selektierung eines Features von Mapbox nicht standard-

mässig unterstützt. Es gibt im Moment keine Events oder Callbacks, auf die sich registriert werden kann. Deshalb haben wir hierzu drei alternative Ansätze ausgearbeitet.

Für die nachfolgenden Ansätze wurde immer die gleiche Methodik verwendet: Für jeden Kartenelement-Typ (Trails, Tracks, Destinations oder Dealers) gibt es zwei Style Layers, die beide dieselben GeoJSON-Daten beinhalten. Der hierarchisch tiefere Style Layer zeigt alle Features mit POI-Markers an. Der hierarchisch höhere Style Layer (auch "Active Style Layer" genannt) dient zur Anzeige von selektierten Kartenelementen. Um explizit nur ein Element (oder mehrere) innerhalb eines Style Layers anzuzeigen, wird mit Prädikaten gearbeitet. Ein Style Layer bietet hierfür die Eigenschaft `.predicate` an.

```
1 // 1. Bei der Auswahl eines Trails wird die ID ausgelesen
2 let id = feature.attributes["id"]
3
4 // 2. Anschliessend wird auf dem "Active Style Layer" ein
      ↪ Prädikat gesetzt
5 layer.predicate = NSPredicate(format: "id == '\(id)'")
```

Ansatz I

Beim Auftreten eines Touch-Ereignis werden alle darunterliegenden Style-Layer überprüft. Da die ganze Karte auf mehreren Schichten basiert, kann hiermit auch auf Strassen oder Häuser geprüft werden. Natürlich sind nur die Kartenelement-Typen erwünscht. Darum wird die Suche auf die entsprechenden Style Layers eingeschränkt.

Da diese Elemente ihre Koordinaten in Attributen enthalten, konnten diese einfach über die Eigenschaft `.features` ausgelesen werden. Auf dem Active Style Layer wird dann die aktive Selektion mithilfe des Prädikates gesetzt.

Bei einer höheren Zoomstufe wurde jedoch festgestellt, dass sich nur einzelne Segmente eines Trails selektieren lassen. Es ist bekannt, dass eine solche Karte aus mehreren Tiles besteht. Es bestand der Verdacht, dass diese Kacheln der Grund für das fehlerhafte Verhalten sind. Nach einer kurzen Recherche wurde auch der entsprechende Eintrag in der Dokumentation gefunden.

> Features come from tiled vector data or GeoJSON data that is converted to tiles internally, so feature geometries are clipped at tile boundaries and features may appear duplicated across tiles. For example, suppose the specified rectangle intersects with a road that spans the screen. The resulting array includes those parts of the road that lie within the map tiles covering the specified rectangle, even if the road extends into other tiles. The portion of the road within each map tile is included individually.
> - Source [19]

Der Verdacht hat sich somit bestätigt. Beim Aufruf der Funktion `-visibleFeaturesInRect:inStyleLayersWithIdentifiers:` werden nur die Informationen eines Features ausgewählt, die sich innerhalb eines Tiles befinden.

Ansatz II

Mit dem exakt gleichen Vorgehen soll auch in diesem Ansatz ein einzelner Punkt oder eine einzelne Polyline identifiziert werden. Dabei werden jedoch nicht gleich die Koordinaten herausgelesen, sondern einen Identifikator in Form einer ID. Mit Hilfe dieser ID wird dann das Feature, welches alle Koordinaten enthält aus der Datenbank geholt.

Das aktive Element soll auch in dieser Variante auf einem separaten Layer gezeichnet werden. Bevor nun ein neues aktives Element ausgezeichnet werden kann, werden alle vorherig aktiven Elemente vom Style Layer entfernt.

Dieses Vorgehen resultierte in einem undefinierbaren Zustand: Es konnte zum Beispiel ein Track ausgewählt werden, aber das Feedback fehlte. Beim Verändern der Zoomstufe tauchte jedoch plötzlich der aktive Track auf.

Auch dieser Ansatz hat sich nicht bewährt.

Ansatz III

Das Touch-Ereignis aus Ansatz I und das Suchen in der GeoJSON Datei aus Ansatz II haben sich bis jetzt bewährt. Deshalb werden diese beiden Vorgehensweisen auch hier eingesetzt.

Es existieren weiterhin einzelne Style Layers für Trails, Tracks und Dealers. Beim Starten der App sind somit alle Elemente bereits geladen. Bei der Auswahl eines Trails wird auf dem Style Layer ein Prädikat gesetzt, welches nur für die eindeutige Identifikation des Kartenelements `true` zurückgibt. Mit unterschiedlicher Farbe wird es als aktiv ausgezeichnet.

Erstaunlicherweise funktioniert das Wechseln des Prädikats zur Laufzeit sehr performant. Auch die bidirektionale Auswahl eines Tracks beim Touch auf einen Trail und umgekehrt funktioniert ohne Probleme.

5.2.2 Szenario 2: Load, Search and Sync

Dem Benutzer wird die Möglichkeit geboten, ausgewählte Karten-Ausschnitte herunterzuladen. Der Download dieser Mapbox Karten-Daten ist Strategie-unabhängig und wird deshalb nicht näher erläutert.

Nachfolgend werden die evaluierten Strategien erklärt und mit Vor- und Nachteilen gewichtet.

5.2.2.1 Online First Strategie

In dieser Strategie liegt der Schwerpunkt auf Einfachheit. Jegliche Kommunikation läuft über die bestehende API. Dazu gehören Suchanfragen oder Tracks in einem bestimmten Kartenausschnitt.

Entscheidet sich der Benutzer, die Traildevils-Daten offline zu nehmen, wird für die Daten eine lokale Datenbank ohne Synchronisations-Logik erstellt. Stehen dem Benutzer zu einem späteren Zeitpunkt neuere Daten zur Verfügung, wird ihm eine Meldung angezeigt. Beim Download wird der ganze lokale Speicher mit den neuen Daten überschrieben. Hierbei werden immer wieder alle Resourcen heruntergeladen.

Ist der Benutzer offline (oder hat schlechten Empfang z.B. GPRS) und möchte gerne einen bestimmten Trail in seinem Kartenausschnitt suchen, werden die Einträge in der lokalen DB durchsucht. Ist der Benutzer online, hat aber einen Offline Ausschnitt auf seinem Gerät gespeichert, wird bei der Suche

über die API kommuniziert.

Zudem besitzen Offline Maps ein Ablaufdatum oder können manuell aktualisiert werden.

Vorteile:

- Geringer Aufwand wegen einfacher Synchronisation
- Der Download aller Resourcen benötigt ca 1.5 MB

Nachteile:

- Es wird immer der ganze Payload vom Server geholt
- Es müssen zwei verschiedene Logiken zur Abfrage von Daten implementiert werden (lokal und remote)

5.2.2.2 Offline First Strategie

Beim ersten Start der App werden initial alle verfügbaren Daten heruntergeladen (Trails, Tracks, Destinations, Dealers). Suchanfragen gehen somit immer auf eine stets aktuelle, lokale Datenbank. Die Synchronisierung zwischen Datenbank und Server geschieht autonom zu definierten Zeitpunkten (z.B. beim Start, bei Wiederaktivierung, alle 30 Minuten, usw.).

Bei der Synchronisierung sendet die App dem Server ein Request mit dem Datum der letzten Abfrage `last-modified` und erhält als Response ein JSON-Array mit allen neuen, geänderten und gelöschten Daten seit diesem Datum.

Beispiel einer Response:

```
22 {
23   "trails": {
24       "create": [ ],
25       "update": [ ],
26       "delete": [ ]
27   },
28   "tracks": { ... },
```

```
29    "destinations": { ... },
30    "dealers": { ... }
31  }
```

Zudem soll die App dem Benutzer die Möglichkeit bieten, die Traildevils-Daten explizit zu aktualisieren (z.B. durch eine Schaltfläche).

Vorteile:

- Suchanfragen sind schneller, da lokal
- Keine spezielle Behandlung zwischen Suchanfragen wenn online und Suchanfragen wenn offline
- Alle Daten sind implizit immer aktuell und auf dem Gerät vorhanden

Nachteile:

- Synchronisierungs-Logik zwischen Datenbank und Server u.U. komplex
- Erster Start dauert länger, da initial mehr Daten heruntergeladen werden müssen

5.2.2.3 Umgesetzte Strategie

Die vorherigen Strategien wurden im Plenum besprochen und diskutiert. Während der Besprechung haben sich Teile aus beiden Strategien als sinnvoll erwiesen. Am Ende hat man sich auf eine Hybrid Lösung geeinigt, welche anschliessend erläutert wird.

Server

- Die Traildevils API liefert die grundlegenden Kartenelemente (Trails, Tracks, Dealers und Destinations) als `.geojson` mit einem zugehörigen Titel-Attribut für die Suche aus
- Die Detailinformationen zu solch einem grundlegenden Element, z.B. zu einem Trail, werden als `.json` ausgeliefert, da dieses keine geografischen Elemente enthalten

Client

- Damit beim ersten Start der App der Benutzer nicht auf eine leere Karte blickt, wird mit dem Download der App aus dem App Store ein Set von grundlegenden Kartenelementen (Trails, Tracks, Dealers und Destinations) mitgeliefert
- Beim Start der App werden diese (veralteten) Elemente über die bestehende Traildevils API im Hintergrund aktualisiert, damit der Benutzer stets aktuelle Daten hat
- Nach dem Download von neuen Daten werden diese Lokal gespeichert und das `expired_at` neu gesetzt

Detailinformation

- Wenn dann z.B. nach einem Trail gesucht wird, wird ausschliesslich in der lokalen DB danach gesucht
- Die `.geojson` files sind bereits Teil der DB und enthalten den Titel für die Suche
- Trifft die Suche zu, wird mit der zugehörigen ID nach den Detailinformation gesucht
- Ist der Eintrag vorhanden, wird seine Validität aufgrund dem gesetzten `expired_at` Flag geprüft
- Ist der Eintrag valide, wird dieser ausgeliefert
- Ist er nicht valid oder noch nicht Teil der lokalen DB, wird die Traildevils API angesprochen
- Der neue Eintrag aktualisiert den bestehenden Eintrag, oder erstellt einen neuen
- Dieser Zyklus kann für jede Detailinformation eines Kartenelementes durchgeführt werden

Assets

- Lokal in der App existiert ein Asset Store
- Assets werden zusätzlich zu den Detailinformationen ebenfalls lokal gecached und haben das gleiche Ablaufdatum

- Die App soll auf keinen Fall durch die Assets aufgebläht werden
- Vielleicht gibt es für die Grösse einen Schwellwert, den wir nicht überschreiten
- Vielleicht löschen wir das älteste Bild von einem Trail um das neue zu laden
- Vielleicht gibt es unterschiedlich grosse Bilder, die remote geholt werden können, um Bandbreite zu sparen

5.3 Traildevils API

Während der Umsetzung hat sich die effiziente Erstellung der Schnittstelle mit Apiary bewährt und das schnelle Anpassen der Spezifikation war sehr hilfreich. Der Austausch mit dem Projektpartner war durch dieses Vorgehen sehr angenehm. Das Feedback konnte direkt wieder in die Spezifikation einfliessen und auf dem Node.js Mock angepasst werden.

Trotzdem hat sich gezeigt, dass die statische API von Apiary schnell an ihre Grenzen stösst. Durch die Definition in einer Datei verliert man schnell den Überblick und vor allem bei einem Track, welcher hunderte Koordinaten der einzelnen Punkte beinhaltet, liegt man schnell bei 150 Zeilen. Dies blähte die Datei unnötig auf.

Kartenelemente treten entweder als `.geojosn` für die BikeMap oder als `.json` für die Detailansicht auf. Jedes dieser Objekte wurde für den Projektpartner spezifiziert.

5.3.1 API Mock

Es wird vermutet, dass die statischen Daten von Apiary nicht ausreichend für die Entwicklung sind. Zudem hat der Projektpartner noch keine produktive Traildevils Schnittstelle in Betrieb.

Aus diesem Grund wurde entschlossen, während der Entwicklung einen API Mock einzusetzen. Dieser soll auf Express basieren und als Gratis-Instanz auf Heroku laufen. So kann schnell und flexibel reagiert werden und die API

unseren Bedürfnissen angepasst werden.

Um nun das Clustering oder die generelle Performance zu testen wurden Daten benötigt, und zwar so viele wie möglich. Der Projektpartner hat die Daten zur Verfügung gestellt. Da Mapbox zwingend Daten im `.geojson` Format benötigt, wurden die gelieferten `.json` Dateien mutiert und in das `.geojson` Format konvertiert.

So konnten die Daten auf dem Mock ausgeliefert werden, um sie anschliessend auf den Endgeräten zu testen.

5.3.2 Wechsel zur produktiven API

Gegen Ende der Arbeit hat der Projektpartner die produktive API publiziert. Noch vor Abschluss der Arbeit konnte die produktive Schnittstelle implementiert und verwendet werden. Der Wechsel der API brachte keine grossen Probleme mit sich.

Einzig waren beim ersten Start der Applikation nach der Umstellung alle verfügbaren Kartenelemente in Ostafrika positioniert. Nach einer kurzen Untersuchung hat sich gezeigt, dass die Breiten- und Längengrade vertauscht waren.

Der ISO 6709 Standard [20] definiert die Repräsentation von geografischen Punkten mit Koordinaten. Dieser Standard besagt, dass Punktkoordinaten in Form von `[Breitengrad, Längengrad]` beschrieben werden.

Im Zusammenhang mit kartesischen Karten werden die Punkte jedoch in Form von `[X, Y]` Koordinaten beschrieben. Viele Geoinformationssysteme basieren auf dieser Notation, wobei die X-Koordinate den Längengrad und die Y-Koordinate den Breitengrad wiederspiegelt. Auch GeoJSON [21] schreibt so die Koordinaten eines Punktes vor. Dies war der Grund, warum die Breiten- und Längengrade vertauscht waren.

5.4 Design

In diesem Kapitel wird auf die grundlegenden Designentscheide und Hot-Spots der iOS Applikation eingegangen.

5.4.1 Farben

> In iOS, color can indicate interactivity, impart vitality, and provide visual continuity. Look to the system's color scheme for guidance when picking app tint colors that look great individually and in combination, on both light and dark backgrounds. - [22]

Aus diesem Grund wurden die Farben von Flat UI Colors verwendet [23], die einerseits den obigen Grundsatz widerspiegeln, andererseits gut mit dem bestehenden Design von Traildevils kombinieren lasse.

Abbildung 5.8: Flat UI Color Farbpalette

5.4.2 Icons

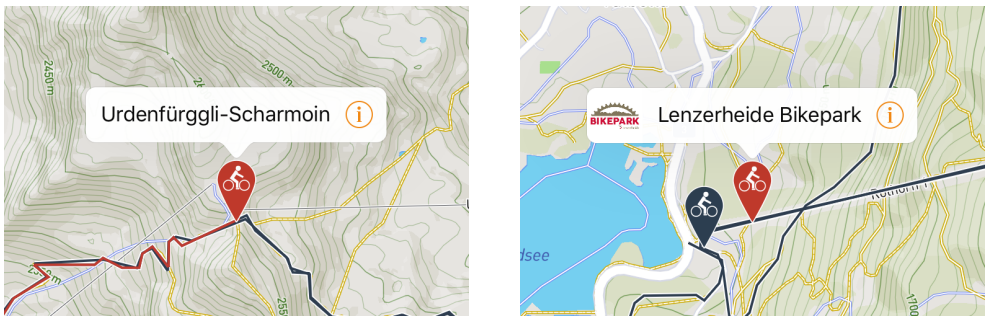
Die Icons wurden in erster Linie von der bestehenden Traildevils-Plattform übernommen. In den meisten Fällen wurden lediglich die Farben gemäss der Palette von oben angepasst. Es wurde zudem darauf geachtet, dass die Icons eine einheitliche Bildsprache besitzen.

Wurde ein zusätzliches Icon benötigt, konnte durch den Erwerb des Icon-Sets von Smashicons [24] immer ein passendes Icon für den Anwendungsfall gefunden werden. Mit der Mighty License (siehe Anhang) sind wir berechtigt, die Icons für sowohl persönliche, als auch professionelle Projekte zu verwenden.

Abbildung 5.9: Verwendete Icons in der Traildevils iOS App

5.4.3 Callout

Für ein aktives Kartenelement wird nebst der Farbänderung auch ein Callout angezeigt. Dieses Callout beinhaltet den Titel des selektierten Elementes und ein Logo auf der linken Seite, falls dieses vorhanden ist. Zusätzlich weist ein Icon auf der rechten Seite vom Callout den Benutzer auf die Detailinformationen hin.



a: Kein Logo vorhanden

b: Logo vorhanden

Abbildung 5.10: Variationen der Callouts

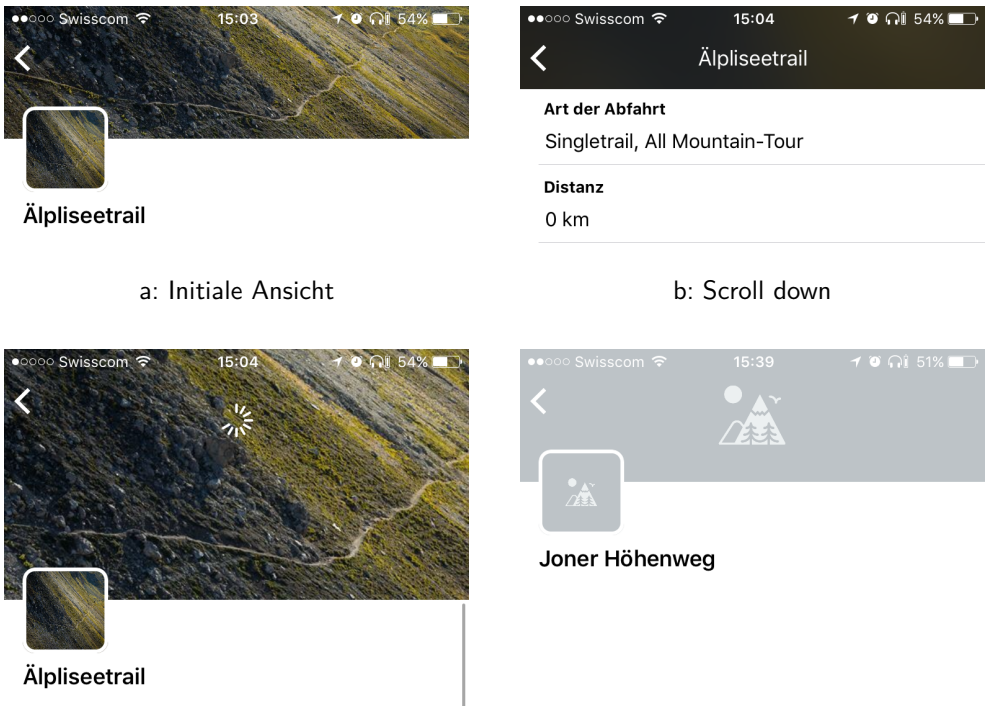
5.4.4 Header in der Detailansicht

Neben den strukturierten Daten einer Detailansicht, mussten auch ein Logo und ein Headerbild in der View platziert werden. Nach einer Recherche über mögliche Varianten, stellte sich die Detailansicht von Twitter als geeignete Lösung dar. Durch den "Sticky Header" verschwinden keine Navigationselemente und sind stets für den Benutzer bedienbar. Das grosse Headerbild kann oberhalb im Screen positioniert werden. Das Logo findet Platz bei der kleineren quadratischen Fläche leicht unter dem Headerbild.

Durch das Navigieren nach unten, wird der Header nach oben geschoben, bis er "sticky" bleibt. Das Logo verschwindet mit dem Namen des Elementes unter dem Bild. Ist der Name nicht mehr sichtbar, erscheint dieser bei weiterem Scrollen nach unten über dem "geblurrtem" Bild wieder.

Besitzt ein Element kein Headerbild oder ein Logo, wird für das nicht vorhandene Bild ein Platzhalter dargestellt. Diese sind im Bild 5.10d ersichtlich.

Nachfolgend sind die einzelnen Stati des Headers visuell präsentiert.



a: Initiale Ansicht

b: Scroll down

c: Force Refresh

d: Kein Logo und Headerbild vorhanden

Abbildung 5.11: Variationen des Headers in der Detailansicht

Stark von den Konzepten von [25] und [26] inspiriert wurde der Anlass genutzt, um einen solchen View Controller zu publizieren. Ganz ohne Abhängigkeiten zu anderen Libraries und komplett in Swift geschrieben wurde der eigene Twitter UI View Controller auf GitHub veröffentlicht [27].

5.4.5 Empty State

Einen Empty State beschreibt die Ansicht, welche der Benutzer in den folgenden Situationen antrifft:

- Der Benutzer hat sich gerade registriert
- Es sind noch keine Daten vorhanden
- Der Benutzer hat die Daten gelöscht
- Ein Fehler ist aufgetreten

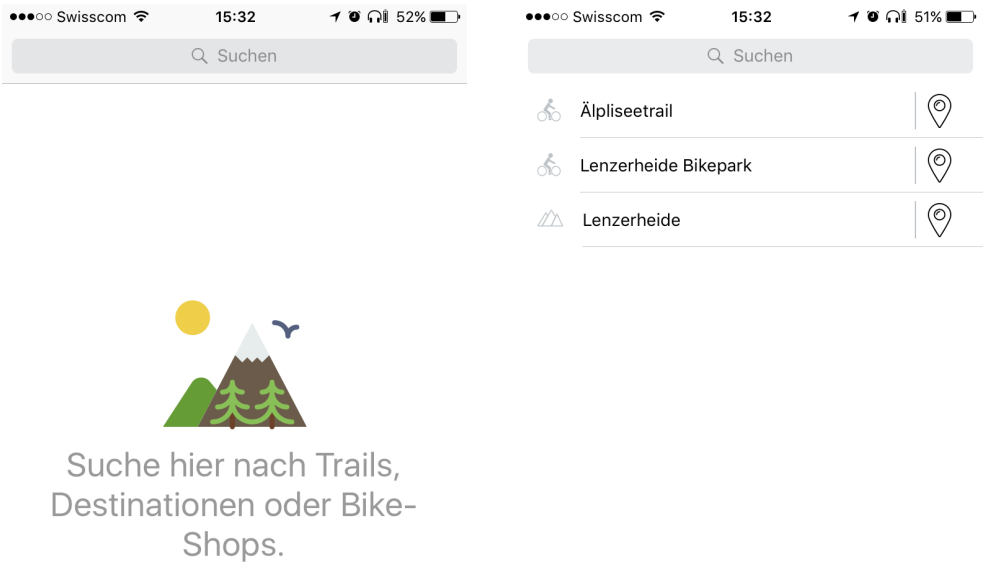
In solch einer Situation soll der Benutzer nicht einfach einen leeren Screen sehen.

Das Google Material Design [28] beschreibt drei verschiedene Alternativen zu einem Empty State:

- Starter content
- Educational content
- Best match

In der Traildevils iOS App kommt der Empty State bei der Ansicht zu den Suchresultaten zum Einsatz. Sind noch keine Suchresultate vorhanden, wird der Benutzer dabei unterstützt, nach Kartenelementen zu suchen. Somit ist dies ein "Educational content", wie es Google Material Design vorschlägt.

Dank der erlernten Kenntnisse in dem von Prof. Dr. Markus Stolze durchgeführten Modul "Web Engineering und Design 2" an der HSR - Hochschule für Technik in Rapperswil - konnte so ein Responsive Web Design Pattern in der Traildevils iOS App implementiert werden.



a: Empty State b: Zuletzt angesehenen Suchresultate

Abbildung 5.12: Ansicht der Suchresultate

5.5 Frameworks

Frameworks bilden eine flexible Grundlage und erleichtern die Programmierarbeit. Das Rad muss nicht immer neu erfunden werden, doch trotzdem ware es das Ziel, den Katalog an eingesetzten Frameworks möglichst klein zu halten.

5.5.1 Verwendung von Frameworks in Xcode

Um externe Frameworks in Xcode zu verwenden, müssen diese in einer sogenannten `.framework`-Datei vorliegen.

> A framework is a bundle (a structured directory) that contains a dynamic shared library along with associated resources, such as nib files, image files, and header files. When you develop an application, your project links to one or more frameworks. For example, iPhone application projects link by default to the Foundation, UIKit, and Core Graphics frameworks. Your code accesses the capabilities of a framework through the application programming interface (API), which is published by the framework through its header files. Because the library is dynamically shared, multiple applications can access the framework code and resources simultaneously. The system loads the code and resources of a framework into memory, as needed, and shares the one copy of a resource among all applications. - [29]

Abbildung 5.13: Veranschaulichung Framework

Frameworks lassen sich in einem iOS Projekt auf viele Arten einbinden. Nebst dem manuellen Kopieren und Einfügen eines Framework in das Projekt, ist die einfachste Möglichkeit der Einsatz eines Package Manager. Nachfolgend werden die drei bekanntesten Package Manager aufgelistet:

- Swift Package Manager (Repo: https://github.com/apple/swift-package-manager)
- Carthage (Repo: https://github.com/Carthage/Carthage)
- CocoaPods (Repo: https://github.com/CocoaPods/CocoaPods)

Seit Version 3.0 ist SPM integriert in Swift. Dieser bietet zur Zeit des Schreibens aber leider noch keinen Support für iOS, watchOS und tvOS Plattformen ([30]). Somit besteht nur noch die Wahl zwischen Carthage und CocoaPods. Beide Package Manager verfolgen dabei einen unterschiedlichen Ansatz. In Carthage werden in einer Datei (dem `Cartfile`) alle Dependencies definiert. Mit dem Ausführen von `carthage update` in der Kommandozeile werden alle Dependencies heruntergeladen, kompiliert und eine `.framework`-Datei erstellt. Der Benutzer kann dann diese Dateien via Drag-n-Drop dem Xcode-Projekt hinzugefügt werden.

CocoaPods macht vieles ähnlich, geht aber einen Schritt weiter. Wie bei Carthage werden in einer Datei (dem `Podfile`) alle Dependencies definiert und mit `pod update` werden alle Dependencies heruntergeladen und kompiliert. CocoaPods macht hier aber noch nicht Schluss, sondern erstellt eine einzige `.framework`-Datei, welche alle Dependencies beinhaltet, und fügt diese dem Xcode-Projekt automatisch hinzu.

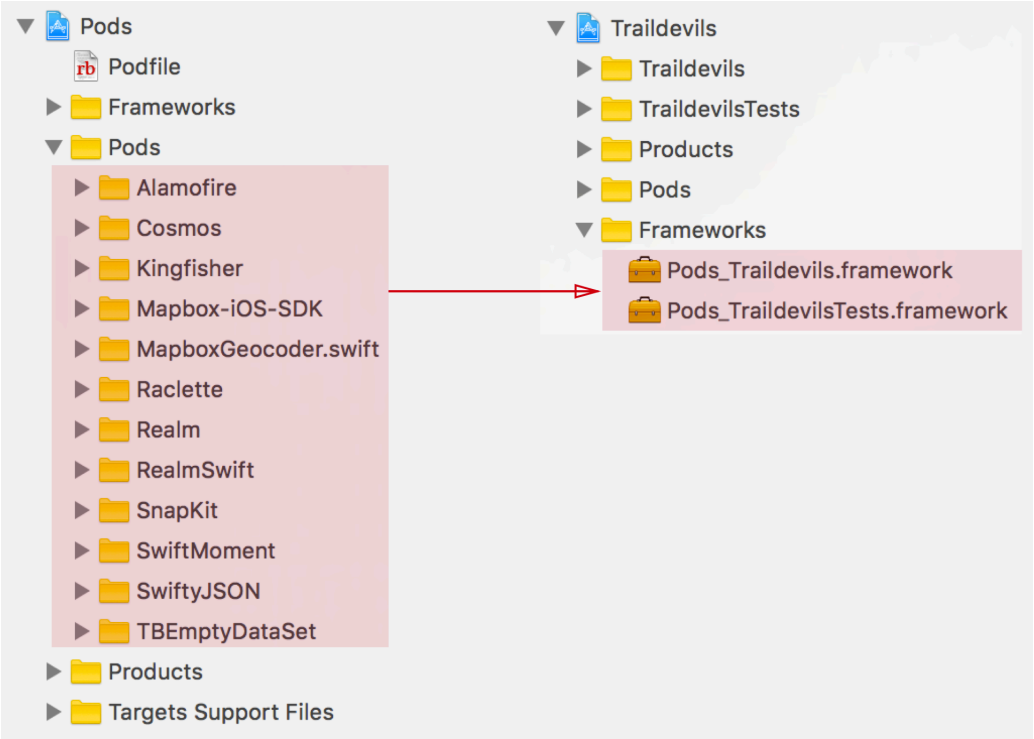


Abbildung 5.14: Aus den Pods wird ein gemeinsames Framework erstellt und automatisch dem Projekt hinzugefügt

Aufgrund dieser bequemen Art zur Verwaltung von Abhängigkeiten und der Tatsache, dass praktisch alle Open Source Projekte ihr Framework auf CocoaPods anbieten, aber nur wenige über Carthage, wurde entschieden, CocoaPods einzusetzen.

5.5.2 Evaluationskriterien

Die zu verwendenden Frameworks wurden anhand der folgenden, unsortierten Kriterienliste berücksichtigt und evaluiert:

- Open Source: Der Quellcode des Frameworks soll jedermann zur Verfügung stehen
- Lizenz: Das Framework darf für beliebige Zwecke, auch kommerziell, verwendet werden
- GitHub Popularität: Eine hohe Anzahl an Stars (“Follower eines Repositories”) bedeutet meist auch eine hohe Popularität, was wiederum

bedeutet, dass das Framework vermutlich aktiv weiterentwickelt und gewartet wird.
- CocoaPods kompatibel: Der Einsatz von CocoaPods erleichtert das Installieren und Aktualisieren von Frameworks
- Swift 3.0 kompatibel: Das Framework muss kompatibel zu Swift 3.0 oder höher sein
- Ausführliche Dokumentation: Das Framework soll ausführlich dokumentiert sein, bevorzugterweise mit einem Beispielprojekt

5.5.3 Eingesetzte Frameworks

Folgende Frameworks werden in der Traildevils iOS App eingesetzt (URL, GitHub Stars, usw. sind Stand 13. Dezember 2016).

5.5.3.1 Mapbox

Stars: 1'596 | Eingesetzte Version: 3.4.0-beta.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.

Nebst Google Maps und Apple Maps ist Mapbox ein weitere, grosse Mapping Plattform zur Anzeige von Karten. Dieses Framework wird verwendet, weil es Vorgabe des Auftraggebers war.

Repository URL: https://github.com/mapbox/mapbox-gl-native

5.5.3.2 MapboxGeocoder

Stars: 32 | Eingesetzte Version: 0.5.1 | Lizenz: ISC

Swift/Objective-C library for the Mapbox Geocoding API.

Um nach Adressen, Ortschaften, usw. im Internet zu suchen, wird der Swift Wrapper für die Mapbox Geocoding API verwendet.

Repository URL: https://github.com/mapbox/MapboxGeocoder.swift

5.5.3.3 RealmSwift

Stars: 9'472 | Eingesetzte Version: 2.1.0 | Lizenz: Apache License

Realm is a mobile database that runs directly inside phones, tablets or wearables.

Realm ist eine der führenden Datenbanken für Mobilgeräte, unter anderem auch für Android, React Native oder Xamarin. In der Traildevils App wird Realm als Offline Cache genutzt, um alle Kartenelemente, Suchresultate und Detailinformationen zu speichern.

Repository URL: https://github.com/realm/realm-cocoa

5.5.3.4 SnapKit

Stars: 8'353 | Eingesetzte Version: 3.0.2 | Lizenz: MIT

A Swift Autolayout DSL for iOS & OS X

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 ([31]). Storyboards in Kombination mit der Software-Architektur VIPER bereitete zu Beginn allerdings erhebliche Probleme, weshalb im weiteren Verlaufe des Projekte komplett auf Storyboards verzichtet wurde. 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.

Repository URL: https://github.com/SnapKit/SnapKit

5.5.3.5 SwiftyJSON

Stars: 12'701 | Eingesetzte Version: 3.1.3 | Lizenz: MIT

The better way to deal with JSON data in Swift

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. Mit SwiftyJSON können vor allem viele Code-Zeilen gespart werden.

Beispiel mit `JSONSerialization`:

```
1 if let JSONObject = try JSONSerialization.jsonObject(with:
      ↪ data, options: .allowFragments) as? [[String: Any
      ↪ ]],
2     let username = (JSONObject[0]["user"] as? [String: Any
      ↪ ])?["name"] as? String {
3       // There's our username
4 }
```

Beispiel mit SwiftyJSON:

```
1 let json = JSON(data: dataFromNetworking)
2 if let userName = json[0]["user"]["name"].string {
3   //Now you got your value
4 }
```

Repository URL: https://github.com/SwiftyJSON/SwiftyJSON

5.5.3.6 TBEmptyDataSet

Stars: 55 | Eingesetzte Version: 2.4.0 | Lizenz: MIT

An extension of UITableView/UICollectionView's super class, it will display a placeholder emptyDataSet when the data of tableView/collectionView is empty.

Ein prädestinierter Ort für den Einsatz des Empty State Patterns ([32]) in der Traildevils Applikation ist die Liste der zuletzt angesehenen Suchresultate. Für eine schnellere Umsetzung wurde TBEmptyDataSet eingesetzt, um ein Bild inklusive Text in der leeren Tabelle anzuzeigen.

Repository URL: https://github.com/teambition/TBEmptyDataSet

5.5.3.7 Alamofire

Stars: 20'961 | Eingesetzte Version: 4.1.0 | Lizenz: MIT

Elegant HTTP Networking in Swift

Das Framework wird eingesetzt, um Web-Requests an die Traildevils-API zu machen.

Repository URL: https://github.com/Alamofire/Alamofire

5.5.3.8 Kingfisher

Stars: 6'646 | Eingesetzte Version: 3.2.1 | Lizenz: MIT

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

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 z.B. bei einem Force-Refresh (oder bei Überschreitung des Ablaufdatums) sowohl die Detailinformationen im Cache von Realm, als auch die Bilder im Cache von Kingfisher invalidiert.

Repository URL: https://github.com/onevcat/Kingfisher

5.5.3.9 Cosmos

Stars: 410 | Eingesetzte Version: 7.0.0 | Lizenz: MIT

A star rating control for iOS / Swift

Cosmos dient zur einfachen Anzeige von Bewertungen mit Sternen.

Abbildung 5.15: Cosmos Star Rating Control

Repository URL: https://github.com/marketplacer/Cosmos

5.5.3.10 SwiftMoment

Stars: 1'387 | Eingesetzte Version: 0.7 | Lizenz: BSD-2-Clause

A time and calendar manipulation library for iOS 9+, macOS 13.11+, tvOS 9+, watchOS 2+, Xcode 8 written in Swift 3.

Inspiriert vom berühmten JavaScript Framework `Moment.js` kann auch mit SwiftMoment einfacher mit Daten und Zeiten gearbeitet werden.

Repository URL: https://github.com/akosma/SwiftMoment

5.5.3.11 Raclette

Stars: 4 | Eingesetzte Version: 1.1.0 | Lizenz: MIT

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

Raclette ist ein Framework zur Erweiterung der `UITableView`. Das Framework wird für die Darstellung der Tabelle in der Detailansicht von Kartenelementen verwendet. Es wurde von Roman Blum während der Studienarbeit entwickelt und unter der MIT Lizenz im CocoaPods-Katalog veröffentlicht. Aber was macht Raclette eigentlich genau? Um dies zu verstehen, muss zuerst auf die Grundlegende Komponenten `UITableView` eingegangen werden.

Was ist UITableView?

Tabellen sind sehr wichtige Komponenten in der iOS-Entwicklung und werden praktisch in jedem Projekt eingesetzt. Prominente Beispiele hierzu sind zum Beispiel das Apple Wetter App oder die Twitter-, Instagram und Facebook-Timeline. Die `UITableView` ist eine beeindruckend vielseitige Komponente zur Darstellung von tabellarischen Daten aus einer Datenquelle und ist seit der ersten iOS Version fest im Apple UIKit Framework verankert.

a: Apple Wetter App

b: Apple Alarm App

Abbildung 5.16: Einsatz der UITableView in bekannten Apple Apps

Im Zusammenhang mit der `UITableView` existieren zwei wichtige Protokolle: `UITableViewDataSource` und `UITableViewDelegate`. Während die Data Source für das Bereitstellen der Daten zuständig ist, dient der Delegate für das Aussehen und die Aktionen der Tabelle.

Einsatz

In der Traildevils App wird im Moment eine `UITableView` für die Suchresultate und die Detailansicht verwendet.

Problem

Der Einsatz der `UITableView` für die Suchresultate bereit keine Probleme. Die Datenquelle ist ein Array von Resultaten aus der Datenbank und der Geocoding API. Probleme gibt es erst in der Detailansicht, denn dort ist die Datenquelle nicht ein Array von Elementen sondern ein einzelnes Objekt mit verschiedenen Eigenschaften (Properties).

```
class MapDetailsTrailEntity: MapDetailsBaseEntity {
    let likes: Int?
    let trailTypes: [String]?
    let distance: String?
    let altitudeUp: String?
    let altitudeDown: String?
    let difficulty: Int?
    ...
}
```

Der Einsatz einer `UITableView` für diese Art von Daten wäre also eigentlich ungeeignet, doch die Daten müssen trotzdem in irgendeiner Art und Weise tabellarisch dargestellt werden.

Lösung

Um die Detailansicht trotzdem mit einer `UITableView` darzustellen, bedarf es einem Framework mit folgenden Anforderungen:

- Einfache, unkomplizierte Verwendung
- Unterstützung von eigenen Zellen (Custom `UITableViewCell`)
- Dynamische Zeilenhöhe

Die evaluierten Frameworks lösen diese Problematik auf unterschiedliche Art und Weise, doch keines erfüllte die Anforderungen komplett.

- **Shoyu**: Keine dynamische Zeilenhöhe, fehlende Features (Pull Request erstellt: https://github.com/yukiasai/Shoyu/pull/35)
- **Hakuba**: Zu komplex für das Einsatzszenario, Dokumentation unverständlich
- **TableManager**: Keine Swift 3.0 Unterstützung (Pull Request von Roman Blum erstellt: https://github.com/Morbix/TableManager/pull/49), komplizierte Implementierung von eigenen Zellen

Anstatt zu einem der bereits vorhandenen Frameworks eine Erweiterung zu programmieren, wurde entschieden, ein eigenes Framework zu erstellen. Ra-

clette enthält ein Minimum an Features, welche in der Traildevils App benötigt werden, trotzdem wurde es universell gehalten und kann, auch dank der guten Dokumentation, schnell von jedem Entwickler mit geringem Aufwand implementiert werden.

Repository URL: https://github.com/rmnblm/Raclette

5.6 Technologien

Nachfolgend werden die während der Entwicklung eingesetzten Technologien und Tools erläutert.

5.6.1 SwiftLint

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

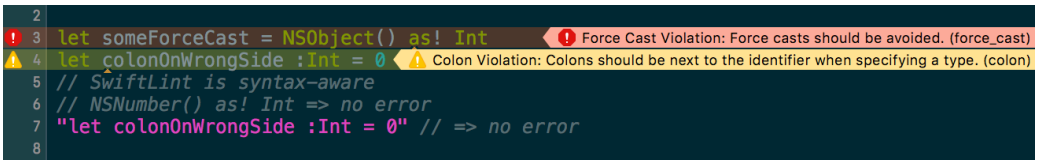


Abbildung 5.17: SwiftLint Integration in Xcode

Repository URL: https://github.com/realm/SwiftLint

5.6.2 Generamba

Mit Generamba kann anhand vordefinierter Templates (`.liquid`-Dateien) Code generiert werden. Da für jedes Modul der VIPER-Archiektur ein Grundgerüst an `.swift`-Dateien benötigt wird, wurde ein Template erstellt. Dadurch kann viel Zeit gespart und das Risiko auf Tippfehler minimiert werden.

```
➜ traildevils-ios git:(development) ✗ generamba gen MyModule swifty_viper

+---------------+----------------------------------+
|              Summary for gen MyModule            |
+---------------+----------------------------------+
| Targets       | Traildevils                      |
| Module path   | Traildevils/Modules/MyModule     |
| Test targets  | TraildevilsTests                 |
| Test file path | TraildevilsTests/Modules/MyModule |
| Template      | swifty_viper                     |
+---------------+----------------------------------+

Creating code files...
Creating test files...
```

Abbildung 5.18: Code-Generierung auf Knopfdruck mit Generamba

Beispiel eines Templates:

```
1  //
2  //  {{ module_info.file_name }}
3  //  {{ module_info.project_name }}
4  //
5  //  Created by {{ developer.name }} on {{ date }}.
6  //  Copyright © {{ year }} {{ developer.company }}. All
       ↪ rights reserved.
7  //
8
9  protocol {{ module_info.name }}ViewInput: class {
10
11 }
```

Repository URL: https://github.com/rambler-digital-solutions/Generamba

5.7 Offline Cache

Mit dem Download der Applikation werden Kartenelemente bereits mitgeliefert, damit beim ersten Start die Karte nicht leer ist, sondern direkt Daten angezeigt werden können. Die Initial-Daten werden als `.geojson`-Dateien ausgeliefert. Beim ersten Start der Applikation werden diese dann geparst

und in die Datenbank abgefüllt. Sofern das Ablaufdatum überschritten wurde, werden anschliessend im Hintergrund (asynchron) neue Daten von der Traildevils API angefordert. Nachdem die Daten heruntergeladen wurden, wird der Offline Cache und das Ablaufdatum aktualisiert.

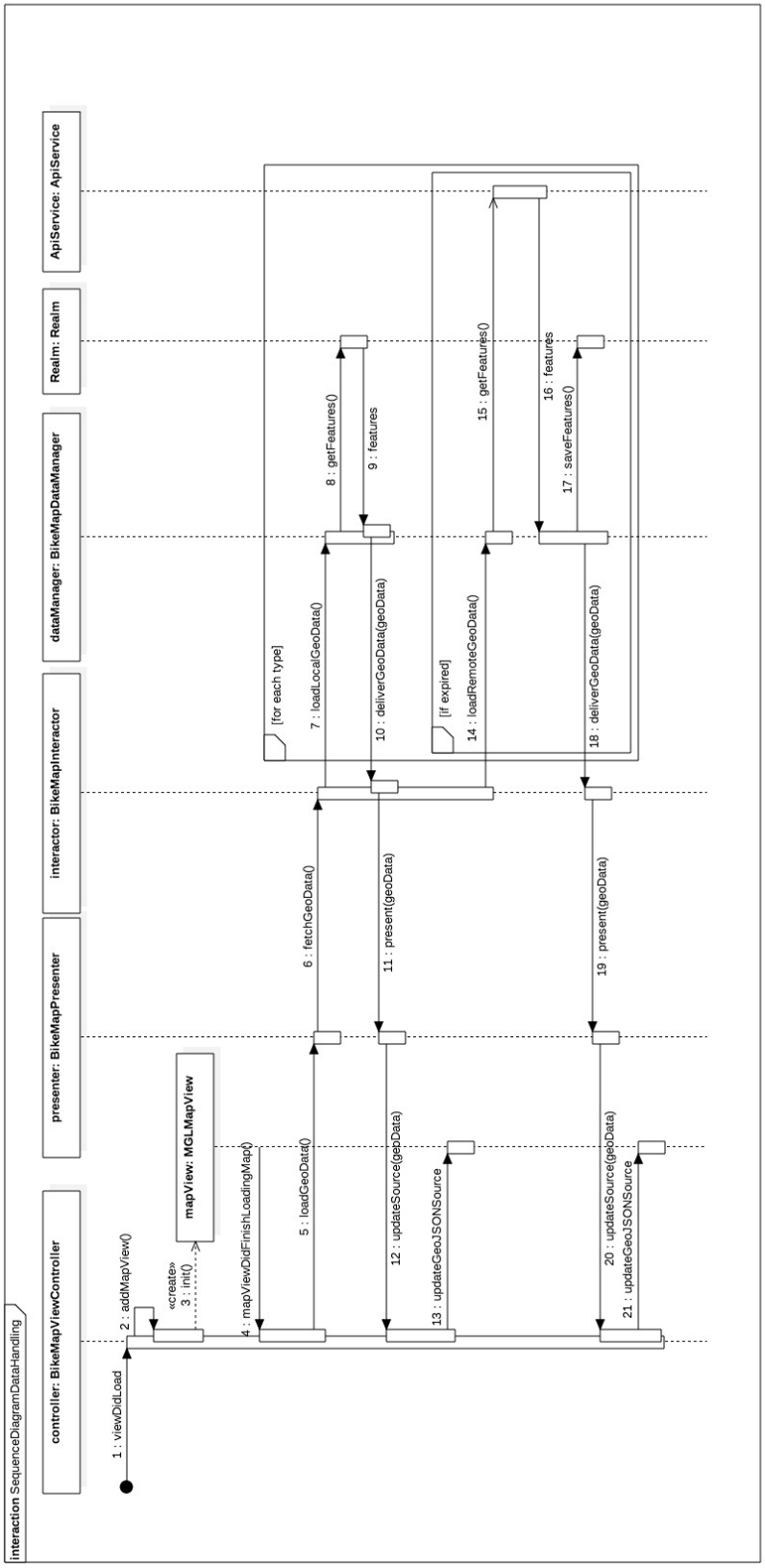


Abbildung 5.19: Sequenz Diagramm für den Ablauf mit lokalen und externen Daten

Der Offline Cache der Traildevils iOS App wurde mit der mobilen Datenbank Realm realisiert. Das Ziel war es, die Datenstruktur möglichst schlank zu halten. Es kommen deshalb auch nur zwei Datenbank-Entitäten in der gesamten Applikation zum Einsatz.

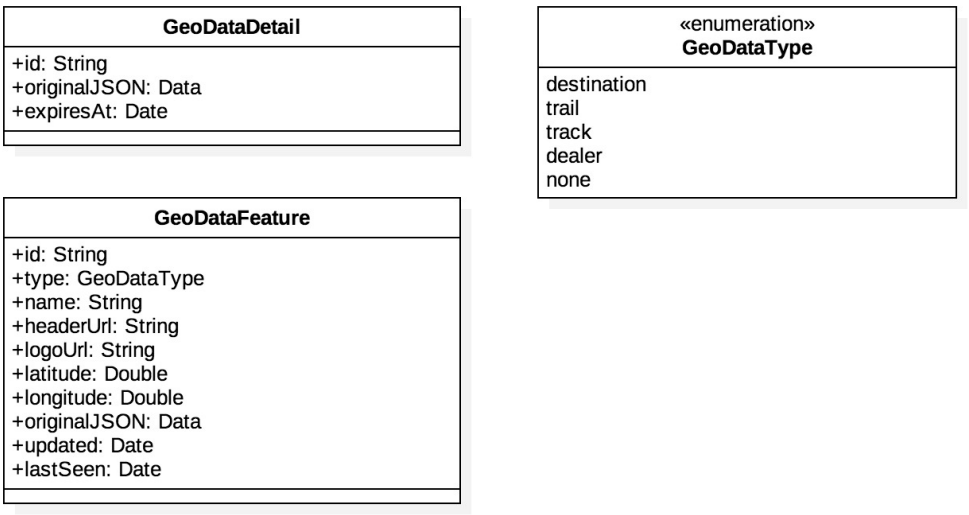


Abbildung 5.20: Klassendiagramm der Datenbank-Entitäten

Erwähnenswert ist hierbei, dass das originale JSON-Objekt auch mitgespeichert wird. Beim Laden einer Entität aus der Datenbank wird dann auf die benötigten Eigenschaften des JSON-Objektes zugegriffen.

5.8 Mapbox Versionssprünge

Während der Umsetzung der Traildevils iOS App kam es zu mehreren Versionssprüngen. Einige der bis dahin identifizierten Probleme wurden durch solche Updates behoben und machten die Studienarbeit erst umsetzbar. Nachfolend eine Liste an Änderungen (Auszug aus den Changelogs), welche unser Projekt besonders positiv beeinflusst haben.

- **alpha.1** A new runtime styling API allows you to adjust the style and content of the base map dynamically. All the options available

in Mapbox Studio are now exposed via MGLStyle and subclasses of MGLStyleLayer and MGLSource. (#5727)

- **alpha.1** GeoJSON sources specified by the stylesheet at design time now support cluster, clusterMaxZoom, and clusterRadius attributes for clustering point features on the base map. (#5724)
- **alpha.4** A style layer's predicate can now use the IN and BETWEEN operators, compare a property to nil, or always evaluate to YES or NO. (#6405)
- **alpha.5** A predicate property on each MGLStyleLayer subclass allows you to access and modify the NSPredicate object that determines which source features are included in the layer. The predicate corresponds to the filter property in the stylesheet. (#6049)
- **alpha.5** Fixed a crash that occurred when attempting to get a style layer or source that isn't part of the current style. (#6408)
- **beta.1** Added methods to MGLStyle for managing the style's images as UIImages. Use these methods in conjunction with MGLSymbolStyleLayer's iconImage property, MGLBackgroundStyleLayer's backgroundPattern property, etc. (#6637)
- **beta.1** Added the -[MGLMapViewDelegate mapView:didFinishLoadingStyle:] delegate method, which offers the earliest opportunity to modify the layout or appearance of the current style before the map view is displayed to the user. (#6636)
- **beta.3** Fixed a crash that occurred when removing an MGLSource or MGLStyleLayer from an MGLStyle and adding it back to the same MGLStyle or another MGLStyle. (#7048)
- **beta.3** Fixed an issue where view-backed annotations added while panning the map could become detached from the map and cause a crash. (#6924)
- **beta.4** Fixed an issue where the compass was not visible if the map was not full screen. (#7084)

Für die Umsetzung der Experimente musste aber oft die Mapbox Community auf GitHub nach möglichen Lösungsansätzen gefragt werden, daraus entstanden mehr als zehn Issues für sowohl Veständnisprobleme, als auch Fehler im Framework. Folgende chronologische Liste zeigt alle während der

Studienarbeit erfassten Issues.

- iOS Xcode “unknown schema version” (#6450)
- Crash when adding and removing the same StyleLayer at runtime (#6460)
- Weird behaviour when changing predicate of MGLCircleStyleLayer at runtime (#6478)
- Interacting with annotation points in style layer (#6505)
- Missing coordinates in MGLPolylineFeature (#6538)
- Is there a built-in way to display the point count on top of each cluster point? (#6609)
- MGLGeoJSONSource should have a constructor without options: (#6639)
- Setting center on MGLMapView initialization is at wrong coordinate (#6753)
- CompassView doesn’t update position when turning device from landscape to portrait mode (#6755)
- Open callout programmatically immediately after adding a MGLPointAnnotation (#6775)
- Clustering Issues: Wrong Point Count / Disappearing Clusters (#7387)

5.9 Deliverable

In diesem Kapitel geht es um den aktuellen Stand des abgegebenen Produktes (Basis-App). Der aktuelle Stand trägt die Versionsnummer 1.0 und Buildnummer 3. Gegen Ende der Realisierungsphase wurde dieser Stand auch ein letztes Mal über TestFlight den Testbenutzern zur Verfügung gestellt. Für diesen Build erhielten wir ein ausführliches Review von Prof. Dr. Markus Stolze und Mischa Trecco. Einzelne Kritikpunkte werden in den nachfolgenden Unterkapiteln aufgegriffen. Weitere Details zu allen Kritikpunkten inklusive einer Aufwandschätzung zur Behebung der Probleme befinden sich im Anhang.

5.9.1 Status: Funktionale Anforderungen

Im Rahmen dieser Studienarbeit konnten 4 von den 15 in Kapitel 3.1 definierten Use Cases umgesetzt werden. Namentlich sind dies:

- Kartenelemente selektieren: Der Benutzer ist in der Lage, Kartenelemente zu selektieren. Bei einem Touch öffnet sich ein Callout. Bei einem erneuten Touch wird er zur Detailansicht weitergeleitet.
- Kartenelemente suchen: Der Benutzer kann nach Kartenelementen suchen. Bereits ausgewählte Kartenelemente werden gespeichert und als "Search-History" angezeigt
- Geodaten suchen: Öffentliches Kartenmaterial wird bei der Suche berücksichtigt und ebenfalls in den Suchresultaten angezeigt. Dazu wird der Mapbox Geocoding Service verwendet.
- Detailinformationen zu einem Kartenelement anzeigen: Der Benutzer kann sich zu allen Kartenelementen Detailinformationen anzeigen lassen. Diese werden entweder aus dem Offline Cache geholt oder bei Nicht-Vorhandensein von der Remote API heruntergeladen.

Zusätzlich zu den Use Cases wurde die Positionsverfolgung implementiert. Beim Start der Applikation ist diese standardmässig aktiv. Das bedeutet, dass die aktuelle Position des Benutzers immer mittig in der Karte liegt. Bewegt sich der Benutzer mit seinem Gerät, so verschiebt sich nur die Karte. Der Benutzer kann diese Verfolgung deaktivieren, indem er mit der Karte interagiert (Panning). Dass die Verfolgung deaktiviert ist, wird dem Benutzer anhand eines typischen "Location"-Icon visualisiert. Mit einem Touch auf das Icon aktiviert sich die Positionsverfolgung wieder.

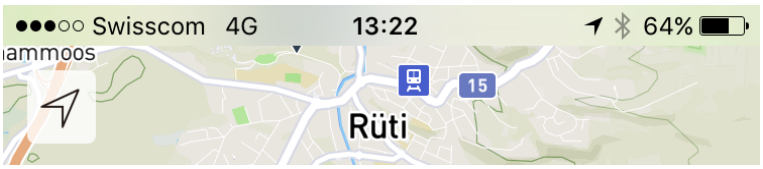


Abbildung 5.21: Ein Icon in der oberen linken Ecke visualisiert dem Benutzer, dass die Positionsverfolgung deaktiviert ist

Diese einfache Logik lässt sich in einem State Diagramm übersichtlich darstellen.

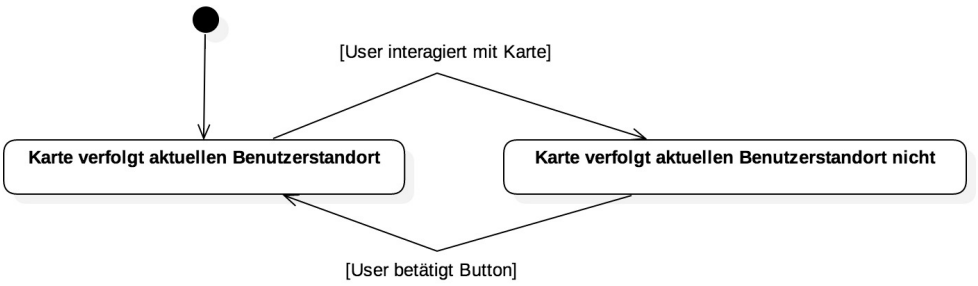


Abbildung 5.22: State Diagramm der Positionsverfolgung

5.9.2 Status: Nicht-funktionale Anforderungen

5.9.2.1 Performance

Die Performance beim Interagieren mit der Karte ist sehr gut, jedoch dauert der Initialstart der App zu lange. Der Download von Karten (Tiles) von Mapbox und das Laden und gleichzeitige Aktualisieren von Kartenelementen muss besser koordiniert werden.

5.9.2.2 Security

Die Basis-App enthält keine Benutzerauthentifizierung (Login/Logout). Es werden deshalb auch noch keine sensiblen Benutzerdaten gespeichert, die verschlüsselt werden müssen.

5.9.2.3 Wartbarkeit

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.

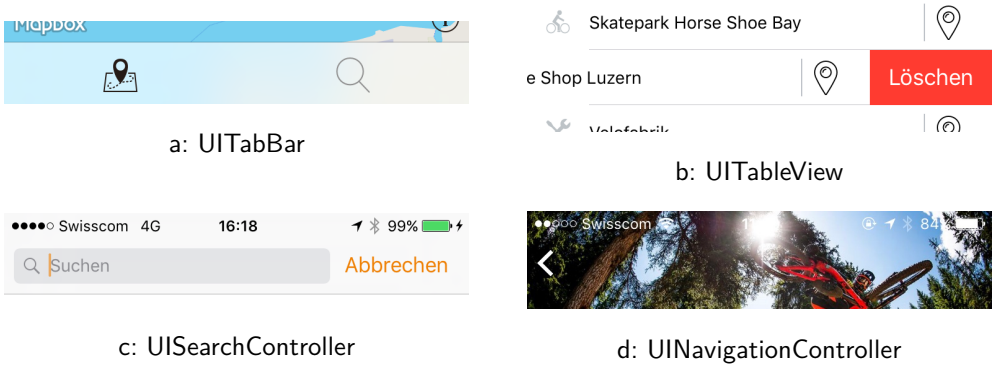
Die klar strukturierte Hierarchie des Xcode-Projektes, die einheitliche Namensgebung von Klassen, Funktionen und Variablen sowie der modulare Aufbau mit VIPER macht den Einstieg und die Weiterentwicklung erheblich leichter.

5.9.2.4 Verfügbarkeit

Die Verfügbarkeit der Karte beziehungsweise einzelnen Tiles ist in der Basis-App nur durch das interne "Ambient Caching" [33] gewährleistet. Eine Funktion zum Offline Download von Kartenausschnitten ist momentan noch nicht umgesetzt. Anders sieht es mit Kartenelementen und Detailinformationen aus. Diese sind sowohl online als auch offline verfügbar.

5.9.2.5 Usability

Auch wenn die Basis-App nicht sonderlich viele Views beinhaltet, wurde sie so gut wie möglich nach den Apple Human Interface Guidelines [8] gearbeitet und viele iOS Standard-Komponenten verwendet.



a: UITabBar

b: UITableView

c: UISearchController

d: UINavigationController

Abbildung 5.23: Viele bekannte iOS Elemente finden sich in der Umsetzung wieder

Betreffend Usability und der Interaktion mit der Karte ging aus dem Review von Build 3 noch ein wichtiger Kritikpunkt hervor: Überlappen sich mehrere Kartenelemente, wird bei einem Touch unter Umständen ein hierarchisch tiefer liegendes Element ausgewählt. Dem Benutzer fällt es dadurch schwer, ein spezifisches Kartenelement zu selektieren und benötigt manchmal mehrere

Anläufe. Die Selektierung muss deshalb bei einem Touch entweder statisch oder hierarchisch priorisiert werden.

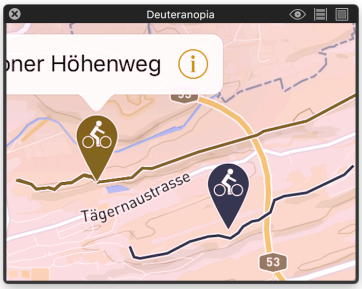
5.9.2.6 Accesibility

Gemäss dem Schweizerischen Zentralverein für das Blindenwesen [34] leben in der Schweiz 320'000 sehbehinderte und blinde Personen, was ungefähr 3% der Schweizer Bevölkerung ausmacht. Die Anzahl sehbehinderter Traildevils Benutzer ist unbekannt, trotzdem dürfen diese beeinträchtigten Mitmenschen nicht vernachlässigt werden. Die Basis-App kann diese handicapierten Benutzer leider noch nicht vollständig befriedigen, wie nachfolgende Tests zeigen.

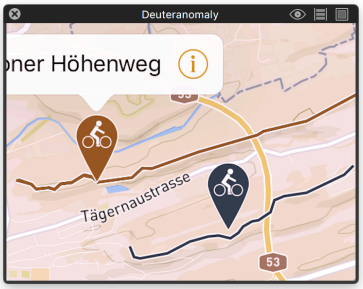
Abbildung 5.24: Ausgangslage: Ein selektierter und nicht-selektierter Trail mit Track

Die Tests wurden mit Sim Daltonism [35], ein Color Blindness Simulator für iOS und macOS, durchgeführt. Mit diesem Simulator lassen sich unterschiedliche Typen von Farbenblindheit visualisieren.

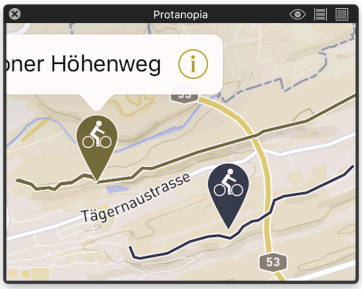
In den nachfolgenden Bildern ist unschwer zu erkennen, dass die momentane Farbwahl für selektierte Kartenelemente unglücklich gewählt wurde.



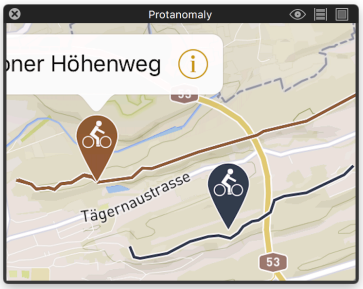
a: Deuteranopia



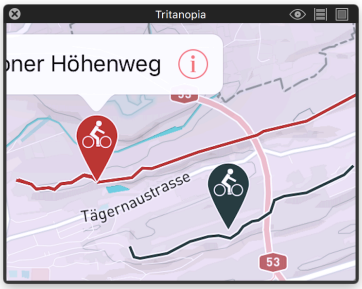
b: Deuteranomaly



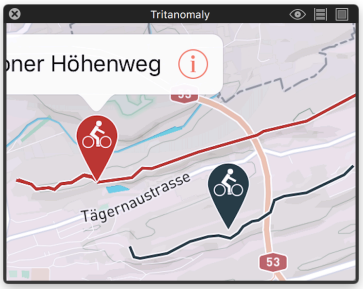
c: Protanopia



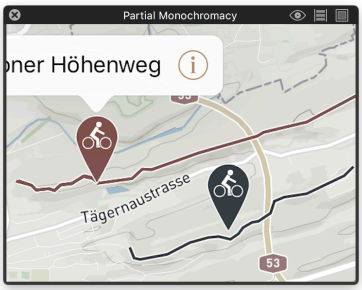
d: Protanomaly



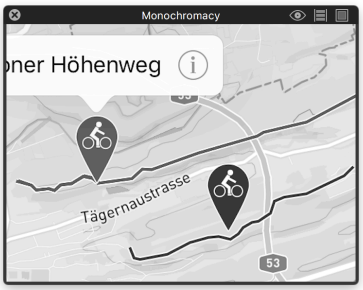
e: Tritanopia



f: Tritanomaly



g: Partial Monochromacy



h: Monochromacy

Abbildung 5.25: Selektion eines Trails unter verschiedenen Typen von Farbenblindheit

Kapitel 6

Zusammenfassung

6.1 Ergebnisse

Nach der Analysephase konnte in der Implementationsphase eine Basis-App umgesetzt und ein Grundstein für eine zukünftige Weiterentwicklung gelegt werden. Zu dem Funktionsumfang der iOS App gehören:

- Anzeigen der Kartenelemente auf dem Kartenmaterial von Mapbox
- Kartenelemente selektieren
- Kartenelemente suchen
- Geodaten suchen
- Detailinformationen zu einem Kartenelement
- Offizielle Traildevils API ist bereits in Betrieb
- Offline Caching der Daten von Traildevils
- Verfolgen des Benutzerstandortes

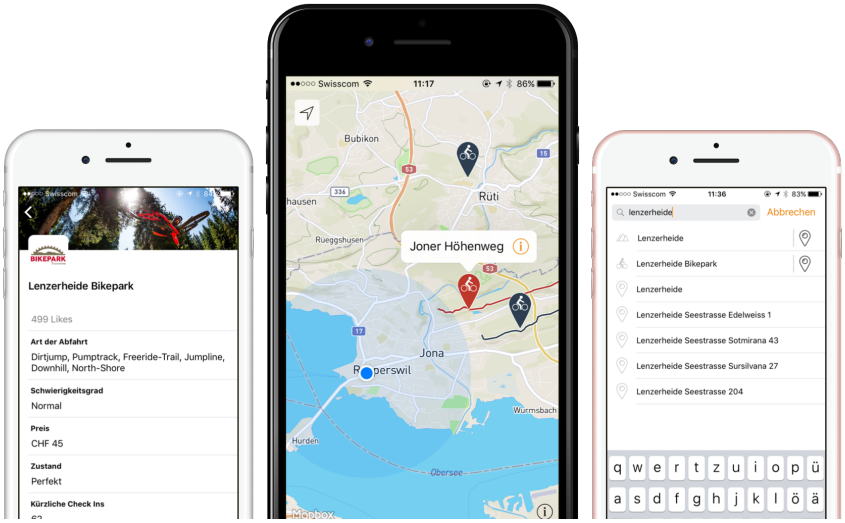


Abbildung 6.1: Traildevils iOS Mockups: Screenshots aus der finalen Applikation

Während der Umsetzung wurde ausschliesslich mit der Mapbox iOS SDK Version `v3.4.0` gearbeitet, welche sich immer noch in Entwicklung befindet. Es ist eindrücklich zu sehen, dass während dieser Zeit fünf Alpha und fünf Beta Versionen erschienen sind. Dabei darf mit Ehre gesagt werden, dass Erweiterungen des SDKs auch auf die Rechnung der Autoren gehen. Es wurden zwei Pull Request erstellt und erfolgreich in das SDK integriert.

Zusätzlich aus der Arbeit sind mehrere Nebenprodukte entstanden. Dazu gehört das Raclette Framework [36], welches mit Sicherheit auf Anklang in der Community stossen wird und vor allem in Bezug auf den Umgang mit einer `UITableView` eine grosse Erleichterung ist. Auch die beiden öffentlichen GitHub Repositories TwitterUIViewController [27] und die Mapbox Beispiele [11] werden bereits von einigen Entwicklern geschätzt. Dies machte sich einerseits durch Stars, Issues und Forks bemerkbar, andererseits auch durch Mails von mehreren Entwicklern über die geleistete Arbeit.

6.2 Ausblick

Nach dem erfolgreichen Prototyp kann mit gutem Gewissen gesagt werden, dass die Traildevils iOS App nach Wunsch erweitert werden kann. Die Autoren, wie auch der Projektpartner und der Betreuer, sind der Ansicht, dass eine native iOS App die Attraktivität der Traildevils Plattform steigern kann.

Der nächste Schritt aus der Sicht der Autoren ist die Umsetzung der Benutzeranmeldung und Benutzerregistrierung. So können Funktionen freigeschaltet werden, welche einen eingeloggten Benutzer voraussetzen, wie z.B. ein Trail Check-In, einen Zustand melden, ein Kartenelement bewerten und kommentieren, oder etwa einen Rider anzeigen. Zudem kann ein Aktivitätenstream gebaut werden, welcher bereits im Einsatz ist auf der Webseite. Je nach Aufenthalt eines Users, können so auch basierend auf dessen Standort, Events in Form von Push Notifications versendet werden, oder ganz einfach Suchresultate in einem Umkreis von 4 km direkt angezeigt werden.

Auch durch die aktive Weiterentwicklung von Mapbox gelangt beinahe täglich neue Funktionalität in das SDK. Dies kommt der Weiterentwicklung der Traildevils iOS App natürlich sehr entgegen.

Der Weiterentwicklung der Traildevils iOS App steht somit nichts im Wege.

Teil II

Projektdokumentation

Kapitel 7

Projektmanagement

7.1 Zeitliche Planung

Das Projekt wird im Rahmen der Studienarbeit durchgeführt. Dies hat zur Folge, dass während 14 Wochen an der iOS App gearbeitet wird. Endgültiger Termin für die Abgabe ist der 23. Dezember 2016 um 17:00 Uhr.

Der Aufwand pro Teammitglied beträgt insgesamt 240 Stunden. Somit entstehen ca. 17 Stunden Arbeitsaufwand pro Woche pro Teammitglied.

7.2 Zeitliche Auswertung

7.2.1 Teammitglied

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

Kalenderwoche	Roman Blum	Philipp Schilter	Total
KW38	23.5	14.5	38
KW39	24	23.5	47.5
KW40	27	15	41.75
KW41	16.5	18	34.5
KW42	29.25	20	49.25
KW43	16.5	20.5	37
KW44	28.75	20	48.75
KW45	25.5	8.25	33.75
KW46	6	7	13
KW47	10.5	10.75	21.25
KW48	29	25	54
KW49	26	28.5	54.5
KW50	25.5	27	52.5
KW51	19	19	38
Total	**307**	**256.75**	**563.75**

Tabelle 7.1: Zeit auf die einzelnen Teammitglieder aufgeteilt

Auswertung

Das zeitliche Tief in den Kalenderwochen 45 bis 47 ist auf die Abgabe eines Testats in einem anderen Modul zurückzuführen. Ebenfalls ist ersichtlich, dass Roman Blum 67h und Philipp Schilter 16.5h mehr als die erforderlichen 240h investiert haben.

7.2.2 Repositories

Die Arbeit wurde in unterschiedliche Repositories auf GitHub unterteilt. So erhielt die Vorstudie (Mapbox Examples), die API, das iOS App und die Dokumentation jeweils ein eigenes Repository.

Repository	Roman Blum	Philipp Schilter	Total
Mapbox Examples	68 (77%)	20.5 (23%)	88.5
API	2.5 (8%)	28.5 (92%)	31
iOS App	130.25 (64%)	73.5 (36%)	203.75
Dokumentation	106.5 (44%)	134 (56%)	240.5

Tabelle 7.2: Zeit auf die einzelnen Repositories aufgeteilt

Auswertung

Aus der Tabelle oben ist zu entnehmen, dass die iOS App, nach der Dokumentation, am meisten Zeit in Anspruch nahm. Zudem ist der prozentuale Anteil eines Teammitglieds pro Projekt sichtbar.

7.3 Phasen

Das Projekt ist grob in zwei Phasen aufgeteilt.

7.3.1 Analysephase

In einer Vorstudie wird mit Experimenten und einem Proof of Concept die Machbarkeit geprüft. Es soll sichergestellt werden, dass alle Anforderungen befriedigt werden können. Diese Anforderungen werden in Form von kleinen Beispielen validiert. Mit einem Proof of Concept soll mit realen Daten gearbeitet werden, damit die Entwicklung näher am schlussendlichen Produkt ist.

7.3.2 Entwicklungsphase

Nach der Absicherung wird in eine "Entwicklungsphase" übergegangen. Hier beginnt das klassische Software Projekt. Arbeitspakete werden vordefiniert, unter den Entwickler geschätzt und zusammen mit dem Projektpartner eingeplant. Wie die Entwicklung durchgeführt wird, ist im nächsten Unterkapitel

ersichtlich.

Auswertung

Lines of Code: 4409 Classes (*.swift Dateien): 108

7.4 Sprints

Für die Entwicklung wurde eine agile Vorgehensweise nach SCRUM festgelegt. Die Entwickler und der Projektpartner legen am Anfang des Sprints die zu entwickelnden Features fest, priorisieren und planen diese ein.

Anhand der Arbeitspakete werden Features erstellt, wobei ein Feature mehrere Tasks enthalten kann. Diese Tasks sind so definiert, dass sie jeweils von einem Gruppenmitglied erledigt werden können. Nach Abschluss des Tasks wird dieser dem anderen Teammitglied in Form eines Pull Requests zu einem Code-Review übergeben. So kann das Vier-Augen-Prinzip angewendet werden um die Qualität des Produktes hoch zu halten.

Die Dauer der Sprints dauern in der Regel zwei Wochen. Dies bietet dem Projektpartner die Möglichkeit zu intervenieren und seine Anforderungen pro Sprint anzubringen.

Die Sprint Reviews passieren jeweils montags um 13:10 Uhr. Dabei wird der aktuelle Stand der Arbeit besprochen. Noch nicht erledigte Tasks kommen in den Backlog und es wird neu priorisiert.

Kapitel 8

Milestones

8.1 MS01: Review des Prototypen

Geplant 03.10.2016
Erreicht

Inhalt

- Kennenlernen der Parteien
- Projektplan und Prototyp

8.2 MS02: Analyse nachführen, Strategie für Offline und Synchronisation

Geplant: 10.10.2016
Erreicht

Inhalt

- Analyse nachführen mit Auswertung der Experimente
- Vorschlag: Offline-Strategien
- Vorschlag: Synchronisation-Strategien

8.3 MS03: Migration des POC in effektives Repo für das iOS App

Geplant: 17.10.2016
Erreicht

Inhalt

- Migration des POC in separates Repo als Grundlage für iOS APP
- API Spezifikation

8.4 MS04: API und iOS Architektur

Geplant: 24.10.2016
Erreicht

Inhalt

- Evaluation und Setup Architektur
- Setup API Mock als Node.js Server auf heroku

8.5 MS05: Map Interaktion und Data Handling

Geplant: 07.11.2016
Erreicht

Inhalt

- Suche nach Traildevils-Kartenelementen
- Speichern der empfangenen Daten in der lokalen DB (caching)
- Use Case: Kartenelemente selektieren
- Use Case: Kartenelemente suchen
- TestFlight Release v1.0, Build 1

8.6 MS06: Detail Views und Authentication/Authorization

Geplant: 21.11.2016
Nicht Erreicht

Inhalt

- Use Case: Geodaten suchen
- Use Case: Detailinformationen zu einem Kartenelement anzeigen
- Use Case: Benutzer anmelden
- Use Case: Benutzer registrieren
- TestFlight Release v1.0, Build 2

Bemerkung

Die aufgetretenen Probleme in Bezug auf die Detailansichten und der `UITableView` verzögerten den Abschluss dieses Milestones. Aus diesem Grund konnten die Use Cases "Benutzer anmelden" und "Benutzer registrieren" nicht zeitgerecht umgesetzt werden.

Die genaue Problematik der `UITableView` wird im Kapitel 5.5.3.11 erläutert.

Während des Sprints traf das Feedback zum Release v1.0 ein. Die gewünschten Änderungen wurden gleich in den Sprint integriert, welches ebenfalls Einfluss auf die Verzögerung hatte.

8.7 MS07: App Flow Polishing

Geplant: 05.12.2016
Erreicht

Inhalt

- Verarbeitung des ersten TestFlight Release

Bemerkung

Durch die genannte Verzögerung im MS06, wurde gemeinsam mit dem Projektpartner entschlossen, dass keine weitere Funktionalität in die Applikation integriert wird. Stattdessen soll der Fokus auf die Optimierung und Lauffähigkeit der bestehenden Features gelegt werden.

8.8 MS08: Finish App

Geplant: 12.12.2016
Erreicht

Inhalt

- Umstellung auf offizielle Traildevils API
- Kleinere Bugfixes

8.9 MS09: Hand-in Documentation

Geplant: 23.12.2016
Erreicht

Inhalt

- Abgabe der Dokumentation

Kapitel 9

Risikomanagement

Zu Beginn jedes Sprints wurden die Risiken für die aktuelle Iteration ermittelt. In den folgenden Abschnitten werden die Risiken erläutert und ausgewertet.

9.1 Risiken

In der folgenden Tabelle werden die Risiken beschrieben und gewichtet. Die Indikatoren der Gewichtung liegen zwischen 1 und 5, wobei 1 die niedrigste Gewichtung und 5 die grösste Gewichtung ist.

ID	Beschreibung	Gewicht
R01	Clustering kann nicht erwartungsgemäss umgesetzt werden	5
R02	Semantic Zooming kann nicht erwartungsgemäss umgesetzt werden	5
R03	Panning kann nicht erwartungsgemäss umgesetzt werden	5
R04	Marker Selection Feedback kann nicht erwartungsgemäss umgesetzt werden	5
R05	Filtering kann nicht erwartungsgemäss umgesetzt werden	5
R06	Path Selection kann nicht erwartungsgemäss umgesetzt werden	3
R07	Icons auf Stylelayer z.B. für Kartenelemente	3
R08	Ausgewählte Frameworks erfüllen die Anforderungen nicht	3

Tabelle 9.1: Risikotabelle

Multipliziert man die Gewichtung mit der Eintrittswahrscheinlichkeit (0% bis 100%), entsteht der schlussendliche Risikowert von 0 bis maximal 500. In der folgenden Grafik ist der Verlauf der Risiken über die Zeit hinweg ersichtlich. Wurde ein Risiko eliminiert, ist dessen Risikowert 0.

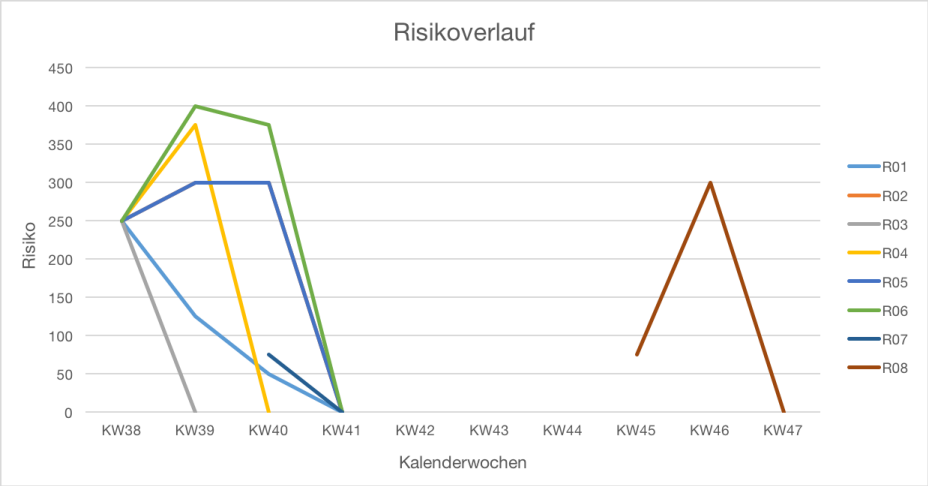


Abbildung 9.1: Verlauf der Risiken über das Projekt

Auswertung

Aus der Grafik oben ist zu entnehmen, dass sich während der Vorstudie die meisten Risiken zeigten. Diese Risiken waren entscheidend für den weiteren Verlauf der Arbeit. Dank Workarounds und den mehreren Alpha- und Betaversion des Mapbox SDKs konnten die Risiken R01 bis R07 während den ersten vier Wochen eliminiert werden. Es ist zudem ersichtlich, dass in KW45 ein weiteres Risiko R08 ermittelt wurde. Durch die Implementation der Detailansichten, musste ein geeignetes Framework evaluiert werden. Leider erfüllte kein Framework die gewünschten Anforderungen (Anforderungen im Kapitel 5.5.3.11 erläutert). Durch die Entwicklung des eignen Frameworks "Raclette" konnte jedoch auch dieses Risiko innerhalb des MS06 eliminiert werden. In den Kalenderwochen bis KW51 sind keine Risiken mehr ermittelt worden und der Projektverlauf brachte keine Schwierigkeiten mehr mit sich.

9.2 Umgang mit Risiken

Das grösste Risiko besteht darin, dass Mapbox die gewünschten Anforderungen nicht erfüllen kann. Darum wird sehr grossen Wert auf das Prototyping gesetzt. Mit konkreten Experimenten wird die Machbarkeit überprüft um so erfolgreich durch die Entwicklungsphase zu gelangen. Zudem werden neue Releases des Mapbox SDKs sofort in die iOS App integriert. Durch die neuen Features kann anschliessend die Funktionalität überprüft und Risiken minimiert oder sogar eliminiert werden.

Kapitel 10

Entwicklungsumgebung

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

Beide Studenten verwenden für die Umsetzung ein MacBook Pro Retina mit macOS Sierra. Als Testgeräte kommen nebst den in Xcode verfügbaren Simulatoren auch physische Geräte zum Einsatz, darunter ein iPhone 6s, ein iPhone 7, ein iPad Mini 2 und ein iPad Pro 9.7“. Auf den physischen Geräten wird grundsätzlich immer mit der neusten iOS Version (ab 10.0.x) getestet. Dank den Simulatoren können aber auch ältere iOS Versionen (bis zu Version 8.0) getestet werden.

10.1 Kosten

Für die Umsetzung der iOS App fallen lediglich die Kosten für die Apple Developer Lizenz an, welche zwingend für Herausgeben von Apps im App Store ist. Mit einem Preis von $110 pro Jahr fällt dieser Preis aber relativ niedrig aus, bedenkt man, dass jegliche Services wie zum Beispiel iCloud, Game Center, Apple Pay, Home Kit, Push Notifications, usw. dann kostenlos genutzt und in der App verwendet werden können.

10.2 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.

10.3 Software

10.3.1 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 App Store heruntergeladen werden.

10.3.2 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 3.0 die aktuellste Version.

10.3.3 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.

10.3.4 GitHub

GitHub wird in erster Linie als zentrales Code Repository verwendet. Zusätzlich planen wir die Milestones und Issues ebenfalls auf der Platform. Mittlerweile besitzen die Repositories sogar einen Projekt-Tab, indem man ein individualisierbares Kanban Board antrifft.

10.3.5 Everhour

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

10.3.6 Travis CI

Continuous Integration wird mit Hilfe des Services von Travis CI eingesetzt. So wird z.B. 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.

Kapitel 11

Richtlinien

Mithilfe von Richtlinien soll eine gewisse Konsistenz erreicht werden.

11.1 Code

Basierend auf dem Swift Style Guide von GitHub [37] wird mithilfe des Tools [38] bei jedem Build geprüft, ob die definierten Code Richtlinien eingehalten wurden. Dabei werden die standardmässigen Einstellungen von SwiftLint übernommen. Folgende Ausnahmen sind in der Datei `.swiftlint.yml`

```
1 disabled_rules: # rule identifiers to exclude from running
2   - trailing_newline
3
4 excluded: # paths to ignore during linting. Takes
      ↪ precedence over `included`.
5   - Pods
6
7 /# rules that have both warning and error levels, can set
      ↪ just the warning level
8 /# implicitly
9 line_length: 200
```

11.2 Design

Die Traildevils iOS App haltet sich prinzipiell an die von Apple vorgeschriebenen iOS Human Interface Guidelines ([8]). Selbstverständlich werden gewisse Farbakzente mit den beiden “Traildevils”-Farben Orange und Braun gesetzt.

11.3 Testing

Tests werden mithilfe von Travis CI automatisch bei jedem Push und Pull Request durchgeführt. Dabei wird das Repository in folgenden Umgebungen/Simulatoren getestet:

- iPhone 7, iOS 10
- iPhone 6, iOS 10

Kapitel 12

Qualitätsmassnahmen

12.1 Entwicklung

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

12.2 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 z.B. 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 [39] 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.

12.3 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.

12.4 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.

12.5 Testing

Unit-Tests werden stets 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.

12.6 Code Style Guide

Um einen einheitlichen Code Style zu gewährleisten, basiert der geschriebene Code auf dem Style Guide von Ray Wenderlich [40].

Teil III

Anhang

Aufgabenstellung Studienarbeit Abteilung I, HS 2016/17 Roman Blum, Philipp Schilter

Traildevils iOS App

1. Betreuer & Praxispartner

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

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

2. Ausgangslage

Ein zentrales Produkt der Frontline Media GmbH ist die Traildevils Website, eine Online Community Site für Mountainbikers. Die Traildevils Website ist schon heute mit dem Webbrowser von Mobilgeräten aus zugreifbar, aber der Auftraggeber meint, dass sich mit einer App die Attraktivität des Angebots an Mountain-Biker noch verbessern lässt und damit die Attraktivität der Community gesteigert werden kann. Im FS16 wurde aus diesem Grund eine Bachelorarbeit durchgeführt mit dem Ziel eine Traildevils Cross Platform App zu entwickeln. Als Implementationsframework für die Cross Platform Entwicklung wurde React Native genutzt. Im Verlaufe des Projektes zeigte sich, dass für die Traildevils App keine Alternative zur Verwendung von Mapbox gibt, wenn es um die Darstellung von Informationen auf Karten geht. Nur über Mapbox lassen sich Karten beziehen, die die für Mountainbiking wichtigen Informationen enthalten. Leider zeigte sich auch, dass bei Mapbox die Implementation der Cross Platform Library hinter der nativen Implementation hinterherhinkt, und somit grundlegende Operationen wie Semantic Zoom und Selektion von Punkten und Pfaden auf der Karte nicht möglich waren. Weitere Experimente zeigten, dass auch die native Implementation von Mapbox für Android noch nicht alle nötigen Funktionen hat. Weitere Analysen zeigten, dass die Entwicklung auf iOS meist weiter fortgeschritten ist, als die auf Android.

3. Ziele der Arbeit

In dieser Studienarbeit soll eine native iOS Traildevils App entwickelt werden. In einer ersten Phase soll analysiert werden, ob (bzw. wie) unter Nutzung des Mapbox APIs die wichtigen Funktionen Semantic Zoom, Selektion von Punkten und Pfaden, sowie Panning und Filtering auf der Karte so umgesetzt werden können, dass eine flüssige Interaktion mit den Karteninformationen möglich wird.

Nach dieser ersten Machbarkeitsstudie soll dann, aufbauend auf den in der vorherigen Bachelorarbeit entwickelten Szenarien und UI-Konzepten, ein Plan für die Umsetzung einer minimalen App entwickelt werden. Das UI-Konzept für die Suche muss hierbei möglicherweise erweitert werden. Zudem ist zu entscheiden, ob Offline-Fähigkeit (und damit Synchronisation) von Daten im Scope einer solchen minimalen App sein soll. Das Gleiche gilt für Notifikationen. Die Anforderungen für diese minimale App und die Diskussion/Priorisierung der umsetzbaren Features mit dem Praxispartner ist eine wichtige Aufgabe für die Studierenden.

Am Ende dieser Arbeit soll ein Prototyp einer Traildevils App entstehen. Dieser Prototyp soll helfen alle technischen und Nutzer-bezogenen Risiken zu eliminieren, bzw. zu bestimmen. Im Idealfall entsteht in dieser Arbeit eine Traildevils App die anschliessend vom Praxispartner zur Publikation im Apple Appstore eingereicht werden kann.

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 ist vollständig elektronisch abzugeben (Exemplar für das Sekretariat Informatik), sowie ein Download-Link für Prof. Stolze und weitere Exemplare nach Absprache mit dem Co-Referenten.

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

5. Weitere Regeln und Termine

Im Weiteren gelten die allgemeinen Regeln zu Bachelor und Studienarbeiten
«Abläufe und Regelungen Studien- und Bachelorarbeiten im Studiengang Informatik» (Skripte Server)

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 Quellcode 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 Quellcode und im Impressum der Anwendung die originale Urheberschaft durch die HSR Studenten weiterhin sichtbar bleibt.
Der Bericht der Studienarbeit (ohne geheime Anhänge) wird von der HSR im E-Prints Respository der HSR (eprints.hsr.ch) elektronisch veröffentlicht. Titel, Abstract und Praxispartner der Arbeit dürfen von der HSR und Studierenden schon während der Arbeit kommuniziert werden.

7. Beurteilung

Eine erfolgreiche Studienarbeit zählt 8 ECTS-Punkte pro Studierenden. Für 1 ECTS Punkt ist eine Arbeitsleistung von ca. 25 bis 30 Stunden budgetiert. Entsprechend sollten ca. 240h Arbeit für die Bachelorarbeit aufgewendet werden. Dies entspricht ungefähr 17h pro Woche (auf 14 Wochen) und damit ca. 2 Tage Arbeit pro Woche pro Student.

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

Die definitive Aufgabenstellung wurde am 24.10.16 beschlossen.

Rapperswil, 24.10.16
Prof. Dr. Markus Stolze, Institut für Software, Hochschule für Technik Rapperswil

Glossar

API	**A**pplication **P**rogramming **I**nterface
JSON	**J**ava**S**cript **O**bject **N**otation
POI	**P**oints **O**f **I**nterest
Framework	Programmierbibliothek
Linter	Tool zur statischen Code-Analyse

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 (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Rapperswil-Jona, 23. Dezember 2016

Roman Blum

Philipp Schilter

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

1. Margarita S. Brose, Mark D. Flood, Dilip Krishna, Bill Nichols. Handbook of financial data and risk information ii: Volume 2: Software and data. Cambridge: Cambridge University Press; 2014.

2. Howard Butler, Martin Daly, Allan Doyle, Sean Gillies, Stefan Hagen, Tim Schaub, et al. The geojson format specification [Internet]. 2008 [cited 2016 Sep 30]. Available from: http://geojson.org/geojson-spec.html

3. Howard Butler, Martin Daly, Allan Doyle, Sean Gillies, Stefan Hagen, Tim Schaub. RFC 7956: The geojson format [Internet]. 2016 [cited 2016 Sep 30]. Available from: http://geojson.org/geojson-spec.html

4. Mapbox. MGLAnnotation protocol reference [Internet]. 2016 [cited 2016 Oct 20]. Available from: https://www.mapbox.com/ios-sdk/api/3.3.7/Protocols/MGLAnnotation.html

5. SwiftyJSON. The better way to deal with json data in swift. [Internet]. 2016 [cited 2016 Oct 10]. Available from: https://github.com/SwiftyJSON/SwiftyJSON

6. Mapbox. Pricing [Internet]. 2016 [cited 2016 Dec 11]. Available from: https://www.mapbox.com/geocoding

7. Mapbox. Mapbox pricing [Internet]. 2016 [cited 2016 Oct 21]. Available from: https://www.mapbox.com/pricing/

8. Apple. IOS human interface guidelines [Internet]. 2016 [cited 2016 Oct 23]. Available from: https://developer.apple.com/ios/human-interface-guidelines/

9. Apple. Cocoa core competencies: Model-view-controller [Internet]. 2015 [cited 2016 Oct 23]. Available from: https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

10. Apple. UIViewController class [Internet]. 2016 [cited 2016 Dec 16]. Available from: https://developer.apple.com/reference/uikit/uiviewcontroller

11. Blum R. Mapbox examples using swift 3 and mapbox iOS sdk [Internet]. 2016 [cited

2016 Oct 4]. Available from: https://github.com/rmnblm/mapbox-ios-examples

12. Rehkemper G. USA mcdonalds locations [Internet]. 2016 [cited 2016 Oct 17]. Available from: https://github.com/gavinr/usa-mcdonalds-locations

13. Mapbox. Amsterdam geojson [Internet]. 2016 [cited 2016 Oct 17]. Available from: https://github.com/mapbox/mapbox-gl-native/tree/9f0584df60efcadd6cd2c12d7eccce206f18e086/platform/ios/test

14. Mapbox. Draw a polyline by parsing a geojson file [Internet]. 2016 [cited 2016 Oct 17]. Available from: https://www.mapbox.com/ios-sdk/examples/line-geojson/

15. RethinkDB. Silicon valley poi geojson [Internet]. 2016 [cited 2016 Oct 17]. Available from: https://github.com/rethinkdb/geojson-streetmaps

16. Mapbox. Mapbox github pull request 'geojson point clustering' [Internet]. 2016 [cited 2016 Oct 4]. Available from: https://github.com/mapbox/mapbox-gl-native/pull/5724

17. Mapbox. Mapbox github pull request '[ios, macos] fixes #5962 added geojson options to support clustering' [Internet]. 2016 [cited 2016 Oct 4]. Available from: https://github.com/mapbox/mapbox-gl-native/pull/6217

18. Ricardo Jota P Albert Ng, Daniel Wigdor. How fast is fast enough? A study of the effects of latency in direct-touch pointing tasks. ACM New York, NY, USA ©2013; 2013.

19. Mapbox. Mapbox api function -visibleFeaturesInRect:inStyleLayersWithIdentifiers: [Internet]. 2016 [cited 2016 Oct 21]. Available from: https://www.mapbox.com/ios-sdk/api/3.4.0-alpha.4/Classes/MGLMapView.html#/c:objc(cs)MGLMapView(im)visibleFeaturesInRect:inStyleLayersWithIdentifiers:

20. Standard representation of geographic point location by coordinates. Vol. 2008. Geneva, CH: International Organization for Standardization; 2008.

21. Howard Butler, Martin Daly, Allan Doyle, Sean Gillies, Stefan Hagen, Tim Schaub, et al. The geojson format specification [Internet]. 2008 [cited 2016 Dec 9]. Available from: http://geojson.org/geojson-spec.html#positions

22. Apple. IOS human interface guidelines [Internet]. 2016 [cited 2016 Dec 13]. Available from: https://developer.apple.com/ios/human-interface-guidelines/visual-design/color/

23. FlatUIColors. Flat ui colors - color pallette from flat ui theme [Internet]. 2016 [cited 2016 Dec 13]. Available from: http://flatuicolors.com/

24. Smashicons. Smashicons | world's largest icon set for designers and developers [Internet]. 2016 [cited 2016 Dec 13]. Available from: http://smashicons.com/

25. ariok. A twitter-inspired uiviewcontroller written entirely in swift [Internet]. 2016 [cited

2016 Dec 13]. Available from: https://github.com/ariok/TB_TwitterUI

26. deanbrindley87. Twitter ui demo for blog post [Internet]. 2016 [cited 2016 Dec 13]. Available from: https://github.com/deanbrindley87/Twitter-UI

27. Blum R. A twitter-inspired uiviewcontroller written entirely in swift [Internet]. 2016 [cited 2016 Dec 13]. Available from: https://github.com/rmnblm/TwitterUIViewController

28. Google. Empty states - patterns - material design guidelines [Internet]. 2016 [cited 2016 Dec 17]. Available from: https://material.io/guidelines/patterns/empty-states.html

29. Apple. Framework definition [Internet]. 2016 [cited 2016 Dec 17]. Available from: https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/Framework.html

30. Apple. Swift package manager readme [Internet]. 2016 [cited 2016 Dec 17]. Available from: https://github.com/apple/swift-package-manager/blob/master/README.md

31. Software R. Should you use storyboards on more complex apps? [Internet]. 2016 [cited 2016 Dec 13]. Available from: http://roadfiresoftware.com/2015/03/the-pros-and-cons-of-using-storyboards/

32. Dina Chaiffetz I. Why empty states deserve more design time [Internet]. 2016 [cited 2016 Dec 13]. Available from: http://blog.invisionapp.com/why-empty-states-deserve-more-design-time/

33. Mapbox. Mapbox ambient caching [Internet]. 2016 [cited 2016 Oct 22]. Available from: https://www.mapbox.com/help/mobile-offline/#ambient-caching

34. Blindenwesen SZ für das. SZG webseite [Internet]. 2016 [cited 2016 Oct 22]. Available from: http://www.szb.ch/

35. Fortin M. Sim daltonism [Internet]. 2016 [cited 2016 Oct 22]. Available from: https://michelf.ca/projects/sim-daltonism/

36. Blum R. Let's get cheesy with raclette, a uitableview extension to add rows and sections on-the-fly. [Internet]. 2016 [cited 2016 Dec 6]. Available from: https://github.com/rmnblm/Raclette

37. GitHub. Style guide & coding conventions for swift projects [Internet]. 2016 [cited 2016 Oct 23]. Available from: https://github.com/github/swift-style-guide

38. Realm. A tool to enforce swift style and conventions. [Internet]. 2016 [cited 2016 Oct 23]. Available from: https://github.com/realm/SwiftLint

39. Beams C. How to write a git commit message [Internet]. 2014 [cited 2016 Sep 28]. Available from: http://chris.beams.io/posts/git-commit/

40. Wenderlich R. The official raywenderlich.com swift style guide [Internet]. 2016 [cited

2016 Oct 21]. Available from: https://github.com/raywenderlich/swift-style-guide