

Projektarbeit 2, Master of Science in Engineering, Major Software and
Systems

Waldmeister - Outdoors

Ein Werkzeug zur Erforschung und Editieren von Waldstandorten

HSR Hochschule für Technik Rapperswil

Frühling 2018

Autor: Daniel Schmider

Betreuer: Prof. Stefan Keller

Kapitel 1

Abstract

"Waldmeister - Outdoors" ist eine Applikation, welche es ermöglicht, Waldstandorte in der Schweiz zu erforschen und zu erfassen. Die Applikation beschleunigt die digitale Erfassung und Publikation von Waldstandorten und erleichtert Experten die Arbeit im Feld. Über die Webapp können sich Benutzer Registrieren, um öffentliche, bzw. private Benutzerflächen von mehreren Geräten aus zu erstellen, speichern und zu teilen. Diese Benutzerflächen können Waldstandorte, aber auch zusätzliche Informationen zu einem Standort beinhalten, welche bei der Feldarbeit benötigt werden. Mithilfe der Geolocation wird die eigene Position bestimmt. Umliegende, bereits erfasste Waldstandorte und Benutzerflächen werden auf verschiedenen Ebenen auf einer Leaflet-basierten Karte in Echtzeit angezeigt.

Keywords: Vue.JS, Django, Rest Framework, Leaflet, Leaflet editable, Geolocation, Progressive Webapp, GIS

Inhaltsverzeichnis

1 Abstract	1
2 Management Summary	5
1 Problemstellung	5
2 Ziel der Arbeit	5
3 Ergebnisse	6
4 Ausblick	7
3 Teil 1 - Technischer Bericht	9
3.1 Einführung	9
3.1.1 Problemstellung, Vision	9
3.1.2 Ziele und Unterziele	9
3.1.3 Rahmenbedingungen	10
3.1.4 Vorgehen, Aufbau der Arbeit	10
3.2 Stand der Technik	11
3.2.1 GIS-Browser	11
3.2.2 Collector for ArcGIS	11
3.2.3 Defizite	12
3.3 Bewertung	13
3.3.1 Kriterien	13
3.3.2 Schlussfolgerungen	13
3.4 Umsetzungskonzept	14
3.4.1 Lösungsansätze	14
3.4.2 Frontend	14
3.4.3 Backend	14
3.4.4 Datenbank	14
3.4.5 JavaScript Libraries und Node Module	14
3.4.6 Python Pakete	15
3.4.7 Werkzeuge und Tools	15
3.4.8 Deployment	15
3.5 Resultate	15
3.5.1 Zielerreichung	15
3.5.2 Ausblick	16

3.5.3 Persönliche Berichte	16
4 Teil 2 - SW-Projektdokumentation	17
4.1 Anforderungsspezifikation	17
4.1.1 Use-Cases	17
4.1.2 Use Case - Diagramm	17
4.1.3 Must-Haves	19
4.1.4 Optional	19
4.1.5 Nicht-funktionale Anforderungen	19
4.2 Domain Modell	20
4.3 Technologien	20
4.3.1 Django	20
4.3.2 PostgreSQL	21
4.3.3 VueJS	21
4.3.4 Docker	26
4.4 Design	27
4.4.1 Architektur	27
4.4.2 Package Struktur	27
4.4.3 Sequenz-Diagramm	27
4.4.4 Komponentendiagramm	30
4.4.5 UI Design	30
4.5 Implementation	36
4.5.1 Vue Komponenten	36
4.5.2 Frontend Design	38
4.5.3 Geolocation	40
4.5.4 API	41
4.5.5 Database	43
4.5.6 Kartenmaterial	44
4.6 Tests	45
4.6.1 Tests in Django	45
4.6.2 Tests in Karma	45
4.6.3 Linter	45
4.6.4 Continuous Integration	46
4.7 Manuelle Tests	49
4.7.1 User-Szenario	49
4.7.2 Testfälle	50
4.7.3 Reproduktion und Auswertung	50
4.8 Resultate und Weiterentwicklung	51
4.8.1 Resultate	51
4.8.2 Möglichkeiten der Weiterentwicklung	51
4.9 Projektmanagement	53
4.9.1 Entwicklungswerkzeuge	53
4.9.2 Prozessmodell	53

4.9.3	Zeitplan	54
4.9.4	Meilensteinplanung	54
4.9.5	Issues	57
4.9.6	Releases	57
4.9.7	Risiken	57
4.9.8	Zusammenfassung Risiken	59
4.10	Projektmonitoring	60
4.10.1	Soll-Ist-Zeitvergleich	60
4.10.2	Code Statistics	60
4.11	Softwaredokumentation	60
4.11.1	Installation	60
4.11.2	Tutorial, Handbuch	60

Kapitel 2

Management Summary

1 Problemstellung

Je nach Untergrund, Bodeneigenschaften, Gelände sowie Klima gedeihen in der Schweiz unterschiedliche Typen von Wäldern. Seit einigen Jahrzehnten werden diese Typen von Experten erhoben und kartiert. Es wurden dabei verschiedene typisierte Waldstandorte festgelegt. Aktuell werden Karten, die im Auftrag der Kantone von Experten angefertigt wurden, nur in grossen Intervallen revidiert, wobei sie oft auch nicht flächendeckend vorhanden sind (z.B. in den Kantonen GR, VS, BE). Einer der Gründe dafür sind u.a. die hohen Kosten, die eine Analyse im Feld mit sich bringt. Zudem ist die Erfassung und Nachführung der Karten geprägt von analogen Vorgängen, da die vorhandenen technischen Geräte und Programme für den Einsatz im Feld ungeeignet sind. Daher muss von Hand Niedergeschriebenes im Büro oder von staatlichen Institutionen digitalisiert werden, bevor es an den Arbeitgeber geschickt werden und später auf kantonal isolierten Plattformen publiziert werden kann.

2 Ziel der Arbeit

Die Erfassung und Publikation von Waldstandorten sollte vereinfacht und beschleunigt werden. Dabei sollen digitale Technologien eingesetzt werden wie Smartphone, GPS und Internet. Diese neuen Instrumente sollen entsprechend geschulten Nutzern die Erfassung von Waldstandorten ermöglichen sowie öffentliche und private Informationen in Form von Flächen und Punkten. Auf einer Basis - Karte wird mittels GPS die eigene Position angezeigt. Darüber werden umliegende, bereits erfasste Waldstandorte, öffentliche Flächen anderer sowie die eigenen, privaten Flächen dargestellt. Diese Flächen können Waldstandorte beschreiben oder aber zusätzliche Informationen über den Standort beinhalten, z.B. eine speziell gekennzeichnete Beobachtungsfläche.

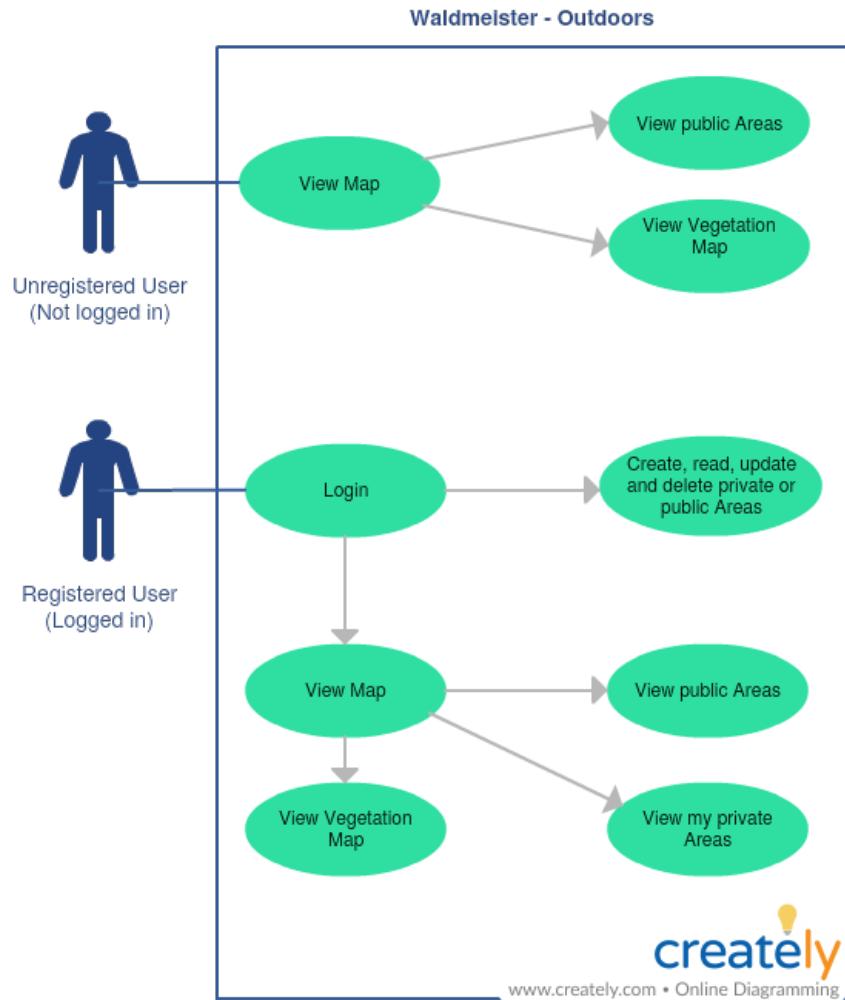


Abbildung 2.1: Use Cases

3 Ergebnisse

Nach einer Evaluation eines Prototyps, erstellt mithilfe eines kommerziellen Produkts, und der Erstellung von Mockups, wurde ein eigenes Webapp 'Waldmeister Outdoors' realisiert. Durch diese App kann die Arbeit der Experten erleichtert werden. Da die Waldstandort-Karte gleichzeitig im Web synchronisiert ist, wird darüber hinaus der Informationsaustausch unter allen Beteiligten erleichtert. Die Webapp wurde für mobile Geräte optimiert und die gewünschten Funktionen wurden umgesetzt. Registrierte Benutzer können Benutzerflächen in Form von Polygonen direkt auf der angezeigten Map erstellen, mit zusätzlichen Informationen versehen und auf einem Server speichern.

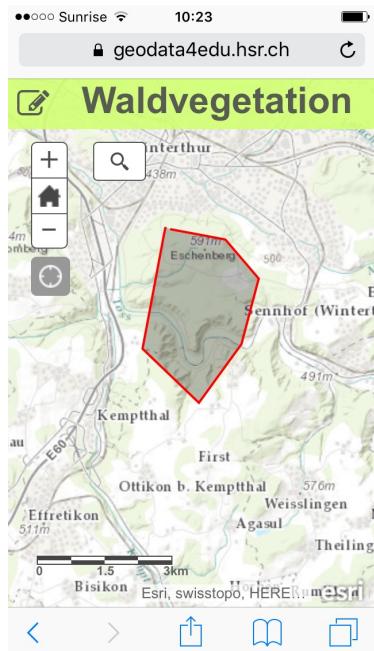


Abbildung 2.2: ESRI Webapp Collector for ArcGIS

4 Ausblick

Grosse Teile der Schweiz sind noch unkartiert, und viele Waldstandorte könnten sich unter dem Einfluss der Klimaerwärmung verändern. Die kontinuierliche Beobachtung solcher Standorte ist Forschungsgegenstand von Forstingenieuren. Die Arbeit im Feld ist unerlässlich. "Waldmeister - Outdoors" kann im Berufsalltag sowie bei der Kommunikation mit Institutionen den Arbeitsfluss beschleunigen. Weitere Features können dazukommen, wie die Verwendung von Plus Codes und offline-Fähigkeiten welche bei Verbindungsproblemen zum Einsatz kommen, bzw. Benutzerflächen automatisch synchronisieren, sobald eine Verbindung besteht. Des weiteren bietet es sich an, dass sich User in Gruppen einklinken können, um unter sich Benutzerflächen zu teilen und zu besprechen, bevor sie veröffentlicht werden. Ebenfalls sollten erstellte Flächen von registrierten Benutzern und deren Gruppen verändert und gelöscht werden können, nachdem sie erstellt wurden.

"Waldmeister - Outdoors" hat das Potential in der Schweiz ein verbreitetes Tool zur Kartierung und Beobachtung von Waldflächen zu werden und stellt eine bereits gefragte Erweiterung der beliebten "Waldmeister" App für mobile Geräte dar.

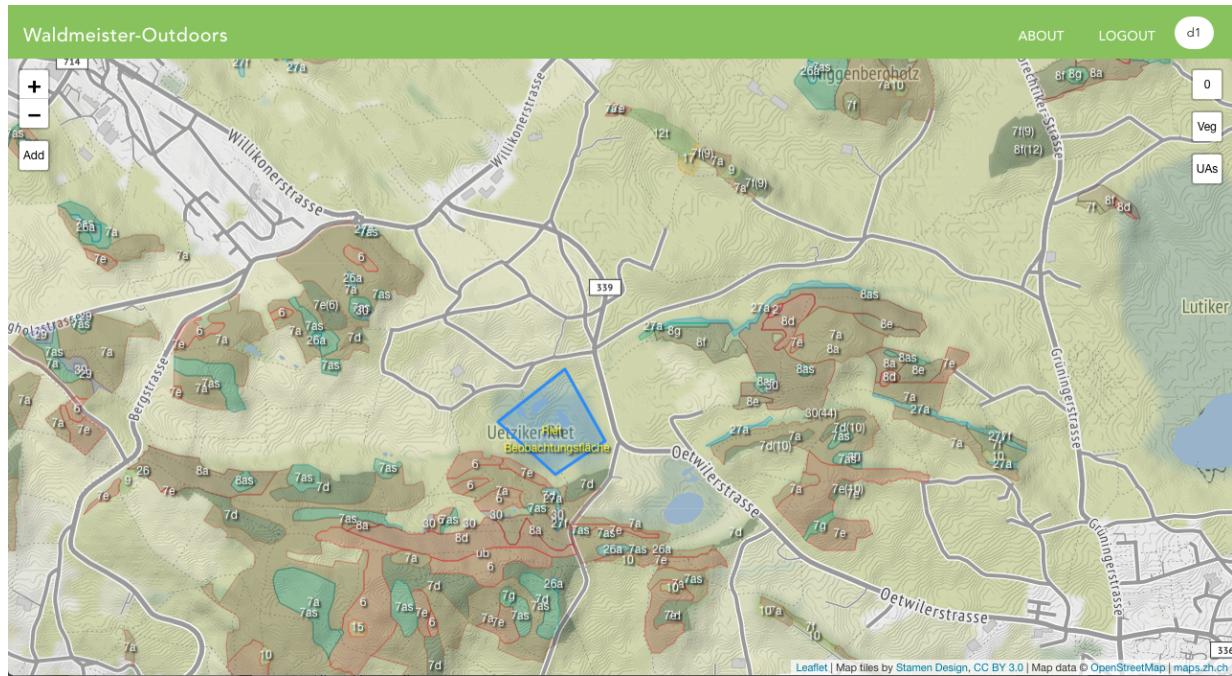


Abbildung 2.3: Erfassung einer neuen Benutzerfläche

Kapitel 3

Teil 1 - Technischer Bericht

3.1 Einführung

3.1.1 Problemstellung, Vision

Arbeit im Feld, wie sie bei der Kartierung von Waldstandorten unerlässlich ist, wird in allen Fällen komplementiert von Arbeit, welche ausschliesslich im Büro erledigt werden kann. Für viele Programme existieren ausschliesslich Desktop-Lösungen, welche auf mobilen Geräten nicht verwendet werden können. Wie im Management Summary beschrieben, müssen oft Arbeitsschritte im Büro wiederholt oder von Hand Niedergeschriebenes am Computer digitalisiert werden. Könnten diese Informationen bereits während der Feldarbeit digital festgehalten werden, würde dies die Arbeit sehr verkürzen und den Informationsaustausch zwischen verschiedenen Personen und Teams einfacher gestalten.

Können sich Personen und Teams untereinander bereits während der Arbeit im Feld über Standorte austauschen, welche es zu kartieren gilt, können sie sich freier und schneller im Feld organisieren und orientieren und so grossflächige Projekte schneller bearbeiten.

Kommt es zu einem späteren Zeitpunkt zu einer Revision von bereits kartierten Flächen, kann bestehendes Kartenmaterial den Experten zur Verfügung gestellt werden, welches sie auf mobilen Geräten gleich im Feld analysieren und gegebenenfalls bearbeiten können. Dies steht in einem starken Kontrast zum bestehenden Arbeitsfluss, welcher auf analogen Medien beruht.

3.1.2 Ziele und Unterziele

Ziel ist es, eine Webapp zu gestalten, welche es dem User erlaubt, bestehende Waldstandorte zu erforschen und nach Belieben neue Flächen, welche Waldstandorte beschreiben können, zu erfassen. Dies soll auf einem mobilen Gerät möglich sein, welches im Feld verwendet wird. Die so erhobenen Daten sollen auf einem zentralen Server gespeichert und sofort dem User selbst sowie anderen Benutzern wieder zur Verfügung gestellt werden. Daten, welche so im Feld eingegeben wurden, können auch im Büro weiterbearbeitet werden. Um dies zu realisieren, wird eine Webapp mit einer Map erstellt, welche je nach User unterschiedliche Informationen auf der Map darstellt. Daten werden vom Client zum Backend übermittelt und in einer Datenbank persistent gemacht, damit sie von einem anderen Gerät aus oder von einem anderen User jederzeit zur Verfügung stehen.

Das technische Grundgerüst der App besteht aus einem Client-Server, welcher dem User das Frontend liefert, und einem Backend, welches Daten vom User empfängt und speichert. Außerdem ermöglicht es dem User auch einen Account zu erstellen. Zum Backend gehört ebenfalls eine Datenbank, in welchem sämtliche Informationen zu den Waldstandorten und registrierten Usern gespeichert werden.

Ziel ist es, dass die Map auf allen Geräten dargestellt wird und dass nach erfolgreichem Login Flächen in Form von Polygonen direkt auf der Map erstellt werden können. Sie werden vom Backend persistent gemacht und mit dem eingeloggten Benutzer verknüpft. Der User kann danach auch auf anderen Geräten auf diese erstellten Flächen zugreifen und sie mit anderen Benutzern teilen.

3.1.3 Rahmenbedingungen

Die Software soll auf mobilen Geräten sowie auf Desktop/Laptop Browsern benutzbar sein. Mobile Geräte sind von der Leistung her meist schwach verglichen mit Desktopgeräten. Server-side rendering kann dieses Problem weniger relevant machen, weil damit die Applikation nicht vollständig vom Client berechnet werden muss.

Die Website ist darauf ausgelegt, dass sie im Chrome Browser verwendet wird, da Chrome der verbreitetste Browser auf Android Geräten ist, und auch seine Desktop Version sehr zuverlässig funktioniert. Apple Geräte können ebenfalls Chrome verwenden. Chrome befindet sich auf den meisten Geräten auf dem neusten Stand und unterstützt daher die meisten Funktionen.

Als Testgeräte werden ein Apple iPad und iPhone verwendet, ein Android Galaxy S5 und mehrere MacBook Pro und Air Laptops (ca. Baujahr 2010 - 2016).

3.1.4 Vorgehen, Aufbau der Arbeit

Es wurde eine Full-Stack Webapp aufbauend aus Frontend, Backend und Datenbank erstellt. Nach der Festlegung des Frontends wurde ein geeignetes Backend eruiert und die Datenbankanbindung für das Backend festgelegt. Als Frontend Framework wurde VueJS zur Erstellung von Single-Page-Applikationen gewählt. Als Backend kommt Django zum Einsatz, welches sich eignet um API Abfragen für das Frontend bereitzustellen. PostgreSQL wurde als Datenbank gewählt.

Vorgehen: Was wurde gemacht? In welchen Teilschritten? Risiken der Arbeit? Wer war involviert (Durchführung, Entscheide usw.)? Details in anderen Kapiteln. Einführung in die Problem- und Aufgabenstellung. Übersicht über die brigen Teile der Abgabe.

3.2 Stand der Technik

3.2.1 GIS-Browser

Ein GIS-Browser, wie er von den verschiedenen Kantonen in der Schweiz eingesetzt wird, ist ein read-only Archiv, bestehend aus öffentlich zugänglichen Daten.

3.2.2 Collector for ArcGIS

Technologien von ESRI (Environmental Systems Research Institute) und insbesondere ArcGIS online (GIS; Geografisches Informationssystem) wurden recherchiert, um einen funktionierenden Prototypen mit offline-caching zu erstellen. Hintergrundkarten (in Form eines Tile-Layers) können auf dem Gerät zwischengespeichert werden. Die Erstellung von editierbaren Vektorlayern funktioniert auch im offline Betrieb und können später, sobald wieder eine stabile Internetverbindung besteht, synchronisiert werden. Dieses Verhalten kann bei einer Webapp durch Service-Worker rekreiert werden.

Ein GeoJSON (Geo JavaScript Object Notation) file kann ebenfalls auf der online Plattform von ArcGIS hochgeladen werden und danach auf der Map dargestellt werden. Das Layer Styling kann so konfiguriert werden, dass die Farbe eines Polygons dem Typ des jeweiligen Waldstandorts entspricht.

Die Map und deren Inhalt wird im online Tool <https://geodata4edu.hsr.ch/> erstellt. Danach kann sie in der App Collector for ArcGIS (native App) verwendet werden. Feature Layer (oder gehostete Feature Layer) wie Areas und Points of Interest sind editierbar und synchronisieren mit dem Server sobald eine Internetverbindung besteht.

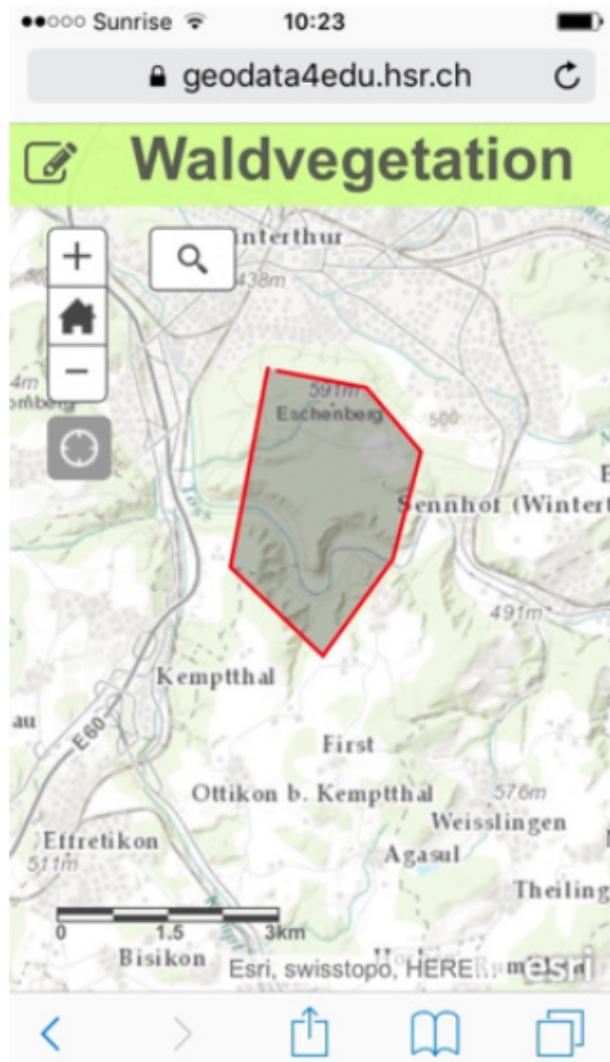


Abbildung 3.1: Prototyp mit ArcGIS online

3.2.3 Defizite

Ein GIS-Browser, wie z.B. vom Kanton Zürich zur Verfügung gestellt, bietet keine Möglichkeit, die eigene Position in Betracht zu ziehen, um damit den Kartenausschnitt zu bestimmen oder genauer noch, den aktuellen Waldstandortstyp per GPS zu bestimmen. Auch ist es nicht möglich, persönliche Notizen oder Flächen wie Beobachtungsflächen definieren, damit sie während der Arbeit im Feld verwendet werden können. Dies liegt daran, dass sich ein User nicht identifizieren kann, und daher alle Benutzer bei jedem Besuch als anonym behandelt werden. Nur durch ein Loginsystem kann sich ein User gegenüber der Webseite gegenüber identifizieren und hat Zugriff auf von ihm erstellten Informationen.

Obwohl Collector for ArcGIS als ein sehr vielfältiges Werkzeug verwendet werden kann, hat es viele Nachteile, vor allem in Kombination ArcGIS online. Die Erstellung einer Map (Hintergrundkarte und Konfiguration der Feature Layer) ist sehr umständlich und für User, welche sich nicht mit den Tools aus-

kennen, nicht ersichtlich. Zum Beispiel mussten gewisse Layer, damit sie korrekt funktionierten, mithilfe der Webseite developers.arcgis.com/ erstellt werden, um sie danach in der Map, welche auf ArcGIS online zusammengeklickt wird, zu importieren. Auf [developers.arcgis](http://developers.arcgis.com/) muss zum Erstellen dieser Layer ein Enterprise Account benutzt werden, welcher eine Organization URL benötigt. Da diese Layer in der ArcGIS online Map und danach in der mobile App "Collector for ArcGIS" verwendet werden, benötigen auch Enduser diesen Login, was eine Verbreitung der App verunmöglicht. Benutzer, welche keinen Enterprise Login haben, können bei ArcGIS online lediglich "Public Accounts" erstellen. Diese sind aber in der Funktionalität eingeschränkt.

3.3 Bewertung

3.3.1 Kriterien

Funktionalität

Maps.zh.ch eignet sich sehr gut um Zugriff auf das sehr umfangreiche Kartenmaterial-Archiv des Kantons zu erhalten. Die Karten sind jedoch als read-only Archiv eingerichtet und auf der Map, welche von maps.zh.ch bereitgestellt wird, lassen sich keine digitalen Anmerkungen, Kommentare oder eigene Flächen oder Points of Interests anbringen. Informationen über eine spezifische Eigenschaft der aktuellen Position im Feld zu erhalten, kann jedoch sehr zeitintensiv sein, da die richtige Karte eingeblendet werden muss und auf der Map manuell an den richtigen Ort hingepant und hineingezoomt werden muss. Da die eigene Position auf der Map nicht ersichtlich ist, muss oft die eigene Position in einem anderen Tool berechnet werden (zum Beispiel Google Maps), um danach per Informationen über einen gewählten Ort zu erhalten. Maps.zh.ch stellt keine API bereit, welche es ermöglicht, Abfragen zu einen gegebenen Ort auszuführen oder Kartenmaterial wie zum Beispiel die Waldvegetationskarte auf einer anderen Map darzustellen. Kartenmaterial muss gekauft und darf nur unter Einhaltung der Nutzungsbedingungen verwendet werden.

Eine native App wie "Collector for ArcGIS" eignet sich, um eine Datensammler Applikation zu erstellen. Das Loginsystem sollte jedoch vereinfacht werden und keine Accounts benötigen, welche an andere Unternehmen gebunden sind. Die Webapp Waldmeister-Outdoors soll ein eigenständiges Projekt werden, welches keine Vorkonfigurierten Map Eigenschaften beinhaltet, welche von in einem Web-Editor wie ArcGIS online erstellt worden sind.

3.3.2 Schlussfolgerungen

Waldmeister-Outdoors kann ein Karten Archiv wie maps.zh.ch anbietet stark aufwerten und durch ein Login System erweitert werden. User sollen wie im "Collector for ArcGIS" die Möglichkeit haben neue Objekte wie Flächen und Points of Interest zu erstellen und Positionsabhängige Funktionen wie den Kartenausschnitt zu zentrieren und den eigenen Standort anzuzeigen. Dies soll bei GPS Funktionen von internetfähigen (Mobilien) Geräten ermöglicht werden und in der Map reflektiert werden.

3.4 Umsetzungskonzept

3.4.1 Lösungsansätze

Damit die App auf mehreren Geräten mit unterschiedlichen Betriebssystemen verwendet werden kann, bietet es sich an eine Webapp zu verwenden, welche in Internet auf einer URL aufgerufen wird. Im Gegensatz zu einer nativen ist diese Webapp plattformunabhängig und muss nicht auf native Umgebungen angepasst werden, da sie vom jeweiligen Browser des Geräts interpretiert wird.

Um die geplanten Funktionalitäten wie das Login und das Erstellen von Benutzeroberflächen umzusetzen, muss die Webapp einer 3-Tier Architektur entsprechen, welche auf Frontend, Backend und einer Datenbank aufbaut.

3.4.2 Frontend

Der Frontend-Client soll als Webapp realisiert werden. Dafür soll ein modernes Webframework gewählt werden, welches zur Erstellung einer Single-Page-Applikation (SPA) eingesetzt wird. Waldmeister-Outdoors soll für Browser von Mobilien und Desktop-Geräten optimiert werden, wie zum Beispiel Chrome oder Firefox.

3.4.3 Backend

Als Backend soll ein flexibles Framework, welches Micro-Services und API Abfragen in Form von REST-Schnittstellen bereitstellen kann, eingesetzt werden. Django wurde gewählt, da es von vielen Projekten des IFS bereits erfolgreich eingesetzt wurde.

3.4.4 Datenbank

PostgreSQL ist ein open-source Datenbank Management System. Es ist dank der PostGIS Erweiterung sehr gut geeignet um raumbezogene Daten (Geospatial Data), wie die Daten der Vegetationskundlichen Karte, zu storen und abzufragen.

3.4.5 JavaScript Libraries und Node Module

Leaflet editable

Damit die User neue Polygone, Punkte und Pfade erfassen können, wird die Library "Leaflet editable" verwendet. Es ermöglicht, neue Objekte (zum Beispiel Polygone und Polys) direkt auf der Leaflet Map zu zeichnen, oder bestehende Objekte zu editieren.

Vuetify

Vuetify ist ein Node Module welches als offizielle Material-Design-Extension von VueJS behandelt wird. Es stellt es viele UI - Komponenten bereit, welche in einer VueJS Applikation verwendet werden können.

Eine vollständige Liste der Node Module befindet sich in der Datei package.json im Ordner client.

3.4.6 Python Pakete

Die wichtigsten Python Pakete sind Django 2.0, welches das Grundgerüst des Backend-Servers darstellt. Django verwendet als Server das Python Paket Djangorestframework 3.8.2 um API Requests des Frontends zu beantworten.

Eine komplette Liste der Verwendeten Python Pakete wird in der file Requirements.txt des backends geführt.

3.4.7 Werkzeuge und Tools

Als IDE bzw. Texteditor wurde das Programm Sublime 2 verwendet, welches die beiden Linter Eslint (für JavaScript Code) und flake8 (für Python Code) unterstützt. Entwickelt wurde auf einer lokalen Node und Python Umgebung, bzw. in dockerisierter Form, um das deployment zu testen.

3.4.8 Deployment

Das Projekt soll auf einem HSR Server deployt werden, damit es per URL im Internet erreichbar ist.

3.5 Resultate

3.5.1 Zielerreichung

User können mittels der Webapp Waldmeister-Outdoors die Vegetationskundliche Karte erforschen und diese mit Ihrem momentanen Standort abgleichen. Darüber hinaus können sie, nachdem sie sich registriert haben, private und öffentliche Benutzerflächen auf einem mobilen Gerät erfassen und sie auf einem zentralen Server hinterlegen. Somit können sie persönliche Orte und Flächen erstellen, welche für Sie im Arbeitsalltag relevant sind und sie zwischen verschiedenen Geräten (Im Feld und im Büro) abrufen oder mit anderen Personen teilen. Erstellte Benutzerflächen können zu einem späteren Zeitpunkt oder von einem anderen Gerät aus editiert oder gelöscht werden.

Ebenfalls erreicht wurde die Geolocation auf der Karte zu repräsentieren. Dies ist im Arbeitsalltag eine grosse Hilfe, und dient zur schnellen Orientierung und Auffindung von bestimmten eingetragenen Flächen und Orten im Feld. Die GPS-Position wird mit einem gesetzten Intervall von fünf Sekunden erneuert, und die erhaltenen Koordinaten werden auf der Map als Element, bestehend aus zwei Circles, dargestellt.

Die Webapp kann sowohl auf Desktop Browsern wie von mobilen Geräten verwendet werden und verhält sich dank VueJS responsive auf eine Änderung des Viewports, zum Beispiel wenn der Screen eines mobilen Geräts während der Verwendung gedreht wird.

Layer und deren Label können ein- und ausgeblendet werden, was zur Übersicht beitragen kann.

Alle Benutzeraccounts und deren Flächen sind im Django Backend ersichtlich und können als Superuser verändert oder gelöscht werden. Wird ein Benutzer aus der Datenbank gelöscht, werden alle von ihm erstellten Benutzerflächen ebenfalls automatisch gelöscht. Alle Passwörter von Benutzeraccounts werden gehasht in der Datenbank gespeichert.

Die Webapp wurde auf dem IFS Server als Dockerimage deployt und ist im Internet erreichbar unter <https://waldmeistermap.sifs0003.infis.ch/>.

3.5.2 Ausblick

Als Wichtigste Punkte zur Weiterentwicklung der Webapp ist die Implementierung einer Vegetationslayer-API, welche das dynamische Laden von Waldstandortsflächen des Clients ermöglicht.

Anpassungen an des UI und weitere Funktionen im Frontend welche die Zusammenarbeit im Team verbessert sollten ebenfalls realisiert werden.

Genauere Ausführungen zur Weiterentwicklung befinden sich im Kapitel 4.9.2 der Projektdokumentation.

3.5.3 Persönliche Berichte

Während der Entwicklung wurde ein Update für das Ubuntu Basisimage veröffentlicht, welche die benötigten Libraries von Gdal entfernte. Gdal (GDAL - Geospatial Data Abstraction Library) setzt diese darunter ein, Geodaten zwischen verschiedenen Formaten zu konvertieren. Die verwendete Version (welche im Dockerfile definiert wird) muss jedoch der Version der Hostmachine von Docker entsprechen. Dieses Update kam zu einem überraschenden Zeitpunkt, und der Build konnte während dieser Zeit nicht ausgeführt werden, bis dass Problem gefunden und behoben wurde. Das Problem wurde erst bemerkt, als das Build von Travis ausgeführt wurde, da die lokale Entwicklung des Build davon nicht beeinträchtigt waren (da in der lokalen Entwicklung kein Ubuntu eingesetzt wurde). Travis ist hier eine grosse Hilfe ein Build auf einem anderen Computer zu testen, bevor die Version live auf dem produktiven Server deployt wird, was in diesem Fall zu einem Unterbruch des Services geführt hätte.

Kapitel 4

Teil 2 - SW-Projektdokumentation

4.1 Anforderungsspezifikation

4.1.1 Use-Cases

Während der Feldforschung kann es sehr hilfreich sein, auf bereits kartierte Waldflächen Zugriff zu haben, um sich am Standort zu orientieren. Dieses Material ist oft in einem Kantonalen Portal z.B. <https://maps.zh.ch> erhältlich, es ist jedoch nur selten kompatibel mit mobilen Geräten, interagieren nicht mit dem GPS des Smartphones oder die Websites sind schwer zu navigieren. "Waldmeister - Outdoors" soll einem gezielten Zweck dienen und nicht mit Kartenmaterial überladen werden. Der Zugriff auf die Vegetationskundlichen Karten sollte vereinfacht und die Navigation beschleunigt werden.

Der über GPS ermittelte eigene Standort soll auf der Karte eingetragen werden und beim Laden der Map soll auf diesen Kartenausschnitt zentriert werden. Diese Funktion des Zentrieren kann jedoch beim Erforschen von Waldstandorten, welche sich nicht in unmittelbarer Nähe befinden als störend empfunden werden. Daher sollte diese Funktion deaktivierbar sein, oder nur einmalig beim Laden der Map ausgeführt werden.

Durch einen Abruf in der Datenbank kann ermittelt werden, ob sich der Benutzer zu diesem Zeitpunkt gerade innerhalb eines Waldstandorts befindet, falls sich die ermittelte Position des Gerts innerhalb eines Polygons befindet, welches in der Vegetationskarte gespeichert ist. Dies soll dem User auf der Map jederzeit mitgeteilt werden, auch wenn er sich zum Zeitpunkt nicht auf dem Kartenausschnitt sichtbar ist.

4.1.2 Use Case - Diagramm

Das Diagramm 4.1 zeigt auf, welche Möglichkeiten ein User hat, mit dem Werkzeug zu interagieren. Ein User, welcher sich nicht registriert, kann keine Flächen generieren oder editieren und kann auch keine privaten Flächen sehen. Er kann jedoch die Vegetationskundliche Karte und öffentliche Flächen aller anderen User sehen.

Nachdem er sich registriert und eingeloggt hat, kann er Flächen erstellen und editieren. Diese teilen sich in öffentliche und private Benutzerflächen auf. Diese Option kann er während der Erfassung der Geometrie der Fläche frei wählen. Standardmäßig ist eine solche Fläche privat und wird nur auf

Wunsch des Users öffentlich gemacht. Eine bereits erstellte Fläche kann aber im Nachhinein auf öffentlich umgeschaltet werden, nachdem sie z.B. durch Änderungen der Geometrie und Position vom User finalisiert wurde. Dies ist vor allem nach Absprache zwischen anderen Experten denkbar, welche über Gruppen auch auf private, noch nicht öffentliche Benutzerflächen Zugriff haben um die Eigenschaften und Lage einer Fläche zu analysieren und über diese zu diskutieren.

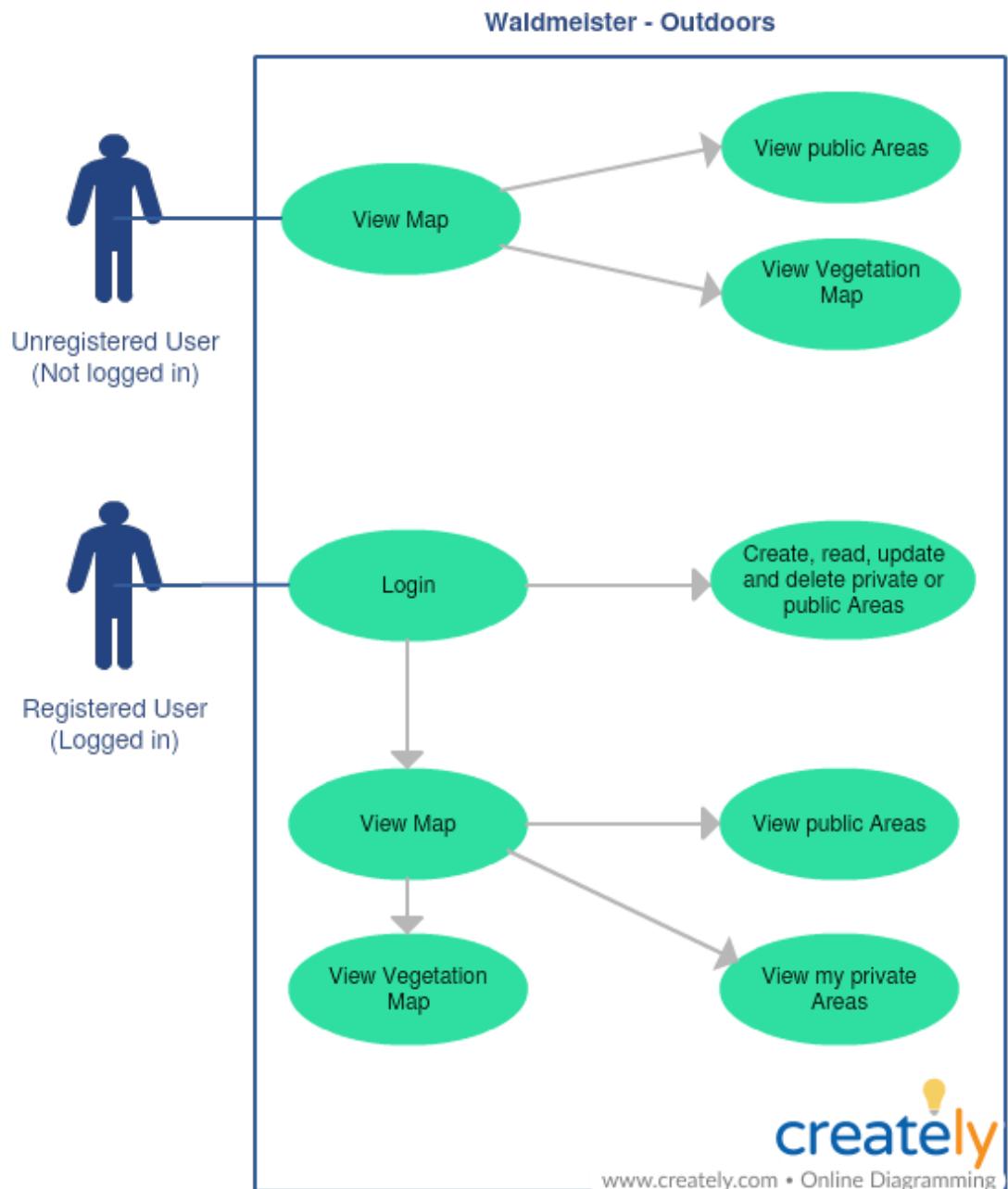


Abbildung 4.1: Use Case Diagram

4.1.3 Must-Haves

Als must-haves werden funktionale Anforderungen beschrieben, welche zentrale Funktionen beschreiben, welche ein User mit der Applikation durchführen möchte. Dazu gehören die Navigation der Map und das Erstellen und Editieren von Benutzerflächen. Ein Login, bzw Registrierungsvorgang ist dabei unerlässlich, damit sich User authentifizieren können, auf welches Kartenmaterial sie Zugriff haben. Bei Benutzerflächen wird hierbei unter persönlichen (privaten) und öffentlichen Benutzerflächen unterschieden.

API-Abfragen sollen wo möglich geschtztet werden und Benutzerflächen, welche von einem User erstellt wurden, sollen nicht von anderen Usern verändert oder gelschtet werden können.

Das Projekt soll per Docker-Container auf dem IFS Server deployt werden und per Link für Jedermann zugänglich sein.

4.1.4 Optional

Als optionale Anforderung wurde die Automatische Anzeige des Waldstandorttyps bezeichnet. Neben der Map, welche alle Waldstandorte am geografisch korrekten Ort darstellt, ist eine ortsbasierte Anzeige des momentanen Waldstandorttypen hilfreich bei der Arbeit im Feld. Diese Anzeige besteht daraus, die GPS Position des Geräts zu verwenden, um in einer Datenbank zu ermitteln, ob der ermittelte Punkt innerhalb eines bekannten Waldstandorts liegt.

4.1.5 Nicht-funktionale Anforderungen

Die Software soll auf mobilen Geräten sowie auf Desktop/Laptop Browsern benutzbar sein. Mobile Geräte sind von der Leistung her meist schwach verglichen mit Desktopgeräten. Server-side rendering kann dieses Problem weniger relevant machen, damit die Applikation nicht vollständig vom Client berechnet werden muss.

Die Website ist darauf ausgelegt, dass sie im Chrome Browser verwendet wird, da Chrome der verbreitetste Browser auf Android Geräten ist, und auch seine Desktop Version sehr zuverlässig funktioniert. Apple Geräte können ebenfalls Chrome verwenden. Chrome befindet sich auf den Meisten Geräten auf dem neusten Stand und unterstützt daher die meisten Funktionen.

Als Testgeräte werden ein Apple iPad und iPhone verwendet, ein Android Galaxy S5 und mehrere MacBook Pro und Air Laptops (ca. Baujahr 2010 - 2016).

Ziel ist es, dass die Map auf allen Geräten dargestellt wird und nach erfolgreichem Login Flächen in Form von Polygonen direkt auf der Map erstellt werden können. Sie werden vom Backend persistent gemacht und mit dem eingeloggten Benutzer verknüpft. Der User kann danach auch auf anderen Geräten auf diese erstellten Flächen zugreifen und sie mit anderen Benutzern teilen.

4.2 Domain Modell

Das Diagramm 4.2 schildert die Relation und den Ausbau der wichtigsten Klassen des Systems. Dies zeigt, dass Benutzerflächen nur von registrierten Usern erstellt werden können und immer einem solchen zugewiesen sind. Wird ein Benutzer aus dem System gelöscht, werden alle Flächen welche diesem Account zugeordnet sind, ebenfalls aus dem System gelöscht.

Die Waldvegetationsflächen sind in der Datenbank als Modell bestehend aus Polygon und einem String mit dem Waldstandorttypen nach EK72 gespeichert. Beim Starten des Backends werden sie aus einem Shapefile geladen und in der Datenbank persistent gemacht, sofern sich der Datensatz nicht bereits in der Datenbank befindet.

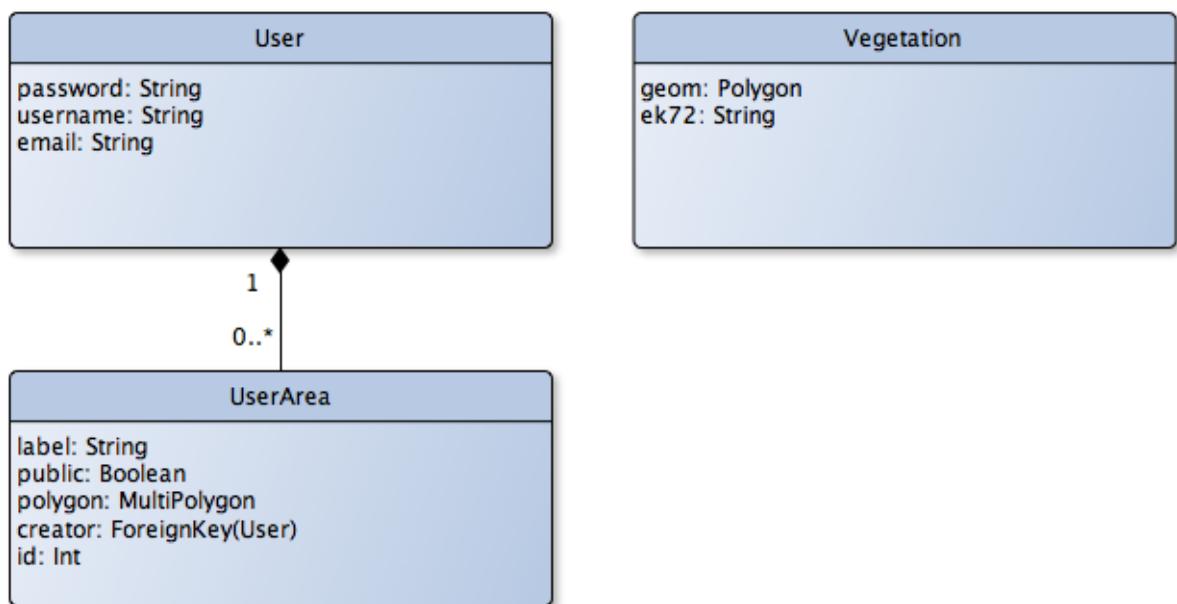


Abbildung 4.2: Domain Modell

4.3 Technologien

4.3.1 Django

Als Server zur Verwaltung der User und der Daten, welche die User generieren und benötigen, kommt Django zum Einsatz. Es ist ein Open-Source Webframework, welches das Python Gegenstück zu Ruby-On-Rails darstellt. Im Kern folgt es dem Model-View-Controller Prinzip, obwohl es eine eigene Namensgebung verwendet. [AH07] In diesem Projekt verwendet Django eine PostgreSQL Datenbank um Daten persistent zu machen. Django wird ebenfalls dazu verwendet, um User einen Account zu geben, damit nur sie selbst Zugriff auf Ihre privaten Benutzerflächen haben, oder um öffentlich erstellte Benutzerflächen mit anderen zu teilen. Zur Authentifizierung wird ein JWT eingesetzt.

Django Rest Framework

Eine SPA kommuniziert hauptsächlich über API Schnittstellen mit dem Server. Hier kommt auf dem Server das Django Rest-Framework (DRF) zum Einsatz.

Es bietet ein sehr flexibles System zur Erstellung von RESTful Web-APIs. Das DRF bietet die Möglichkeit CRUD Operationen auf eine Ressource auszuführen. "Waldmeister - Outdoors" verwendet die REST Api beispielsweise um usergenerierte Flächen, Pfade oder Punkte in der Datenbank zu speichern oder diese zur Darstellung in der Map aus der Datenbank zu laden.

Ebenfalls werden Benutzer, welche sich registrieren, mit Username, Password und ggf. Emailadresse in der Datenbank eingetragen.

4.3.2 PostgreSQL

Postgres ist das Datenbank Management System, welches mit Django zusammen die Daten persistent macht, welche die User per API in der PWA generieren. Hierzu wird das Django Packet psycopg2 verwendet. Django kann durch Models ein Datenbankschema beschreiben, welches von PostgreSQL generiert und in einer lokalen PostgreSQL Instanz gespeichert wird. [Har01] [JF08]

PostGIS

Um Geoinformationsdaten wie z.B. Polygone und Pfade korrekt zu speichern wird auf der Datenbank das Plugin PostGIS installiert. Dadurch kann Django die benötigten Datenbankmodelle erstellen und per REST Schnittstelle speichern. Anhand des Django Models werden Geodaten in einem gewählten Format gespeichert. Standardmäßig wird von Django die Spatial Reference Identifier (SRID) Nummer 4326 (World Geodetic System, WGS84) verwendet. Dieses System benutzt Zahlen von -180, -90 bis 180, 90 um Positionen auf der Erde (in Latitude und Longitude) zu beschreiben.

Postgis wird ebenfalls dazu verwendet, dass die Daten, welche per REST-Schnittstelle an den Client geschickt werden, in richtigem Format (Multipolygone in GeoJSON) übermittelt werden.

4.3.3 VueJS

Vue.JS ist ein JavaScript Framework, welches sich zum Erstellen von Single-Page-Webapplikationen in Form einer PWA eignet. Es wurde im Jahr 2013 erstmals veröffentlicht und wurde am 19. Dezember 2017 auf die aktuellste Version 2.5.13 gepatcht. Vue.JS folgt einer Variation des Model-View-Controller-Entwurfsmusters, dem Model - View - ViewModel Muster. Wie auch das MVC client MVVM der Trennung von Darstellung und der Logik der Benutzerschnittstelle. Dies erlaubt dem nutzenden Entwickler, die Struktur der Anwendung nach eigenen Ansprüchen zu richten.

Entwickler beschreiben es daher als "less opinionated" im Vergleich zu anderen populären JavaScript Webframeworks wie Anglar.JS oder React. Vue.JS kann von Entwicklern eingesetzt werden, welche HTML und JavaScript beherrschen und erfordert keine weiteren Webtechnologien. Vue.JS setzt eine

Website aus Instanzen und Komponenten, bzw Single File Components zusammen. Single File Components sind bei VueJS, welche Architekturprobleme von mittel bis grossen Webapps, welche vollständig von JavaScript getrieben werden, zu verbessern versucht.

Folgende Probleme tauchen dabei auf:

1. Global definitions

Global definitions force unique names for every component

2. String templates

String templates lack syntax highlighting and require ugly slashes for multiline HTML

3. No CSS support

No CSS (Cascading Style Sheets) support means that while HTML and JavaScript are modularized into components, CSS is conspicuously left out

4. No build step

No build step restricts us to HTML and ES5 JavaScript, rather than preprocessors like Pug (formerly Jade) and Babel

VueJS besagt, dass all diese Probleme von Single File Components (mit .vue extension) dank Werkzeugen wie Webpack und Browserify gelöst werden. Eine solche Komponente besteht auf HTML Template, JavaScript und CSS in einer eigenen, abgekapselten Datei.

Durch das erzielt VueJS

1. Complete syntax highlighting

2. CommonJS modules

3. und Component-scoped CSS

Wem diese Idee Abkapselung nicht gefällt, der kann weiterhin ein CSS auslagern und in eine Komponente (innerhalb des HTML Templates) importieren:

```
<!-- my-component.vue -->
<template>
  <div>This will be pre-compiled </div>
</template>
<script src="./my-component.js"></script>
<style src="./my-component.css"></style>
```

Separation of Concern

Was ist gemeint mit Separation of Concern (SoC) und bricht der Aufbau von Single-File-Components nicht dieses Pattern? Eine bekannte Vorgehensweise bei Softwareengineering ist es, ein Computerprogramm in logische Abschnitte einzuteilen und zu separieren. Diese Teile sollten sich um einen Zweck oder Belang (Concern) kümmern. Dies heisst jedoch nicht, dass die verschiedenen Dateitypen unbedingt in separate Dateien aufgeteilt werden. In der modernen User-Interface-Entwicklung und den Entwicklern von VueJS ist es oft leichter gefallen, verschiedene Komponenten, welche lose gekoppelt sind, zu komponieren, statt sie auf drei riesigen Layern (HTML, JS und CSS) getrennt zu halten, sie aber in den Komponenten zu verflechten. [Vue]

Auf diesem Weg sind Komponenten (Template, die Logik und das Styling) zusammenhängender und auch einfacher zu warten, obwohl dies nicht den Prinzipien von SoC folgt. Traditionelles SoC unterteilt dies in die Gruppen der Zwecke Organisation (HTML), Präsentation (CSS) und Interaktion bzw. Verhalten (JavaScript).

Vue-Router

Der Vue-Router ist das Herzstück einer Single-Page-Applikation (SPA). Der offizielle Vue-Router ist ein Client-seitiger Router, welcher mithilfe der HTML5 History API voll funktionsfähiges Client-side routing macht.

In der HTML Definition der Hauptkomponente kann <router-view> als Platzhalter verwendet werden, um die Komponenten anzuzeigen, welche abhängig von der momentanen Route an dieser Stelle angezeigt werden sollen. Ein Wechsel zwischen diesen Routen bewirkt kein Page-Restart, da dies von Vue.JS lediglich innerhalb derselben Page Änderungen bewirkt und keine tatsächlichen URL Aufrufe ausführt.

Der Vue Router wird innerhalb einer Hauptseite angezeigt welche die Basis der Webseite darstellt. Methoden von Komponenten können bewirken, dass sich der Inhalt verändert, welcher an der Stelle des Router-Views angezeigt wird. Menupunkte im Header (z.B Register, Login, About, Map), bewirken mit .push() dass der Vue-Router mithilfe des index.js files die korrekten Komponenten an dieser Stelle anzeigt. Dies kann auch direkt über eine URL Eingabe /register oder /login erfolgen, ohne dass der Aufruf über eine interne Komponente ausgeführt werden muss. Die Datei index.js beinhaltet daher alle möglichen Pfade, welche von der Webapp aufgelöst werden und bestimmt die angezeigten Komponenten, welche mit dieser Route verknüpft sind.

Alternativ könnten die ähnlichen Lösungen von Page.js oder Director als Third-Party Produkte an dieser Stelle integriert werden, es ist jedoch zu empfehlen, die offizielle Vue-Router Library zu verwenden.

VueX

VueX ist eine offizielle Erweiterung von Vue.JS und fungiert als Statusmanager. VueX arbeitet mit einem Store, welcher die Zustände aller Komponenten in einer Vue Applikation über Regeln definiert. VueX besteht aus Actions, Mutations und States. Aktionen können in dieser Reihenfolge eine Auswirkung auf die Vue Komponenten haben.

VueX hat Vorteile bei mittel bis grossen Projekten, welche auf dem Single-Page-Application Prinzip basieren. VueX bietet auch die Möglichkeit, einen zentralen Store in kleinere Module aufzuteilen, jedes

mit ihren eigenen State, Mutations, Actions Werten.

Da Komponenten in Vue abgekapselt sind, können sie standardmässig nicht auf Daten zugreifen ,welche in anderen Komponenten definiert werden. Solche Daten müssen per Store verfügbar gemacht werden, damit sie über mehrere Instanzen geteilt werden können.

```
const sourceOfTruth = {}

const vmA = new Vue({
  data: sourceOfTruth
})

const vmB = new Vue({
  data: sourceOfTruth
})
```

Wird nun sourceOfTruth verändert, wird sie in allen Komponenten, in welcher sie verwendet wird, automatisch auf den neuen Stand gebracht. Dies kann in grösseren Applikationen schnell unübersichtlich werden, da jede Komponente diese sourceOfTruth verändern kann, ohne eine nachvollziehbare Spur zu hinterlassen. Es wird daher empfohlen, das Store Pattern von VueX zu implementieren, welche Veränderungen am Store nur über Mutationen zulässt. Somit wird es klarer, zu welcher Zeit welche Mutationen aufgerufen werden können und wie sie durchgeführt wurden:

```
var store = {
  debug: true,
  state: {
    message: 'Hello!'
  },
  setMessageAction (newValue) {
    if (this.debug) console.log('setMessageAction_triggered_with', newValue)
    this.state.message = newValue
  },
  clearMessageAction () {
    if (this.debug) console.log('clearMessageAction_triggered')
    this.state.message = ''
  }
}
```

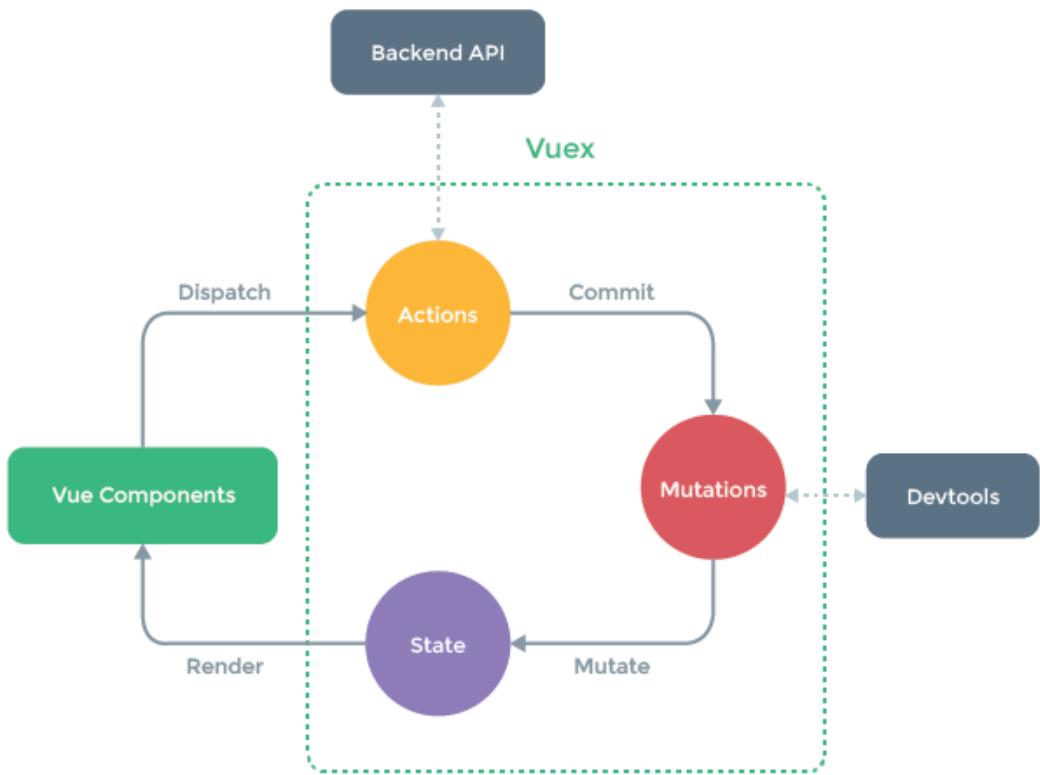


Abbildung 4.3: Vuex Action-Mutations-State Diagram

Komponenten können auch private Zustände haben, dies wird mit "privateState" erreicht. In diesem Fall muss der sharedState ebenfalls definiert werden.

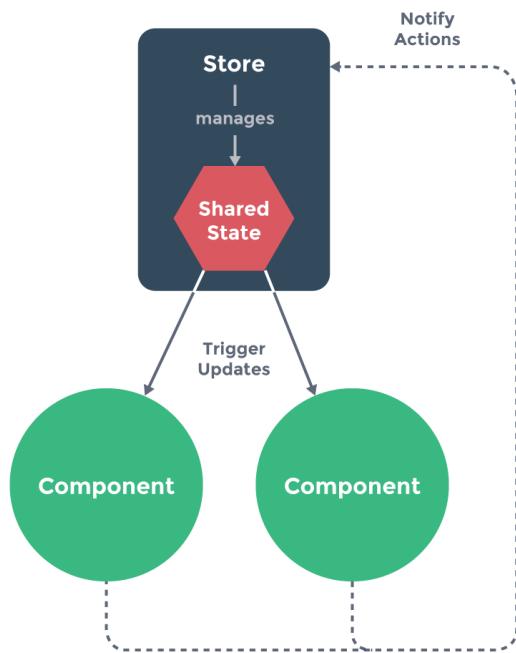


Abbildung 4.4: Vuex private

Dies bewirkt, dass eine Komponente nicht direkt einen Wert oder Zustand im Store verändern kann, sondern dies einen Event aufruft. Dieser informiert den Store welchen State es über eine Mutation zu verändern gilt.

4.3.4 Docker

Docker wird dazu verwendet, um die fertige Webapp plattformunabhängig laufen zu lassen. Es wird sowohl beim lokalen builden der Webapp verwendet, sowie auch in der deployten Form, auf einem Linux-basierten Server. Docker garantiert, dass sich alle benötigten Systeme gleich verhalten, da sie in einer virtualisierten Form (einem Docker-Container) laufen.

4.4 Design

4.4.1 Architektur

Als Frontend folgt Vue.JS einer Variation des Model-View-Controller-Entwurfsmusters, dem Model - View - ViewModel Muster. Wie auch das MVC folgt MVVM dient es der Trennung von Darstellung und der Logik der Benutzerschnittstelle. Dies erlaubt dem nutzenden Entwickler, die Struktur der Anwendung nach eigenen Ansprüchen zu richten. Vue setzt Stores seinen eigenen Vue Store ein. Stores enthalten den Applikationszustand und die Applikationslogik. Verglichen mit dem MVC-Pattern entsprechen sie am ehesten dem Model.

Wie die meisten SPAs bezieht VueJS dynamische Daten über eine REST-Schnittstelle, welche Requests an das Backend schickt, welches die benötigten Daten bereitstellt.

4.4.2 Package Struktur

Der Source-Code von "Waldmeister-Outdoors" ist auf GitHub unter "<https://github.com/dschmide/Waldmeister-Outdoors>" erhältlich. Die Paket-Struktur des Projekts ist in die Teile "backen", "client", "Dokumentation" und "secured-local-nginx" aufgeteilt.

4.4.3 Sequenz-Diagramm

Die folgenden Sequenzdiagramme geben detaillierten Einblick in den Ablauf des Registrierung - und Loginvorgangs, sowie das Laden der Public und Private Areas und deren Darstellung im Client.

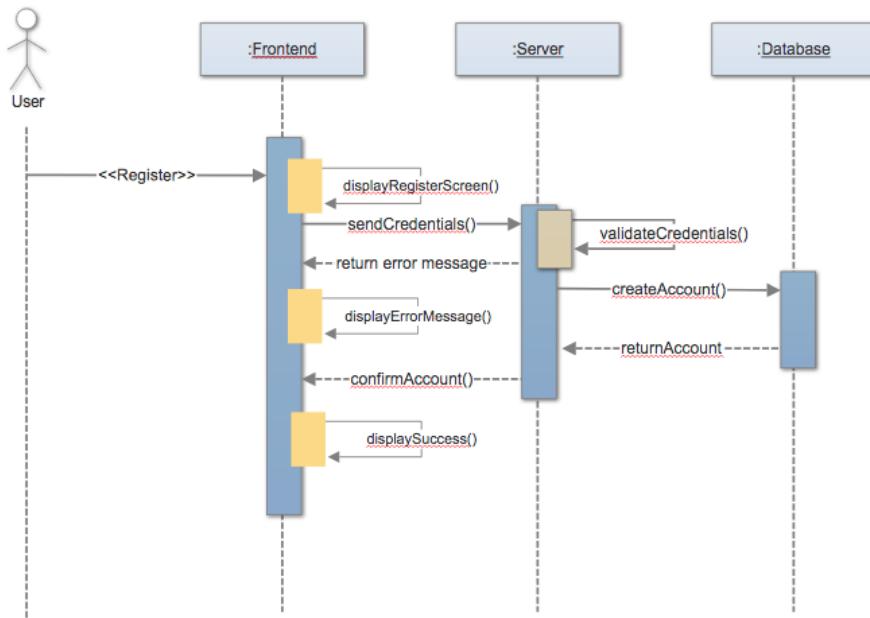


Abbildung 4.5: Sequenzdiagramm, Register

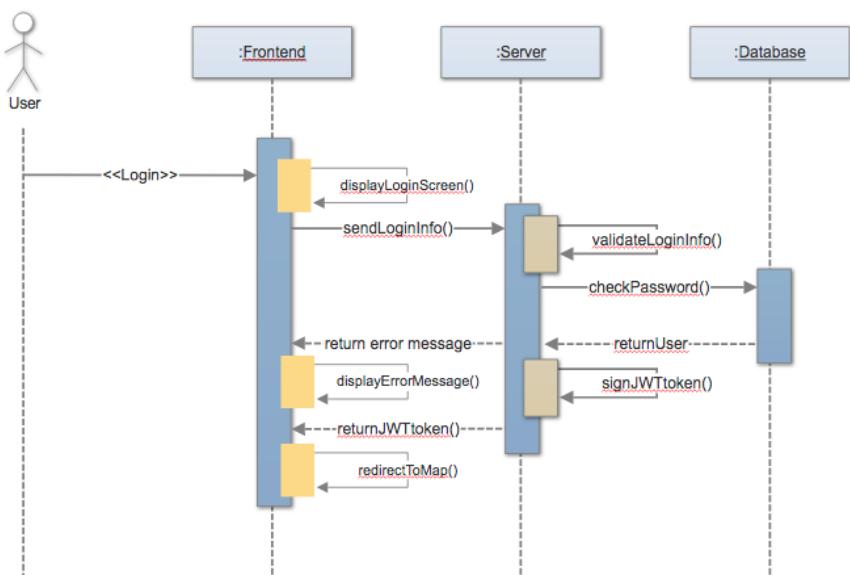


Abbildung 4.6: Sequenzdiagramm, Login

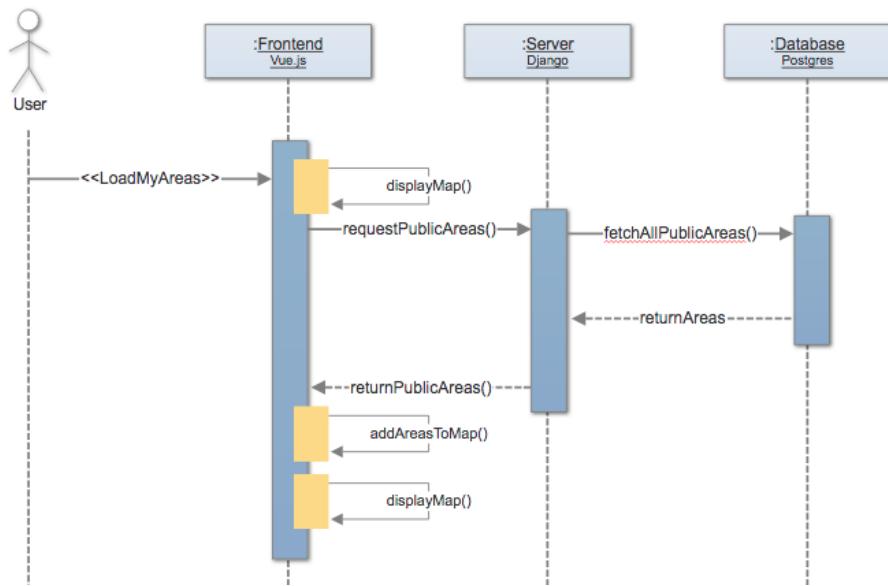


Abbildung 4.7: Sequenzdiagramm, Public Areas

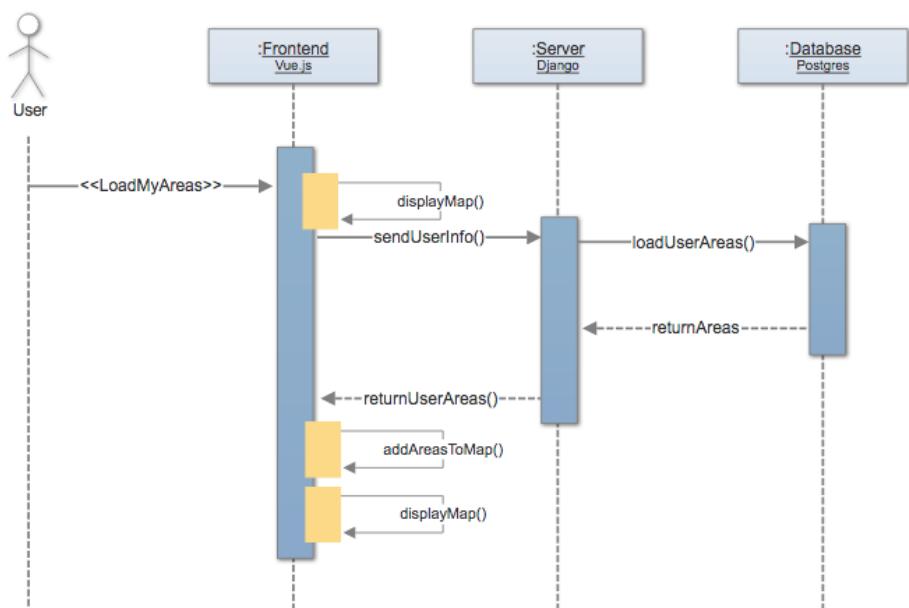


Abbildung 4.8: Sequenzdiagramm, My Areas

4.4.4 Komponentendiagramm

Das Komponentendiagramm zeigt die Beziehungen zwischen verschiedenen Komponenten der Webapp. Die Waldmeistermap bezieht Daten über verschiedene Interfaces welche auf der Map dargestellt oder verwendet werden.

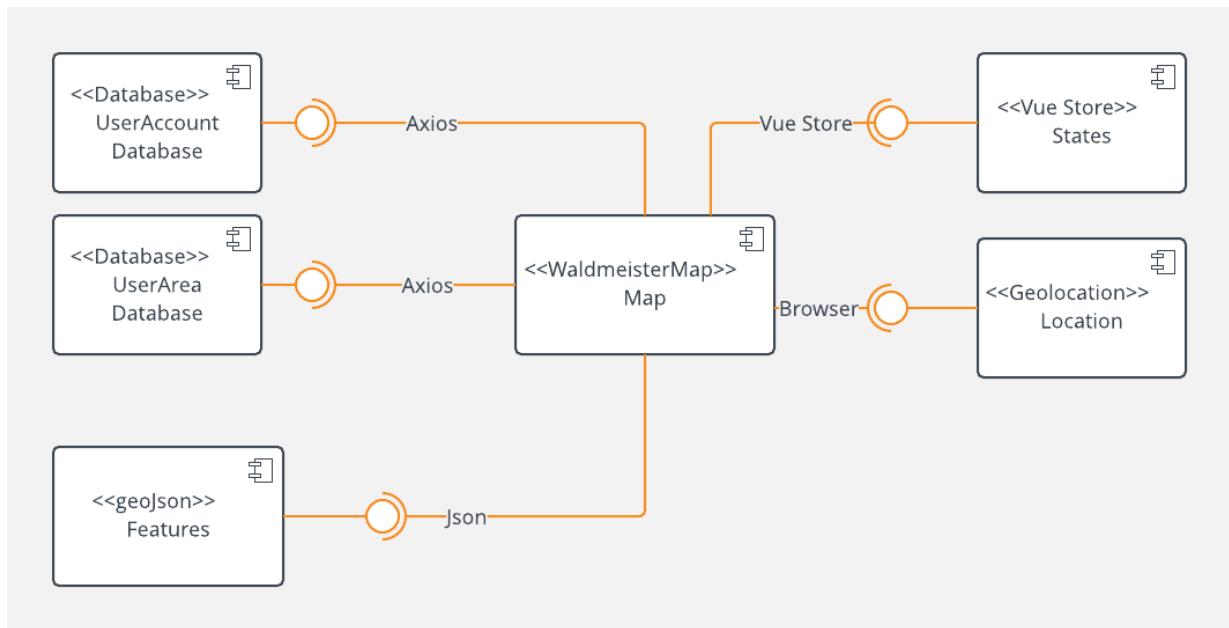


Abbildung 4.9: Komponentendiagramm, Waldmeister Map

4.4.5 UI Design

Mockups

Die Mockups wurden vor der Implementation erstellt, um Screendesign und Layout klarer zu definieren, bevor es um die technische Implementation von "Waldmeister - Outdoors" ging. In den Abbildungen 1 bis 9 kann man den Arbeitsschritt Einloggen und Erstellen einer neuen Fläche und eines Points of Interests (POI) sehen. Zusätzlich sieht der User seine eigene Location auf der Map eingetragen und hat über das Menu "My Places" Zugriff auf eine Liste seiner erstellten Flächen. Ein Kontextmenu gibt bei der Anzeige eines ausgewählten Objekts zusätzliche Informationen.

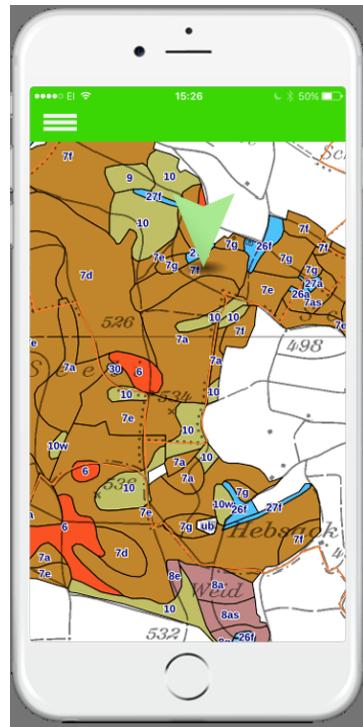


Abbildung 4.10: Mockup Screen 1, Anzeige des Eigenen Standorts auf der Map

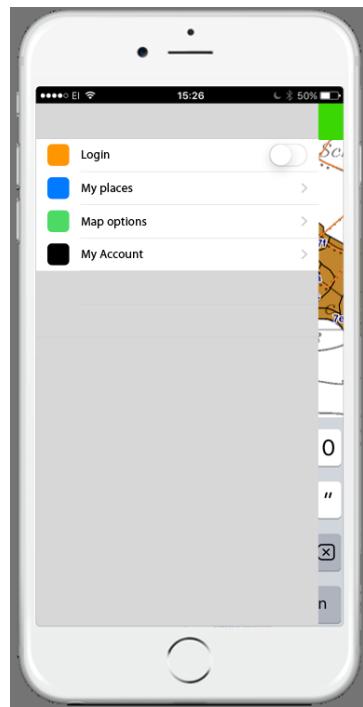


Abbildung 4.11: Mockup Screen 2, öffnen des Menus, Login

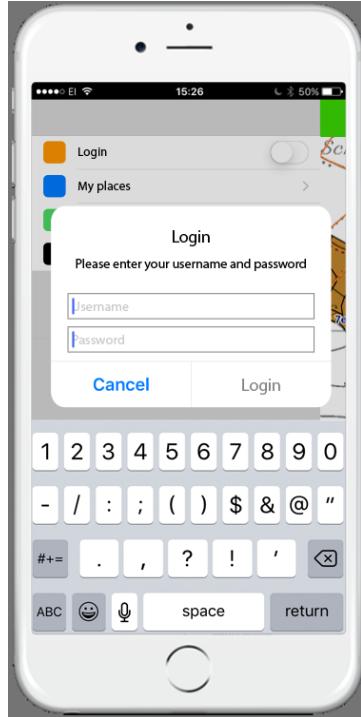


Abbildung 4.12: Mockup Screen 3, Eingeben der Accountdetails

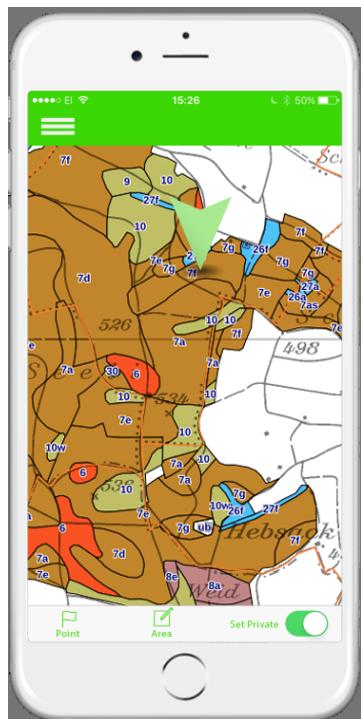


Abbildung 4.13: Mockup Screen 4, Map Ansicht nach dem Login

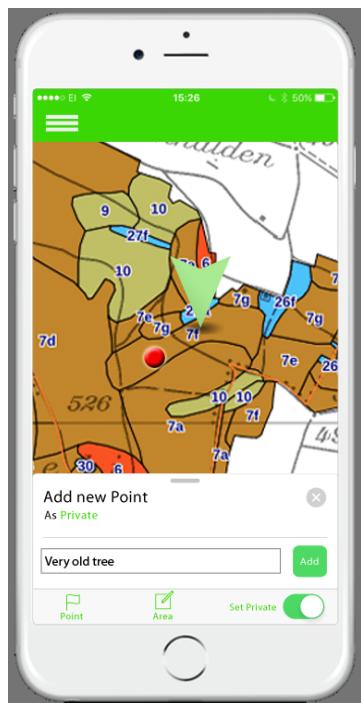


Abbildung 4.14: Mockup Screen 5, Erfassung eines Points of Interests

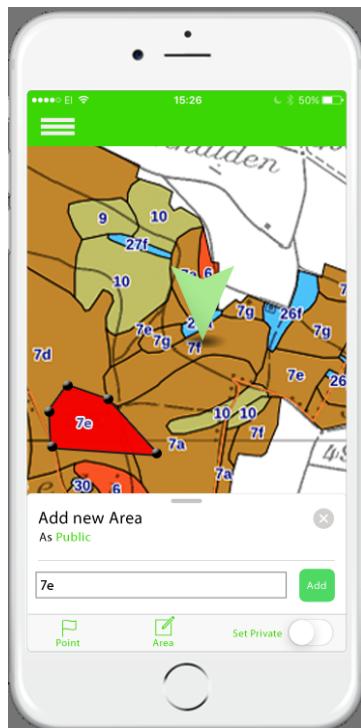


Abbildung 4.15: Mockup Screen 6, Erfassung einer Benutzerfläche

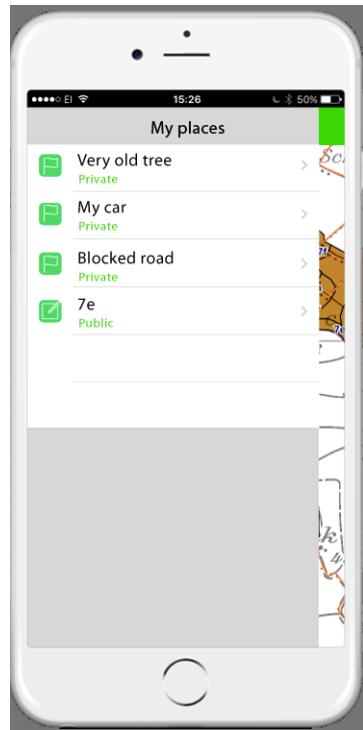


Abbildung 4.16: Mockup Screen 7, Auflistung aller Benutzerflächen dieses Users

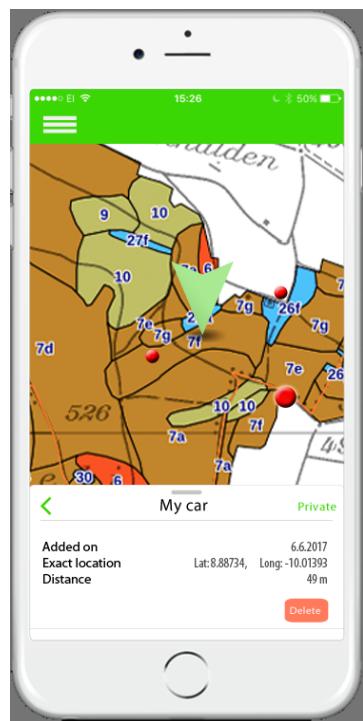


Abbildung 4.17: Mockup Screen 8, Detailansicht eines selektierten Points of Interests

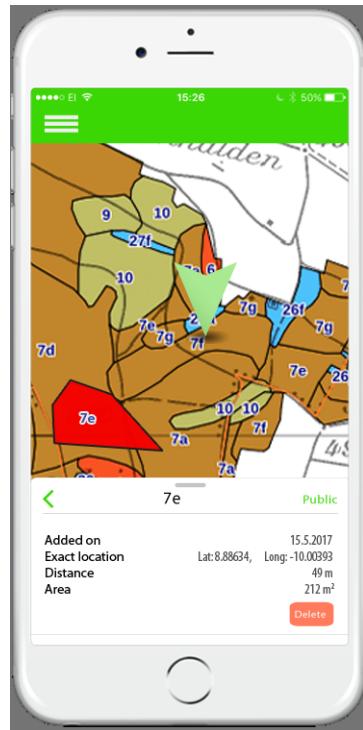


Abbildung 4.18: Mockup Screen 9, Detailansicht einer selektierten Benutzeroberfläche

4.5 Implementation

4.5.1 Vue Komponenten

Die Applikation "Waldmeister - Outdoors" setzt sich aus den Komponenten Register, Login, About, der WaldmeisterMap und einem Page Header zusammen. Darüber hinaus bildet die Komponente "App.vue" die Basis, um den Page Header und Router-View Komponenten zu laden.

Vue Map Komponente

Die Komponente WaldmeisterMap ist die Vue Komponente, auf welcher die Leaflet Map und darauf die Layer Vegetationskarte (als geoJSON) und der Layer für die Benutzerflächen (als Polygone) dargestellt werden. Das Leaflet Map Objekt beinhaltet auch die Kontrollbuttons, welche mit der Map assoziiert sind. Durch die Buttons 'Veg' und 'UAs' in der oberen linken Ecke der Map können die GeoJSON bzw die Benutzerflächen ein - und ausgeschaltet werden. Dadurch werden die Layer, auf welchen sich die Polygone befinden gecleared und alle erstellten Labels von der Map gelöscht. Betätigt der User den Button noch einmal, werden sie erneut erstellt. Sie sind standardmäßig eingeschaltet und werden daher auch erstellt, wenn das Map Objekt auf die Page gemountet wird (beim Laden der WaldmeisterMap Komponente). Damit Label und Polygone der verschiedenen Layer separat ein- und ausgeschaltet werden können, werden sie zuerst in Gruppen unterteilt, bevor diese Gruppen auf die Map hinzugefügt werden.

Oberhalb dieser zwei Buttons befindet sich der "Add" Button, welcher es dem User ermöglicht, eine neue Benutzerfläche auf der Map einzutragen. Er kann, während er im Zeichnungsmodus ist, per click event dem Polygon einen neuen Eckpunkt anhängen. Klickt der User während des Zeichnungsmodus auf den ersten erstellten Eckpunkt, schliesst sich das Polygon und der User wird per Dialogbox aufgefordert, dem erstellten Objekt die benötigten Attribute zuzuweisen (label, public). Drückt er in diesem Dialog auf "Save" wird die Fläche an den Server geschickt und in der Datenbank gespeichert.

Registrierung und Login

Registrierung und Login wurden als separate Single-File-Komponenten aufgebaut, damit sie vom Vue-Router dargestellt werden können. Sie beinhalten die Felder Username, Passwort und bei der Registrierung auch ein Email Feld, welches optional ist. Bei der erfolgreichen Registrierung wird von Djoser ein Benutzer angelegt und sein Passwort im verschlüsselten Zustand hinterlegt. Loggt sich der Benutzer mit den korrekten Daten ein, erhält er vom Server ein JWT (JSON Web Token), welches von Djoser basierend auf dem hinterlegten Accounts erstellt wird. Der Vue-Client Server hinterlegt dieses Token im Vue Store, damit es in verschiedenen Komponenten verwendet werden kann. Um sich dem Server gegenüber zu authentifizieren wird es bei einem Request mitgeschickt. Loggt er sich aus, wird dieses Token aus dem Store gelöscht. Der User kann sich erneut oder unter einem anderen Benutzernamen und Passwort einloggen.

Bei nicht erfolgreichen Versuchen, einen Account zu erstellen oder sich einzuloggen, werden dem User

diverse Fehlermeldungen angezeigt, welche Auskunft über fehlerhafte Passworteingabe oder Benutzerdaten bei der Erstellung des Accounts geben.

Nach einem erfolgreichen Login wird der User per Vue-Router auf die Komponente WaldmeisterMap weitergeleitet und alle Benutzeroberflächen werden vom Server angefordert, auf die er Zugriff hat. Der Server liefert per REST Interface alle öffentlichen Benutzeroberflächen, sowie die privaten Benutzeroberflächen, welche diesem User zugeordnet sind. [djo]

About

Die Komponente About enthält wichtige Informationen zu den dargestellten Daten und beinhaltet die Links zu den nativen iOS und Android Apps, unter welchen das Nachschlagewerk "Waldmeister" erhältlich ist. Unterhalb des dargestellten Logos befinden sich ausserdem Informationen zum Projekt und den Rahmenbedingungen, unter welchen es erstellt wurde.

Hier wird auch dem Kanton Zürich gedankt für die Nutzungsrechte der Daten "Vegetationskundliche Kartierung der Waldflächen im Kanton Zürich".



Abbildung 4.19: About in der Waldmeister-Outdoors Webapp

4.5.2 Frontend Design

Das User Interface wurde grösstenteils mit Vuetify realisiert, welches die äussere Erscheinung der Komponenten Header, Registrierung, Login, About, etc... bestimmt. Es wurde auf das Farbschema der existierenden Waldmeister App angepasst und eignet sich, um eine Webapp responsive zu gestalten, damit sie auf mobilen Geräten ebenfalls korrekt dargestellt wird.

Weitere Teile, vor allem Elemente welche sich auf der Leaflet Map befinden, wurden mit CSS Styling aufgewertet. Der GeoJSON Layer, welcher die Flächen der Waldstandorte anzeigt, wird durch CSS dazu verwendet jedes Polygon auf diesem Layer aufgrund von einem Property des Polygons einzufärben. Dieses Property ist die EK72 Kennzahl, bzw. Kürzel. Jedem dieser Kürzel kann eine bestimmte Farbe zugewiesen werden, welche als hexadezimale, 6-stellige Zahl im Code repräsentiert wird. Jedes Polygon erhält darüber hinaus auch ein Label, welches dieses Kürzel ebenfalls darstellt. Diese Property Label haben eine weisse Schrift, sind nicht transparent und haben einen 2 Pixel weiten Schatten, damit sie sich von der Hintergrundfarbe abheben.



Abbildung 4.20: Einfärbung von GeoJSON Flächen und Darstellung von PropertyLabels

Benutzerflächen werden blau dargestellt und haben ein gefärbtes Label, damit sie sich von den Flächen des geoJSON Layers unterscheiden. Private Flächen haben ein gelbes Label, öffentliche sind grün. Darüber hinaus haben sie eine breitere Outline als Flächen des GeoJSON Layers.

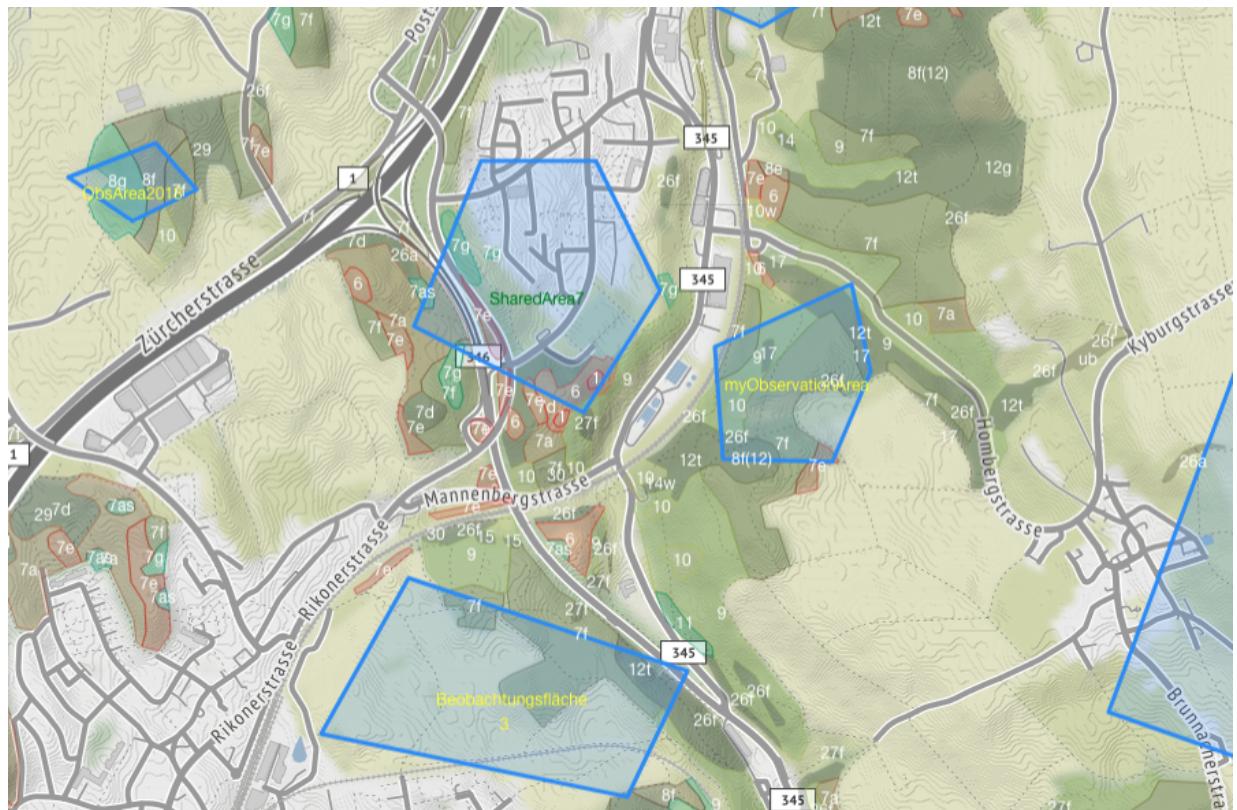


Abbildung 4.21: Benutzerflächen mit Labels

Die Geolocation wird als blauer "Circle" dargestellt. Die Grösse des Kreises repräsentiert die Genauigkeit der Berechnung der Geolocation in Metern.

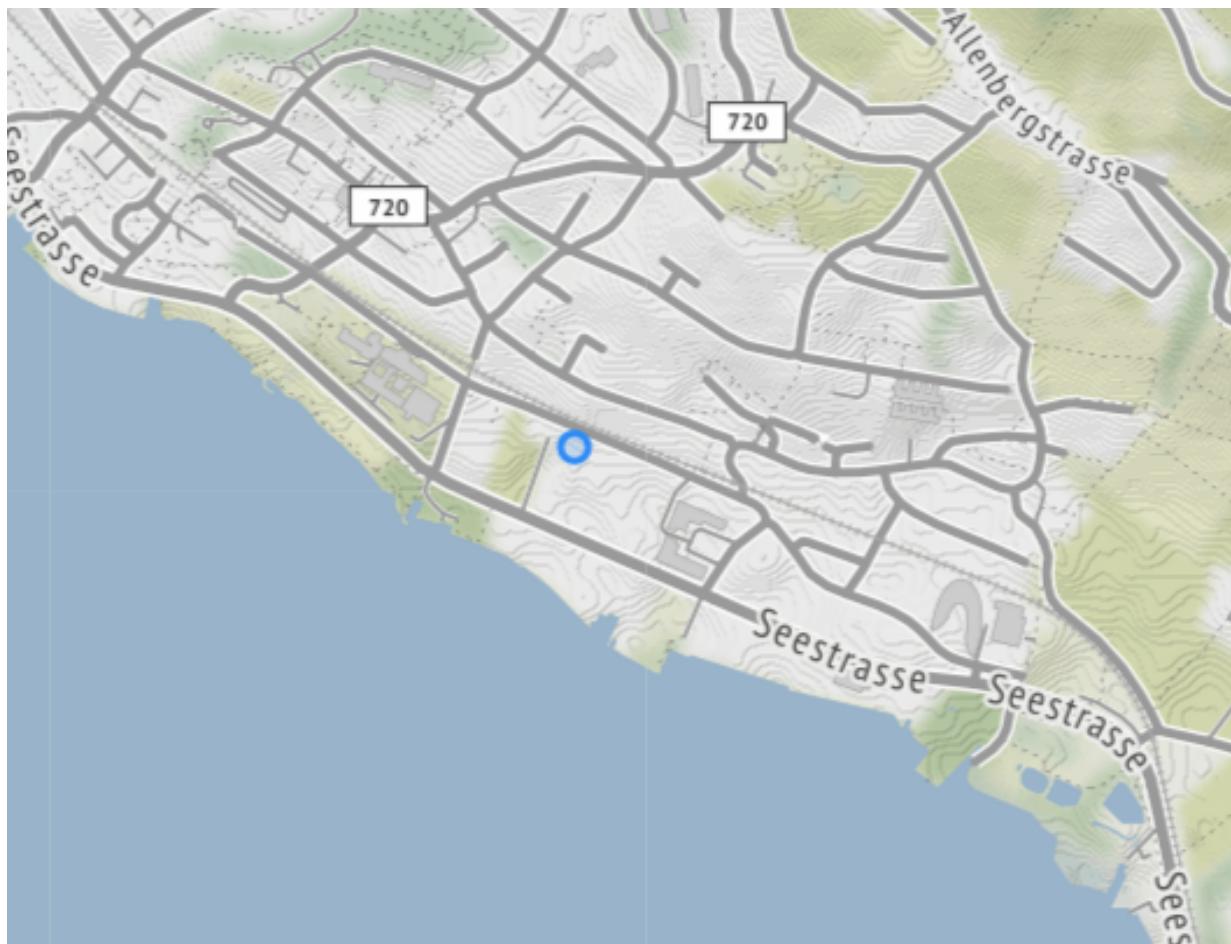


Abbildung 4.22: Geolocation mit Circle

Die Buttons welche den Zeichnungsmodus starten oder beenden oder Ebenen ein- und ausblenden sind in der oberen linken Ecke, unterhalb der Zoombuttons vertikal aufgereiht. Der Zeichnungsbutton ist mit "Add" beschriftet und wechselt auf "Edit" solange der Zeichnungsmodus aktiviert ist. Die Buttons zum Ein- und Ausblenden von Ebenen sind mit "Veg" und "Areas" beschriftet, haben aber längere Tooltips, welche die Funktion genauer beschreiben.

Die "Save" Dialogbox wird nur angezeigt, wenn eine gezeichnete Fläche durch einen Klick auf den ersten gezeichneten Eckpunkt geschlossen wird. Durch den "Save" Button innerhalb der Dialogbox werden die Werte für Label und Public übernommen und das gezeichnete Polygon wird per REST Schnittstelle an den Server geschickt, damit es in der Datenbank gespeichert wird.

4.5.3 Geolocation

Damit der eigene Standort auf der Map eingetragen oder die Map auf diesen Punkt zentriert werden kann, muss die Webapp HTTPS Secure (Https) verwenden, da die meisten Webbroweser es nicht zulas-

sen, den Standort eines Users zu ermitteln, ohne dass die Verbindung geschützt ist. Um den Transport Layer zu verschlüsseln wird ein self-signed Zertifikat verwendet, welches auf dem Vue-Client hinterlegt wird. Es besteht aus einer cert.pem und einer key.pem file. Browser werden zwar eine Warnung für solche Zertifikate einblenden, können aber eine verschlüsselte Verbindung aufbauen, nachdem sie zugelassen wird.

Danach können die Funktionen von HTML 5, z.B navigator.geolocation.getCurrentPosition() verwendet werden, welche einen Punkt mit Latitude und Longitude zurückgibt. Zu diesem Punkt gibt es zusätzlich einen Accuracy Wert, welcher die Genauigkeit der Berechnung in Metern angibt.

4.5.4 API

Diese Anfragen werden vom Client über eine Axios API abgesetzt. Nach der Initialisierung des Axios Services wird eine Base Url bestimmt, welche auf das Backend im Django Server zielt. Mit jedem Request, welches über diesen Service ausgeführt wird, wird auch das im Store hinterlegte JWT Token mitgeschickt, sofern es existiert. Falls nicht, wird der User als nicht autorisiert identifiziert. Neben der Authentication Service, welcher die Methoden Register und Login auslösen kann, besteht der AreaService, welcher per getAreas und postAreas Benutzerflächen vom Server anfordern oder an den Server schicken kann, damit sie in der Datenbank erstellt werden. Die Erstellung von neuen Flächen ist jedoch nur möglich, wenn der User sich eingeloggt hat.

Abhängig vom Request fügt Axios Service eine URL-Endung an die Basisurl an, damit Django per Routingsystem erfassen kann, um welche Art Request es sich handelt. Post Requests innerhalb des AuthenticationServices kommunizieren mit den Djoser definierten URL-Endungen /auth/users/create (um beispielsweise einen neuen Benutzer zu registrieren). Diesem Post Request werden die credentials mitgeschickt, welche vom User im Client eingegeben wurden. Dieses credentials-Objekt besteht aus username, password und email. Sie werden als JSON Objekt an den Django Server geschickt.

Neben dem AuthenticationService regelt der AreaService per get und post requests über die URL-Endungen '/api/areas/' Requests bezüglich Benutzerflächen. Axios kommuniziert mit dem Server über eine REST-Schnittstelle. Benutzerflächen werden per Get-Request an den Client übertragen, neu erfasste Benutzerflächen werden über einen Post-Request als JSON Objekte an den Server übertragen. Django deserialisiert diese JSON Objekte und speichert sie anschliessend in der PostgreSQL Datenbank.

API Dokumentation, Swagger

Die API Dokumentation wurde mit dem Python Package Swagger implementiert und kann über die URL WaldmeisterMap/swagger/ aufgerufen werden. Sie gibt Information über alle verfügbaren API Requests und deren Funktionen. Die API gliedert sich in die URLs der Benutzeraccount Erstellung (WaldmeisterMap/auth/) und die API Requests welche für die Erstellung von Benutzerflächen zuständig sind (WaldmeisterMap/api/areas/).

Benutzeraccounts werden vom Python Package Djoser erstellt. Dieses Package beinhaltet unter anderem die URL /WaldmeisterMap/auth/users/create/, welches mit den Attributen "email, username, password" einen neuen User in der Datenbank registriert. Nach der erfolgreichen Registrierung kann über

die URL /WaldmeisterMap/auth/token/create/ ein JWT Token vom Server angefordert werden. Dieser Request hat die Attribute "username, password". Stimmen die Werte mit einem existierenden Account in der Datenbank überein, wird dem Client als Antwort ein JWT Token geliefert, welches der Client mit darauffolgenden Requests mitschicken kann, damit der Server den User mit einem registrierten User in der Datenbank verknüpfen kann. JWT Tokens laufen nach einer bestimmten Zeit ab oder z.B. wenn der Django Server sich neu initialisiert. Ein JWT Token kann über die URL /WaldmeisterMap/auth/jwt/refresh/ erneuert werden, in dem das existierende Token im POST Request mitgeschickt wird. Benutzerflächen werden über die URL /WaldmeisterMap/api/areas/ als GET Request abgefragt. Der Request liefert anhand vom eingeloggten oder nicht eingeloggten Usern öffentliche Flächen, bzw. auch die privaten Flächen des authentifizierten Users, welche unter seinem Account in der Datenbank gespeichert sind. Über die gleiche URL kann mit einem POST Request eine neue Benutzerfläche in der Datenbank erstellt werden. Über die URLs /WaldmeisterMap/api/areas/{id} kann auf eine spezifische, existierende Benutzerfläche die Requests GET, DELETE, PATCH und PUT ausgeführt werden, um diese bestehende Benutzerfläche zu bearbeiten, bzw. zu löschen. Sie wird über die URL-Endung "id" referenziert.

Über die API kann auch abgefragt werden, ob sich die ermittelte GPS Position des Gerts zu diesem Zeitpunkt gerade innerhalb eines Waldstandorts befindet, welcher in der Datenbank eingetragen ist.

api		Show/Hide	List Operations	Expand Operations
GET	/api/waldmeister-map/api/areas/			
POST	/api/waldmeister-map/api/areas/			
DELETE	/api/waldmeister-map/api/areas/{id}/			
GET	/api/waldmeister-map/api/areas/{id}/			
PUT	/api/waldmeister-map/api/areas/{id}/			
GET	/api/waldmeister-map/api/vegetation/			
GET	/api/waldmeister-map/api/vegetation/{id}/			

Abbildung 4.23: Swagger API Authorization

auth		Show/Hide List Operations Expand Operations
GET	/api/waldmeister-map/auth/	Root endpoint - use one of sub endpoints.
POST	/api/waldmeister-map/auth/jwt/create/	API View that receives a POST with a user's username and password.
POST	/api/waldmeister-map/auth/jwt/refresh/	API View that returns a refreshed token (with new expiration) based on
POST	/api/waldmeister-map/auth/jwt/verify/	API View that checks the veracity of a token, returning the token if it
POST	/api/waldmeister-map/auth/password/reset/	Use this endpoint to send email to user with password reset link.
POST	/api/waldmeister-map/auth/password/reset/confirm/	Use this endpoint to finish reset password process.
POST	/api/waldmeister-map/auth/token/create/	Use this endpoint to obtain user authentication token.
POST	/api/waldmeister-map/auth/users/activate/	Use this endpoint to activate user account.
POST	/api/waldmeister-map/auth/users/create/	Use this endpoint to register new user.

Abbildung 4.24: Swagger API UserAreas

Implementation Notes
Use this endpoint to register new user.

Parameters

Parameter	Value	Description	Parameter Type	Data Type
data	{ "email": "testemail@gmail.com", "username": "itsmyusername", "password": "dontusethispassword135" }		body	Model Example Value { "email": "string", "username": "string", "password": "string" }

Parameter content type: application/json

Response Messages

HTTP Status Code	Reason	Response Model	Headers
201			

Try it out!

Abbildung 4.25: Swagger API user create Details

4.5.5 Database

Die Benutzerflächen und Vegetationsflächen werden von Django durch ein Django-Databatase-Model beschrieben und haben die Attribute "label, public, polygon und creator".

"Label" ist ein Feld, welches aus Charakteren und Ziffern besteht.

Das Feld 'public' ist ein Boolean Wert, welcher bestimmt, ob eine Fläche öffentlich oder privat gehandhabt wird.

'polygon' beinhaltet die Geometrie, die Art der Fläche und das Format, in welcher sie in der Datenbank gespeichert wird. Die Geometrie ist Teil des polygon field und beinhaltet ein Array, welches Punkte in Longitude / Latitude Paaren enthält, welche die Eckpunkte des Polygons beschreiben. Dies können beliebig viele sein und im Falle eines Multipolygons kann dieses Feld auch mehrere Arrays beinhalten, welche wiederum ein einzelnes Polygon beschreiben. Überschneiden sich mehrere Polygone entstehen "Inseln", welche nicht zu der Fläche des Multipolygons gehören.

Also Koordinatensystem wird das System WGS84 mit der SRID 4326 verwendet. [?]

Vegetationsflächen werden in der Datenbank als Multipolygone und einem Krzel nach EK72 in der Form von Zahlen und Charakteren gespeichert.

Die Django Datenbank Modells befinden sich im Ordner "restful_{api}" des backends in der Datei *models.py*

Database, PostgreSQL

Für die Kommunikation zwischen Django und PostgreSQL wird das Python Package Psycopg2 verwendet. Dieses Package ist der verbreitetste PostgreSQL Datenbank Adaptor für die Python Programmiersprache.

Als Datenbank wird PostgreSQL 10 verwendet. Die Datenbank Verbindungsdetails sind in der Python settings.py Datei eingetragen. Waldmeister-Outdoors wird versuchen, auf eine lokale Datenbank mit dem Namen 'dschwaldmeister' zuzugreifen. Username ist postgres und der Port ist standardmäßig auf 5432 gesetzt. Die "Engine" wurde ebenfalls auf 'django.contrib.gis.db.backends.postgis' geändert, da postgis auf der Datenbank verwendet wird und als Extension installiert werden muss. Mit psql über die Konsole oder PGAdmin4 können auf die lokale PostgreSQL Datenbank zugegriffen, Daten manuell gelöscht oder eingetragen werden.

4.5.6 Kartenmaterial

Die Daten für den Vegetationslayer wurden vom Kanton Zürich (maps.zh.ch) übernommen und werden von der Webapp als Polygone auf der Leaflet Map dargestellt. Sie werden vom Django Backend nach erfolgreichem Starten des Servers in die Datenbank geladen.

4.6 Tests

Testgetriebene Entwicklung (TDD) ist eine Methode zur Softwareentwicklung, welche oft bei der agilen Entwicklung von Computerprogrammen eingesetzt wird. Bei TDD erstellt der Programmierer Software-Tests, um das korrekte Verhalten von Softwarekomponenten zu planen und zu überprüfen. Unit Tests (auch als Unitest oder als Komponententest bezeichnet), werden verwendet um die funktionalen Einzelteile eines Softwareprojekts zu prüfen.

4.6.1 Tests in Django

Test für das Django wurden mit dem Standard Django Modul "unittest" erstellt. Es wurden Tests zu Login, Registrierung und Erstellung von Benutzeroberflächen erstellt. Es werden unvollständige, fehlerhafte und nicht autorisierte Requests (zum Beispiel ohne sich einzuloggen) getestet. HTTP-Statuscode geben Auskunft über die korrekte Ausführung oder fehlerhaftes Verhalten des Servers. Sie werden verglichen mit zu erwartenden Codes und bestimmen auf diese Weise ob der Test bestanden oder gescheitert ist.

Es wird getestet ob Email und Passwort vorhanden ist und ob die Anforderungen an die Werte erfüllt sind, ob Accounts und JWTs erstellt werden können und ob Benutzeroberflächen vom System korrekt erstellt werden.

Die Tests können mit dem Befehl "python3 manage.py test WaldmeisterMap" ausgelöst werden. Fehlerhafte Test werden von Django gemeldet.

4.6.2 Tests in Karma

Tests für Vue.js wurden mit dem Test Runner Karma und dem Testframework Jasmine erstellt. Sie können über den Befehl "npm run test" ausgeführt werden, aus dem "client" folder des Vue-Servers. Getestet werden Komponenten und deren Inhalt, welche von VueJS verwendet werden.

Damit Jasmin die Tests ausführen kann, muss der Code von Webpack zuerst kompiliert werden. Danach wird er in einer Browserumgebung (z.B Chrome oder Firefox) ausgeführt und das Testframework sammelt die zu testenden Werte und Verhalten. Es vergleicht diese mit gegebenen Werten und Zuständen, welche vom Programmierer erwartet werden. Stimmen sie überein, ist der Test bestanden. Stimmen sie nicht überein, ist der Test gescheitert und Jasmine meldet einen Fehler.

4.6.3 Linter

Linter werden als Werkzeuge eingesetzt, welche während der Entwicklung eine statische Analyse Quellcode ausführen.

Beim Linten von Code wird dieser auf gewisse Qualitätskontrollen oder Linting-Regeln geprüft. Linter helfen Entwicklern, Code auf höherem Qualitätsstandard zu schreiben, reduzieren die Anzahl Bugs, Syntax-Errors oder schlechte Formatierungen welche im Source-Code vorkommen. Dadurch wird auch die Lesbarkeit und Wartbarkeit des Codes erhöht. Vor allem bei Teamarbeiten ist das Einsetzen und

Durchsetzen von Styleguides sehr wichtig. Durch das Einsetzen von Lintern sparen Entwickler Zeit und Code-Reviews können schneller durchgeführt werden.

Eslint

Eslint ist ein Code-Linting Tool für JavaScript, welches dafür eingesetzt wird gewisse Regeln und Styleguides während der Entwicklung des Projekts zu überprüfen. Eslint (in Kombination mit einem Texteditor wie Sublime), macht auf Codestellen aufmerksam, welche nicht den Konfigurationen entsprechen. Eingesetzt wurde der AirBnB Styleguide. Das Frontend, entwickelt mit Vue.js ist mit JavaScript erstellt. Daher kann das gesamte Frontend mit ESLint gelintet werden.

flake8

Flake8 ist Linter für Python Projekte, welches dafür eingesetzt wird Code-Guidelines für Python Quellcode

4.6.4 Continuous Integration

Travis

Um Continuous mit automated build und testing zu realisieren, wurde Travis verwendet. Die Software Travis CI wurde 2011 in Berlin erstellt und 2013 veröffentlicht. Auf der Website travis-ci.org wird das Repo, welches auf Github gehostet wird, verknüpft. Jedes mal wenn auf das remote Repo (auf GitHub) gepusht wird, führt Travis die config in Form einer .yml Datei aus, welche im root Ordner des Repos hinterlegt ist. Sie installiert alle Programme und Dependancies auf einer Linux Maschine, bildet das Projekt und testet ob der Build in der aktuellen Version funktioniert. Es entsteht eine Version History mit dazugehöriger config. Travis meldet, ob der Build und die Tests erfolgreich durchgeführt werden konnten. Dies geschieht auch auf separaten Branches des Projekts.

Nachdem das Projekt erfolgreich gebaut worden ist, werden die erstellten Dockerimages von Travis auf DockerHub hochgeladen. Images, welche auf DockerHub gehostet werden können später von Docker gepulkt werden, zum Beispiel wenn das Projekt auf einem Server deployt wird.

	ci_pgport	ci	#44 failed	4 min 15 sec	
	Daniel Schmider		999ddd4	about an hour ago	
	ci	Continous Integration with python and vue.js	#42 passed	2 min 35 sec	
	Daniel Schmider		18bfa3e	about an hour ago	
	ci	ci	#40 passed	3 min 35 sec	
	Daniel Schmider		a284575	about 2 hours ago	
	ci	ci	#39 errored	12 min 31 sec	
	Daniel Schmider		8f331cc	about 3 hours ago	
	ci	ci	#38 errored	12 min 11 sec	
	Daniel Schmider		54ed094	about 3 hours ago	
	ci	ci	#37 errored	12 min 1 sec	
	Daniel Schmider		418df28	about 3 hours ago	
	ci	ci	#36 errored	12 min 24 sec	
	Daniel Schmider		45a47af	about 3 hours ago	
	ci	ci	#35 passed	1 min 28 sec	
	Daniel Schmider		f3d06b6	about 15 hours ago	

Abbildung 4.26: TravisCI Version History

```
{  
  "language": "python",  
  "python": 3.4,  
  "virtualenv": {  
    "system_site_packages": true  
  },  
  "addons": {  
    "postgresql": "9.5",  
    "apt": {  
      "packages": [  
        "postgresql-9.5-postgis-2.3"  
      ]  
    },  
    "firefox": "latest"  
  },  
  "before_install": [  
    "sudo apt-get -qq update",  
    "sudo apt-get install binutils libproj-dev gdal-bin python-gdal"  
  ],  
  "install": [  
    "pip install -r requirements.txt",  
    "(cd client && npm install)"  
  ],  
  "script": [  
    "python manage.py test WaldmeisterMap",  
    "(cd client && npm test)"  
  ],  
  "global_env": "MOZ_HEADLESS=1 PGPORT=5435",  
  "os": "linux",  
  "group": "stable",  
  "dist": "trusty"  
}
```

Abbildung 4.27: Travis config .yaml, Version 45

Default Branch						
 master	# 33 failed	7028d0c ↗				
33 builds						
 about 15 hours ago	Daniel Schmider					
Active Branches						
 ci_pgport	# 45 failed	1afa786 ↗				
2 builds	about an hour ago	Daniel Schmider				
 ci	# 42 passed	18bfa3e ↗				
8 builds	about 2 hours ago	Daniel Schmider				

Abbildung 4.28: Travis branches

4.7 Manuelle Tests

4.7.1 User-Szenario

Das User-Szenario ist ein möglicher Interaktionsablauf eines Users, welcher die Webapp "Waldmeister-Outdoors" verwendet. Es ist in die verschiedenen Schritte eingeteilt, welcher der User durchführt.

1. Aufrufen der URL
2. Pan Zoom der Map auf einen beliebigen Bereich
3. Aufruf der "About" Seite
4. Öffnen der Map-Legende in einem separaten Tab
5. Registrierung eines neuen Users, Eingabe von Benutzernamen und Passwort
6. Login mit dem neu erstellten Benutzer
7. Erstellung einer neuen privaten Benutzerfläche
8. Update der gerade erstellten Benutzerfläche
9. Erstellen einer neuen öffentlichen Benutzerfläche
10. Erstellen einer zweiten öffentlichen Benutzerfläche
11. Löschen einer öffentlichen Benutzerfläche
12. Ausloggen des Benutzers
13. Aufruf der Map im ausgeloggtem Zustand
14. Anzeige der öffentlichen Benutzerfläche
15. Registrierung eines zweiten Benutzers
16. Einloggen des zweiten Benutzers
17. Löschungsversuch einer öffentlichen Benutzerfläche, welche vom ersten Nutzer erstellt wurde
18. Erstellen einer öffentlichen Benutzerfläche
19. Update der erstellen Benutzerfläche, wechseln des Labels und öffentlich auf privat

4.7.2 Testfälle

Testfall A:

Aufruf der URL

Erwartet: Laden der Webapp

Erfüllt: Ja/Nein

Testfall B:

Pan und Zoom der Map

Erwartet: Kartenausschnitt bewegt sich durch drag drop, zoomen durch Mousewheel (Gesture bei mobilen Geräten) oder Zoom Buttons Erfüllt: Ja/Nein

Testfall C:

Registrierung und Login

Erwartet: Registrierung durch Eingabe eines noch nicht verwendeten Usernamens und Passworts, Login durch die selben Informationen Erfüllt: Ja/Nein

Testfall D:

Erstellen von Benutzerflächen

Erwartet: Erstellung einer beliebigen Benutzerfläche durch aktivieren des "Add" buttons. Eckpunkte des Polygons bezeichnen. Durch erneute Auswahl eines bereits existierenden Eckpunkts wird die Fläche geschlossen und Dialogbox zum speichern der Fläche öffnet sich. Erfüllt: Ja/Nein

Testfall D:

Update einer existierenden Benutzerfläche

Erwartet: Update einer existierenden Benutzerfläche, in dem im eingeloggtem Zustand eine eigene Benutzerfläche (öffentlich oder privat) ausgewählt wird.ndern der Eckpunkte der Benutzerfläche durch drag drop. Klick der Schaltfläche "Upd" öffnet Dialogbox "Update". User kann der Fläche ein neues Label geben oder Schaltfläche privat / öffentlich wechseln. Durch Klick auf "Update" wird die Fläche und das Label neu gezeichnet. Erfüllt: Ja/Nein

4.7.3 Reproduktion und Auswertung

Falls ein Testfall von einem Testuser nicht korrekt ausgeführt wurde, sollte der Zustand welcher nach dem Testfall vorliegt reproduziert werden, in dem alle Interaktionen, welche der User tatsächlich ausgeführt hat, festgehalten werden. Falls es sich um einen Bug handelt sollte ein issue erfasst werden. Falls der Fehler nicht durch einen Bug verursacht wurde, sollte es als negative User Experience festgehalten werden. Führt der Testfall in vielen Fällen zu einer negativen User Experience, sollte das Applikationsdesign verändert werden, Buttons, Labels oder Hints angepasst werden.

4.8 Resultate und Weiterentwicklung

4.8.1 Resultate

User können mittels der Webapp Waldmeister-Outdoors die Vegetationskundliche Karte erforschen und diese mit Ihrem momentanen Standort abgleichen. User können, nachdem sie sich registriert haben, private und öffentliche Benutzerflächen erfassen und sie auf einem zentralen Server hinterlegen. Dies können sowohl bisher nicht erfasste Waldstandorte, oder persönliche Orte und Flächen beschreiben, welche für Sie im Arbeitsalltag relevant sind. Solche Benutzerflächen können sie zwischen verschiedenen Geräten (Im Feld und im Büro) abrufen oder mit anderen Personen teilen, sobald sie die Fläche öffentlich und daher von allen Usern abrufbar machen.

Ebenfalls wurde erreicht die Geolocation auf der Karte darzustellen. Dies ist im Arbeitsalltag eine grosse Hilfe, und dient zur schnellen Orientierung und Auffindung von bestimmten eingetragenen Flächen und Orten im Feld.

Die Webapp kann sowohl auf Desktop Browsern wie von mobilen Geräten verwendet werden und verhält sich dank Vue.js responsive auf eine Änderung des Viewports, zum Beispiel wenn der Screen eines mobilen Geräts während der Verwendung gedreht wird.

Der Vegetationslayer und Benutzerflächen-Layer und deren Label können ein - und ausgeblendet werden, was zur Übersicht beitragen kann.

Alle Benutzeraccounts und deren Flächen sind im Django Backend ersichtlich und können als Superuser verändert oder gelöscht werden. Wird ein Benutzer aus der Datenbank gelöscht, werden alle von ihm erstellten Benutzerflächen ebenfalls automatisch gelöscht. Alle Passwörter von Benutzeraccounts werden verschlüsselt in der Datenbank gespeichert.

Zur automatischen Bestimmung des Ortabhängigen Waldstandorttyps wurde eine separate API entwickelt. Der Client schickt die Koordinaten an denen er sich befindet als Argumente im API Request mit, und erhält vom Backend eine Response, welche beinhaltet ob an diesem Ort ein Waldstandort eingetragen ist. Der Server berechnet dies anhand der Geometrien (Polygone) des Vegetationslayers. Befindet sich der Punkt innerhalb eines Polygons, wird der Waldstandortstyp zurück an den Client geschickt. Dies funktioniert auch, wenn der User sich ausserhalb der sichtbaren Map befindet, da diese Berechnung auf dem ganzen Datensatz des Vegetationslayers ausgeführt wird, nicht nur auf den Waldfächern welche momentan auf dem Kartenausschnitt des Frontends angezeigt werden.

4.8.2 Möglichkeiten der Weiterentwicklung

Vegetationsflächen per API Requests

Es wird noch ein statisches GeoJSON verwendet, welches die Vegetationsflächen beinhaltet, welche auf der Map zu jedem Zeitpunkt dargestellt werden. Dies beschränkt sich nicht auf dem momentanen Kartenausschnitt und führt zu einer erhöhten Ladezeit beim Aufruf der Webseite, da das JSON (ca. 1mb gross) als statische Datei an den Client geschickt wird. Besonders für mobile Geräte ist es sinnvoll, dies dynamisch zu gestalten und nur die benötigten Vegetationsflächen des Kartenausschnitts per API-Request an den Client zu schicken.

Auflistung der Benutzerflächen

Um die Lokalisierung von bereits erstellten Benutzerflächen zu erleichtern, wäre es von Vorteil, diese in einer geordneten Liste darzustellen, z.B. in alphabetischer Reihenfolge oder nach ihrem Erstellungsdatum. Klickt der User auf eine dieser Listeneinträge, würde die Map auf die Lage dieser Fläche fokussieren.

Eine solche Auflistung wäre als aufklappbares Menu denkbar, welches sich über einen Button im Header der Applikation auf - und zuklappen lässt.

Plus Codes

Plus Codes können verwendet werden, um Koordinaten in eine Zeichenfolge umzuwandeln. Es wird dazu verwendet, einem Ort auf der Welt gewissermassen eine Adresse zu geben, ähnlich wie eine Straßenadresse. Statt einen Punkt in geografische Breite und Länge zu beschreiben, ist die Zeichenfolge lediglich 7 oder 11 stellig (Eine Stelle ist dabei immer das namengebende + oder - Zeichen, welches die Zeichenkette auch von internationalen Postleitzahlen abhebt). Je nachdem, ob der Code einen Ort lokal oder weltweit beschreibt, können die vier ersten Ziffern weggelassen werden, da es ersichtlich ist, um welche weltweite Region (ca. 100 Kilometern) es sich handelt. Dies funktioniert ähnlich wie eine Vorwahl bei Telefonnummern, welche regional nicht benötigt wird. Die vollständige Zeichenfolge beschreibt einen Ort auf dem Planeten mit einer Genauigkeit von 14 auf 14 Metern. Es kann jedoch ein weiteres Zeichen angehängt werden, um die Genauigkeit auf drei auf drei Meter zu erhöhen. Plus Codes beschreiben immer eine Fläche, keinen genauen Punkt.

Plus Codes können offline en- oder dekodiert werden. [Plu]

Sie basieren auf Open Location Code (OLC), ein open-source Projekt, welches von Google in Zürich entwickelt wurde. Das Projekt ist auf Github gehostet unter <https://github.com/google/open-location-code>. Es existieren Libraries für JavaScript wie auch fr Python.

"Waldmeister - Outdoors" kann diese Codes gebrauchen, damit Experten unter sich einfacher Geolocations austauschen können. Die Webapp kann jeder Fläche einen solchen Code zuweisen, nachdem die Benutzerfläche vom User erstellt wurde. Der Code repräsentiert das geometrische Zentrum der Benutzerfläche, welches von Leaflet berechnet wird.

Gruppen

User von Waldmeister-Outdoors sollen Gruppen erstellen können und andere Benutzer einladen, um Flächen unter sich zu teilen. Dies hat zum Vorteil, dass die Benutzerflächen nicht öffentlich abrufbar, aber innerhalb einer Gruppe sichtbar sind. Die Gruppe kann so über mehrere Geräte gleichzeitig über eine Benutzerfläche diskutieren oder sie auch bearbeiten, bevor sie veröffentlicht wird.

Das Kreieren, Betreten oder Verlassen einer Gruppe sollte möglichst einfach über einen Punkt im Menu möglich sein. Der User, welcher die Gruppe erstellt hat, sollte dabei Einladungen verschicken bzw. Gruppenmitglieder wieder aus der eigenen Gruppe löschen können. Denkbar ist auch, anderen Gruppenmitgliedern Rechte zu erteilen, andere User einzuladen.

Pfade und Orte

Neben Flächen können auch Pfade und Orte eine unterstützende Funktion im Feld darstellen. Um Pfade zu erstellen, können beispielsweise vergangene Geolocations des Users zu einem Pfad zusammengefügt werden, welcher die zurückgelegte Strecke darstellt. Mithilfe von Timestamps wäre es möglich, den Tagesablauf zu protokollieren und sie mit einer Tätigkeit an einem Ort zu verknüpfen.

Orte, sogenannte Points of Interests (PoI), könnten Teammitglieder oder andere Experten auf bestimmte Indikatoren, wichtige örtliche Gegebenheiten hinweisen, den Standort des nächstgelegenen Parkplatzes oder eine unterbrochene Zufahrtsstrasse kennzeichnen. Diese Orte müssen nicht durch eine Fläche beschrieben werden, sondern nur durch einen einzigen zweidimensionalen Punkt.

Zu Orten, bzw auch Flächen könnte ein Bild hochgeladen werden, um die Stelle genauer zu kennzeichnen, oder um mit anderen Experten über den Standort zu diskutieren.

Export von erfassten Benutzerflächen

Um die Handhabung von erhobenen Daten zu erleichtern, wäre die Möglichkeit erwünscht, alle erstellten Benutzerflächen eines Users (oder einer Gruppe) in einem speziellen Format zu exportieren, damit sie auf alternativen Wegen ausgetauscht und verschickt werden können.

GUI Verbesserungen

Das Frontend von Waldmeister-Outdoors kann vor allem innerhalb der Map Komponente von VueJS verbessert werden. Labels, welche zu Waldstandorte auf dem Vegetationslayer oder Benutzerflächen angezeigt werden sollten Zoom-abhängig ein und aus-geblendet werden. Dies hat neben Performanzvorteilen auch Vorteile der Übersicht, da sich die Labels bei kleinen Zoomstufen berlagern und unlesbar werden. Obwohl sich die Waldstandorte des Vegetationslayers nicht berlagern, sollten auch sie zu größeren Flächen zusammengefasst werden, bzw. ausgeblendet werden, wenn die Geometrien der Polygone sich zu nahe beinander befinden.

4.9 Projektmanagement

4.9.1 Entwicklungswerkzeuge

Als Texteditor, bzw. IDE wurde Sublime eingesetzt welches Linter für JavaScript und Python unterstützt. Sublime ist open-source und stützt sich auf viele Pakete, welche im integrierten Package-Manager heruntergeladen und konfiguriert werden können.

GitHub wurde verwendet um eine Versionierung des Programmcodes zu ermöglichen. Issues werden auf GitHub erstellt.

4.9.2 Prozessmodell

Scrum wurde eingesetzt um bei der Entwicklung des Projekts agil vorzugehen. Scrum und agile Development erhöhen die Transparenz während der Entwicklungszeit und ermöglichen dem Kunden oder

Auftragsgeber flexibler Anpassungen am Projekt vorzunehmen. In der Entwicklungszeit wurde Scrum in sehr vereinfachter Form umgesetzt. Sprints wurden in 1 oder 2 wchentlichen Abständen Reviewt und die Fortschritte (Was wurde erreicht, wo gab es Probleme, was sind die nächsten Schritte) wurden per Mail dokumentiert. Beschlussprotokolle wurden auf das HSR Wiki eingetragen.

Der Product Backlog wurde auf GitHub auf einem Projektboard nachgeführt, auf welchem alle projektrelevanten Issues aufgeführt sind.

Docker-Deployment bei Sprintende (als Testsystem ?)

GitHub

4.9.3 Zeitplan

Eine grobe Planung wurde Anfang April festgehalten. Es wurde eine Meilensteinplanung sowie eine Aufwandschätzung und ein Projektplan als Gantt-Diagramm festgehalten. Issues wurden am Anfang des Projekts bereits einige definiert, wurden jedoch laufend im GitHub Projektboard auf dem neusten Stand gehalten.

4.9.4 Meilensteinplanung

Meilensteine beschreiben Wichtige geplante Daten im Projektablauf. Im Projekt "Waldmeister-Outdoors" wurden folgende Meilensteine definiert:

1. Projektplan erstellen 18.4.
2. Refactoring der Codebase, 2.5.
3. Feature Freeze / Feature Demo 23.5.
4. Abgabe Projekt 8.6.

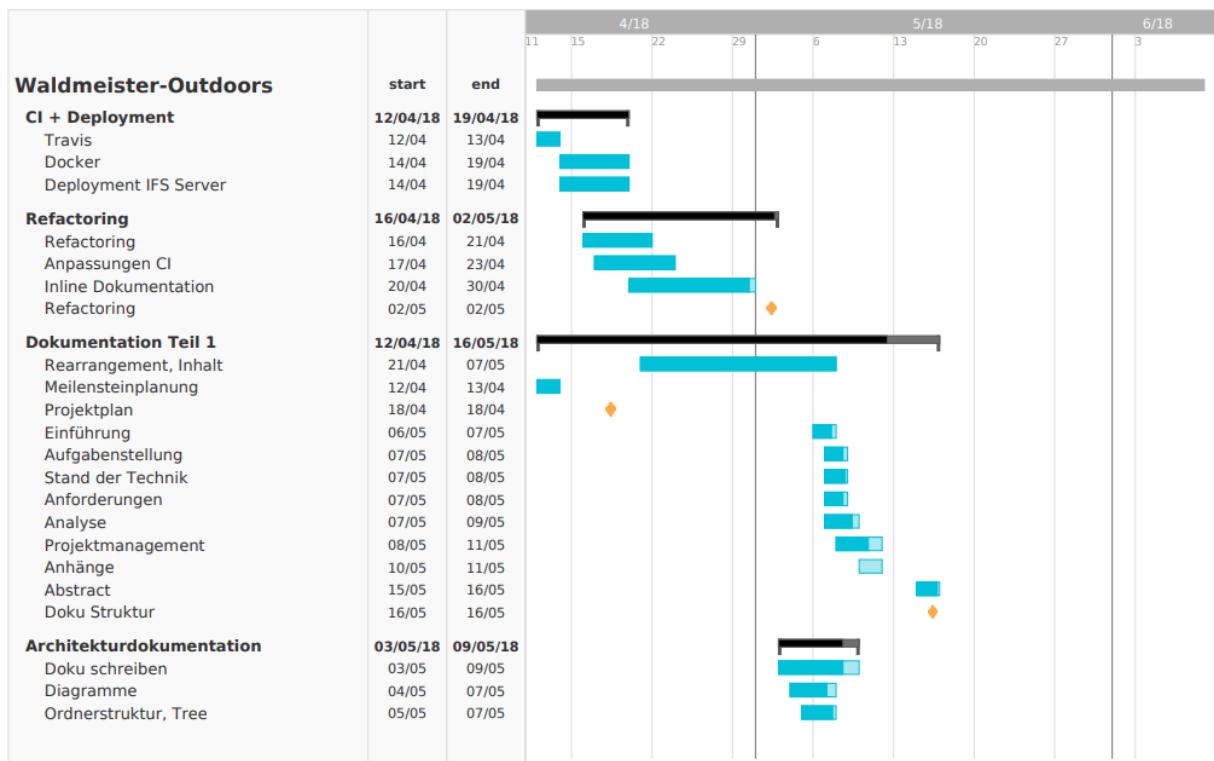


Abbildung 4.29: Gantt Diagramm Teil 1



Abbildung 4.30: Gantt Diagramm Teil 2

4.9.5 Issues

Während dem Projekt und insbesondere während der Projektplanungsphase werden Issues erfasst, welche geplante Schritte, Funktionen oder Arbeitsschritte beschreiben. Issues sind in die Kategorien "To Do", "In Progress", "Ready to Review" und "Done and Reviewed" unterteilt. Issues können während der Entwicklung mehrmals Ihre Kategorie wechseln, zum Beispiel bei einem Review wieder in die Kategorie "To Do" eingeteilt werden. Issues haben einen "Label", welche sie einer Gruppierung wie "Bug", "Must Have" oder "Dokumentation" zuordnen.

Issues sind auch einem Meilenstein zugeordnet, damit sie

4.9.6 Releases

Als Release wird das Deployment der Webapp auf den Produktions Server bezeichnet. Grundsätzlich wird die Webapp auf einem Lokalen Testserver entwickelt und getestet. Bei einem Release wird die Lauffähige Webapp auf das Livesystem portiert und öffentlichen Usern zugänglich gemacht.

4.9.7 Risiken

Technologien Frameworks

R01: Mangelnde Erfahrung mit VueJS	
Beschreibung	Wirkt sich negativ auf Design und Programmcode aus.
Schadenspotential	30h
Eintrittswahrscheinlichkeit	80%
Auswirkung	VueJS wird als Frontend Framework eingesetzt. Es umfasst die ganze visuelle Darstellung der Webapplikation und ist dafür zuständig, dass die Navigation und Aufruf der API Requests korrekt ausgeführt wird.
Vorbeugung	Vue kann an vielen Orten eingesetzt werden um visuelle oder funktionelle Anforderungen zu bernehmen. Mittels JavaScript Libraries wie Vuetify kann bei der visuellen Darstellung auf Vorgefertigte Komponenten zurückgegriffen werden.

Tabelle 4.1: VueJS Risiken

R02: Fehlende Erfahrung mit Python / Django	
Beschreibung	Wirkt sich negativ auf Performance und Security aus
Schadenspotential	20h
Eintrittswahrscheinlichkeit	80%
Auswirkung	Das Django-Backend basiert auf Python. Das Frontend fragt Informationen zu Login und Benutzerflchen beim Backend ab. Rest-Schnittstellen und Serializer werden vom Backend bereitgestellt und mssen korrekt funktionieren, damit keine Sicherheitslcken entstehen. Bei einer falschen Konfiguration des Backends knnen auch geschwindigkeitsrelevante Probleme beim Frontend auftreten.
Vorbeugung	Python Pakete verwenden, welche die Authorisierung von Usern korrekt ausfhren und die geschwindigkeitsrelevante API Abfragen in realen Umgebungen testen und ggf. verbessern. Auf Tutorial und Guides setzen um das API zu erstellen.

Tabelle 4.2: Django / Python Risiko

R03: Fehlende Erfahrung mit Leaflet & Leaflet Editable	
Beschreibung	Wirkt sich negativ auf Usability und Funktionalitt aus
Schadenspotential	30h
Eintrittswahrscheinlichkeit	70%
Auswirkung	Ein Grossteil der User-Interaktion findet beim Frontend auf dem Map Objekt statt. Das Erstellen, bzw. Bearbeiten von Benutzerflchen wird von Leaflet Editable bereitgestellt, muss jedoch korrekt implementiert werden, damit die Zusammenarbeit mit dem Backend korrekt funktioniert.
Vorbeugung	Leaflet Beispiele verwenden um Benutzerflchen und GeoJSON Layer korrekt darzustellen. Leaflet-editable Code korrekt implementieren damit beim Aufruf von API Abfragen keine Fehler passieren. Bestehende Features testen bevor neue Features implementiert werden. Codebase oft Refaktorieren. Zusammenarbeit mit VueJS berprfen.

Tabelle 4.3: Risiko Leaflet Leaflet-editable

R04: Leistung von mobilen Gerten schwcher	
Beschreibung	Wirkt sich negativ auf Usability aus
Schadenspotential	15h
Eintrittswahrscheinlichkeit	80%
Auswirkung	Mobile Gerte haben Schwierigkeiten viele Objekte auf der Leaflet Map darzustellen. Waldstandortsflchen des Vegetationslayers knnen das Gert berfordern, auch wenn die Darstellung und Usability bei Desktopgerten gewhrleistet ist.
Vorbeugung	Anzahl an Objekten, welche auf der Leaflet Map dargestellt werden reduzieren. Flchen welche nicht im Kartenausschnitt sichtbar sind nicht laden um das Mobile Gert nicht zu berfordern. Komplexitt (Genauigkeit) der Polygone des Vegetationslayers reduzieren

Tabelle 4.4: Risiko Leistung Mobile Gerte

4.9.8 Zusammenfassung Risiken

Aus den Risiken gehen Schadenspotentiale hervor, welche in der Projektplanung bercksichtigt werden mssen. Da die Risiken aus Technologien hervorgehen, ist vorhersehbar in welchem Projektabschnitt sie auftreten knnten. Neben einer Reservezeit werden diese Risiken vorallem dazu verwendet den jeweiligen Meilensteinen mehr Zeit zuzurechnen.

Risiko 1:

$$30 * 0.8 = 24$$

Risiko 2:

$$20 * 0.8 = 16$$

Risiko 3:

$$30 * 0.7 = 21$$

Risiko 4:

$$15 * 0.8 = 12$$

Total: 73 Stunden

Um diesen Risiken vorzubeugen wurde vor den Sprints abgeklrt und getestet wie relevant die Risiken bei der Implementation sein werden. Falls die Risiken bei der Abklrung auftauchen wurden die Sprints in kleinere Teile zerlegt bzw. mehr Zeit eingeplant.

4.10 Projektmonitoring

4.10.1 Soll-Ist-Zeitvergleich

4.10.2 Code Statistics

4.11 Softwaredokumentation

4.11.1 Installation

Um die App zu verwenden, sollte die Anleitung auf GitHub (Readme) verwendet werden, oder die .yml config Datei, um das Projekt auf einer Linux Machine zu builden.

Github Repo klonen

"npm install"

"pip install -r requirements.txt"

"npm start" im client folder startet den Vue-Server

"python manage.py runserver" (als separates Window oder Tab) im root Folder startet den Django Server

Damit Django die PostgreSQL Datenbank verwenden kann, müssen die Werte in der settings.py angepasst werden, oder eine Datenbank mit dem Namen dschwaldmeister, user Postgres und Port 5432 erstellt werden. Von PostgreSQL soll auch die Extension postgis erzeugt werden.

Mit "manage.py migrate" kann die Datenbank auf den neusten Stand gebracht werden, damit sie verwendet werden kann.

Die App kann im Browser unter der angezeigten Adresse des Vue-Servers aufgerufen werden.

Alternativ kann das Projekt mit dem Befehl "docker-compose build" in Dockercontainern gebaut und mit "docker-compose up" ausgeführt werden.

Das Projekt ist im Internet unter <https://waldmeistermap.sifs0003.infos.ch/> deployt.

4.11.2 Tutorial, Handbuch

Registrierung:

Um einen neuen Benutzer zu registrieren, muss ein bereits eingeloggter User sich ausloggen. Danach kann mit der Schaltfläche "Register" ein neuer Account erstellt werden. Username und Passwort werden benötigt, Email ist optional und kann leer gelassen werden.

Fläche Erstellen:

Ein eingeloggter User hat die Möglichkeit eine Neue Benutzerfläche zu erstellen. Um dies zu tun, muss

er auf die Schaltfläche "Add" auf der linken Seite, unterhalb der Zoom Buttons klicken. Im Editiermodus kann er durch Klicks auf die Map neue Eckpunkte zu einem Polygon hinzufügen. Er kann die Fläche schliessen, in dem er auf den ersten erstellten Eckpunkt klickt. Es öffnet sich eine Dialogbox, in welchem der User der erstellten Fläche einen Namen geben kann, welcher auf der Map erscheint. In dieser Dialogbox kann der User auch entscheiden, ob die Fläche öffentlich (für alle sichtbar) oder privat dargestellt werden soll (nur für den User sichtbar, welcher die Fläche erstellt hat).

Fläche Bearbeiten:

Um eine bereits erstellte Fläche zu bearbeiten und zu verändern muss die zu ändernde Fläche ausgewählt werden, während der User, welcher die Fläche erstellt hat, eingeloggt ist. Nur der User, der die Benutzerfläche erstellt hat, kann diese Verändern oder Updaten. Im Bearbeitungsmodus kann der User die Eckpunkte des Polygons beliebig verschieben. Durch einen Klick auf die Schaltfläche "Upd" (Update), öffnet sich die Dialogbox, um der Benutzerfläche einen neuen Namen zu geben, oder Ihren Zustand von öffentlich oder privat zu wechseln.

Fläche Löschen:

Im eingeloggten Zustand auf eine mit diesem User erstellte Benutzerfläche klicken. Durch Klick auf den Button "Del" am linken Rand der Map wird die ausgewählte Fläche gelöscht.

Abbildungsverzeichnis

2.1 Use Cases	6
2.2 ESRI Webapp Collector for ArcGIS	7
2.3 Erfassung einer neuen Benutzerfläche	8
3.1 Prototyp mit ArcGIS online	12
4.1 Use Case Diagram	18
4.2 Domain Modell	20
4.3 Vuex Action-Mutations-State Diagram	25
4.4 Vuex private	26
4.5 Sequenzdiagramm, Register	28
4.6 Sequenzdiagramm, Login	28
4.7 Sequenzdiagramm, Public Areas	29
4.8 Sequenzdiagramm, My Areas	29
4.9 Komponentendiagramm, Waldmeister Map	30
4.10 Mockup Screen 1, Anzeige des Eigenen Standorts auf der Map	31
4.11 Mockup Screen 2, öffnen des Menus, Login	31
4.12 Mockup Screen 3, Eingeben der Accountdetails	32
4.13 Mockup Screen 4, Map Ansicht nach dem Login	32
4.14 Mockup Screen 5, Erfassung eines Points of Interests	33
4.15 Mockup Screen 6, Erfassung einer Benutzerfläche	33
4.16 Mockup Screen 7, Auflistung aller Benutzerflächen dieses Users	34
4.17 Mockup Screen 8, Detailansicht eines selektierten Points of Interests	34
4.18 Mockup Screen 9, Detailansicht einer selektierten Benutzerfläche	35
4.19 About in der Waldmeister-Outoors Webapp	37
4.20 Einfärbung von GeoJSON Flächen und Darstellung von PropertyLabels	38
4.21 Benutzerflächen mit Labels	39
4.22 Geolocation mit Circle	40
4.23 Swagger API Authorization	42
4.24 Swagger API UserAreas	43
4.25 Swagger API user create Details	43
4.26 TravisCI Version History	47
4.27 Travis config .yaml, Version 45	48
4.28 Travis branches	49

4.29 Gantt Diagramm Teil 1	55
4.30 Gantt Diagramm Teil 2	56

Glossar

Begriffserklärung

Waldmeister-Outdoors:

Name der Webapp welche in dieser Projektarbeit erstellt wurde.

Waldmeister Map:

Name der Leaflet Map, welche innerhalb der Applikation dargestellt wird.

Waldmeister App:

Native App iOS und Android App, welche im Appstore und Android Playstore erhältlich ist.

Waldstandort:

Eine Fläche auf der Erdoberfläche, welche einen Wald bezeichnet.

Waldstandorttyp:

Einer von vielen Typen, welcher einen Waldstandort einen bestimmten Typ zuordnet. Meist wird hier der Kürzel aus den Definitionen von EK72 verwendet, z.B. Kürzel "7e". Der Name des Waldstandorttyps "7e" ist "Waldmeister-Buchenwald mit Hornstrauch"

Vegetationslayer:

Erster Layer oberhalb der Hintergrundkarte, welcher auf der Map dargestellt wird. Enthält eingefärbte Polygone mit Labels welche einen Waldstandorttyp beschreiben. Die Daten, welche auf dem Vegetationslayer dargestellt werden, basieren auf der "Vegetationskundliche Kartierung der Wälder im Kanton Zürich", bzw. der "Waldvegetationskarte". Library:

Eine Library (Programmbibliothek) bezeichnet in der Programmierung eine Sammlung von Unterprogrammen, die Lösungen anbieten. Bibliotheken laufen im Unterschied zu Programmen nicht eigenständig, sondern sie enthalten Hilfsmodule, die angefordert (importiert) werden können

Framework:

Ein Framework ist noch kein fertiges Programm. Es ist ein Rahmen (ein Gerüst) welches der Programmierer verwenden kann um sein eigenes, persönliches Programm zu erstellen

Akronyme

SRID = Spatial Reference Identifier

GIS = Geografisches Informationssystem

MVC = Model - View - Controller

MVVM = Model - View - ViewModel

HTML = Hypertext Markup Language

CSS = Cascading Style Sheets

ES5 = ECMA Script 5

ECMA = European Computer Manufacturers Association

SoC = Separation of Concern

SPA = Single-Page-Applikation

EK72 = Ellenberg Klötzli

POI = Points of Interests

UML = Unified Modeling Language

JWT = (JSON Web Token)

JSON = JavaScript Object Notation

REST = Representational state transfer

CRUD = Create, Read, Update, Delete

DRF = Django Rest-Framework

CI = Continuous Integration

Gdal = GDAL - Geospatial Data Abstraction Library

IFS = Institut fr Software

Literaturverzeichnis

[AH07] Jacob Kaplan-Moss Adrian Holovaty. *The Definitive Guide to Django*. Apress, 2007.

[djo] Djoser, rest implementation of django authentication system.
<https://github.com/sunscrapers/djoser>.

[Geo] Geojson geometry lookup
<https://github.com/simonepri/geojson-geometries-lookup>.

[git] Github projektboards automatisierung
<https://help.github.com/articles/about-automation-for-project-boards/>.

[Har01] Jens Hartwig. *PostgreSQL - professionell und praxisnah*. Addison-Wesley, 2001.

[JF08] Wesley Chun Jeff Forcier, Paul Bissex. *Python Web Development with Django*. Addison-Wesley Professional, 2008.

[Plu] Plus codes website fuer developer
<https://plus.codes/developers>.

[Ser] Introduction to service worker
<https://developers.google.com/web/fundamentals/primers/service-workers/>.

[Vue] Vue single file components
<https://vuejs.org/v2/guide/single-file-components.html>.

[ZHG] Geometadaten gis-zh - geolion
http://www.parcs.ch/wpz/pdf_public/2013/9601_20131015_131512_gds_110.pdf.