

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

## Waldmeister - Outdoors

Ein Werkzeug zur Erfassung und Publikation von Waldstandorten

HSR Hochschule für Technik Rapperswil

Fall 2017

*Author:* Daniel Schmider  
*Supervisor:* 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, relevante Informationen zu einem Standort beinhalten. Mithilfe der Geolocation wird die eigene Position bestimmt. Umliegende, bereits erfasste Waldstandorte und Benutzerflächen werden auf verschiedenen Ebenen auf einer Leaflet-basierten Karte angezeigt.

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

# Inhaltsverzeichnis

<b>1 Abstract</b>	<b>1</b>
<b>2 Management summary</b>	<b>5</b>
2.1 Ausgangslage . . . . .	5
2.2 Ziel der Arbeit . . . . .	5
2.3 Ergebnisse . . . . .	5
2.4 Ausblick . . . . .	6
<b>3 Ausgangslage und Vision</b>	<b>7</b>
3.1 Ein mobiles App für Feldforschung im Wald . . . . .	7
3.2 Use Cases . . . . .	8
3.2.1 Zugriff auf Vegetationskundliche Karten . . . . .	8
3.2.2 Bearbeitung der Vegetationskundlichen Karten . . . . .	8
3.2.3 Unterstützung während der Untersuchung . . . . .	8
3.2.4 Automatische Bestimmung eines Waldstandorts . . . . .	9
3.3 Mobile Limits . . . . .	9
<b>4 Projektmanagement</b>	<b>10</b>
4.1 Git, Github . . . . .	10
4.2 Projektboards . . . . .	11
4.3 Issues . . . . .	12
4.4 Continuous Integration . . . . .	12
<b>5 Anforderungen, Technologien</b>	<b>16</b>
5.1 Anforderungsspezifikationen . . . . .	16
5.2 Progressive Webapp . . . . .	16
5.3 Mobile Field Solution mit ArcGIS Online . . . . .	17
5.4 Vue.JS . . . . .	18
5.4.1 Separation of Concern . . . . .	20
5.4.2 Vue-Router . . . . .	20
5.4.3 Vuex . . . . .	20
5.4.4 Vuetify . . . . .	24
5.5 Axios . . . . .	24
5.6 Django . . . . .	24

5.6.1	Django Rest Framework . . . . .	24
5.7	PostgreSQL . . . . .	25
5.7.1	PostGIS . . . . .	25
5.8	Leaflet . . . . .	25
5.8.1	TileLayer, Hintergrundkarte . . . . .	25
5.8.2	GeoJSON Layer, Vegetationskundliche Karte . . . . .	25
5.8.3	Leaflet editable . . . . .	26
5.8.4	HTTPS, Geolocation . . . . .	26
<b>6</b>	<b>Implementation</b>	<b>27</b>
6.1	Mockup . . . . .	27
6.2	Modellierung mit UML Diagrammen . . . . .	37
6.2.1	Use Case - Diagramm . . . . .	37
6.2.2	Klassendiagramm, Datenbankdiagramm . . . . .	37
6.2.3	Sequenzdiagramm . . . . .	38
6.2.4	Komponentendiagramm . . . . .	41
6.3	Technische Implementation . . . . .	43
6.3.1	VueJS Komponenten . . . . .	43
6.3.2	WaldmeisterMap . . . . .	43
6.3.3	Login und Registrierung . . . . .	43
6.3.4	About . . . . .	44
6.3.5	User Interface / Frontend . . . . .	45
6.3.6	Axios Requests . . . . .	48
6.3.7	Database Models . . . . .	48
6.3.8	Database, PostgreSQL . . . . .	48
6.3.9	API Dokumentation, Swagger . . . . .	49
6.4	Tests . . . . .	55
6.4.1	Unit Tests, TDD . . . . .	55
6.4.2	VueJS . . . . .	55
6.4.3	Django . . . . .	55
<b>7</b>	<b>Resultate</b>	<b>56</b>
7.1	Erreichte Ziele . . . . .	56
7.2	Verbesserungen, Weiterentwicklungen . . . . .	57
7.2.1	Automatische Standortbestimmung . . . . .	57
7.2.2	Editieren und Löschen von erstellten Benutzerflächen . . . . .	57
7.2.3	Auflistung der Benutzerflächen . . . . .	57
7.2.4	Gruppen . . . . .	57
7.2.5	Pfade und Orte . . . . .	58
7.2.6	Continuous Tracking . . . . .	58
7.2.7	Plus Codes . . . . .	58
7.3	Screenshots . . . . .	60

<b>8</b>	<b>Softwaredokumentation</b>	<b>62</b>
8.1	Installation . . . . .	62
<b>9</b>	<b>Links</b>	<b>63</b>
9.0.1	Projektrelevante Links . . . . .	63

# **Kapitel 2**

## **Management summary**

### **2.1 Ausgangslage**

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.

### **2.2 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.

### **2.3 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.

## 2.4 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 und die Arbeit im Feld ist unerlässlich. "Waldmeister - Outdoors" kann im Berufsalltag sowie bei der Kommunikation mit Institutionen den Arbeitsfluss beschleunigen. Weitere Features 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.

# **Kapitel 3**

## **Ausgangslage und Vision**

### **3.1 Ein mobiles App für Feldforschung im Wald**

Aufbauend auf der existierenden "Waldmeister" App, welches von Studenten und Professoren als Nachschlagewerk für Waldstandortbestimmungen in der Praxis verwendet wird, existieren viele, teils digitalisierte Karten, welche den Stand der Forschung in sogenannten Vegetationskundlichen Karten beschreiben. Waldstandortbestimmung ist ein sich ständig im Wechsel befindendes Thema. Standorte oder deren Befund können sich je nach Standort ständig ändern. Vegetationskundliche Karten, welche von Experten im Auftrag des Kantons angefertigt werden, befinden sich in statischen oder ungewarteten Zuständen in Archiven des Kantons, oder werden in grossen Intervallen (5-10 Jahre) revidiert. Teilweise sind diese Daten daher gar nicht, oder nur oder in sehr veraltetem Zustand zugänglich. Dies liegt hauptsächlich am grossen Kostenaufwand, welche eine Analyse im Feld durch Experten mit sich bringt. Der Datenfluss ist geprägt von analogen Vorgängen, vor allem da viele technische Geräte nicht für den Einsatz im Feld geeignet sind und kantonale Organisationen als Mittelsmänner etabliert sind, welche die Daten von Experten archivieren und ggf. in digitaler Form veröffentlichen.

"Waldmeister - Outdoors" zielt darauf hin, den Arbeitszyklus der Analyse und Publikation von Waldstandorten zu vereinfachen und zu beschleunigen. Dies soll erreicht werden durch neue technische Möglichkeiten und digitale Medien wie dem Smartphone, GPS und Mobiles Internet.

Das Instrument "Waldmeister - Outdoors" soll auch dazu verwendet werden, die Erfassung von Geoinformationsdaten bezüglich der Waldstandortbestimmung zu standardisieren und deren Übermittlung an die zuständige Behörde, Forschern und anderen Experten zu beschleunigen. Anstelle eines jahrelangen Projekts, einen bestimmten Standort in Waldstandorte zu kartieren, soll ein System entwickelt werden, welches eine inkrementelle digitale Erfassung und Publikation ermöglicht, den Arbeitsaufwand der Experten erleichtert und den Informationsaustausch, und -abgleich beschleunigt.

## **3.2 Use Cases**

### **3.2.1 Zugriff auf Vegetationskundliche Karten**

Während der Feldforschung kann es sehr hilfreich sein, auf bereits kategorisierte 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 Webseiten 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 dazu beitragen in dem die eigene Position in der Karte eingetragen werden soll und die Map darauf zentriert wird. Zusätzlich soll es möglich sein, dies über einen bestimmten Intervall zu wiederholen, bzw. die eigene Position als Pfad zu speichern. Diese Funktionen sind jedoch stark abhängig von der Genauigkeit des GPS, welche innerhalb eines Waldes relativ oft ungenau sein kann, und sollten daher deaktivierbar sein.

### **3.2.2 Bearbeitung der Vegetationskundlichen Karten**

Kantonale Vegetationskundliche Karten sind statisch (d.h. Read-Only) und können von Usern nicht bearbeitet oder erweitert werden. Dies führt dazu, dass Kartenmaterial in veraltetem Zustand vorliegt und Fehler oder Veränderungen nur mit grossem Aufwand upgedatet werden können. Nicht nur führt dies zu Problemen bei der Kommunikation mit anderen Experten, Forschern und Studenten, es behindert auch den Arbeitsfluss der Person, welche sich im Feld befindet. Da andere Mittel zur temporären Festhaltung der Ergebnisse verwendet werden, um an einem späteren Zeitpunkt wieder darauf zugreifen zu können, kann dies umständlich werden. Zum Beispiel einer halbjährlichen Untersuchung eines Standorts. Dies geschieht oft analog, auf Papier im Feld und muss erst zu einem späteren Zeitpunkt zum persönlichen Gebrauch digitalisiert werden. Das führt zu einem erhöhten Arbeitsaufwand. Durch die Verwendung von "Waldmeister - Outdoors" kann das gleiche Werkzeug benutzt werden, um eine Fläche während der Untersuchung sowohl im Feld als auch im Büro zu beschreiben, sowie die finalen Befunde am Ende der Untersuchung zu publizieren und zu teilen.

### **3.2.3 Unterstützung während der Untersuchung**

Feldforschung hat oft mit der Orientierung und der Manövrierung des Standorts an sich zu tun und auch hier kann "Waldmeister - Outdoors" den Arbeitsaufwand simplifizieren und reduzieren. Durch die direkte digitale Erfassung von beliebigen Notizen bezüglich des Standorts müssen diese nicht mehr analog erfasst werden und können direkt der Position in der realen Welt zugeordnet werden. Ist beispielsweise ein Gebiet schwer befahrbar oder schwer zugänglich, kann dies direkt auf der digitalen Karte erfasst und für persönliche Zwecke gespeichert werden. Solche Daten können aber ebenfalls publiziert werden falls sie für Andere von Interesse sind. Es kann sich hierbei auch um Pfade oder einzelne Standorte von Indikatoren handeln, bzw. eine genaue Lage der Observationsfläche, welche untersucht werden soll.

### **3.2.4 Automatische Bestimmung eines Waldstandorts**

Durch den Zugriff auf eine Datenbank, in welcher jeder Waldstandort einem Typ zugeordnet ist, kann mithilfe des Mobilen Geräts der Typ des Waldstandorts in welcher sich ein User gerade befindet, automatisch bestimmt werden. Dies kann mithilfe des GPS Sensors des Mobilen Geräts und einer Datenbankabfrage zu jedem Zeitpunkt geschehen in dem überprüft wird, ob die momentan per GPS ermittelte Position innerhalb einer Fläche liegt, welche einen Waldstandort beschreibt. Dies kann über einen Intervall stetig wiederholt werden und somit dem Benutzer ohne jegliche Aufforderung darüber informieren, in welchem Gebiet er sich befindet.

## **3.3 Mobile Limits**

In vielen Fällen ist eine Standortbestimmung durch GPS im Wald sehr ungenau und die Internetverbindung kann instabil sein. Im Idealfall überträgt das Werkzeug so wenig Daten wie möglich, speichert diese auf dem Gerät und wartet auf eine stabile Verbindung, um Daten auf den Server zu übertragen. Kartenmaterial sollte wenn möglich permanent auf das Mobile Gerät geladen werden können, damit Datenvolumen bei der Verwendung im Feld nicht strapaziert werden.

# **Kapitel 4**

## **Projektmanagement**

### **4.1 Git, Github**

Git habe ich zur Versionenkontrolle des Projekts verwendet. Versionenkontrolle erlaubt es, verschiedene Versionen eines Projekts zu haben und zeigt die Änderungen, welche im Code über Zeit gemacht wurden. Git erlaubt es auch, Änderungen rückgängig zu machen und ist vor allem bei grösseren Projekten von unerlässlichem Nutzen. Bei Git, anstelle eines zentralen Servers, welcher die aktuellste Version hostet, haben bei Git alle beteiligten Entwickler die gesamte "version history" des Projekts auf all Ihren Geräten, statt nur die aktuellste Version, welche sie Lokal gespeichert haben. User haben jederzeit Zugriff auf alle Versionen aller Dateien im Git-Repository. Mehrere Entwickler können gleichzeitig an einem Projekt arbeiten, ohne sich gegenseitig zu stören, und haben keine Angst, die von einem Kollegen vorgenommenen Änderungen zu verlieren. In Git sind die Möglichkeiten der Zusammenarbeit unbegrenzt. Git ist gratis und open-source.

Github ist ein Remote Server für Git Projekte, welcher gleichzeitig ein community Hub für Entwickler ist. Auf Github können Repositories mit einem grafischen web Interface bearbeitet werden, und Entwickler können sich über Projekte informieren und teilnehmen.

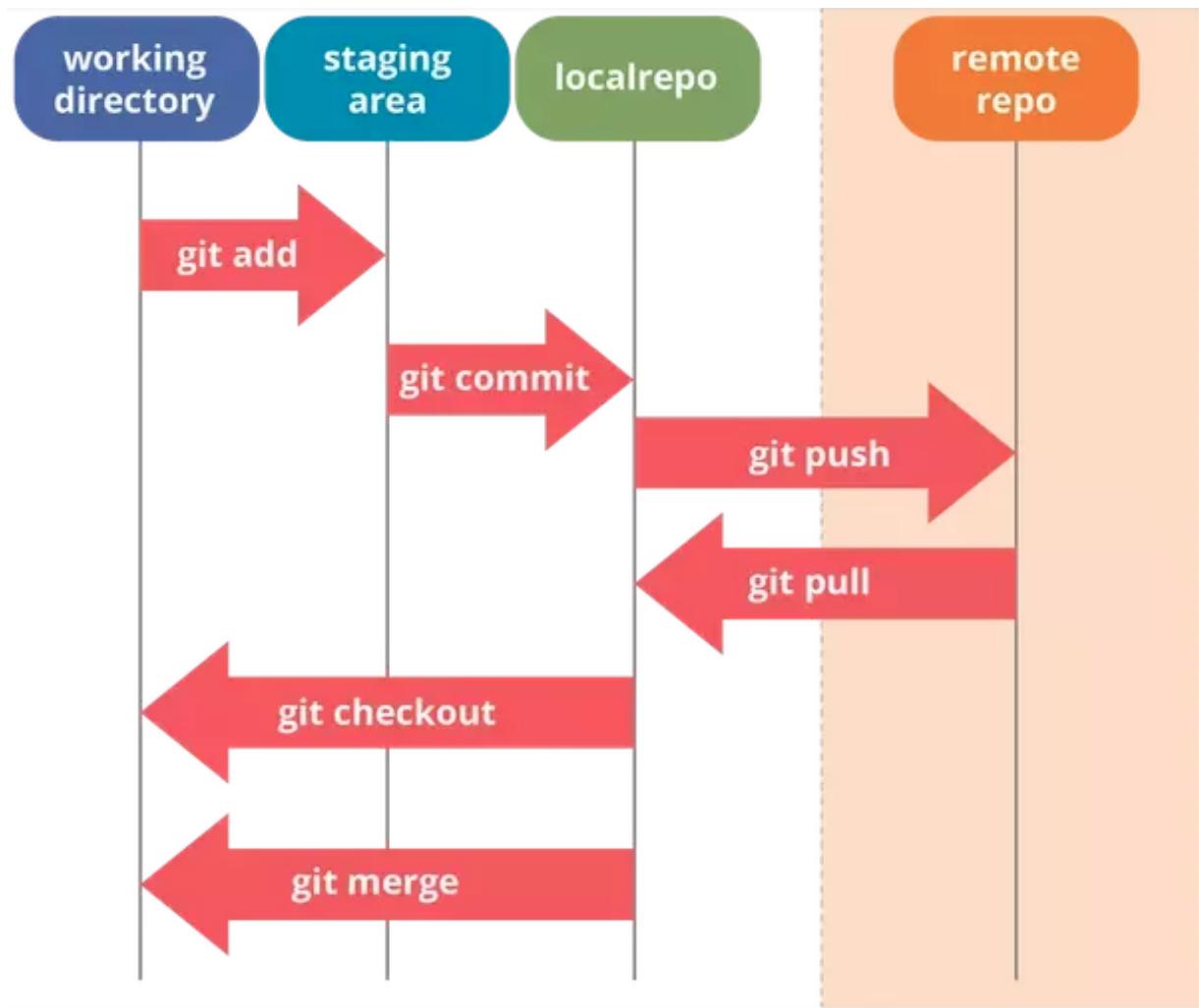


Abbildung 4.1: Git und Github Operationen

## 4.2 Projektboards

Mit den Projektboards auf GitHub können Softwareprojekte organisiert und Aspekte priorisiert werden. Projektboards können für spezifische Funktionen, umfassende Roadmaps oder sogar Release-Checklisten erstellt werden. Projektboards bestehen aus Themen, Pull-Requests und Notizen, die in Spalten als Karten kategorisiert werden, zum Beispiel "To Do", "In Progress", "Ready to Review" und "Done". Notizen und Issues können innerhalb einer Karte dargestellt werden und wechseln während ihrer Entwicklung die Karten / Spalten bis sie in der Karte "Done" bleiben. Dies kann sogar automatisiert werden. [git]

Notizen und Issues können Bugs, Features, Aufgabenerinnerungen oder Tasks welche in der Planungsphase anfallen, müssen daher nicht direkt mit der Codebase zu tun haben. Neben Repository-weiten

Projektboards gibt es auch Organisations-weite Projektboards, welche Probleme und Anfragen von allen Repositories beinhalten, die zu einer Organisation gehört.

## 4.3 Issues

Viele Projekte sammeln Benutzerfeedback über einen zentralen Bugtracker. GitHubs Tracker heit "Issues" und kann mit jedem Repository verwendet werden. Issues werden verwendet um Ideen, Erweiterungen, Aufgaben oder Fehler für die Arbeit an GitHub zu verfolgen. Issues können von Usern und Entwicklern erstellt werden und können zum Projektboard hinzugefügt werden. Issues können mehr als nur ein Ort zum Melden von Softwarefehlern sein. Ein Projektbetreuer kann mithilfe von Issues Tasks organisieren, die er ausführen möchte, z. B. neue Funktionen hinzufügen oder alte überwachen. Issues können mit Pull-Requests verknüpft werden, damit sie bei der Ausführung des Pull-Requests automatisch geschlossen werden. Issues können auch anderen Benutzern zugewiesen, mit Labels (zum Beispiel "Bug", "Enhancement" oder "question") für eine schnellere Suche versehen und mit Meilensteinen gruppiert werden.

## 4.4 Continuous Integration

Um Continuous mit automated build und testing zu realisieren, habe ich 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 gepusht wird, führt Travis die config in Form einer .yaml Datei aus, welche im root folder des Repos hinterlegt ist. Sie installiert alle Programme und Dependancies auf einer Linux Maschine und testet ob der Build in der aktuellen Version funktioniert. Es entsteht eine Version History mit dazugehörender config. Travis meldet, ob der Build und die Tests erfolgreich durchgeführt werden konnten. Dies geschieht auch auf separaten Branches des Projekts.

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

Abbildung 4.2: 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 4.3: Travis config .yaml, Version 45

### Default Branch

 ✘ master	# 33 failed ⌚ about 15 hours ago	-o- 7028d0c ⓘ 👤 Daniel Schmider	    
--	-------------------------------------	------------------------------------	---

### Active Branches

 ✘ ci_pgport	# 45 failed ⌚ about an hour ago	-o- 1afa786 ⓘ 👤 Daniel Schmider	 				
 ✓ ci	# 42 passed ⌚ about 2 hours ago	-o- 18bfa3e ⓘ 👤 Daniel Schmider	    				

Abbildung 4.4: Travis branches

# Kapitel 5

## Anforderungen, Technologien

### 5.1 Anforderungsspezifikationen

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. 2012 - 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.

### 5.2 Progressive Webapp

Um das Werkzeug "Waldmeister - Outdoors" nicht auf eine Plattform von mobilen Geräten zu beschränken (iOS oder Android), setzt es auf die Prinzipien der Progressive Webapps (PWA). Diese beschreiben eine Website, die viele Merkmale besitzt, welche bisher den nativen Apps vorbehalten waren. Eine PWA ist gewissermaßen eine responsive Website, welche auch offline verwendet werden kann. Sie schliesst dadurch eine zusätzliche Entwicklung einer nativen App, parallel zur Webseite überflüssig, da sämtliche Funktionen einer nativen App in der Webapp repliziert werden können. Eine PWA erreicht diese offline Fähigkeiten durch den Einsatz von Service Workern; ein JavaScript, welches von Web-Browersen im Hintergrund ausgeführt wird. Einmal online aufgerufen, können die Inhalte beim nächsten Besuch der Seite auch dann angezeigt werden, wenn eine schlechte oder sogar gar keine Internet-

verbindung besteht (Offline-Betrieb). Auch die von nativen Apps bekannten Push-Benachrichtigungen sind mit Service Workern möglich. [Ser]

Grundlegende Charakteristiken der PWA sind Offline Funktionalität, Push Notifications, Add-To-Homescreen, jedoch ist keine Installation notwendig (beispielsweise über den Apple Appstore oder Google Playstore). Verbreitete Mobile Browser wie Firefox und Google Chrome haben bereits eine vollständige Unterstützung von PWAs, Safari sollte dies bis Ende Februar 2018 ebenfalls implementiert haben und kann daher auch auf iOS verwendet werden. PWAs können auf den Home-Screen des mobilen Geräts hinzugefügt werden.

## 5.3 Mobile Field Solution mit ArcGIS Online

Technologien von ESRI (Environmental Systems Research Institute) und insbesondere ArcGIS (GIS; Geografisches Informationssystem) online 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 beim offline Betrieb und können später, sobald wieder eine stabile Internetverbindung besteht, synchronisiert werden. Dieses Verhalten kann bei einer PWA 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.

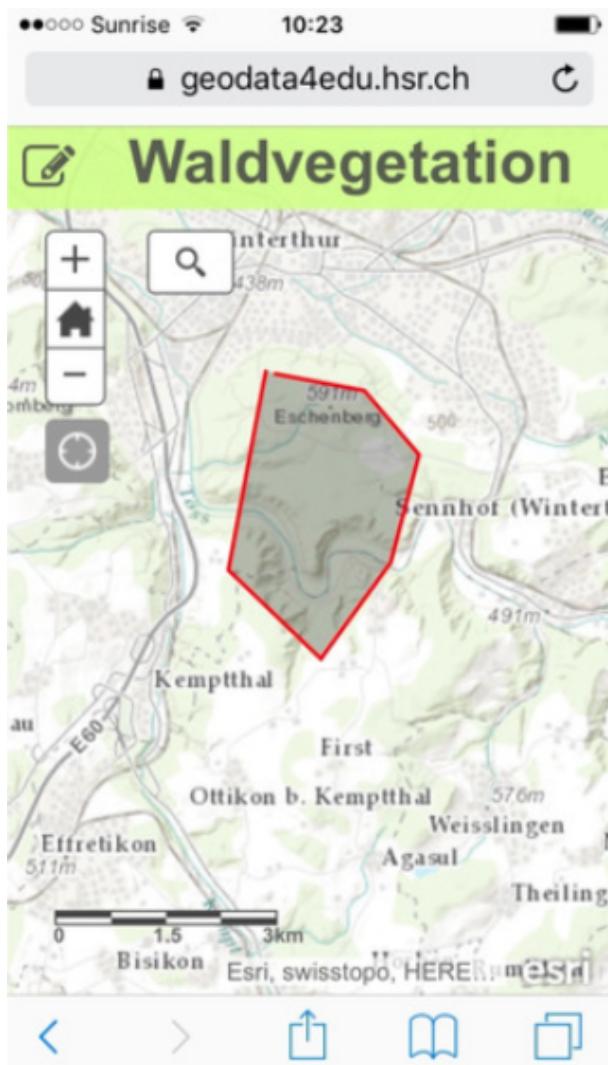


Abbildung 5.1: Prototyp mit ArcGIS online

## 5.4 Vue.JS

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 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.

Entwickler beschreiben es daher als "less opinionated" im Vergleich zu anderen populären JavaScript Webframeworks wie Angular.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>
```

### 5.4.1 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 einfacher 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).

### 5.4.2 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-Reload, 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.

### 5.4.3 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 ba-

sieren. 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 = ''
  }
}
```

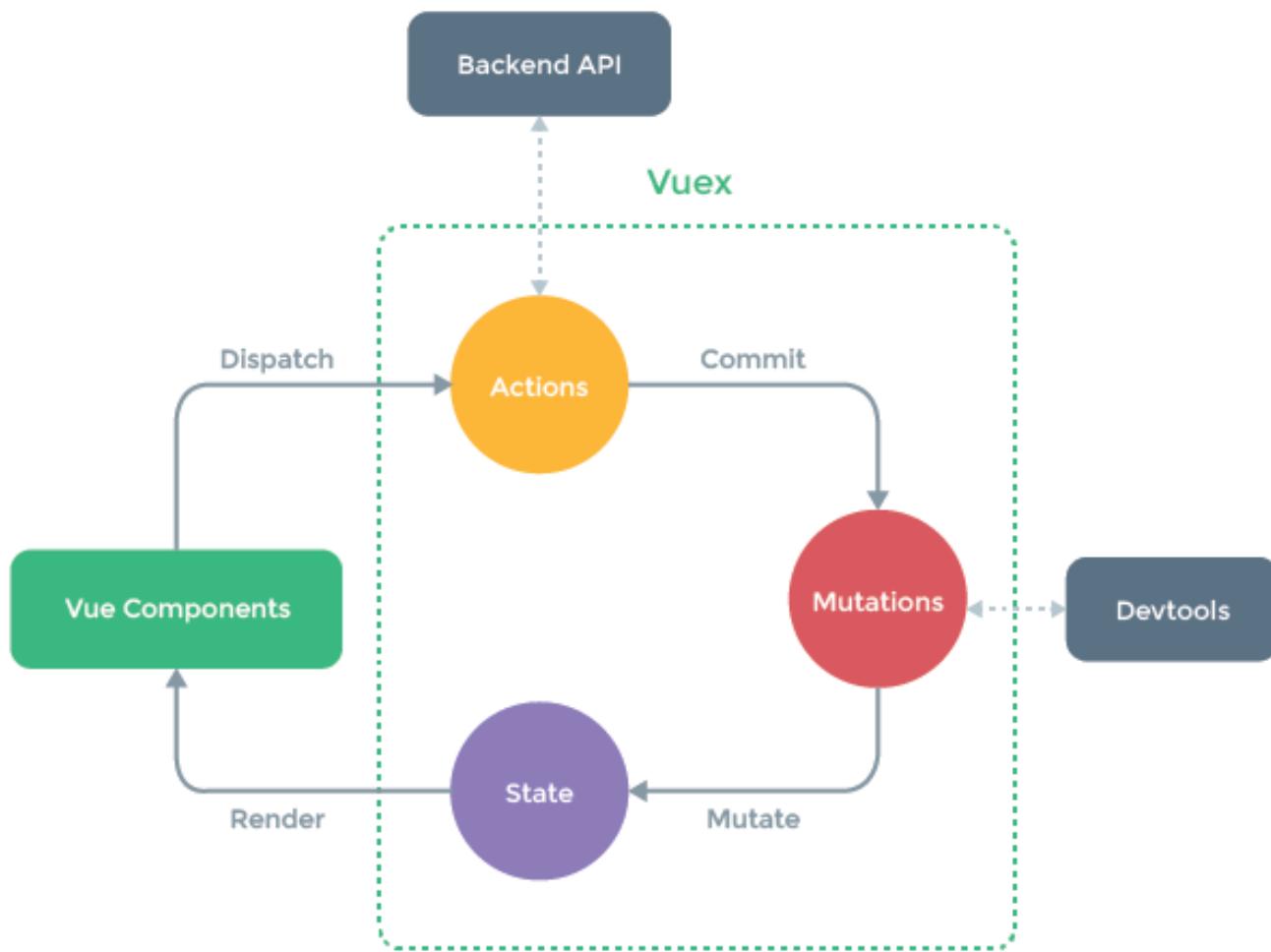


Abbildung 5.2: Vuex Action-Mutations-State Diagram

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

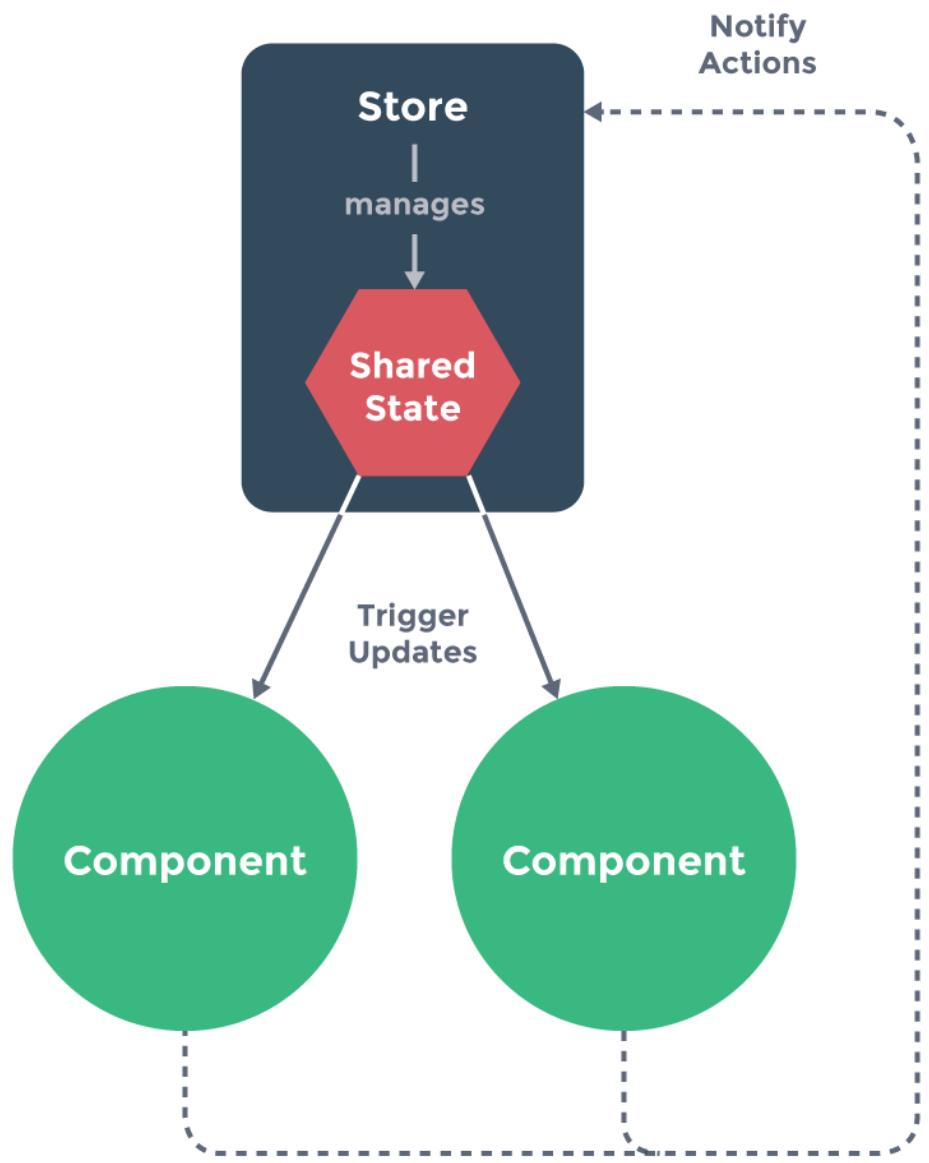


Abbildung 5.3: Vuex private

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.

#### 5.4.4 Vuetify

Vuetify ist eine Vue.JS UI Framework, welches das Frontend aus vielen Material-Design basierten UI Bausteinen zusammenbaut.

### 5.5 Axios

Axios ist eine JavaScript library zur Erstellung von Promise-Based HTTP Requests, welche "Waldmeister - Outdoors" dazu verwendet mit dem Server zu kommunizieren. Axios ermöglicht es, asynchrone HTTP requests zu REST (Representational state transfer) Endpunkten abzusetzen oder Requests wie Create, Read, Update, Delete (CRUD) Operationen auszuführen. Axios kann in puren JavaScript Projekten verwendet werden oder auch in Projekten, welche auf Vue.JS basieren. Ein Promise-Objekt repräsentiert die in der Zukunft zu geschehende Komplettierung einer asynchronen Operation (oder deren Abbruch durch einen Fehler).

### 5.6 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 Benutzeroberflächen haben, oder um öffentlich erstellte Benutzeroberflächen mit anderen zu teilen.

#### 5.6.1 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.

## 5.7 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]

### 5.7.1 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.

## 5.8 Leaflet

Leaflet ist eine JavaScript Library, welche es ermöglicht, eine Map auf dem Client darzustellen. Es fokussiert sich auf Simplizität, Performanz und ist sehr schlank, was einer PWA sehr entgegen kommt. Die Library ist nur 38 KB gross und ermöglicht es, viele Features zu verwenden, welche bei der Darstellung einer Map benötigt werden, zum Beispiel Marker, GeoJSON Layer oder Buttons welche das Verhalten der Map beeinflussen.

### 5.8.1 TileLayer, Hintergrundkarte

Der TileLayer fungiert als Hintergrundkarte, welche dynamisch geladen wird. Je nach benötigtem Kartenausschnitt werden die Tiles als .pngs geladen und auf der Karte dargestellt. Dies führt dazu, dass je nach Grösse und Zoomstufe des Kartenausschnitts nur minimaler Datenaufwand betätigt wird. Als Hintergrundkarte werden die "Terrain" Tiles von "stamen-tiles" verwendet.

### 5.8.2 GeoJSON Layer, Vegetationskundliche Karte

Oberhalb der Hintergrundkarte wird ein GeoJSON layer dargestellt, welcher die bereits bekannten Waldstandorte des Kantons darstellt. Er kann wahlweise ausgeblendet werden. Jedes Polygon dieses Layers wird durch ein Label beschriftet, welches den Kürzel (EK72) dieses Waldstandorts beschreibt. Sie sind Ausdruck der Standorteigenschaften, und beziehen sich nicht auf die aktuelle Bestockung, sondern auf eine potentielle natürliche Vegetation. Die Farbe des Polygons wird ebenfalls über diese Bezeichnung definiert und ist Teil des Standards Ellenberg Klötzli (EK72).

Diese Daten wurden vom Kanton Zürich bezogen und haben den Stand 31.12.1997. Viele Geometadaten haben jedoch den Stand 04. 04. 2011.

Grundlage für die Digitalisierung bildeten masshaltige Folien aus der analogen Kartierung im Massstab 1:5'000. Der grösste Teil wurde am damaligen Oberforstamt digitalisiert. Die Lagegenauigkeit ist c.a. 1 Meter. Die Daten wurden in der Form einer ESRI-Shapefile geliefert und mussten zuerst in das GeoJSON Format transformiert werden, damit sie auf der Leaflet Map (als GeoJSON Layer) dargestellt werden können. Die Attribute (darunter die Kurzbezeichnung EK72) bleiben dabei erhalten. [ZHG]

### 5.8.3 Leaflet editable

Damit die User neue Polygone, Punkte und Pfade erfassen können, benötigt Leaflet das Plugin "Leaflet editable", welches es ermöglicht, neue Objekte direkt auf der Map zu zeichnen, oder bestehende Objekte zu editieren. Der User hat die Möglichkeit, dem Objekt per Dialogbox einen Namen als Label zuzuweisen, und zwischen den Zuständen privat oder public zu wechseln. Jeder User hat nur Zugriff auf seine eigenen privaten Flächen, bzw. Pfade und Punkte, ausser er wählt es diese zu veröffentlichen, bzw "public" zu machen, damit sie alle User auf der Map sehen. Private Benutzerflächen beschreiben oft Orte, welche während der Feldarbeit von persönlichem Nutzen sind, jedoch für andere Experten nicht von Interesse sind oder bewusst privat gehalten werden.

Sobald ein Objekt erfasst ist, wird es per REST Schnittstelle an den Server übertragen und in der Datenbank gespeichert.

### 5.8.4 HTTPS, 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 Webbrower es nicht zulassen, 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.

# **Kapitel 6**

## **Implementation**

### **6.1 Mockup**

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.

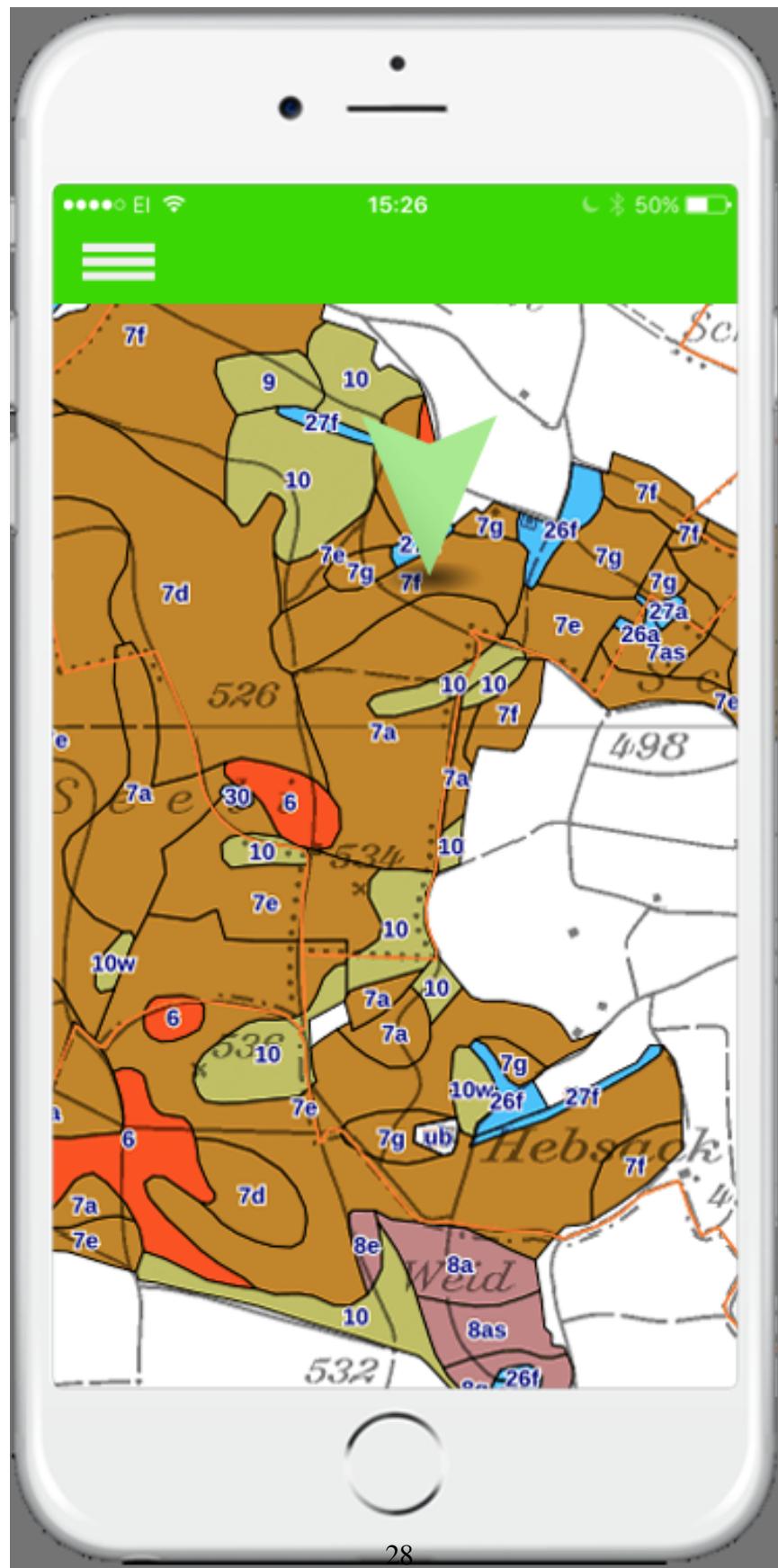
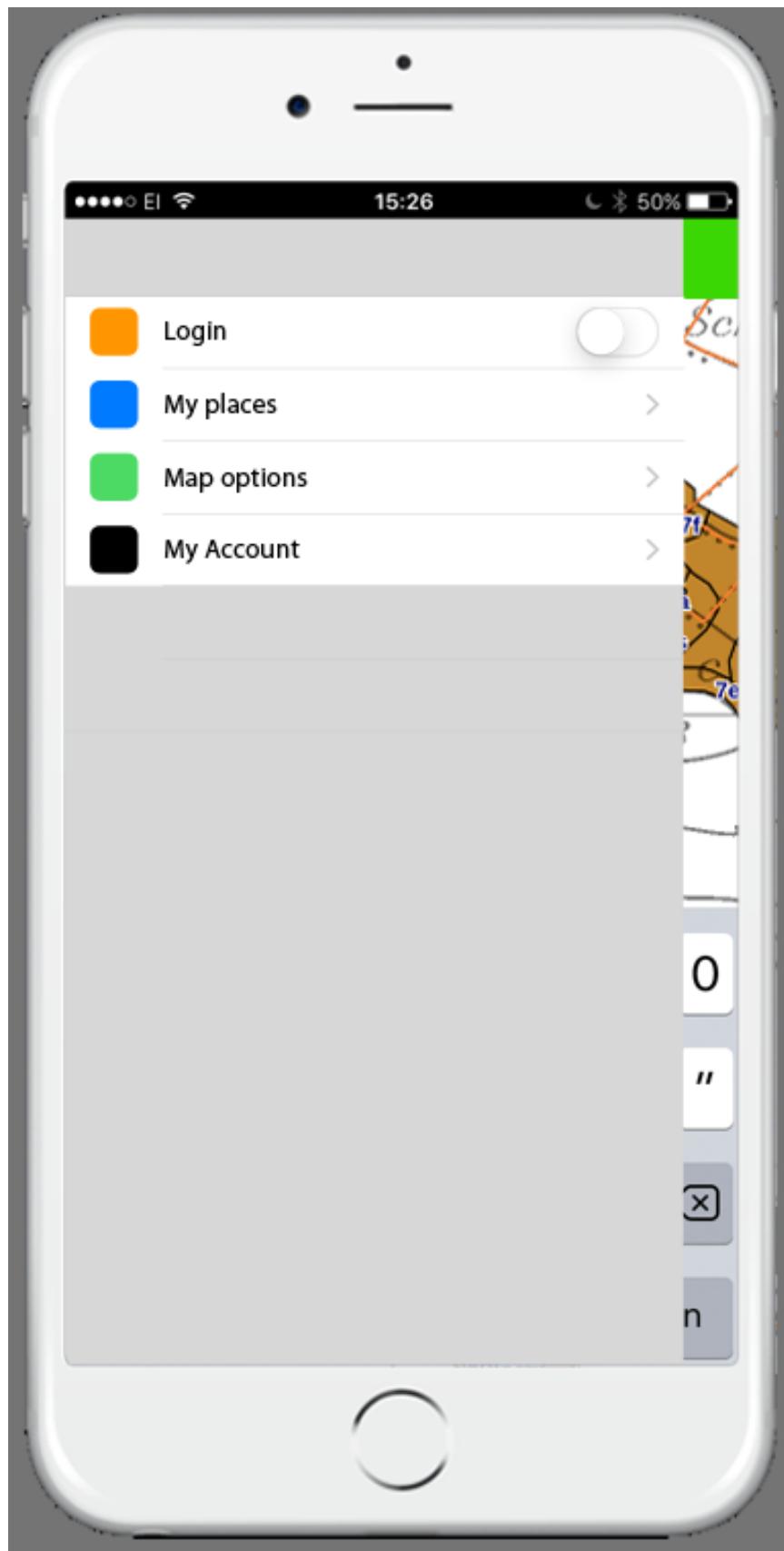


Abbildung 6.1: Mockup Screen 1



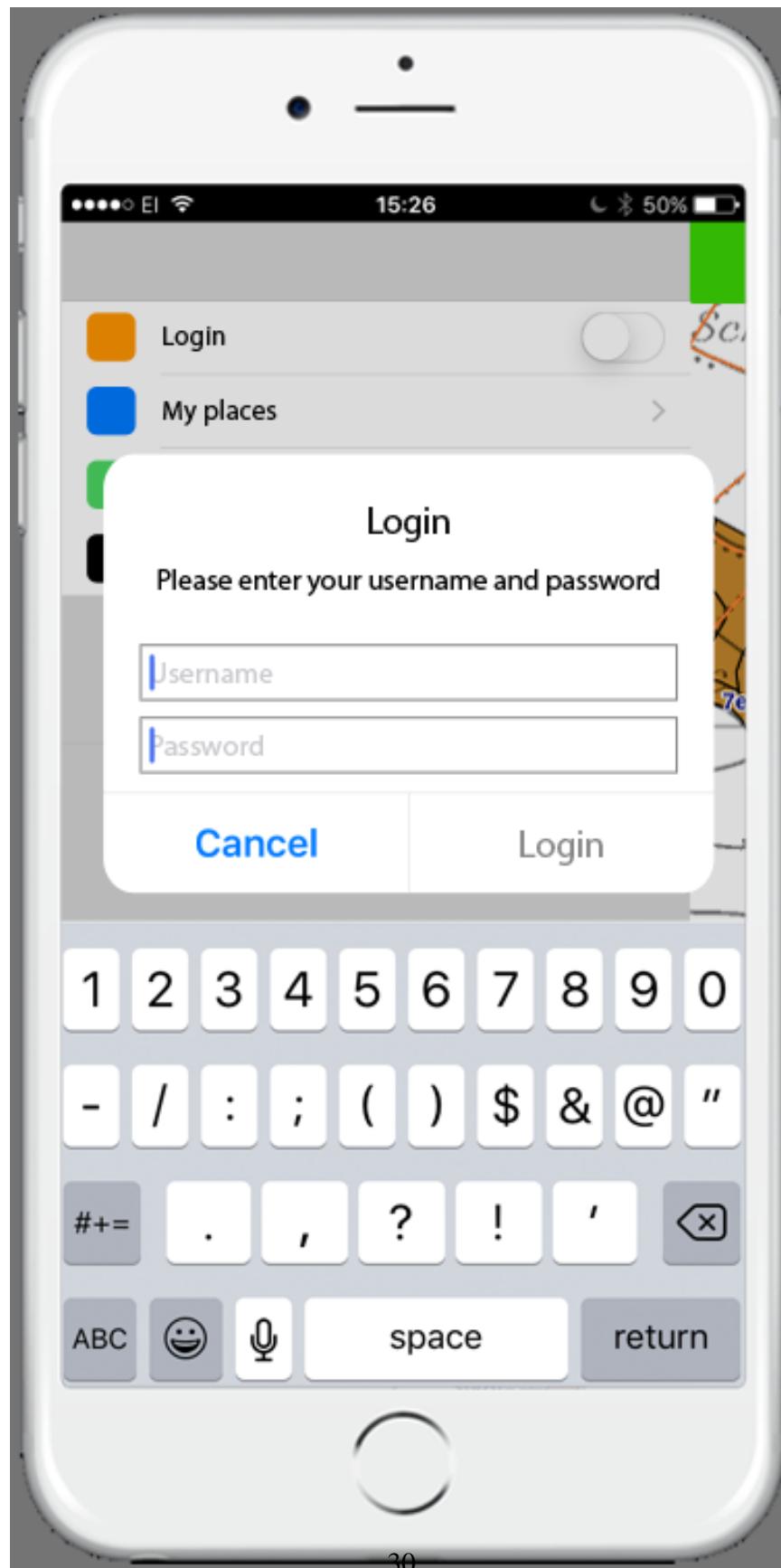


Abbildung 6.3: Mockup Screen 3

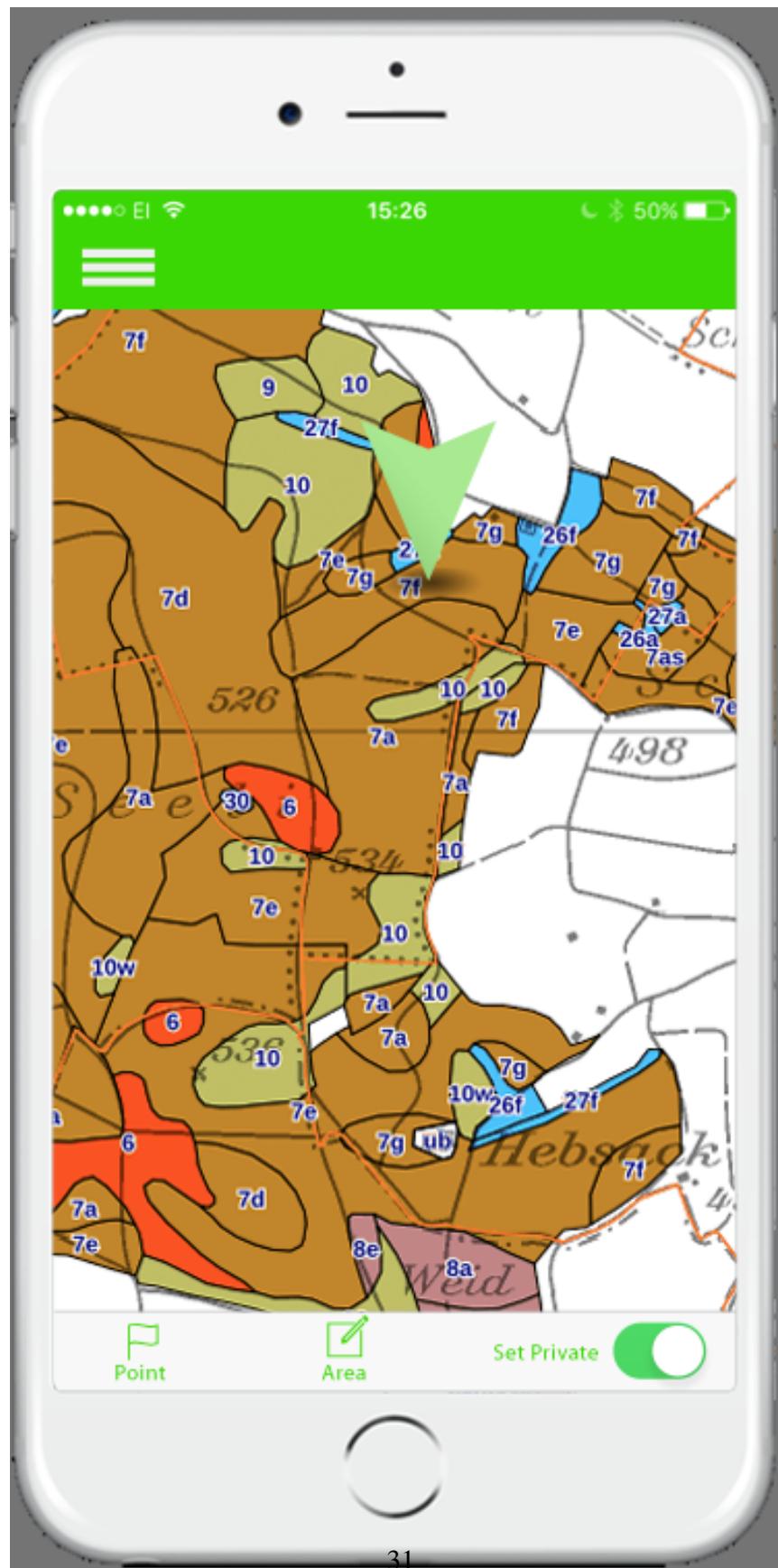


Abbildung 6.4: Mockup Screen 4

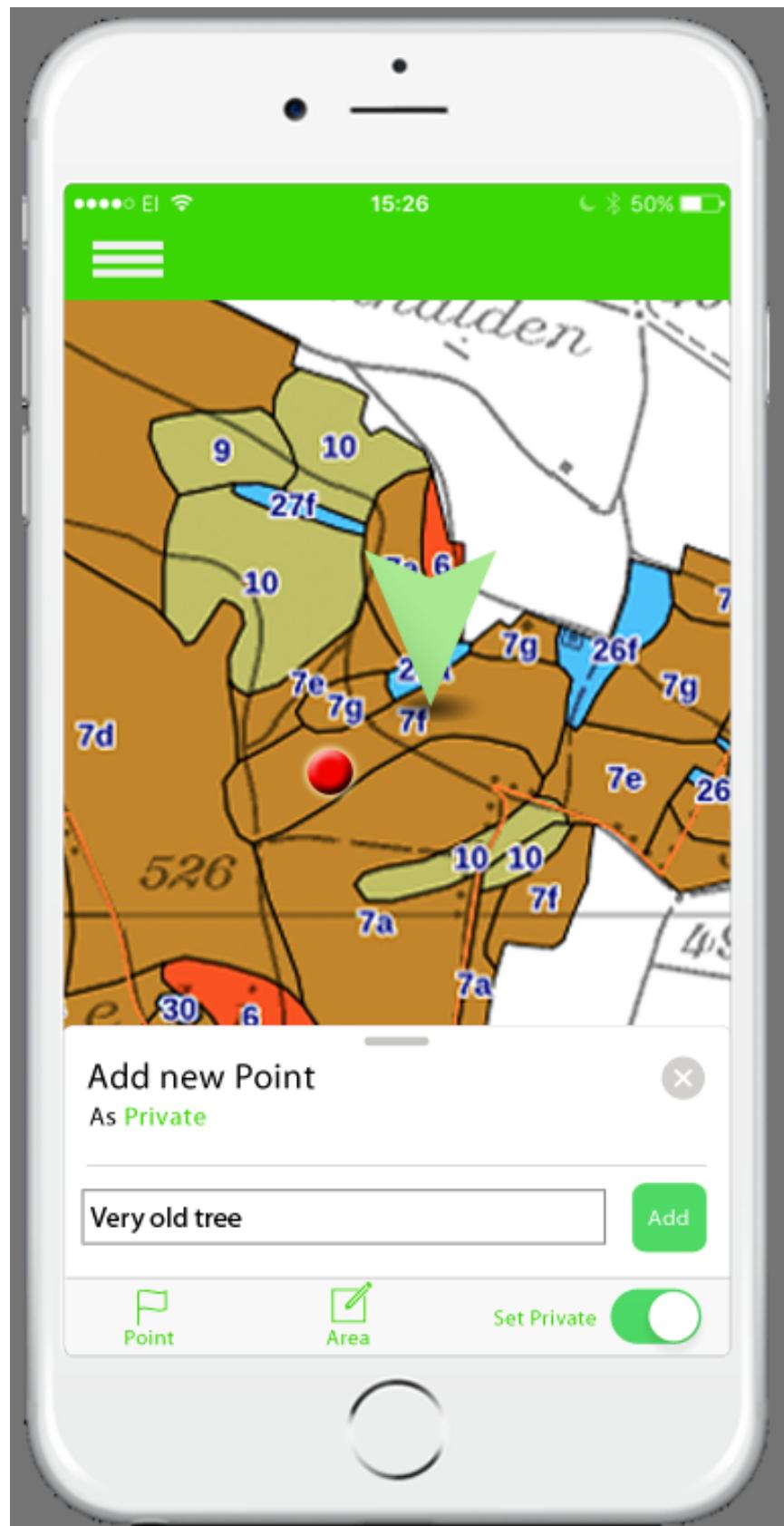


Abbildung 6.5: <sup>32</sup> Mockup Screen 5

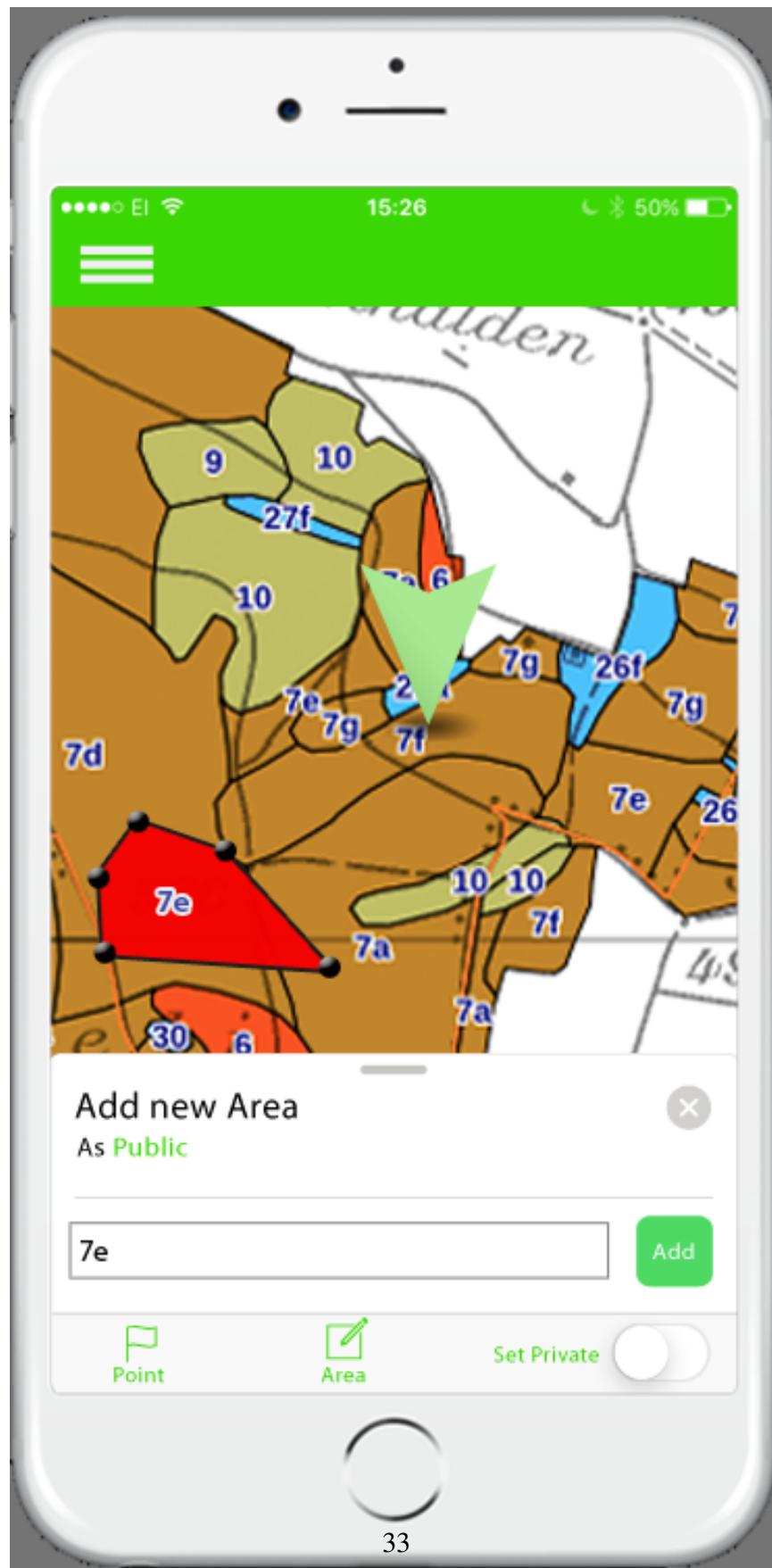


Abbildung 6.6: Mockup Screen 6

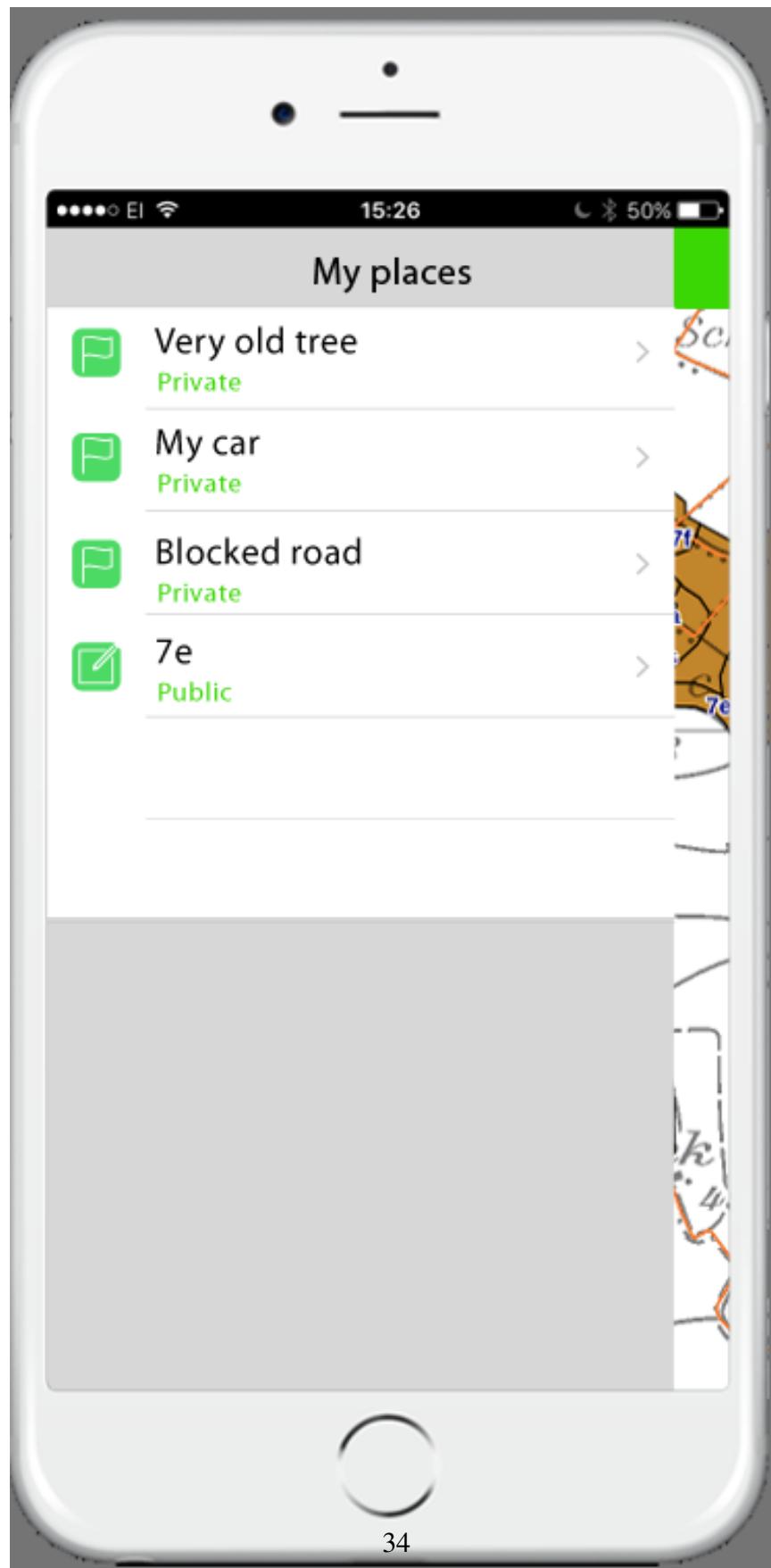


Abbildung 6.7: Mockup Screen 7

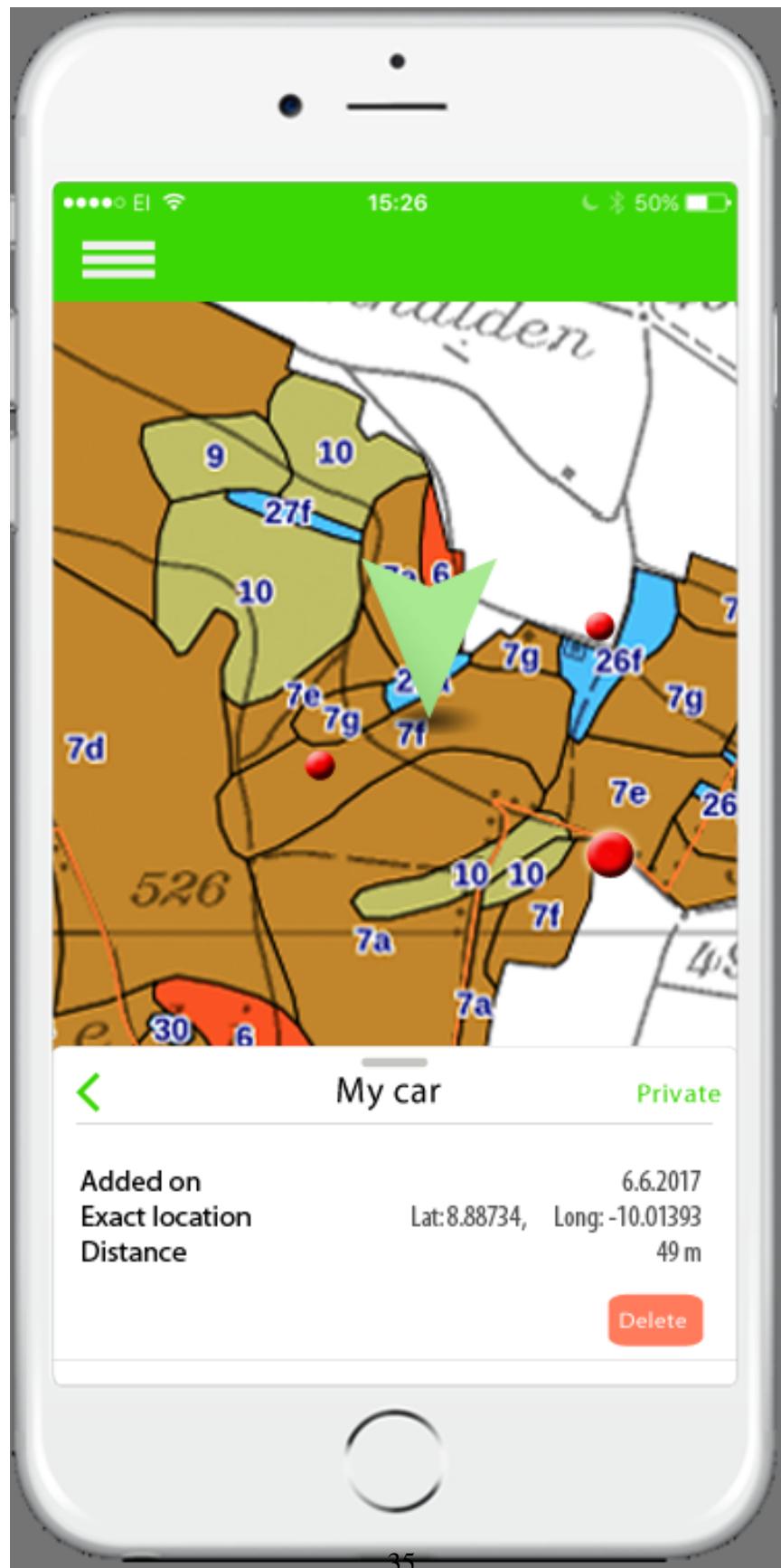


Abbildung 6.8: Mockup Screen 8

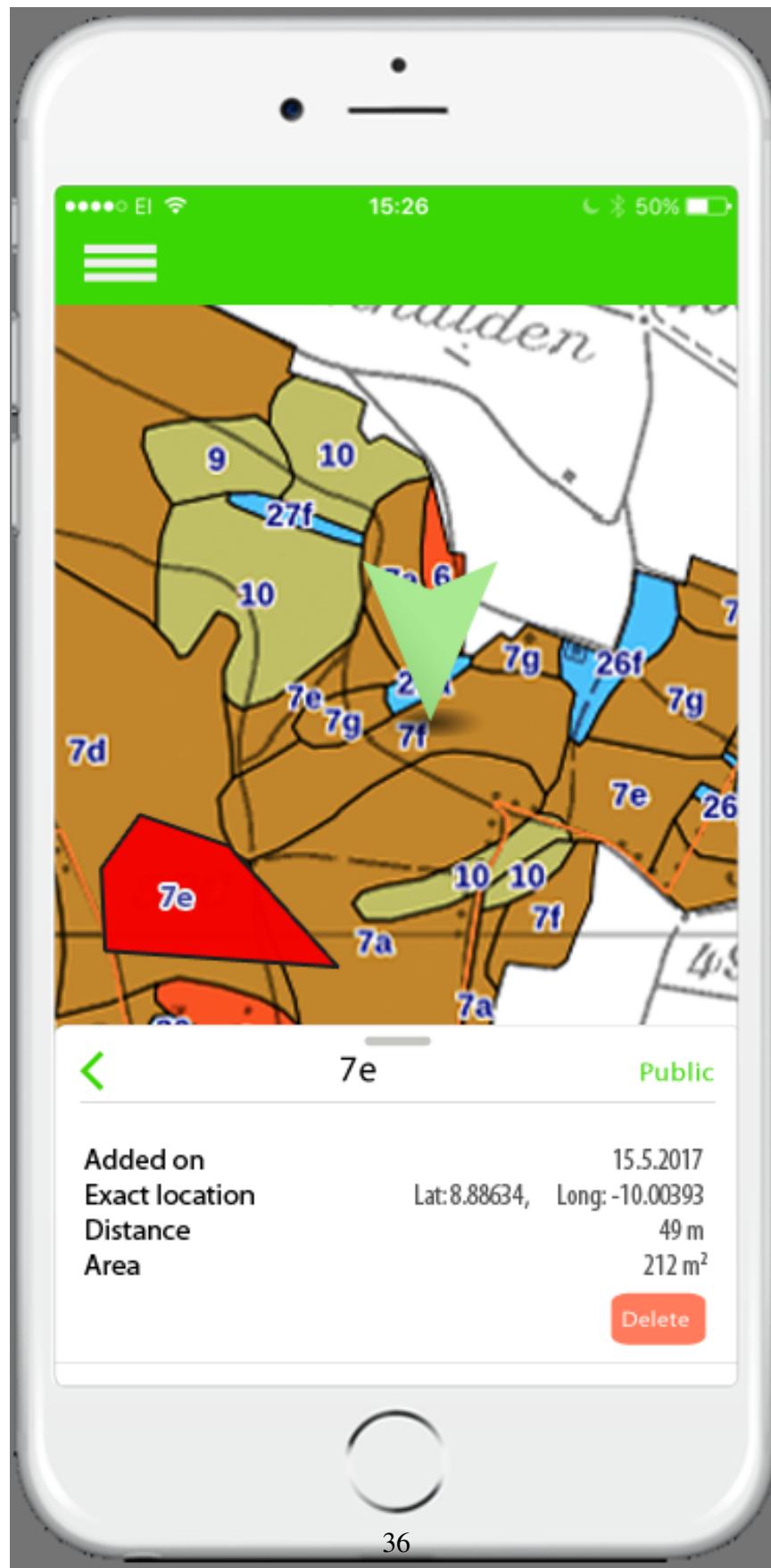


Abbildung 6.9: Mockup Screen 9

## 6.2 Modellierung mit UML Diagrammen

UML (Unified Modelling Language) Diagramme geben Auskunft über die Architektur und Abläufe des Systems. Es ist eine grafische Repräsentation in Form von Diagrammen, welche in der Sprache nach den Definitionen der "Unified Modeling Language" produziert sind. Diese bestimmt bei der Modellierung Beziehungen und Abhängigkeiten zwischen Begriffen und Notationen für diese Begriffe und statischen Strukturen und dynamischen Abläufen. Modellierung und UML finden Verwendung und UML ist die dominierende Sprache für die Softwaresystem-Modellierung. Sie dient zur Kommunikation von Ideen zwischen Projektauftraggeber, Softwareentwickler und Systemingenieuren.

In den 90er Jahren wurde die erste Spezifikation unter dem Namen UML 1.x entwickelt, im Jahr 2000 revidiert und in UML2 umbenannt. Seit Februar 2008 liegt Version 2.2 in der finalen Version vor. Seit Juni 2015 wird die aktuellste Version 2.5 verwendet.

Hauptsächlich definiert die Sprache Aktionen, Aktivitäten, Allgemeines Verhalten, Anwendungsfälle, Informationsflüsse, Interaktionen, Klassen, Komponenten, Kompositionsstrukturen, Modelle, Profile, Schablonen, Verteilungen und Zustandsautomaten. Diese kommen bei Strukturdiagrammen und Verhaltensdiagrammen zum Einsatz, wie u.a. den Klassen und Komponentendiagramm, oder Aktivitätsdiagramm, Sequenzdiagramm, Kommunikationsdiagramm, etc... Die Grenzen zwischen den über vierzehn Diagrammtypen verlaufen weniger scharf. UML2 verbietet nicht, dass ein Diagramm grafische Elemente enthält, die eigentlich zu unterschiedlichen Diagrammtypen gehören. Elemente aus einem Strukturdiagramm und aus einem Verhaltensdiagramm können nach Anwendungsfall auf dem gleichen Diagramm dargestellt werden, falls damit eine besonders treffende Aussage zu einem Modell gemacht werden kann.

### 6.2.1 Use Case - Diagramm

Das Diagramm 6.10 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.

### 6.2.2 Klassendiagramm, Datenbankdiagramm

Das Diagramm 6.11 schildert die Relation und den Ausbau der wichtigsten Klassen des Systems. Dies zeigt, dass Benutzerflächen nur von registrierten Users 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.

### 6.2.3 Sequenzdiagramm

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.

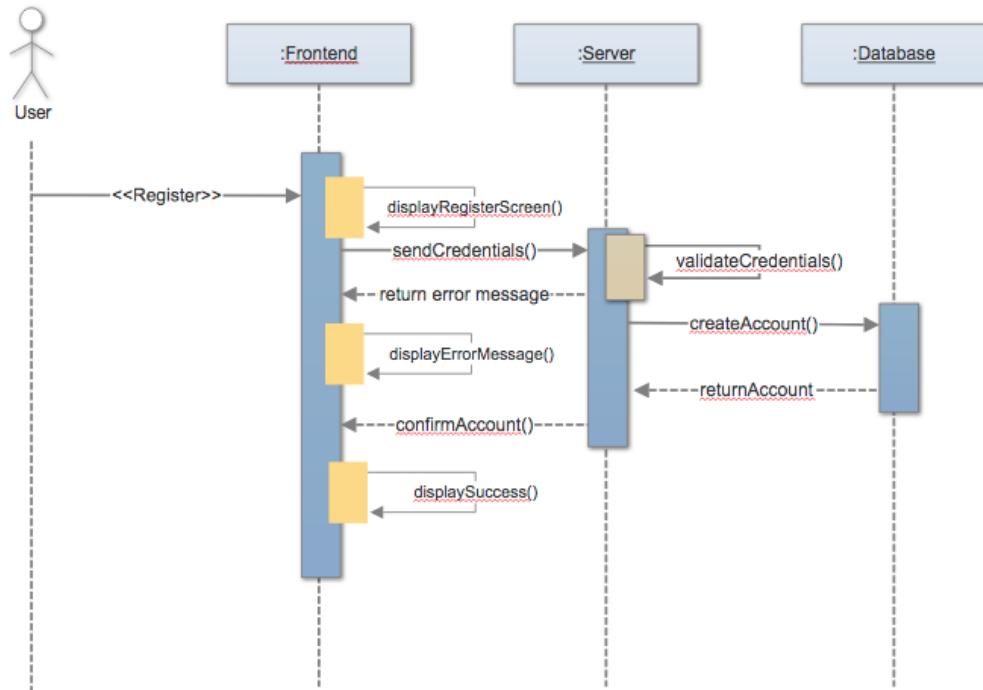


Abbildung 6.12: Sequenzdiagramm, Register

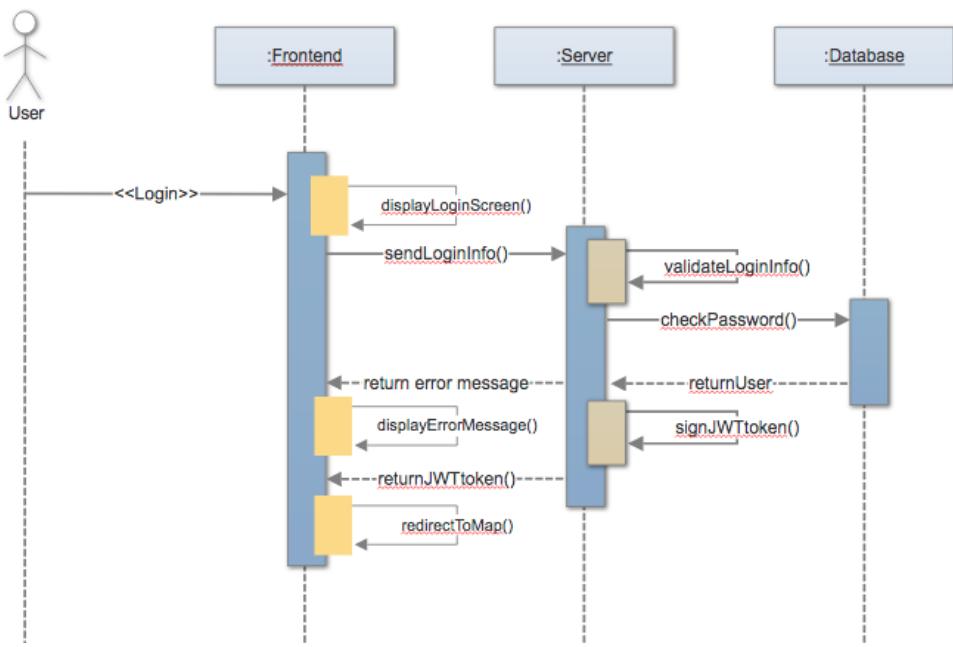


Abbildung 6.13: Sequenzdiagramm, Login

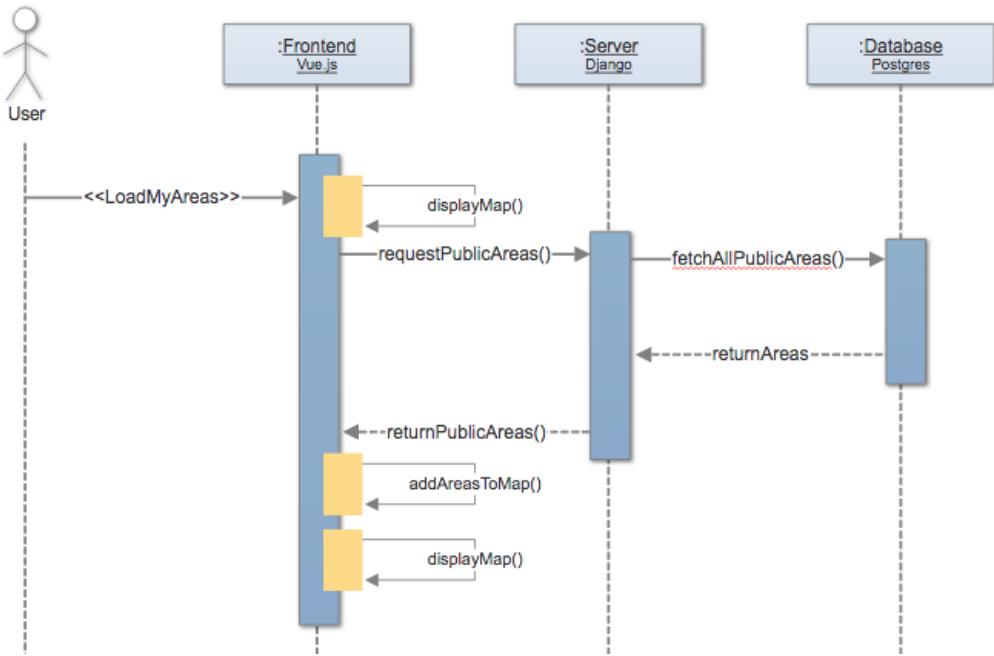


Abbildung 6.14: Sequenzdiagramm, Public Areas

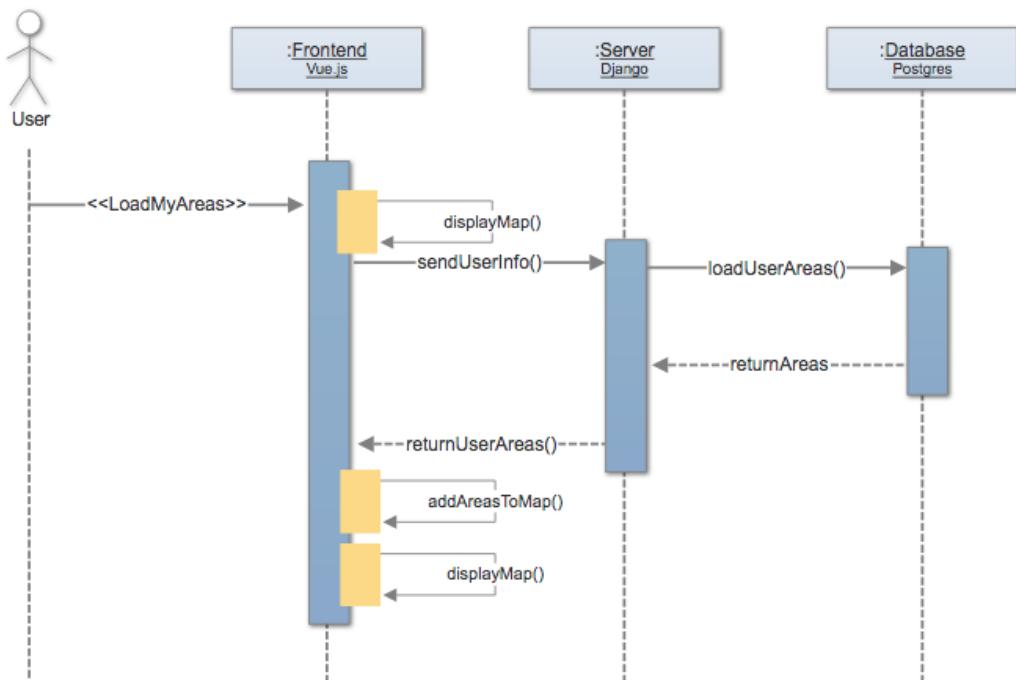


Abbildung 6.15: Sequenzdiagramm, My Areas

#### 6.2.4 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.

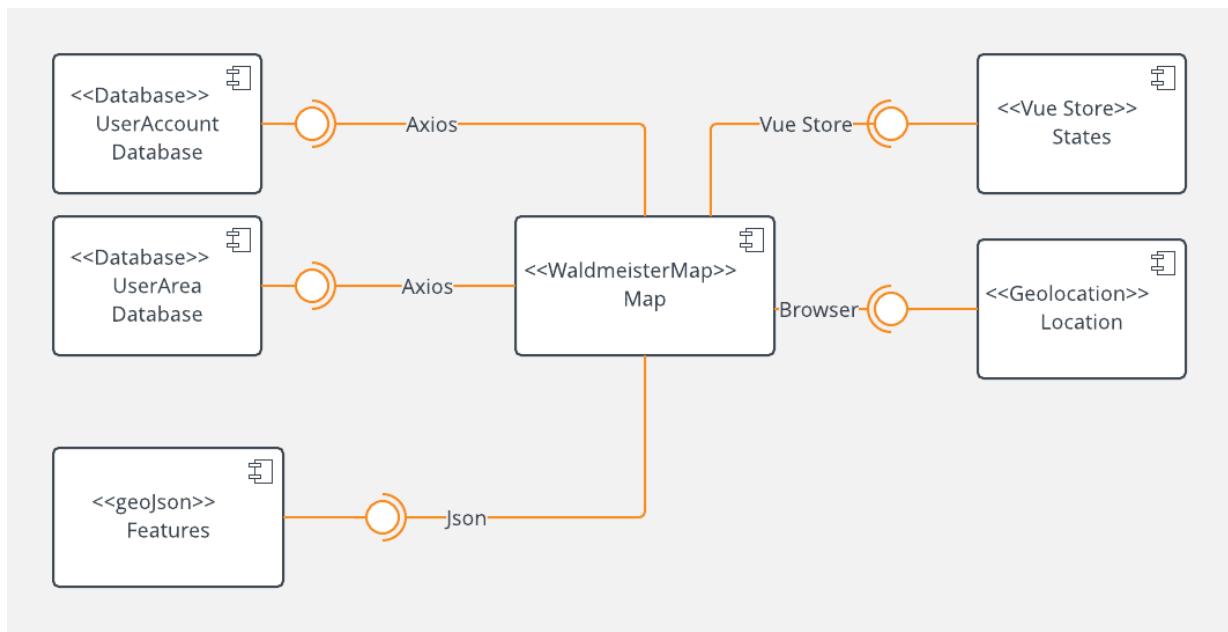


Abbildung 6.16: Komponentendiagramm, Waldmeister Map

## 6.3 Technische Implementation

### 6.3.1 VueJS 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.

### 6.3.2 WaldmeisterMap

Die Komponente WaldmeisterMap ist die 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 'Areas' 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 geclearnt 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.

### 6.3.3 Login und Registrierung

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 dies in einem 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 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]

### 6.3.4 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 6.17: About in der Waldmeister-Outdoors Webapp

### **6.3.5 User Interface / Frontend**

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-tellige 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.

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.

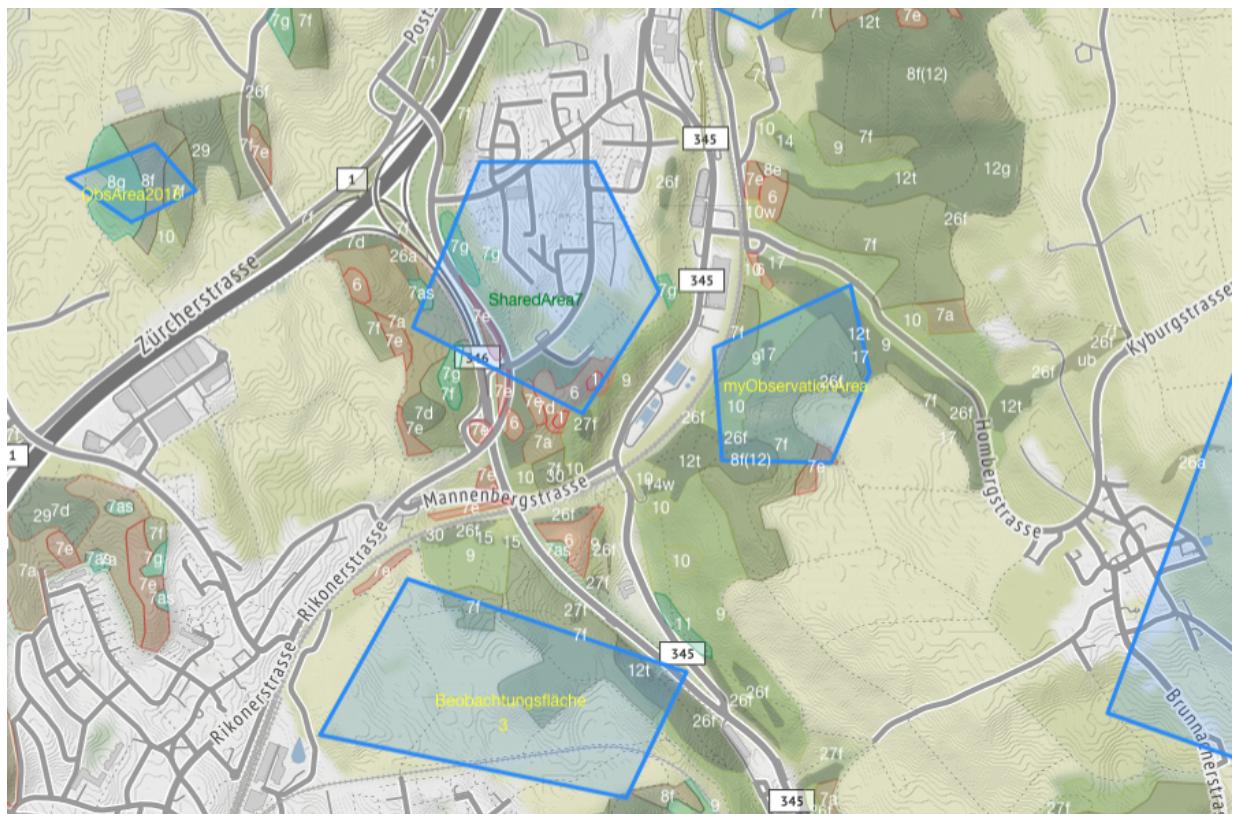


Abbildung 6.19: Benutzerflächen mit Labels

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

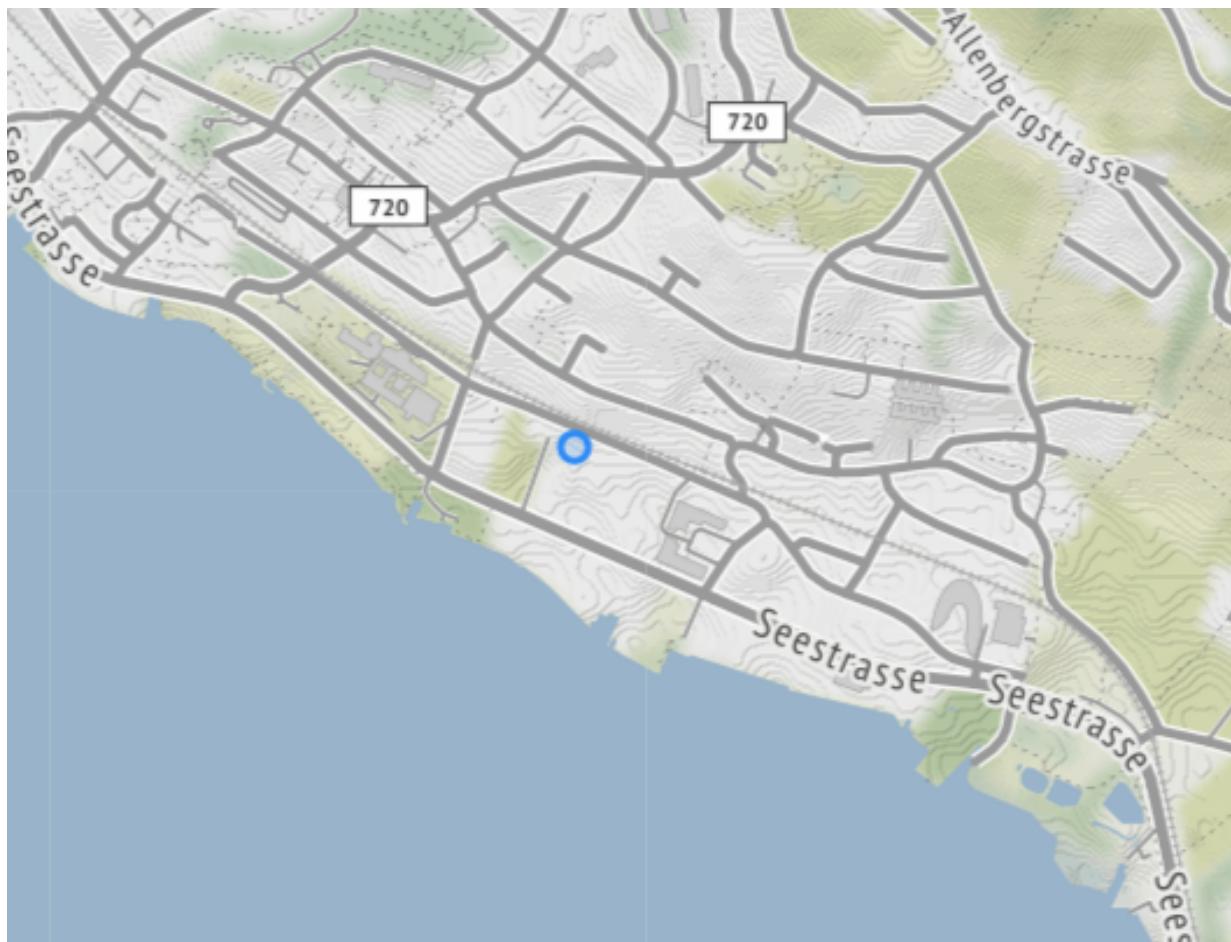


Abbildung 6.20: Geolocation mit Circle

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 Zeichnungsbutton 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.

### **6.3.6 Axios Requests**

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.

### **6.3.7 Database Models**

Die Benutzerflächen 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. Die Maximallänge ist auf 50 Zeichen gesetzt, kann aber beliebig erhöht werden.

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, die die Ecken 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. srid

### **6.3.8 Database, PostgreSQL**

Nachdem die Datenbank Modelle von Django definiert wurden, müssen sie über einen migrate Befehl in die PostgreSQL Datenbank übertragen werden, damit sie erstellt werden. 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 auf 5435 gesetzt, um Komplikationen mit anderen lokalen PostgreSQL Installationen zu vermeiden. 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.

### 6.3.9 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 User 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.

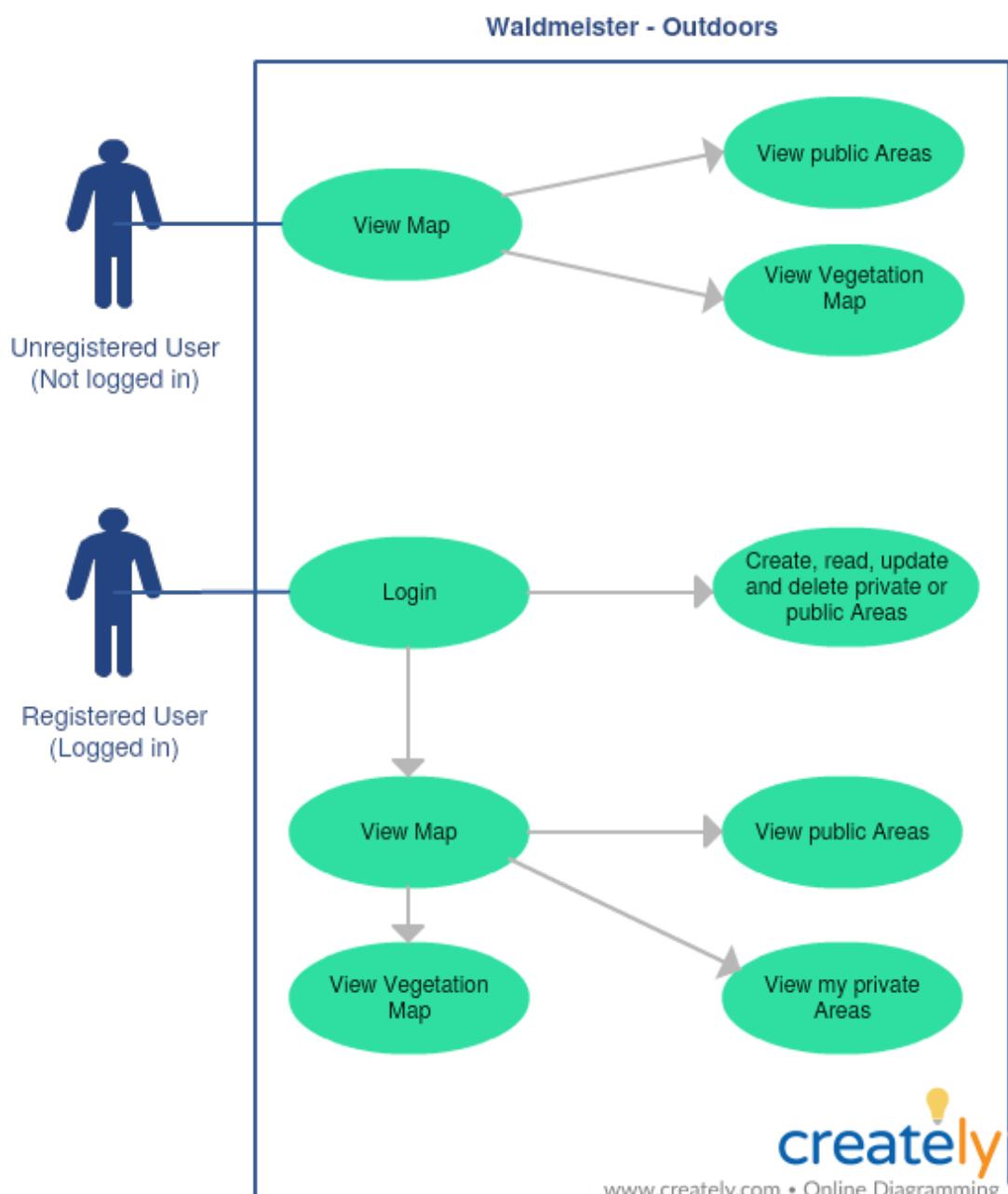


Abbildung 6.10: Use Case Diagram

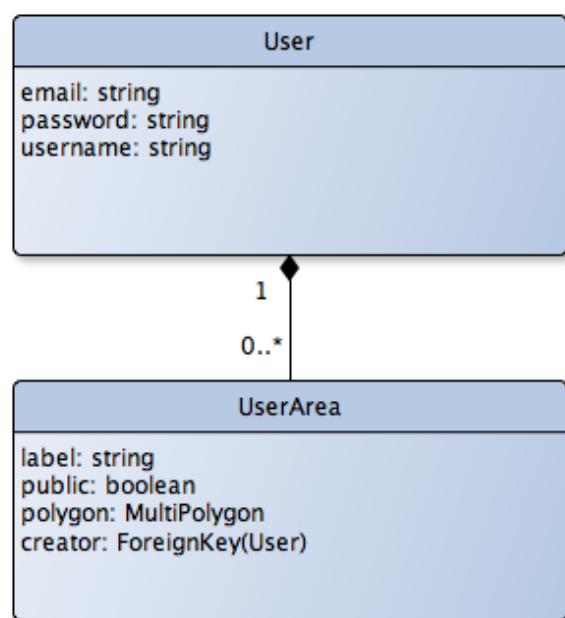


Abbildung 6.11: Klassendiagramm



Abbildung 6.18: Einfärbung von GeoJSON Flächen und Darstellung von PropertyLabels

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

Abbildung 6.21: Swagger API Authorization

api		Show/Hide   List Operations   Expand Operations
GET	/WaldmeisterMap/api/areas/	
POST	/WaldmeisterMap/api/areas/	
DELETE	/WaldmeisterMap/api/areas/{id}/	
GET	/WaldmeisterMap/api/areas/{id}/	
PATCH	/WaldmeisterMap/api/areas/{id}/	
PUT	/WaldmeisterMap/api/areas/{id}/	

Abbildung 6.22: Swagger API UserAreas

**POST** /WaldmeisterMap/auth/users/create/ Use this endpoint to register new user.

**Implementation Notes**  
Use this endpoint to register new user.

**Parameters**

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

Parameter content type: application/json  

**Response Messages**

HTTP Status Code	Reason	Response Model	Headers
201			

Try it out!

Abbildung 6.23: Swagger API user create Details

## **6.4 Tests**

### **6.4.1 Unit Tests, TDD**

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.

### **6.4.2 VueJS**

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.

### **6.4.3 Django**

Test für das Django wurden mit dem standard Django module "unittest" erstellt. Es wurden Tests zu Login, Registrierung und Erstellung von Benutzerflächen 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 Benutzerflächen vom System korrekt erstellt werden.

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

# Kapitel 7

## Resultate

Das Projekt wurde mit genannten Technologien umgesetzt und wird auf Github gehostet:  
<https://github.com/dschmide/Waldmeister> Installationsanleitung im Readme und im Kapitel Softwaredokumentation weiter unten in diesem Dokument.

### 7.1 Erreichte Ziele

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 gebrauchen, welche für Sie im Arbeitsalltag relevant sind und sie zwischen verschiedenen Geräten (Im Feld und im Büro) abrufen oder mit anderen Personen teilen, wenn 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.

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.

## 7.2 Verbesserungen, Weiterentwicklungen

### 7.2.1 Automatische Standortbestimmung

Die automatische Bestimmung, in welchem Waldstandort sich ein User anhand der Geolocation befindet, ist ein wünschenswertes Feature, da es den User laufend darüber informiert, in was für einem Waldtyp er sich befindet, ohne dass er dies durch einen Aufruf selbst ausführen muss. Dies kann erreicht werden durch eine periodische Abfrage der Geolocation, gekoppelt mit einer Berechnung in welchem GeoJSON Polygon sich dieser Punkt befindet. Eine solche Berechnung ist zum Beispiel über die Library geojson-geometries-lookup möglich, welche alle Polygone aus einem GeoJSON zurückgibt, welche den gegebenen Punkt beinhalten. Danach kann aus dem EK72 Feld der Kürzel ausgelesen und dem Benutzer angezeigt werden. [Geo]

Um diesen Ablauf für den User so angenehm wie möglich auszuführen, könnte er in der Toolbar in einen Tab wechseln, welcher keine Map, sondern lediglich die Information darstellt, in welcher Waldfläche er sich befindet. Dies kann in einem gegebenen Intervall automatisch wiederholt werden.

### 7.2.2 Editieren und Löschen von erstellten Benutzerflächen

Ein weiteres geplantes Features ist es, die Geometrie oder das Label von erstellten Benutzerflächen zu editieren, sie vom erstellten Zustand (öffentliche oder privat) in einen anderen zu wechseln oder sie wieder zu löschen, damit sie nicht mehr in der Datenbank eingetragen sind. Dies vervollständigt die CRUD Operationen, welche in einer solchen Anwendung wünschenswert sind.

Der Server, bzw. die REST-Schnittstelle lässt diese Operationen bereits zu (siehe API Dokumentation), sie sind jedoch im Frontend noch nicht eingebaut.

### 7.2.3 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.

### 7.2.4 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 Menü 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.

### 7.2.5 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.

### 7.2.6 Continuous Tracking

Die Map wird nur in dem Zeitpunkt in dem sie geladen wird auf die momentane Geolocation des Users zentriert. Dieses Feature könnte erweitern werden, indem der angezeigte Kartenausschnitt gewissermassen dem User folgt. Wiederholend in einem kurzen Intervall soll die map automatisch auf die Geolocation des Geräts zentriert werden. Auf diese Weise wird der User stetig in der Mitte der Karte angezeigt.

Dieses Feature kann jedoch je nach Anwendungsfall als störend empfunden werden und sollte deaktivierbar sein, bzw. nur auf Aufforderung des Users aktiviert werden.

### 7.2.7 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 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.

## 7.3 Screenshots

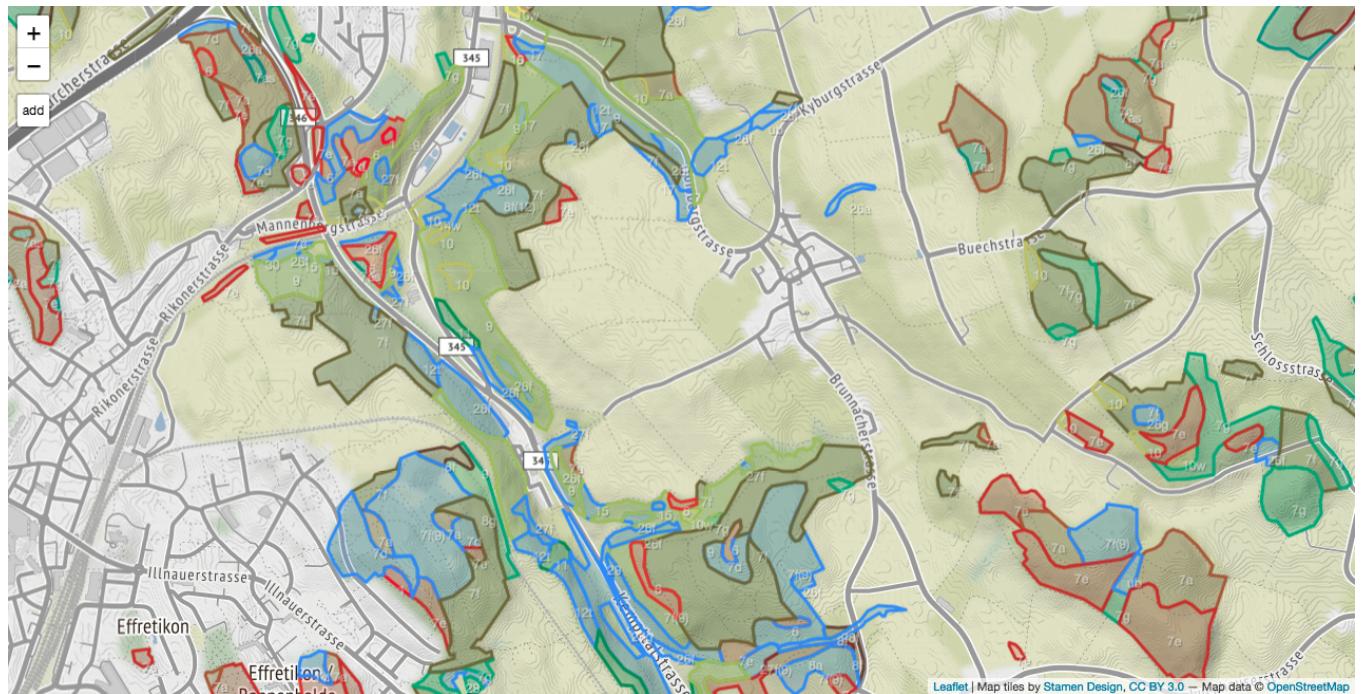


Abbildung 7.1: Screenshot Desktop, Vegetationskundliche Karte

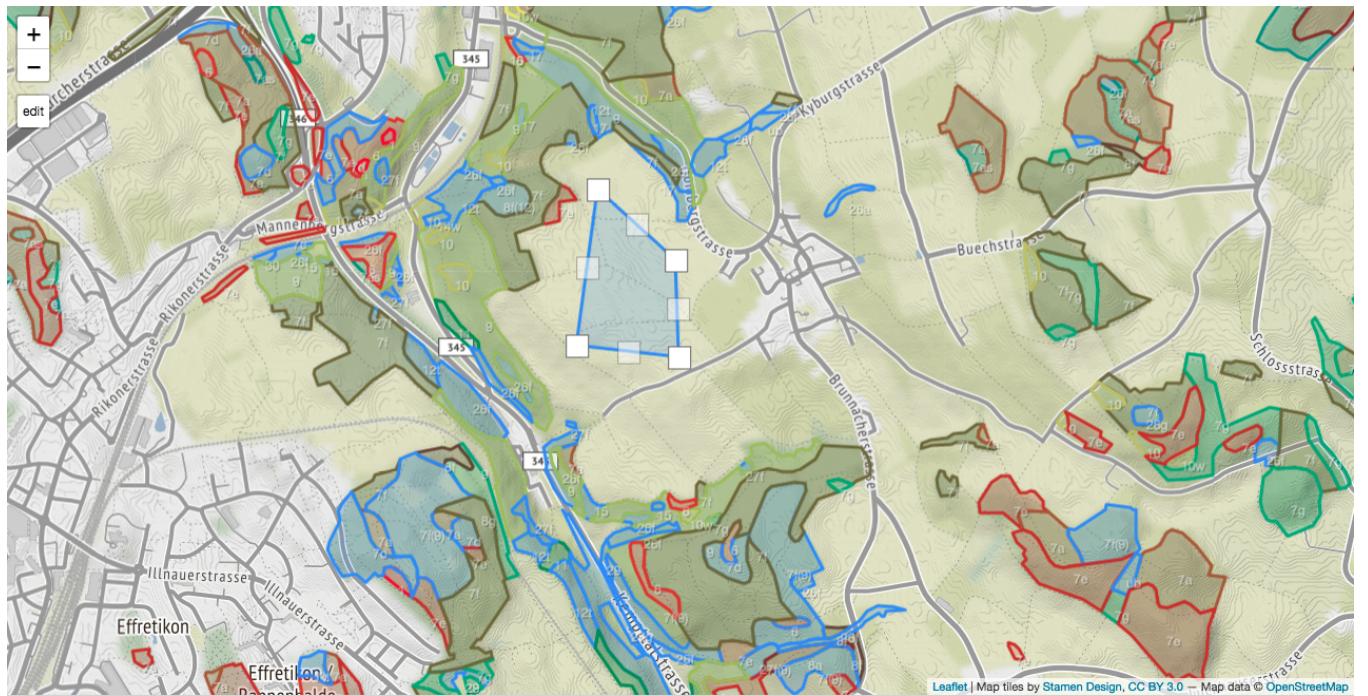


Abbildung 7.2: Screenshot Desktop, EditArea

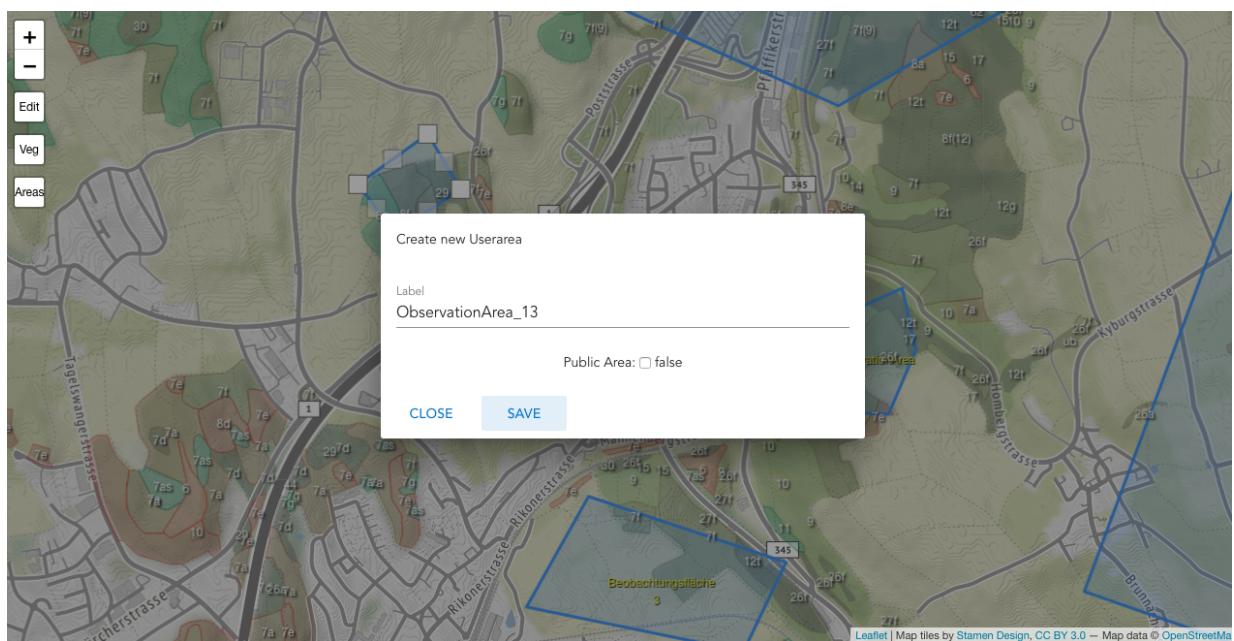


Abbildung 7.3: Screenshot Desktop, Dialogbox

# Kapitel 8

## Softwaredokumentation

### 8.1 Installation

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

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.

# **Kapitel 9**

## **Links**

### **9.0.1 Projektrelevante Links**

Projekt Wiki

[https://wiki.hsr.ch/StefanKeller/wiki.cgi?PA2\\_Schmider\\_Aufgabenstellung](https://wiki.hsr.ch/StefanKeller/wiki.cgi?PA2_Schmider_Aufgabenstellung)

<https://giswiki.hsr.ch/Koordinatensystem>

Github und Github Projekt

<https://github.com/dschmide/Waldmeister>

<https://github.com/dschmide/Waldmeister/projects/1>

Gis-Browser Kanton Zürich

<https://maps.zh.ch?topic=WaldVKZHscale=18634x=2706590.13y=1251180.39srid=2056>

Native Apps für Android und iOS Geräte

<https://itunes.apple.com/bn/app/waldmeister/id825753160?mt=8>

[https://play.google.com/store/apps/details?id=bgu\\_schmider.waldmeister](https://play.google.com/store/apps/details?id=bgu_schmider.waldmeister)

# Abbildungsverzeichnis

4.1	Git und Github Operationen . . . . .	11
4.2	TravisCI Version History . . . . .	13
4.3	Travis config .yaml, Version 45 . . . . .	14
4.4	Travis branches . . . . .	15
5.1	Prototyp mit ArcGIS online . . . . .	18
5.2	Vuex Action-Mutations-State Diagram . . . . .	22
5.3	Vuex private . . . . .	23
6.1	Mockup Screen 1 . . . . .	28
6.2	Mockup Screen 2 . . . . .	29
6.3	Mockup Screen 3 . . . . .	30
6.4	Mockup Screen 4 . . . . .	31
6.5	Mockup Screen 5 . . . . .	32
6.6	Mockup Screen 6 . . . . .	33
6.7	Mockup Screen 7 . . . . .	34
6.8	Mockup Screen 8 . . . . .	35
6.9	Mockup Screen 9 . . . . .	36
6.12	Sequenzdiagramm, Register . . . . .	38
6.13	Sequenzdiagramm, Login . . . . .	39
6.14	Sequenzdiagramm, Public Areas . . . . .	40
6.15	Sequenzdiagramm, My Areas . . . . .	41
6.16	Komponentendiagramm, Waldmeister Map . . . . .	42
6.17	About in der Waldmeister-Outoors Webapp . . . . .	44
6.19	Benutzerflächen mit Labels . . . . .	46
6.20	Geolocation mit Circle . . . . .	47
6.10	Use Case Diagram . . . . .	50
6.11	Klassendiagramm . . . . .	51
6.18	Einfärbung von GeoJSON Flächen und Darstellung von PropertyLabels . . . . .	52
6.21	Swagger API Authorization . . . . .	53
6.22	Swagger API UserAreas . . . . .	53
6.23	Swagger API user create Details . . . . .	54
7.1	Screenshot Desktop, Vegetationskundliche Karte . . . . .	60

7.2 Screenshot Desktop, EditArea . . . . .	61
7.3 Screenshot Desktop, Dialogbox . . . . .	61

# Glossar

## 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

# 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](http://www.parcs.ch/wpz/pdf_public/2013/9601_20131015_131512_gds_110.pdf).