2014

Maps3D

3D-Höhenprofil eines Kartenausschnittes mit Google Maps

SEMESTERARBEIT

Daniel Tiefenauer (daniel@tiefenauer.info)

ZHAW SCHOOL OF ENGINEERING | 28. Mai 2014

Versionierung

Version	Datum	Kommentar
1.4.0	28.05.2014	Abgabeversion
1.3.0	21.04.2014	Überarbeitung und Kürzung
1.2.0	07.04.2014	Technischer Teil fertiggestellt
1.1.0	03.03.2014	Konzeptioneller Teil fertiggestellt
1.0.1	08.01.2014	Ausgangslage beschrieben
1.0.0	20.12.2013	Initiale Version mit Dokumentstruktur

Tabelle 1: Versionsprotokoll

Abstract

In dieser Semesterarbeit wurde eine Web-Applikation namens „*Maps3D*“ realisiert, welche für einen benutzerdefinierten Kartenausschnitt das Höhenprofil dreidimensional darstellt.

Der Kartenausschnitt kann in Form eines Rechtecks durch den Benutzer definiert werden. Zur Darstellung der Karte wird die *Google Maps* API verwendet. Sowohl die Karte als auch das Höhenprofil werden vollständig im Browser dargestellt.

Die Applikation ist so aufgebaut, dass Höhendaten von einer beliebigen Quelle bezogen werden könnten. Zu Demonstrationszwecken wurde der Bezug über die *Google Elevations* API bereits implementiert.

Inhaltsverzeichnis

1 Über diese Arbeit 5
1.1 Ausgangslage und Motivation 5
1.2 Ziel der Arbeit 6
1.3 Aufgabenstellung 7
1.4 Aufbau dieser Arbeit 8
2 Projektplanung 9
2.1 Phasen 9
2.2 Projektplan 10
2.3 Termine 10
3 Bestehende Lösungen 12
3.1 Auswahl von Referenzlösungen 12
3.2 Beschreibung der Referenzlösungen 12
3.3 Google Maps 13
3.4 Google Earth 15
3.5 3D Outdoor Guides 17
3.6 HERE Maps 18
3.7 Zusammenfassung und Vergleich 19
4 Wahl der Technologie 22
4.1 Varianten 22
5 Anforderungsanalyse 25
5.1 Stakeholder 25
5.2 Use Cases 26
5.3 Anforderungen 30
5.4 Akzeptanztests 37
5.6 GUI-Mockup 44
6 Architektur 46
6.1 Systemumfeld (System- und Kontextabgrenzung) 46
6.2 Architekturstil 46
6.3 Architekturmuster/Grobkonzept 47
6.4 Komponenten 48
7 Risiken 50
7.1 Risiko 1: Abhängigkeit von Google und Limitierung der Nutzung 50
7.2 Risiko 2: Innovationsgrad 50
8 Umsetzung 53
8.1 Wahl der Technologie zur 3D-Modellierung 53
8.2 Technologie-Stack 56
8.3 Klassendiagramm 59

8.4 Unit Testing 67
8.5 Ablauf bei Ausführung des Haupt-Use Cases 67
8.6 Aufgetretene Probleme 69
9 Fazit 75
10 Glossar 76
11 Links 78
12 Abbildungsverzeichnis 79
13 Tabellenverzeichnis 80
14 Formelverzeichnis 82
15 Scriptverzeichnis 82
16 Literaturverzeichnis 83

1 Über diese Arbeit

Dieses Kapitel gibt einen Überblick über das Projekt und den Aufbau der vorliegenden Arbeit.

1.1 Ausgangslage und Motivation

Google Maps ist die wohl bekannteste Applikation zur Darstellung von Kartenmaterial im Web. Ob es darum geht, herauszufinden, wie weit das Ferienhotel vom Strand entfernt ist, ob momentan gerade Stau auf dem Arbeitsweg ist oder einfach zur Planung der nächsten Wanderung: *Google Maps* ist zu einem Hilfsmittel für viele Aktivitäten des Alltags geworden. Die Kartendaten sind öffentlich zugänglich und können über eine API[1] bezogen werden.

Seit 2010[2] bietet Google auch den Bezug von Höhendaten über diese API an. Damit kann für einen beliebigen Punkt auf der Erde die Höhe über oder unter Meer herausgefunden werden. Google nutzt dieselbe Datengrundlage für eigene Applikationen, beispielsweise für *Google Earth* oder die 3D-Darstellung in *Google Maps*.

Seither wurden diverse Tools durch Drittanbieter entwickelt, welche die Daten von Google nutzen und zur Darstellung des Höhenprofils verwenden. Diese Tools haben aber meist eine oder mehrere der folgenden Einschränkungen:

- Das Tool stellt die Daten nur zweidimensional dar. So kann zwar das Höhenprofil entlang eines Pfades berechnet werden, nicht aber für eine Fläche.
- Das Tool stellt die Daten zwar dreidimensional dar, die Darstellung kann aber nicht auf einen Kartenausschnitt beschränkt werden.
- Der Blickwinkel auf die dargestellten Daten kann vom Benutzer nicht oder nur mit Einschränkungen verändert werden.
- Das Tool kann nur gegen Bezahlung verwendet werden.

[1] Application Programming Interface: Schnittstelle zu einem System
[2] http://googlegeodevelopers.blogspot.ch/2010/03/aint-no-mountain-high-enough.html

1.2 Ziel der Arbeit

Es soll eine Applikation erstellt werden, welche das Höhenprofil für einen beliebigen Kartenausschnitt als 3D-Modell darstellen kann. Sämtliche Geodaten stammen von Google. Das Profil kann vom Benutzer gedreht und vergrössert werden.

Sowohl die Applikation als auch der Quellcode sollen öffentlich zugänglich sein. Der Arbeitstitel für die Applikation lautet „*Maps3D*“.

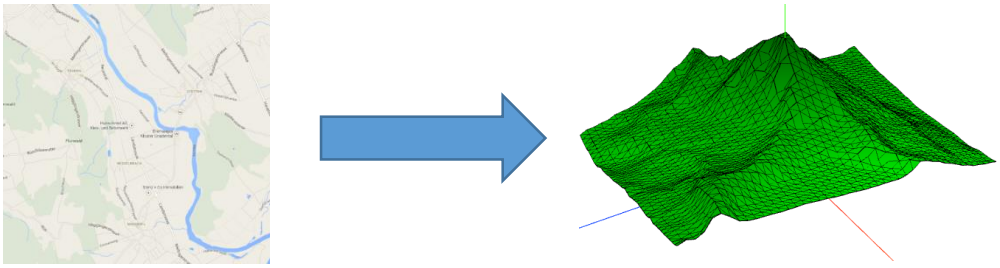

Abbildung 1: Grobe Visualisierung des Hauptanwendungsfalls der fertigen Applikation

1.2.1 Abgrenzung

Hinter dem Projekt stehen keine Auftraggeber und keine kommerziellen Interessen. Das Produkt ist ein POC[3] und deshalb in seinem Funktionsumfang limitiert. Insbesondere hat die Applikation nicht zum Ziel, das berechnete 3D-Modell realitätsgetreu darzustellen, sondern lediglich als Wireframe (Gitternetz-Darstellung) und ohne Überlagerung mit Kartenmaterial. Ebenso kann die Auflösung relativ grob sein, jedoch immer noch so hoch, dass die ursprüngliche Geländeform erkennbar bleibt.

[3] „Proof of Concept“, Machbarkeitsanalyse

1.3 Aufgabenstellung

Folgende Teilaufgaben wurden im EBS[4] definiert. Deren Umsetzung ist in dieser Arbeit beschrieben:

- Vergleich und Analyse von Unzulänglichkeiten bestehender, ähnlicher Produkte
- Evaluieren von Möglichkeiten zur Darstellung dreidimensionaler Inhalte im Web
- Einarbeitung in die Schnittstellendefinition und Funktionsweise der *Google Maps* API und *Google Elevation* API
- Implementieren einer Applikation, mit welcher eine Fläche über einem Kartenausschnitt definiert werden kann.
- Anbindung der *Google Maps*/*Google Elevation* API an die Applikation zum Bezug von Höhendaten
- Implementieren einer Applikation, welche aus den von Google bezogenen Daten verarbeitet und in ein für die Weiterverarbeitung optimiertes Format bringt
- Dreidimensionale Darstellung der verarbeiteten Daten

[4] Einschreibe- und Bewertungssystem für Projekte an der ZHAW

1.4 Aufbau dieser Arbeit

Diese Arbeit besteht in Anlehnung an die Aufgabenstellung aus folgenden drei Teilen:

- Analyse
- Konzept
- Realisierung

Im Analyseteil werden bestehende Lösungen miteinander verglichen, welche eine ähnliche Funktionalität wie *Maps3D* haben. Dieser Teil fokussiert auf funktionale Überschneidungen sowie Vor- und Nachteile jeder Applikation.
Ziel des Analyseteils ist die Identifikation von Alleinstellungsmerkmalen von *Maps3D* gegenüber bestehenden Produkten („unique selling points“). Dieser Teil ist vor allem durch Kapitel 3 abgedeckt.

Der zweite Teil enthält die konzeptionelle Sicht auf *Maps3D*. Ausgehend von Anwendungsfällen werden die Anforderungen an die Applikation beschrieben. Diese umfassen auch Testfälle, Akzeptanzkriterien und GUI-Mockups. Basierend auf den Anforderungen wird die Architektur, der Systemkontext und Schnittstellen zu externen Systemen dokumentiert. Ausserdem wird auf Risiken eingegangen.
Ziel dieses Teils ist ein Grobkonzept, welches als Grundlage für die Implementierung dient. Er umfasst die Kapitel 4 bis 7.

Der Realisierungsteil fokussiert die technische Umsetzung von *Maps3D*. So sind darin die innere Struktur, die einzelnen Bestandteile von *Maps3D* sowie verwendete Tools und Libraries beschrieben. Technische Details werden erklärt, soweit sie für das Verständnis der Applikation hilfreich sind. Insbesondere wird auf zwei Probleme während der Implementierungsphase vertieft eingegangen.
Ziel des Umsetzungsteils ist eine technische Dokumentation von *Maps3D*. Dieser Teil ist in Kapitel 8 beschrieben.

Zum Schluss ziehe ich ein persönliches Fazit, in welchem ich den Projektverlauf und meinen Lernerfolg reflektiere.

In der Dokumentation verwendeten Bilder, Tabellen, Formeln und Skripte sind am Schluss der Arbeit in separaten Verzeichnissen aufgeführt, zusammen mit einem Glossar, in welchem verwendete Fachbegriffe kurz erläutert werden.

2 Projektplanung

Laut Reglement[5] ist für eine Semesterarbeit ein Aufwand von 120 Stunden vorgesehen. Zu Projektbeginn wusste ich bereits, dass ich während zwei Wochen in den Ferien sein würde. Diese geplanten Abwesenheiten wurden in der Planung berücksichtigt. Insgesamt standen mir also 24 Wochen zu Verfügung, wovon 4 Wochen als Reserve eingeplant wurden. Als Planungsgrundlage rechnete ich somit mit einem durchschnittlichen Aufwand von 6 Stunden pro Woche.

2.1 Phasen

Der Projektverlauf war grob in folgende Phasen gegliedert.

Phase	Tätigkeiten
Informationsbeschaffung	Einlesen in einschlägige Literatur, Vergleich bestehender Lösungen, Analysieren von Umsetzungsvarianten, Ausprobieren von Technologien
Konzeption	Klärung von Anforderungen an die Applikation, skizzieren der Architektur, Formulieren von Lösungsansätzen, Technologieentscheid
Web-Applikation	Implementieren einer webbasierten Applikation, mit welcher Geodaten (Kartenmaterial, Koordinaten, Höhendaten) von Google bezogen werden können und welche die bezogenen Daten für die Weiterverarbeitung in ein internes Format bringt.
3D-Modellierung	Implementieren einer Applikation, welche die Geodaten dreidimensional darstellen kann
Dokumentation	Festhalten des Prozesses, gewonnene Erkenntnisse, Anknüpfungspunkte für Erweiterungen

Tabelle 2: Beschreibung der Projektphasen

Diese Phasen waren nicht als sequentielle Zeitabschnitte geplant, sondern als sich überlagernde oder iterativ durchlaufene Projektabschnitte. Insbesondere die Dokumentation erfolgte projektbegleitend und nicht als in sich abgeschlossene Phase. Tabelle 3 visualisiert die Zeiträume, in welcher der Hauptteil der Arbeit für die jeweilige Phase erfolgte.

[5] https://ebs.zhaw.ch/files/documents/informatik/Reglemente/Bachelor/Semesterarbeit/a_Reglement-Semesterarbeit_Studiengang-Informatik_V4.1.docx

2.2 Projektplan

Die folgende Tabelle visualisiert die zeitliche Verteilung der oben beschriebenen Phasen und erlaubt einen Vergleich zwischen geplantem und tatsächlichem Aufwand.

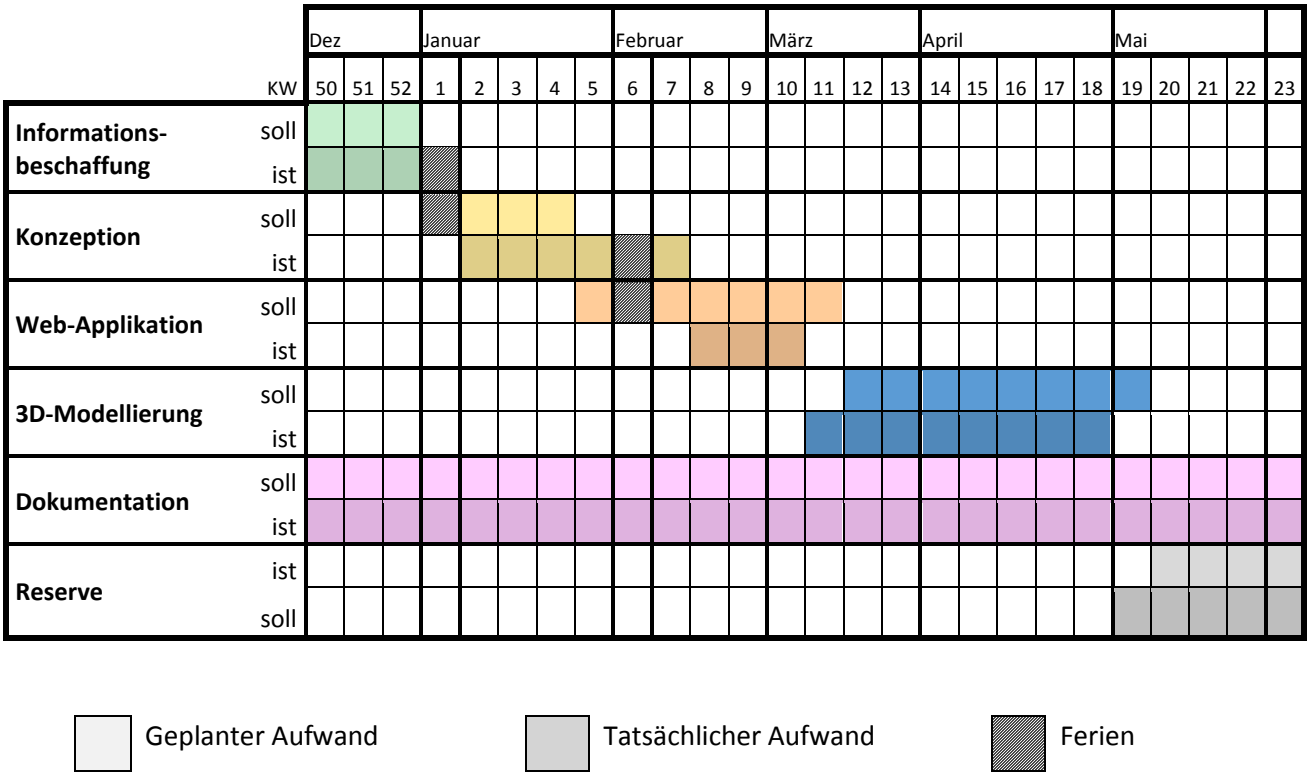

Tabelle 3: Projektplan

2.3 Termine

Folgende Termine wurden zu Beginn des Projektes vereinbart:

- 06.12.2013: Projekteingabe EBS
- 08.01.2014: Kick-Off
- 05.03.2014: Design Review
- 07.06.2014: letzter möglicher Abgabetermin

Teil 1: Analyse

3 Bestehende Lösungen

In diesem Kapitel werden bestehende Lösungen verglichen. Es gibt bereits eine Vielzahl von Produkten, welche eine ähnliche Funktionalität wie *Maps3D* bieten oder sogar darüber hinausgehen. Der Vergleich wurde gemacht, um Alleinstellungsmerkmale für *Maps3D* zu identifizieren und so *Maps3D* von bestehenden Lösungen abzugrenzen. Das Resultat des Vergleichs war die Ausgangslage für die anschliessende Anforderungsanalyse.

3.1 Auswahl von Referenzlösungen

Die Beschreibung sämtlicher existierender Softwareprodukte zur Verarbeitung von Karten- bzw. Höhendaten ist kaum möglich und würde den Rahmen dieser Arbeit bei weitem sprengen. Deshalb habe ich mich auf die Auflistung einiger Vertreter der 3D-Modellierung von Kartenmaterial beschränkt. Die Auswahl erfolgte unter folgenden Gesichtspunkten:

- Ist der Hauptanwendungsfall demjenigen von *Maps3D* ähnlich?
- Ist das Produkt oder zumindest Teile davon kostenlos verfügbar?

Da zum Zeitpunkt der Recherche die Anforderungen an *Maps3D* noch nicht im Detail geklärt waren, erfolgte die Auswahl nach eigenem Ermessen und ist in der Folge zu einem gewissen Grad subjektiv. Die Liste der verglichenen Lösungen hat deshalb exemplarischen Charakter und ist keineswegs als abschliessend zu betrachten.

3.2 Beschreibung der Referenzlösungen

Die ausgewählten Referenzprodukte sind im Folgenden kurz beschrieben. Jede Beschreibung beinhaltet folgende Teile:

- Übersicht über die Anwendungsbereiche und Möglichkeiten des Produktes
- Vorteile des Produktes gegenüber den anderen Referenzprodukten
- Nachteile des Produktes gegenüber den anderen Referenzprodukten

3.3 Google Maps

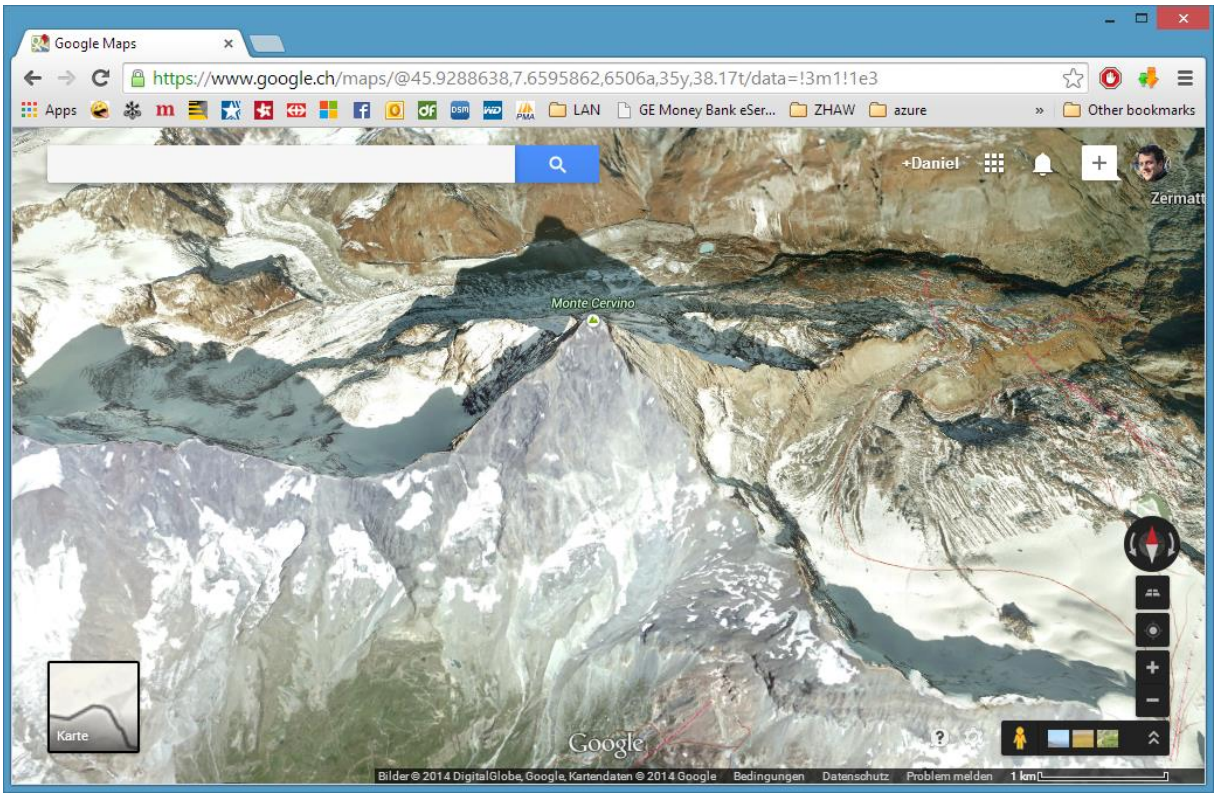

Abbildung 2: 3D-Darstellung des Matterhorns mit Google Maps

Hauptlieferant von Kartendaten für *Maps3D* und gleichzeitig grösster Konkurrent von *Maps3D* ist *Google Maps* selber. Nebst der Möglichkeit, Karten aus der Vogelperspektive im Browser darzustellen bietet *Google Maps* seit 2010 auch die Möglichkeit, die Ansicht zu kippen und die Karte als dreidimensionales Modell darzustellen. *Google Maps* verwendet eigene Daten für die Darstellung des 3D-Modells.

3.3.1 Vorteile von Google Maps

Google Maps kann kostenlos und ohne Installation verwendet werden. Bedienung und Darstellung erfolgen vollständig im Browser. Nebst der Darstellung eines dreidimensionalen Höhenprofils bietet *Google Maps* eine Vielzahl weiterer Funktionalitäten wie die Suche nach Adressen, Routenplanung und Integration anderer Google-Dienste wie z.B. *StreetView*. Das dargestellte Höhenprofil kann dabei mit Satellitenbildern überlagert werden und erhält so ein sehr realistisches Aussehen. *Google Maps* geht ausserdem über die Darstellung eines reinen Höhenprofils hinaus, indem dieses zusätzlich mit Metadaten wie zum Beispiel Orts- und Strassennamen sowie der momentanen Verkehrslage angereichert werden kann.

3.3.2 Nachteile von Google Maps

Trotz der äusserst vielseitigen Anwendungsmöglichkeiten bietet *Google Maps* in einigen Punkten Raum für Erweiterungen. Folgendes ist mit *Google Maps* nicht möglich:

- **Kein stufenloses Rotieren in der Horizontalen**: Das Höhenprofil kann zwar über Schaltflächen gedreht werden, dies aber nur in Intervallen von 90 Grad. Eine Änderung in kleineren Schritten ist nicht möglich. Somit ist lediglich eine Blickrichtung nach Norden, Osten, Süden oder Westen möglich.
- **Kein stufenloses Rotieren in der Vertikalen**: Die Kartenansicht kann zwar gekippt werden, wodurch die Karte dreidimensional dargestellt wird. Dies ist allerdings nur in zwei Abstufungen möglich. Es ist nicht möglich, die Höhe des Blickwinkels frei einzustellen.
- **Kein stufenloses Zoomen:** Der Benutzer kann in das Höhenprofil hinein- oder hinauszoomen. Wie bei der Rotation ist dies allerdings nur in Abstufungen möglich.
- **Keine Isolierung des Kartenausschnittes:** Nachdem die Karte gekippt wurde, reicht der sichtbare Bereich bis zu einem virtuellen Horizont. Das Höhenprofil umfasst somit auch Bereiche, die an den ursprünglich angezeigten Kartenausschnitt angrenzen. Die Beschränkung des Höhenprofils auf einen definierten Kartenausschnitt ist nicht möglich.

3.4 Google Earth

Abbildung 3: Darstellung des Matterhorns in Google Earth (Windows)

Eine der bekanntesten Applikationen zur dreidimensionalen Darstellung von realem Terrain ist *Google Earth*. Unter dem Namen *Keyhole* 2001 entwickelt wurde die Applikation 2004 von Google gekauft und umbenannt[6]. Seither steht sie als *Google Earth* kostenlos zum Download zu Verfügung und kann auf diversen Plattformen und Betriebssystemen installiert werden. *Google Earth* umfasst eine Basisversion, welche kostenlos heruntergeladen werden kann, sowie eine Pro-Version mit zusätzlichen Features. Diese erlauben beispielsweise den Einbezug von demographischen Daten, den Import von GIS[7]-Daten, Vermessung von Strecken und Flächen sowie das Erstellen von Videos. Die Pro-Version ist kostenpflichtig und richtet sich vor allem an Unternehmen. Sie findet beispielsweise Anwendung in der Visualisierung von Karten in der Tagesschau.

3.4.1 Vorteile von Google Earth

Im Gegensatz zum firmeneigenen Konkurrenten *Google Maps* kann das Höhenprofil in *Google Earth* stufenlos gedreht und gekippt werden. Der Zoomfaktor kann ausserdem in viel kleineren Intervallen eingestellt werden. Ähnlich wie *Google Maps* unterstützt *Google Earth* ausserdem das Überlagern des Kartenmaterials mit Metadaten. Im Gegensatz zu *Google Maps* müssen diese aber nicht notwendigerweise von Google selbst stammen, sondern können über eine Importfunktion von externen Quellen hinzugefügt werden.

[6] Quelle: http://de.wikipedia.org/wiki/Google_Earth
[7] Geographic Information System (s. Glossar)

Neben den erwähnten Funktionalitäten, welche auch in *Google Maps* vorhanden sind, bietet *Google Earth* eine Fülle von Features, die weit über die geplanten Möglichkeiten von *Google Maps* hinausgehen. So kann zum Beispiel der Verlauf der Sonne simuliert werden, die Applikation als Flugsimulator verwendet werden oder die Oberfläche des Mondes oder des Mars angezeigt werden.

3.4.2 Nachteile von Google Earth

Google Earth ist von den verglichenen Referenzprodukten bezüglich der Funktionsvielfalt wohl das umfangreichste. So umfangreich die Funktionalitäten aber auch sein mögen, so bietet auch *Google Earth* keine Möglichkeit, einen Kartenausschnitt isoliert und ohne angrenzende Bereiche darzustellen. Einige der Features sind ausserdem nur gegen Bezahlung erhältlich.

3.5 3D Outdoor Guides

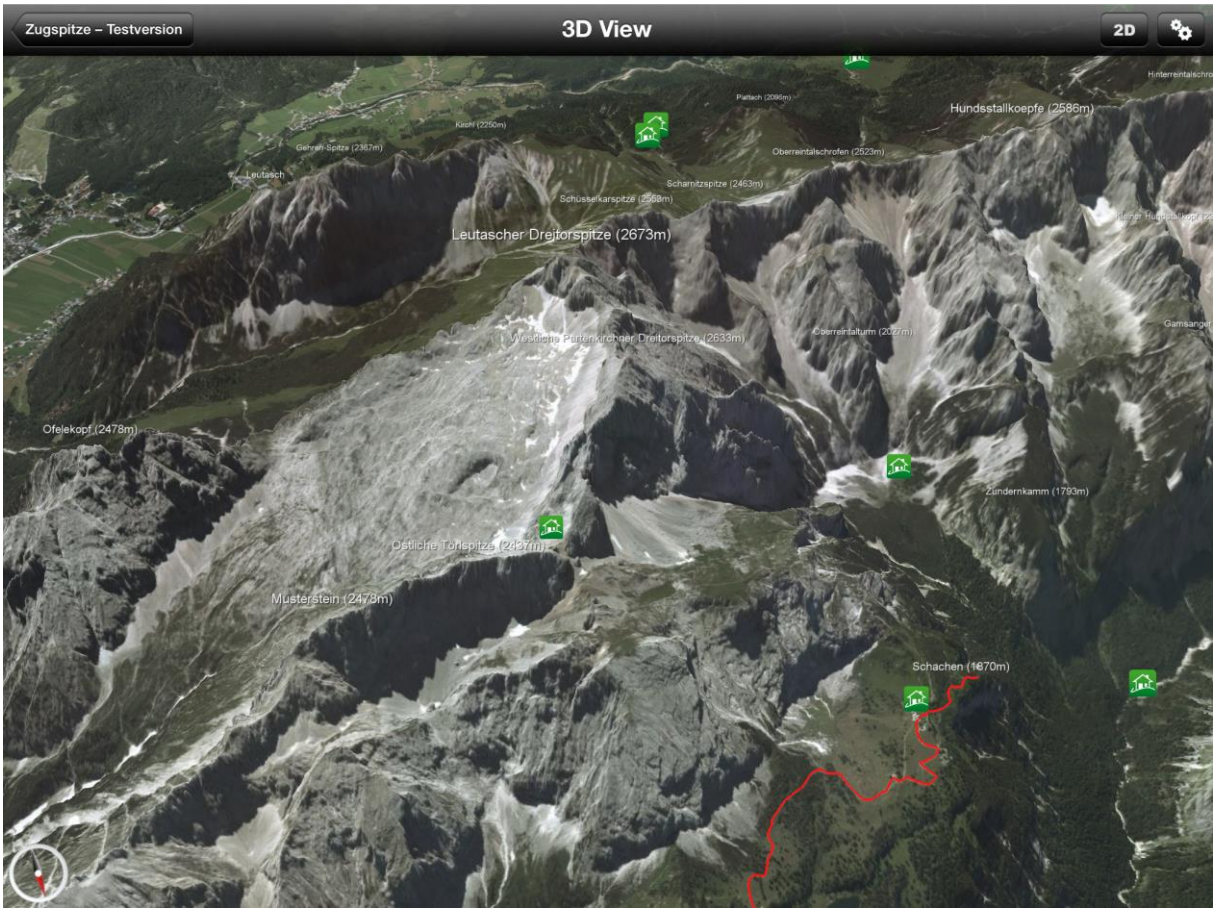

Abbildung 4: Screenshot der App "3D Outdoor Guides"

3D Outdoor Guides ist ein Produkt der Firma *3D Reality Maps* und richtet sich vor allem an Wanderer und Bergsteiger. Es erlaubt die Darstellung von Kartenmaterial auf mobilen Endgeräten (Android und iOS).

3.5.1 Vorteile von 3D Outdoor Guides

3D Outdoor Guides kommt der Funktionsweise von *Google Earth* sehr nahe. So unterstützt die App beispielsweise stufenloses Zoomen und Rotieren. Des Weiteren geht *3D Outdoor Guides* über den geplanten Funktionsumfang von *Maps3D* hinaus, indem es die Planung von Bergtouren mithilfe der mitgelieferten Routendaten ermöglicht.

3.5.2 Nachteile von 3D Outdoor Guides

Nachteilig bei der Benutzung von *3D Outdoor Guides* ist die Abhängigkeit von der Plattform. Die Applikation steht ausschliesslich für mobile Endgeräte und nicht für Desktop-Geräte zu Verfügung. Ebenfalls nachteilig wirkt sich aus, dass nur ein Teil des Kartenmaterials kostenlos visualisiert werden kann und sich auf die Darstellung der Alpenregion beschränkt.

3.6 HERE Maps

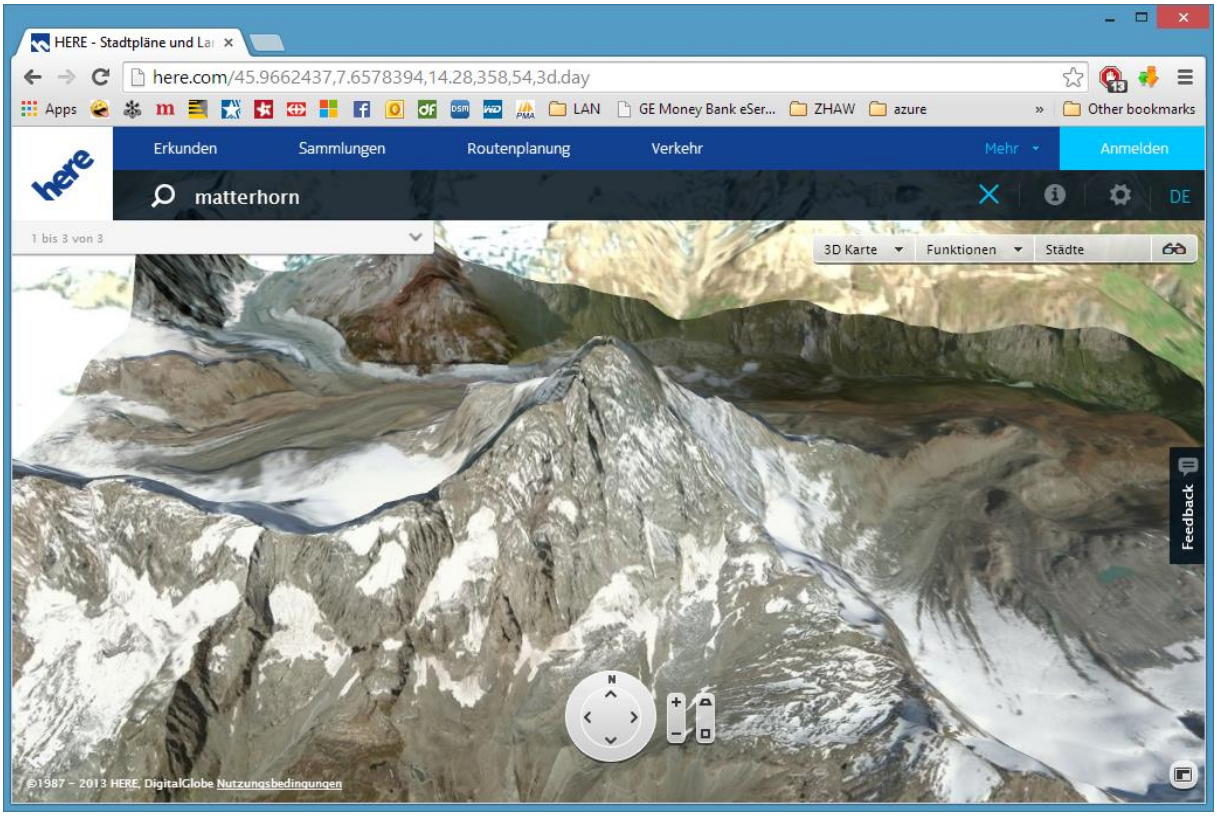

Abbildung 5: Darstellung des Matterhorns mit HERE Maps

Erstaunlicherweise kommt die Applikation, welche *Maps3D* funktionell am nächsten steht, nicht von Google, sondern von Nokia. Nokia hat durch den Kauf diverser Firmen das Know-How und die Datengrundlage zur Darstellung von interaktiven Karten aufgebaut. Die dazu notwendige Technologie finden sich heutzutage in vielen Navigationsgeräten wieder. Nokia stellt mit *HERE* eine Web-Applikation zu Verfügung, welche *Google Maps* nicht nur optisch, sondern auch funktionell sehr ähnlich ist. Im Gegensatz zu *Google Maps* kann die Ansicht aber in feineren Abstufungen gekippt werden

3.6.1 Vorteile von HERE Maps

Wie *Google Earth* bietet *HERE* Maps die Möglichkeit, die Karte stufenlos zu drehen und zoomen. Im Gegensatz zu *Google Earth* muss HERE aber nicht installiert werden, sondern ist als Web-Applikation in allen modernen Browsern lauffähig. Die Applikation kann kostenlos verwendet werden.

3.6.2 Nachteile von HERE Maps

Wie die übrigen Referenzprodukte bietet auch HERE Maps keine Möglichkeit, das Höhenprofil für einen Kartenausschnitt isoliert darzustellen.

3.7 Zusammenfassung und Vergleich

Die folgende Tabelle fasst den Vergleich der oben beschriebenen Produkte zusammen. Als Vergleichskriterien haben sich aufgrund der individuellen Vor- und Nachteile folgende Punkte herauskristallisiert:

- **Keine Installation**: Muss die Anwendung lokal installiert werden, bevor sie verwendet werden kann?
- **Realitätstreue**: Wird dem Höhenprofil mittel Satellitenbilder ein realistisches Aussehen verliehen?
- **Freigestelltes Profil**: Kann das Höhenprofil für einen Kartenausschnitt isoliert (d.h. ohne angrenzende Bereiche) dargestellt werden?
- **Zoomen**: Kann in das Höhenprofil hineingezoomt werden?
- **Rotieren**: Kann das Höhenprofil gedreht werden?
- **Wahl der Datenquelle**: Besteht eine Möglichkeit, Höhen- oder andere Geodaten von alternativen Quellen zu beziehen und in die Darstellung des Profils zu integrieren?
- **Kostenfrei**: Kann die Applikation gratis verwendet werden?

3.7.1 Innovation

In der letzten Spalte von Tabelle 4 ist *Maps3D* aufgeführt. Die eingetragenen Werte sind das Resultat des Vergleichs und repräsentieren den Mehrwert bzw. die Art der Innovation, welche *Maps3D* gegenüber ähnlichen Produkten bietet.

	Google Earth	Google Earth Pro	Google Maps	3D Outdoor Guides	HERE	Maps3D
Keine Installation	⊘	⊘	✓	⊘	✓	✓
Realitätstreue	✓	✓	✓	✓	✓	⊘
Freigestelltes Profil	⊘	⊘	⊘	⊘	⊘	✓
Zoomen	✓	✓	✓[1]	✓	✓	✓
Rotieren	✓	✓	✓[2]	⊘	✓	✓
Wahl der Datenquelle	⊘	✓[3]	⊘	⊘	⊘	✓[4]
Kostenfrei	✓	⊘	✓	✓[5]	✓	✓

Tabelle 4: Vergleich von ähnlichen Produkten mit Maps3D

3.7.2 Legende und Hinweise

✓ Feature verfügbar

⊘ Feature nicht verfügbar

✓(Zahl) Feature nur eingeschränkt verfügbar (siehe Hinweis zur entsprechenden Zahl)

1 Zoomen nur in Abstufungen möglich (kein freies Zoomen)

2 Rotation nur in Intervallen von 90 Grad möglich (kein freies Rotieren)

3 Als Importfunktion von GIS-Daten, deren Darstellung über die Kartendarstellung überlagert werden kann. Keine Wahl der Datenquelle für Geodaten.

4 Anbindung zusätzlicher Datenquellen für Höhendaten technisch vorbereitet, die Realisierung umfasst aber nur die Anbindung von *Google Elevations* als Lieferant von Höhendaten

5 App mit einigen Beispielen kostenfrei, zusätzliche Regionen müssen separat gekauft werden.

3.7.3 Fazit

Als Alleinstellungsmerkmale bietet *Maps3D* die Möglichkeit, ein Höhenprofil freigestellt, also ohne angrenzende Bereiche darzustellen sowie stufenlos zu zoomen oder rotieren. Im Gegensatz dazu müssen Abstriche bei der Darstellung des Höhenprofils gemacht werden, welches nicht gleich fein abgestuft ist und ohne realitätsnahe Darstellung (z.B. durch Überlagerung von Satellitenbildern) auskommen muss.

Teil 2: Konzept

4 Wahl der Technologie

In diesem Kapitel wird die Wahl der Technologie begründet, mit denen *Maps3D* realisiert wurde. Von besonderer Bedeutung war die Frage, ob die Datenbeschaffung und –Darstellung als getrennte Applikationen realisiert werden sollten (wie dies die Projektphasen vermuten lassen). Da dieser Punkt einen entscheidenden Einfluss auf die Anforderungen und die Architektur hatte, musste er vorgängig geklärt werden.

4.1 Varianten

Da ich beruflich fast ausschliesslich mit kompilierten Sprachen (hauptsächlich ActionScript) arbeite und mein Interesse im Bereich Web-Technologien eher privater Natur ist, standen mir zwei Möglichkeiten zu Verfügung:

1. Umsetzung als Web-Applikation in Kombination mit einer Applikation in ActionScript (AS3)
2. Umsetzung als reine Web-Applikation

Beide Varianten hatten Vor- und Nachteile, welche im Folgenden kurz beschrieben sind.

4.1.1 Umsetzung als Web-Applikation in Kombination mit einer AS3-Applikation

In dieser Variante würde der Bezug der Geodaten in einer einfachen (Web-)Applikation realisiert, welche lediglich die für das Höhenprofil benötigten Daten ermittelt. Diese würden anschliessend über eine Schnittstelle durch eine AS3-Applikation mithilfe einer entsprechende Library (z.B. *Away3D*[8]) dargestellt. Die AS3-Applikation wäre mit der AIR-Runtime als Standalone-App oder mit Flash in einem Browser ausführbar.

Der Vorteil bei dieser Variante läge darin, dass die Implementierung der 3D-Darstellung des Höhenprofils in einer mir nicht völlig unbekannten Technologie erfolgen würde. Zwar müsste ich mich in die *Away3D*-API einarbeiten, dafür kenne ich die Programmiersprache und verschiedene Frameworks bereits aus dem beruflichen Alltag.

Der Nachteil dieser Variante liegt in einer umständlichen Programmierung. Da die *GoogleMaps*-API nur für JavaScript verfügbar ist, müssten die Daten mit der Webapplikation beschafft und von dort irgendwie in die ActionScript-Applikation gelangen. Da dies über eine File-Schnittstelle nicht möglich ist (JavaScript unterstützt keinen Schreibzugriff auf das Dateisystem) müsste eine andere Lösung gefunden werden. Dies führt wahrscheinlich zu einer komplizierten Applikation, die im Grunde genommen aus zwei unabhängigen Teilapplikationen bestünde, welche behelfsmässig miteinander verbunden sind.

[8] AS3-Library zur Darstellung von 3D-Inhalten mit Flash: http://away3d.com/

4.1.2 Umsetzung als reine Web-Applikation

Bei dieser Variante würden sowohl die Datenbeschaffung als auch Darstellung des Höhenprofils in derselben Applikation stattfinden. Zur Darstellung des Höhenprofils als dreidimensionales Modell gäbe es wiederum mehrere Varianten, z.B. WebGL oder SVG[9].

Der Vorteil hierbei wäre, dass nur eine Applikation implementiert werden müsste, wobei das Risiko des Aufwands durch die Integration zweier unterschiedlicher Technologien eliminiert würde. Die Applikation wäre ausserdem einfacher zu bedienen, da alles im gleichen Browserfenster erfolgen könnte.

Nachteilig bei dieser Variante wirkt sich aus, dass mir Technologien zur Darstellung von 3D-Inhalten im Web noch völlig unbekannt sind und zuerst ein Initialaufwand für die Einarbeitung geleistet werden müsste (siehe auch Abschnitt *Risiken*).

4.1.3 Nutzwertanalyse

Zur Entscheidungsfindung führte ich eine Nutzwertanalyse durch, in welcher die beiden Varianten einander gegenüberstellt wurden. Dazu wurden die oben beschriebenen Punkte gewichtet und beide Varianten danach bewertet.

Die Gewichtung entspricht einem Faktor, den ich Aufgrund persönlicher Präferenz oder Einschätzung setzte.

Die Bewertung erfolgte auf einer Skala von 1 bis 5, wobei 1 für eine sehr schlechte Erfüllung des Kriteriums und 5 für eine sehr gute Erfüllung des Kriteriums mit der entsprechenden Variante steht.

[9] Mehr zum Vergleich WebGL oder SVG im Umsetzungsteil

Kriterium	Gewicht	AS3-Applikation		Web-Applikation	
		Einfach	Gewichtet	Einfach	gewichtet
Einarbeitungsaufwand	1.3	4	5.2	3	3.9
Plattformunabhängigkeit	1	2	2	4	4
Einfachheit der Applikation	1.5	2	3	5	7.5
Risiko durch fehlende Kenntnis	1.4	2	2.8	3	4.2
Risiko inkompatibler Technologien	1.5	2	3	4	6
Total		**12**	**16**	**19**	**25.6**

Tabelle 5: Nutzwertanalyse für Technologieentscheid

4.1.4 Fazit

Sowohl bei den ungewichteten als auch bei den gewichteten Werten schlägt die Variante der reinen Web-Applikation die AS3-Applikation in der Summe deutlich.

Somit wird die Applikation als reine Web-Applikation realisiert. Das Ergebnis entspricht meinem Bauchgefühl und meiner Erfahrung als Entwickler, wobei Applikationen so einfach wie möglich gehalten werden sollen (KISS-Prinzip[10]). Die Implementierung als reine Web-Applikation stellt zudem sicher, dass *Maps3D* ohne zusätzlichen Installationsaufwand verwendet werden kann.

Ein angenehmer Nebeneffekt ist, dass ich mich im Bereich Web-Entwicklung vertiefen kann und gleichzeitig eine neue Technologie kennen lerne.

[10] *Keep It Simple and Stupid*: Prinzip (u.a. in der Softwareentwicklung), wobei für ein Problem eine möglichst einfache Lösung gewählt werden sollte

5 Anforderungsanalyse

Dieses Kapitel enthält die Anforderungen an *Maps3D*. Als Grundlage dienten die Use Cases (Anwendungsfälle) des Systems, aus welchen die konkreten Anforderungen abgeleitet wurden. Für jede Anforderungen wurden Akzeptanztests definiert, anhand derer die Erfüllung der Anforderung überprüft werden kann.

Das Kapitel schliesst mit den Mockups[11] ab, welche die Benutzeroberfläche des Systems schematisch veranschaulicht. Die Mockups sollten eine Vorstellung darüber vermitteln, wie die Applikation schlussendlich aussehen würde. Da die Benutzerschnittstelle beim Erstellen der Mockups noch nicht implementiert war, sind Abweichungen zur umgesetzten Version möglich.

5.1 Stakeholder

Da ich gleichzeitig Auftraggeber und Umsetzer bin und das vorliegende Projekt eine Einzelarbeit ist, existieren ausser mir keine weiteren Stakeholder.

[11] Attrappe, Nachbildung zu Präsentationszwecken

5.2 Use Cases

Das folgende Diagramm zeigt die Anwendungsfälle (Use Cases) von *Maps3D,* bei denen ein Akteur (Benutzer oder externes System) in Interaktion mit *Maps3D* tritt.

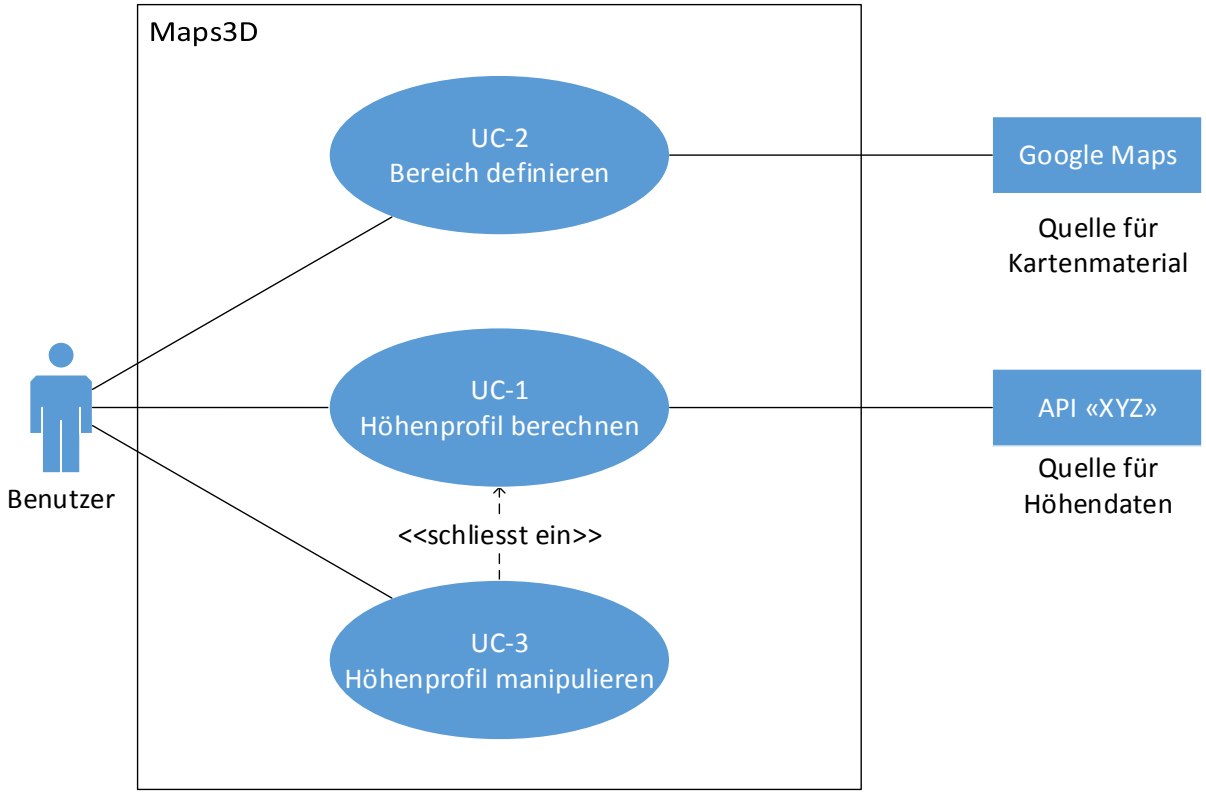

Abbildung 6: Use Case-Diagramm

Jeder Use Case ist anschliessend genauer nach folgendem Schema spezifiziert.

ID	UC-x
Beschreibung	Kurze Beschreibung im Stil einer User Story.
Abhängigkeiten	Referenzen auf andere Use Cases
Auslöser	Aktion oder Ereignis, welches den Use Case initiiert
Vorbedingung	Bedingungen die vor der Ausführung des Use Case erfüllt sein müssen.
Szenarien	Ablauf des Use Cases (oder Varianten davon)
Ergebnis	Erwartetes Resultat
Bemerkungen	Allfällige Kommentare

Tabelle 6: Schema der Use Case Spezifikation

5.2.1 UC-1: Höhenprofil berechnen

ID	UC-1
Beschreibung	Als Benutzer kann ich mit der Applikation ein Höhenprofil berechnen lassen, welches dreidimensional dargestellt wird.
Abhängigkeiten	UC-2
Auslöser	Der Benutzer klickt auf die Schaltfläche mit der Beschriftung „Profil berechnen“
Vorbedingung	Es ist ein Kartenausschnitt gemäss UC-2 definiert
Szenarien	• Benutzer definiert einen Kartenausschnitt gemäss UC-2 • Benutzer klickt Schaltfläche zum Generieren des Höhenprofils
Ergebnis	Das Höhenprofil wird angezeigt
Bemerkungen	Dies ist der Hauptanwendungsfall. Die Technologie zur Darstellung des Höhenprofils ist noch zu bestimmen.

Tabelle 7: Spezifikation Use Case UC-1

5.2.2 UC-2: Bereich definieren

ID	UC-2
Beschreibung	Als Benutzer kann ich mit einem Auswahlrechteck auf einer Karte einen Bereich definieren, für den das Höhenprofil berechnet wird.
Abhängigkeiten	keine
Auslöser	Der Benutzer startet die Applikation.
Vorbedingung	Es ist eine Karte mit einem Auswahlrechteck sichtbar.
Szenarien	Szenario 1: Ändern der Position des Auswahlrechtecks • Klick auf die Mitte des Auswahlrechtecks • Verschieben des Auswahlrechtecks Szenario 2: Ändern der Grösse des Auswahlrechtecks • Klick auf einen Eckpunkt des Auswahlrechtecks • Verschieben des Eckpunktes Szenario 3: Suche nach einem Ort • Eingabe eines Ortsnamens in ein Textfeld • Drücken der Enter-Taste
Ergebnis	Szenario 1: Das Auswahlrechteck bleibt an der Zielposition Szenario 2: Das Auswahlrechteck behält seine neue Grösse Szenario 3: Der gefundene Ort wird im Kartenausschnitt zentriert und das Auswahlrechteck liegt mittig darüber
Bemerkungen	• Das Auswahlrechteck ist immer sichtbar und kann vom Benutzer nicht entfernt werden. • Nachdem ein Ort gefunden wurde nimmt das Auswahlrechteck wieder seine ursprüngliche Grösse ein.

Tabelle 8: Spezifikation Use Case UC-2

5.2.3 UC-3: Höhenprofil manipulieren

ID	UC-3
Beschreibung	Als Benutzer kann ich die Ansicht auf das Höhenprofils verändern.
Abhängigkeiten	UC-1
Auslöser	Der Benutzer klickt mit der Maus auf das Höhenprofil.
Vorbedingung	Es wurde ein Höhenprofil berechnet, welches nun sichtbar ist.
Szenarien	Szenario 1: Rotation • Klicken und Halten der linken Maustaste auf dem Höhenprofil • Bewegen der Maus in eine beliebige Richtung Szenario 2: Zoom • Klicken und Halten der mittleren Maustaste auf dem Höhenprofil • Bewegen der Maus nach oben oder unten
Ergebnis	• Szenario 1: Der Blickwinkel auf das Höhenprofil wird geändert • Szenario 2: Das Höhenprofil wird herangezoomt (Bewegung nach oben) bzw. weggezoomt (Bewegung nach unten)
Bemerkungen	Die Bedienung mit der Maus entspricht einer intuitiven Bedienung, wie sie auch z.B. in Google Earth verwendet wird und sollte wenn möglich so übernommen werden.

Tabelle 9: Spezifikation Use Case UC-3

5.3 Anforderungen

Aus den oben formulierten Use Cases wurden Anforderungen abgeleitet. Diese sind in folgende Kategorien unterteilt:

- **Funktionale Anforderungen**: Funktionalitäten, welche das System aufweisen soll.
- **Nicht-Funktionale Anforderungen**: Qualitätsanforderungen sowie weitere Anforderungen an das System, welche keine funktionalen Anforderungen sind.
- **Rahmenbedingungen**: Faktoren, welche direkt oder indirekt Einfluss auf die Anforderungen haben

Zur Dokumentation der Anforderungen wurde folgendes Schema verwendet:

ID	Eindeutige Identifikation gemäss folgendem Muster • *REQ-F-…* für funktionale Anforderungen • *REQ-NF-…* für nicht-funktionale Anforderungen
Name	Eindeutiger Bezeichner
Beschreibung	Kurzbeschreibung der Anforderung in natürlicher Sprache
Priorität	Prioritäten können niedrig, mittel oder hoch sein
Vorbedingungen	Allfällige Punkte, die vorgängig erfüllt sein müssen
Akzeptanztests	Akzeptanztests, welche erfüllt sein müssen, damit die Anforderung als umgesetzt markiert werden kann
Beziehungen	Auflistung allfälliger Bezüge zu Use Cases oder anderen Anforderungen

Tabelle 10: Schema für die Beschreibung der Anforderungen

Um eine möglichst vollständige und einheitliche Erfassung zu gewährleisten und gleichzeitig Zweideutigkeiten aufgrund sprachlicher Transformationseffekte auf ein Minimum zu reduzieren, verwende ich für die Beschreibung die folgende Satzschablone gemäss Pohl&Rupp[12].

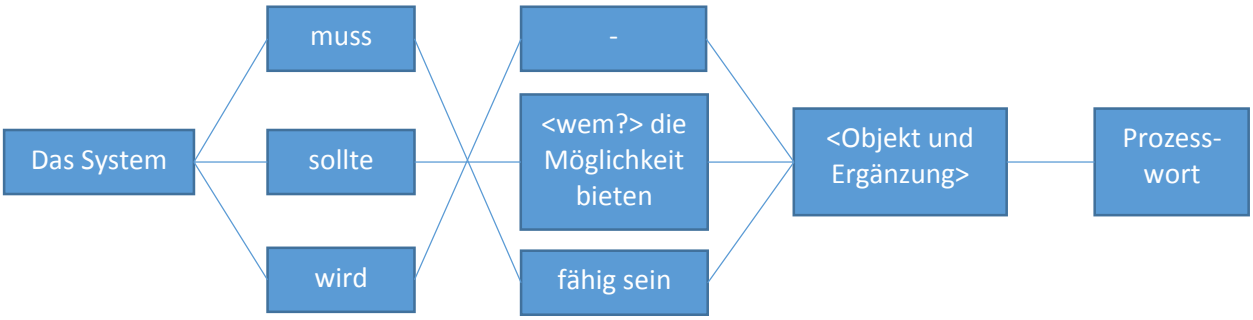

Abbildung 7: Natürlichsprachliche Satzschablone gemäss Pohl/Rupp Kap. 5.2

[12] Vgl. Pohl/Rupp, *Basiswissen Requirements Engineering*, Kapitel 5

5.3.1 Funktionale Anforderungen

Folgende funktionale Anforderungen wurden identifiziert.

ID	REQ-F-1
Name	Kartenausschnitt anzeigen
Bezug Use Cases	UC-2
Akzeptanztests	T-01, T-03
Beschreibung	Das System muss dem Benutzer die Möglichkeit bieten, mit einem Auswahlrechteck auf einer Karte einen beliebigen Punkt auf der Erde anzuzeigen.
Priorität	Hoch
Vorbedingungen	Der Benutzer hat die Applikation gestartet und es besteht eine Internetverbindung.
Bemerkungen	• Die Karte muss nicht zoombar sein. • Die Karte definiert nicht den Bereich, für welchen das Höhenprofil berechnet werden soll. Dieser wird durch das Auswahlrechteck definiert, welches über der Karte liegt und vom Benutzer verändert werden kann (s. REQ-F-2 und REQ-F-3).

Tabelle 11: Beschreibung der Anforderung REQ-F-1: "Bereich definieren"

ID	REQ-F-2
Name	Position des Auswahlrechtecks ändern
Bezug Use Cases	UC-2
Akzeptanztests	T-02
Beschreibung	Das System muss dem Benutzer die Möglichkeit bieten, die Position des Auswahlrechtecks manuell zu verändern.
Priorität	Hoch
Vorbedingungen	Es ist ein Auswahlrechteck vorhanden (REQ-F-1)
Bemerkungen	Das Auswahlrechteck muss nicht durch den Benutzer hinzugefügt werden, sondern soll schon vorhanden sein, muss aber vom Benutzer verändert werden können.

Tabelle 12: Beschreibung der Anforderung REQ-F-2 "Position des Auswahlrechtecks ändern"

ID	REQ-F-3
Name	Grösse des Auswahlrechtecks ändern
Bezug Use Cases	UC-2
Akzeptanztests	T-04
Beschreibung	Das System muss dem Benutzer die Möglichkeit bieten, die Grösse des Auswahlrechtecks manuell zu verändern.
Priorität	Hoch
Vorbedingungen	Es ist ein Auswahlrechteck vorhanden (REQ-F-1)
Bemerkungen	Die Grösse des Auswahlrechtecks beeinflusst nicht die Auflösung des Höhenprofils. Das bedeutet, dass aus einem grösseren Auswahlrechteck nicht eine grössere Anzahl Abtastpunkte resultiert.

Tabelle 13: Beschreibung der Anforderung REQ-F-3 "Grösse des Auswahlrechtecks ändern"

ID	REQ-F-4
Name	Ort suchen
Bezug Use Cases	UC-2
Akzeptanztests	T-05
Beschreibung	Das System muss dem Benutzer die Möglichkeit bieten, nach einem Ort zu suchen.
Priorität	mittel
Vorbedingungen	Es wird eine Karte angezeigt (REQ-F-1).
Bemerkungen	Nach Möglichkeit sollen dem Benutzer bereits während der Eingabe des Ortsnamens Vorschläge gemacht werden.

Tabelle 14: Beschreibung der Anforderung REQ-F-4: "Ort suchen"

ID	REQ-F-5
Name	Höhenmodell berechnen und darstellen
Bezug Use Cases	UC-1
Akzeptanztests	T-06, T-07
Beschreibung	Das System muss fähig sein, für den definierten Kartenausschnitt ein dreidimensionales Höhenprofil zu berechnen.
Priorität	Hoch
Vorbedingungen	Der Benutzer hat die Applikation gestartet und einen Bereich für das Höhenprofil definiert (REQ-F-1).
Bemerkungen	-

Tabelle 15: Beschreibung der Anforderung REQ-F-5: "Höhenmodell berechnen und darstellen"

ID	REQ-F-6
Name	Daten beziehen
Bezug zu Use Cases	UC-1
Akzeptanztests	T-06
Beschreibung	Das System muss fähig sein, Daten von einem Datenlieferanten über eine API zu beziehen.
Priorität	hoch
Vorbedingungen	Es wurde ein Bereich für die Berechnung des Höhenprofils definiert. (REQ-F-1)
Bemerkungen	-

Tabelle 16: Beschreibung der Anforderung REQ-F-6: Daten beziehen

ID	REQ-F-7
Name	Höhenprofil rotieren
Bezug Use Cases	UC-3
Akzeptanztests	T-08
Beschreibung	Das System muss dem Benutzer die Möglichkeit bieten, die Ansicht auf das Höhenprofil in einem beliebigen Winkel stufenlos zu drehen.
Priorität	hoch
Vorbedingungen	Es wurde ein Höhenprofil berechnet und dargestellt (REQ-F-5)
Bemerkungen	Die Bedienung sollte wenn möglich mit der Maus und möglichst intuitiv erfolgen

Tabelle 17: Beschreibung der Anforderung REQ-F-7: Höhenprofil rotieren

ID	REQ-F-8
Name	Höhenprofil zoomen
Bezug Use Cases	UC-3
Akzeptanztests	T-09
Beschreibung	Das System muss dem Benutzer die Möglichkeit bieten, das Höhenprofil stufenlos zu zoomen.
Priorität	hoch
Vorbedingungen	Es wurde ein Höhenprofil berechnet und dargestellt (REQ-F-5)
Bemerkungen	Die Bedienung sollte wenn möglich mit der Maus und möglichst intuitiv erfolgen

Tabelle 18: Beschreibung der Anforderung REQ-F-8: Höhenprofil zoomen

5.3.2 Nicht-Funktionale Anforderungen

Die folgenden nicht-funktionalen Anforderungen müssen vom System „*Maps3D*“ erfüllt werden.

ID	REQ-NF-1
Name	Limitierung der Anfragen
Querbezüge	UC-1, R-2, Risiko 1 (Abhängigkeit von Google)
Akzeptanztests	T-10
Beschreibung	Das System sollte für die Berechnung eines Höhenprofils pro Request zur *Google Elevations* API nicht mehr als 512 Punkte abfragen. Insgesamt dürfen für ein Höhenprofil nicht mehr als 2‘500 Requests oder 25‘000 Punkte abgefragt werden (das zuerst Erreichte gilt).
Priorität	hoch
Vorbedingungen	Die Anbindung an die Google-API wurde implementiert
Bemerkungen	Diese Einschränkung ist aufgrund der Limitierung der Nutzung der Google-API nötig und gilt nur für dessen Anbindung als Quelle von Höhendaten. Für die Anbindung anderer Quellen gilt diese Einschränkung möglicherweise nicht.

Tabelle 19: Beschreibung der Anforderung REQ-NF-1: Limitierung der Anfragen

ID	REQ-NF-2
Name	Erweiterbarkeit
Querbezüge	UC-1, Risiko 1 (Abhängigkeit von Google)
Akzeptanztests	T-11
Beschreibung	Das System sollte dem Entwickler die Möglichkeit bieten, eine zusätzliche Datenquelle für Höhendaten anzubinden.
Priorität	hoch
Vorbedingungen	Der Entwickler kennt die Funktionsweise des Systems.
Bemerkungen	-

Tabelle 20: Beschreibung der Anforderung REQ-NF-2: Erweiterbarkeit

ID	REQ-NF-3
Name	Plattformunabhängigkeit und Installation
Querbezüge	-
Akzeptanztests	T-12
Beschreibung	Das System muss plattformunabhängig sein und dem Benutzer die Funktionalität ohne Installation eines Programmes anbieten
Priorität	niedrig
Vorbedingungen	keine
Bemerkungen	Diese Anforderung ist „nice to have“, muss aber nicht unbedingt umgesetzt werden. Im Notfall müsste der Benutzer einmalig ein Plugin installieren, bevor er die Applikation starten kann.

Tabelle 21: Beschreibung der Anforderung REQ-NF-3: Plattformunabhängigkeit

5.3.3 Rahmenbedingungen

Folgende Rahmenbedingungen beeinflussen die Anforderungen bzw. schränken diese ein.

ID	Beschreibung	Abhängigkeiten
R-1	Innerhalb von 24 Stunden können maximal 2‘500 Requests zur *Google Maps* API gemacht werden	REQ-F-6, REQ-NF-1, Risiko 1
R-2	Pro Request dürfen die Höhendaten von maximal 512 Punkten abgefragt werden	REQ-F-6, REQ-NF-1, Risiko 1
R-3	Innerhalb von 24 Stunden dürfen die Höhendaten von maximal 25‘000 Punkten abgefragt werden.	REQ-F-6, REQ-NF-1, Risiko 1
R-4	WebGL wird nicht auf allen Browsern unterstützt[13]	REQ-NF-3

Tabelle 22: Rahmenbedingungen

[13] Dieses Rahmenbedingung kam erst nach der Wahl der Technologie hinzu

5.4 Akzeptanztests

ID	T-01
Bezeichnung	Anzeige eines Kartenausschnittes
Geprüfte Anforderungen	REQ-F-1
Status	Erfüllt
Ablauf	• Starten der Applikation durch Aufrufen in einem Browser • Überprüfen des dargestellten Kartenausschnittes und des darauf definierten Auswahlrechtecks
Soll	• Der Kartenausschnitt zeigt das Matterhorn zentriert. • Ein Auswahlrechteck wird mittig über dem Kartenausschnitt angezeigt.
Ist	Das Verhalten entspricht dem Soll-Verhalten
Bemerkung	-

Tabelle 23: Akzeptanztest T-01

ID	T-02
Bezeichnung	Verschieben des Auswahlrechtecks
Geprüfte Anforderungen	REQ-F-2
Status	Erfüllt
Ablauf	• Positionierung des Maus-Cursor innerhalb der Fläche des Rechtecks • Verschieben des Rechtecks durch „Drag’n’Drop“
Soll	• Das Rechteck lässt sich mit der Maus verschieben. • Falls das Auswahlrechtecks über die Kartengrenzen hinaus verschoben wird, verschiebt sich der angezeigte Kartenausschnitt zusammen mit dem Rechteck
Ist	Das Verhalten entspricht dem Soll-Verhalten
Bemerkung	Der Cursor wechselt über dem Auswahlrechteckt zu einer Hand (vom Browser für Links verwendeter Cursor).

Tabelle 24: Akzeptanztest T-02

ID	T-03
Bezeichnung	Verschieben des angezeigten Kartenausschnittes
Geprüfte Anforderungen	REQ-F-1
Status	Erfüllt
Testablauf	Ändern des dargestellten Kartenausschnitts durch Klick auf die Karte und Drag’n’Drop
Soll	• Die angezeigte Karte lässt sich verschieben • Das Auswahlrechteck verschiebt sich mit der Karte und bleibt zentriert.
Ist	Das Verhalten entspricht dem Soll-Verhalten.
Bemerkung	-

Tabelle 25: Akzeptanztest T-03

ID	T-04
Bezeichnung	Ändern der Grösse des Auswahlrechtecks
Geprüfte Anforderungen	REQ-F-1
Status	Erfüllt
Ablauf	Ändern der Grösse des Auswahlrechtecks durch Klicken und Ziehen einer der Eckpunkte
Soll	Die Grösse des Rechtecks lässt sich verändern
Ist	Die Grösse des Rechtecks lässt sich verändern. Die veränderte Grösse lässt sich durch Klick auf das entsprechende Icon neben dem gezogenen Eckpunkt auf die ursprüngliche Grösse zurücksetzen.
Bemerkung	-

Tabelle 26: Akzeptanztest T-04

ID	T-05
Bezeichnung	Suche nach einem Ort
Geprüfte Anforderungen	REQ-F-4
Status	Erfüllt
Ablauf	Suche eines Ortes durch Eingabe eines Ortsnamens im Suchfeld und Bestätigen durch Drücken der Enter-Taste.
Soll	• Der gesuchte/gewählte Ort wird zentriert auf der Karte angezeigt • Das Auswahlrechteck liegt über dem gefundenen Punkt auf der Karte
Ist	Das Verhalten entspricht dem Soll-Verhalten. Zusätzlich werden dem Benutzer schon bei der Suche Vorschläge gemacht.
Bemerkung	Das Auswahlrechteck hat nach Anzeige des Ortes dieselbe Grösse wie beim Starten der Applikation (T-01).

Tabelle 27: Akzeptanztest T-05

ID	T-06
Bezeichnung	Anzeige des Fortschrittes beim Bezug der Höhendaten
Geprüfte Anforderungen	REQ-F-5, REQ-F-6
Status	Erfüllt
Ablauf	Klicken des Buttons zum Berechnen des Höhenprofils
Soll	Der Fortschritt des Bezuges der Höhendaten und der Berechnung des Höhenprofils wird mit einem Fortschrittsbalken angezeigt
Ist	Der Fortschritt wird durch ein Popup mit einem Fortschrittsbalken angezeigt. Nach Beenden des Vorganges schliesst das Popup automatisch.
Bemerkung	Der Vorgang lässt sich durch Schliessen des Popups abbrechen.

Tabelle 28: Akzeptanztest T-06

ID	T-07
Bezeichnung	Korrektheit des Höhenprofils
Geprüfte Anforderungen	REQ-F-5
Status	Erfüllt
Ablauf	Darstellung des Höhenprofils für das Matterhorn durch Starten der Applikation und Klicken des Buttons für die Berechnung des Höhenprofils.
Soll	• Das Matterhorn ist durch das Höhenprofil als solches erkennbar. • Der tiefste Punkt des Höhenprofils hat die Höhe 0 • Das Seitenverhältnis der Grundfläche des Höhenprofils entspricht demjenigen des Auswahlrechtecks
Ist	Die Darstellung entspricht der Soll-Darstellung
Bemerkung	Da nicht sämtliche Punkte oder auf der Welt als Testkandidaten herangezogen werden können, wird die Korrektheit des Höhenprofils anhand einiger Berge mit charakteristischer Morphologie überprüft. Der Test wurde mit folgenden weiteren Kandidaten wiederholt: • Eiger, Mönch und Jungfrau • Finsteraarhorn

Tabelle 29: Akzeptanztest T-07

ID	T-08
Bezeichnung	Rotation des Höhenprofils
Geprüfte Anforderungen	REQ-F-7
Status	Erfüllt
Ablauf	Berechnung eines beliebigen Höhenprofils und Rotation durch Klick mit der linken Maustaste auf das Profil und Bewegen der Maus.
Soll	• Das Höhenprofil lässt sich ohne wahrnehmbare Abstufungen auf allen Achsen drehen. • Die Achsen können auch kombiniert werden („schräges“ Rotieren) • Das Zentrum während und nach der Rotation bleibt immer der Mittelpunkt des Höhenprofils
Ist	Das Verhalten entspricht dem Soll-Verhalten
Bemerkung	-

Tabelle 30: Akzeptanztest T-08

ID	T-09
Bezeichnung	Zoomen des Höhenprofils
Geprüfte Anforderungen	REQ-F-8
Status	Erfüllt
Ablauf	Berechnung eines beliebigen Höhenprofils und Zoom durch Klick mit der mittleren Maustaste auf das Profil und Bewegen der Maus vom Benutzer weg oder zum Benutzer hin.
Soll	Die virtuelle Kamera bewegt sich ohne wahrnehmbare Abstufung zum Höhenprofil hin (Mausbewegung vom Benutzer weg) oder weg (Mausbewegung zum Benutzer hin)
Ist	Das Verhalten entspricht dem Soll-Verhalten
Bemerkung	Der Benutzer bewegt mit der Maus die virtuelle Kamera und nicht das Höhenprofil selber.

Tabelle 31: Beschreibung des Akzeptanztests T-09

ID	T-10
Bezeichnung	Limitierung der Anfragen
Geprüfte Anforderungen	REQ-NF-1
Status	Erfüllt
Ablauf	Bezug der Höhendaten durch die *Google Elevations* API.
Soll	1. Die Berechnung eines Höhenprofils löst keinen Request zur *Google Elevations* API mit mehr als 512 Punkten aus. 2. Die Berechnung eines Höhenprofils ermittelt die Höhendaten von insgesamt nicht mehr als 25‘000 Punkten über die *Google Elevations* API. 3. Die Berechnung eines Höhenprofils löst nicht mehr als 2‘500 Requests zur *Google Elevations* API aus.
Ist	1. Pro Request werden maximal 150 Punkte abgefragt (Aufgrund der Beschränkung der Länge des Request-URL musste die Anzahl Punkt pro Request reduziert werden) 2. Pro Höhenprofil werden maximal 10‘000 Punkte abgefragt 3. Pro Höhenprofil werden maximal 67 Requests abgesetzt
Bemerkung	Punkt 2 und 3 wurden durch die Beschränkung der Auflösung des Höhenprofils auf maximal 100x100 Punkte (Total 10‘000 Punkte mit 150 Punkten pro Request) erreicht.

Tabelle 32: Beschreibung des Akzeptanztests T-10

ID	T-11
Bezeichnung	Überprüfen der Erweiterbarkeit
Geprüfte Anforderungen	REQ-NF-2
Status	Erfüllt
Ablauf	Technisches Review der Architektur
Soll	Die Einbindung einer alternativen Quelle von Höhendaten darf lediglich Änderungen an einer einzigen Klasse zur Folge haben.
Ist	Ein alternativer Lieferant von Höhendaten kann durch Implementierung eines entsprechenden Adapters realisiert werden. Um diesen einzubinden sind lediglich Änderungen an der *ProfileModel*-Klasse notwendig ⇨ vgl. Kapitel 6 (Architektur) und 8 (Umsetzung)
Bemerkung	Die Wahl der Quelle für Höhendaten kann nicht durch den Benutzer ausgewählt werden, sondern muss in der Applikationslogik implementiert werden

Tabelle 33: Beschreibung des Akzeptanztests T-11

ID	T-12
Bezeichnung	Sicherstellung der Plattformunabhängigkeit und Minimierung des Installationsaufwands
Geprüfte Anforderungen	REQ-NF-3
Status	Erfüllt
Ablauf	Starten der Applikation in einem Browser
Soll	Der Benutzer kann die Applikation unter Windows, OSX und Linux starten, ohne zuerst Plugins oder weitere Programme installieren zu müssen.
Ist	Die Applikation ist eine reine Web-Applikation und kann in allen modernen Browsern unter Windows, OSX und Linux ohne Installation eines Plugins gestartet werden.
Bemerkung	-

Tabelle 34: Beschreibung des Akzeptanztests T-12

5.6 GUI-Mockup

Das folgende Bild zeigt ein Mockup des GUI, über welches der Benutzer mit der Applikation interagiert. Das Mockup diente als Anhaltspunkt für Anzahl und Art der einzelnen GUI-Elemente, welche anschliessend kurz beschrieben sind. Die Darstellung ist konzeptionell und erfolgte vor der Umsetzung, deshalb entsprechen Aussehen und Anordnung nicht exakt der umgesetzten Variante.

Die Benutzeroberfläche wurde bewusst simpel gehalten, um die Bedienung zu vereinfachen und den Benutzer nicht mit der Darstellung unnötiger Bedienelemente zu verwirren.

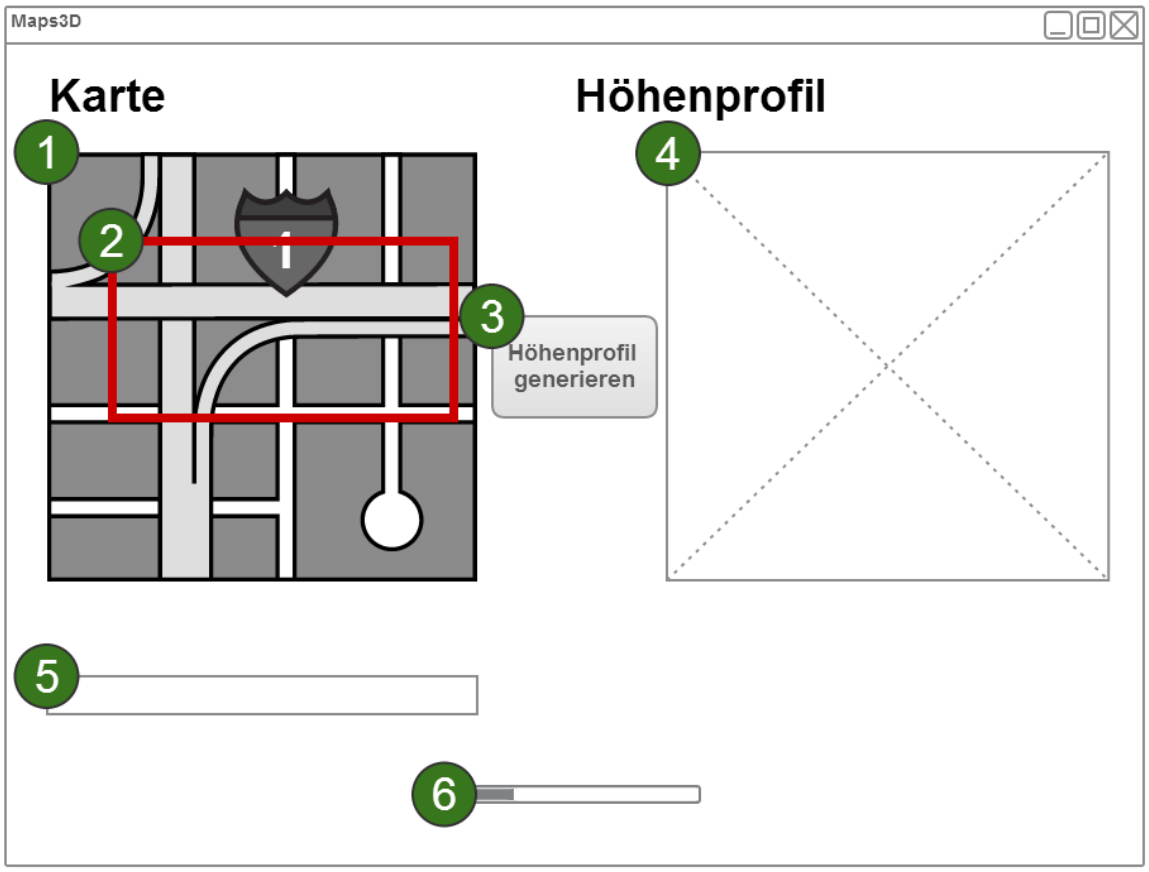

Abbildung 8: Mockup der Benutzerschnittstelle von Maps3D

5.6.1 UI-Elemente

	Typ/Bezeichnung	Beschreibung
1	Kartenausschnitt	Bereich zur Darstellung der Karte. Das Kartenmaterial stammt von *Google Maps*. Auf dem Kartenausschnitt kann der Benutzer einen beliebigen Bereich definieren, für welchen das Höhenprofil berechnet wird.
2	Auswahlrechteck	Ein vom Benutzer definierter, rechteckiger Rahmen. Grösse und Seitenverhältnis müssen nicht demjenigen des Kartenausschnitts entsprechen
3	Schaltfläche „Höhenprofil generieren“	Ein Klick auf diese Schaltfläche startet den Bezug der Höhendaten und die Berechnung des Höhenprofils.
4	Höhenprofil	Fläche zur Darstellung des Höhenprofils.
5	Textfeld	Mit diesem Textfeld kann nach Orten gesucht werden.
6	Fortschrittsbalken	Dieser Balken zeigt den Fortschritt der Berechnung des Höhenprofils nach Klick auf die Schaltfläche „Höhenprofil generieren“

Tabelle 35: UI-Elemente

6 Architektur

Im folgenden Abschnitt wird der schematische Aufbau von *Maps3D* beschrieben. Im Laufe des Entwicklungsprozesses wurden diverse Entscheidungen getroffen, welche zur gewählten Architektur führten. Diese Entscheidungen werden beschrieben und begründet.

6.1 Systemumfeld (System- und Kontextabgrenzung)

Das System besteht aus einer Web-Applikation, welche die Applikationslogik für den Bezug, die Verarbeitung und die Darstellung der Geodaten enthält und clientseitig ausgeführt wird. Die Daten werden über eine API bezogen und nicht persistent gespeichert. In der Abgabeversion bezieht *Maps3D* die Daten von Google. Weitere Lieferanten sind denkbar.

Das folgende Diagramm stellt *Maps3D* im Kontext mit externen System und Benutzern dar.

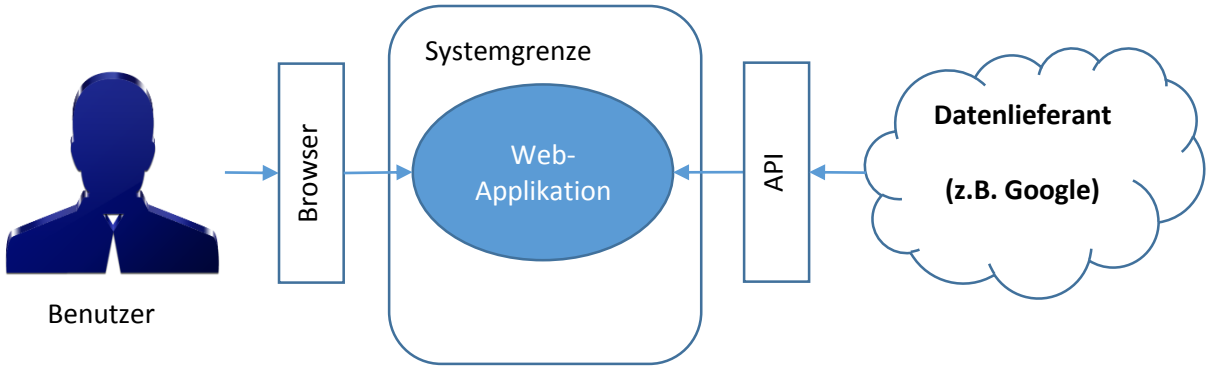

Abbildung 9: Systemkontext

6.2 Architekturstil

Um die Applikation für den Benutzer möglichst einfach zu halten, habe ich mich für eine Single-Page-Applikation[14] entschieden. Die im GUI verwendeten DOM-Elemente werden mit JavaScript erstellt und manipuliert. Da die Applikationslogik ebenfalls in JavaScript realisiert wurde und die Daten nicht zwischengespeichert werden, konnte *Maps3D* ohne eigenes Backend realisiert werden.
Maps3D ist somit eine reine Client-Applikation, die nur für den Bezug von Höhendaten Verbindung zu einem Server aufnehmen muss.

[14] Webapplikation, welcher aus einem einzigen HTML-Dokument besteht, dessen Inhalte dynamisch nachgeladen und verändert werden

6.3 Architekturmuster/Grobkonzept

Um die Applikationslogik möglichst lose gekoppelt von der Präsentationslogik zu halten, habe ich das MVC[15]-Architekturmuster verwendet. Dadurch erhoffe ich mir eine möglichst grosse Flexibilität im Hinblick auf zukünftige Erweiterungen. So soll es beispielsweise möglich sein, später auch eine andere Datenquelle als Google für den Bezug von Höhendaten zu verwenden (nicht-funktionales Requirement REQ-NF-2), ohne dafür die ganze Applikation umschreiben zu müssen.

Die folgende Grafik zeigt die von der Applikation verwendeten Elemente sowie deren Einbettung im MVC-Kontext.

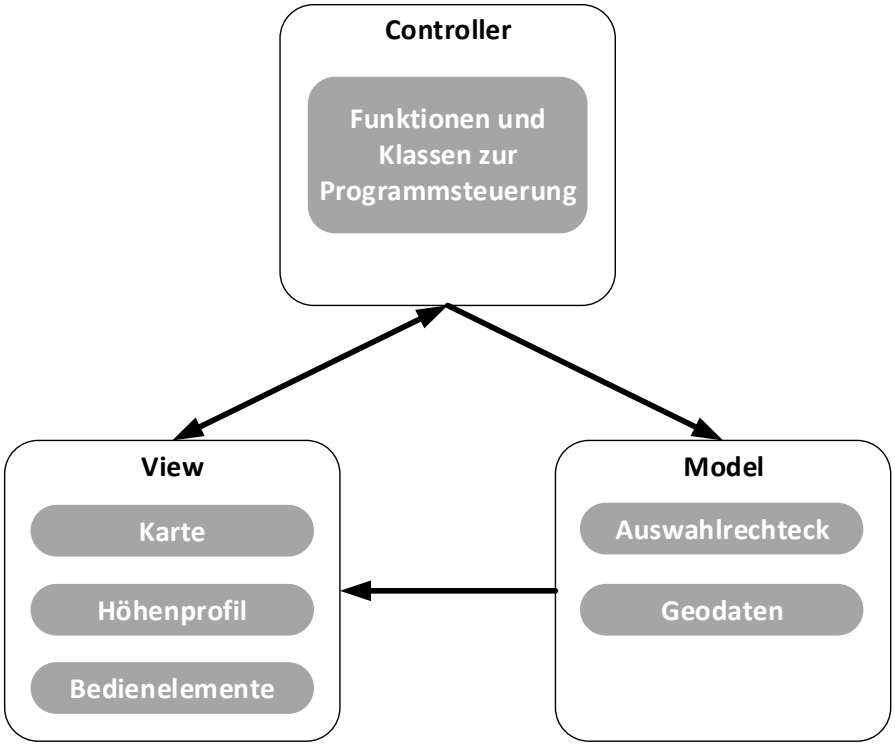

Abbildung 10: Grobkonzept von Maps3D

[15] Model View Controller: Architekturmuster in der Softwareentwicklung

6.4 Komponenten

Das folgende Diagramm visualisiert die Komponenten, aus denen *Maps3D* besteht sowie ihre Schnittstellen zu externen Systemen. Zusätzlich zum Systemkontext aus Abbildung 9 ist in diesem Diagramm die interne Struktur ersichtlich (White-Box). Im Gegensatz zum (abstrakten) Grobkonzept aus Abbildung 10 zeigt diese Darstellung die Aufteilung der Applikation in konkrete Bausteine. Die einzelnen Komponenten sind weiter unten detailliert beschrieben.

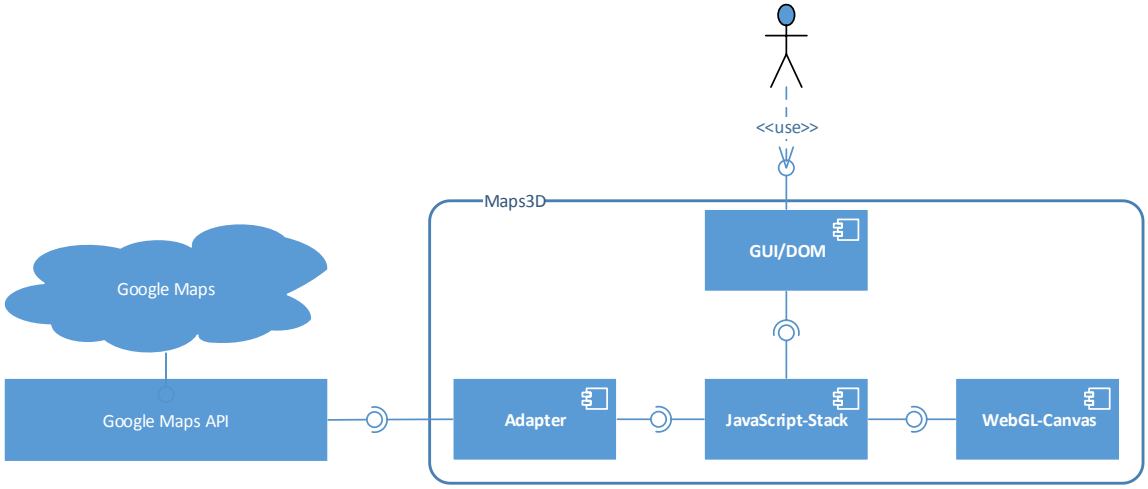

Abbildung 11: Komponentendiagramm von Maps3D

6.4.1 GUI/DOM

Das GUI ist eine in HTML realisierte Komponente, welche ausschliesslich dazu dient, dem Benutzer die für die Interaktion benötigten visuellen Bedienelemente in Form eines DOM[16] bereitzustellen. Zum DOM gehören:

- **eine Google-Maps Karte**, auf welcher ein beliebiger Ausschnitt definiert werden kann. Die Technologie dazu stammt von Google selber und sorgt dafür, dass die Karte innerhalb der HTML-Seite angezeigt wird und navigierbar ist.
- **Schaltflächen** zur Bestätigung der Eingabe bzw. Auslösen der Berechnung des Höhenprofils
- **Informative Komponenten** zur Darstellung von Metadaten wie z.B. ein Popup mit einem Ladebalken, der zur Anzeige des Fortschritts eines beliebigen Vorgangs verwendet werden kann.

Das GUI stellt lediglich das DOM zu Verfügung und wird durch entsprechende Klassen und Funktionen im JavaScript-Stack verändert, ohne selber JavaScript zu enthalten. Dies entspricht dem MVC-Paradigma, gemäss welchem der Presentation-Layer keine Applikationslogik enthalten sollte.

[16] Document Object Model

6.4.2 WebGL-Canvas

Die Darstellung des Höhenprofils erfolgt innerhalb der Web-Applikation in einer so genannten *Canvas*, einem speziellen HTML-Element, in welchem mittels JavaScript gezeichnet werden kann. Somit gehört die WebGL-Canvas zwar eigentlich zum DOM. Da die dargestellten Inhalte jedoch durch die Grafikkarte und nicht durch die vom Browser verwendete Layout-Engine gerendert werden, ist dieses Element separat aufgeführt.

6.4.3 JavaScript-Stack

Der JavaScript-Stack ist eine Sammlung von JavaScript-Files. Die Files definieren die Klassen[17] und Funktionen, aus denen *Maps3D* besteht. Sie beinhalten die Applikationslogik.

6.4.4 Adapter zum Bezug von Höhendaten

Die Anbindung zu einem externen Lieferanten von Höhendaten erfolgt über einen so genannten Adapter. Dieser ist eine ausgezeichnete JavaScript-Klasse und kapselt die Logik zum Bezug von Höhendaten. Der Adapter gehört streng genommen zum JavaScript-Stack, ist aber aufgrund seiner Funktion gesondert aufgeführt. Pro Datenlieferant muss eine eigene Adapter-Klasse implementiert werden. Falls die Höhendaten von einer anderen Quelle bezogen werden sollen, muss lediglich die Adapter-Implementation ausgetauscht werden.

Gegen aussen müssen sämtliche Adapter-Klassen eine einheitliche Schnittstelle aufweisen. Die interne Programmlogik ist pro Adapter-Klasse bzw. Datenlieferant jedoch individuell. Da in JavaScript keine Interfaces existieren, ist die Definition der Schnittstelle für einen Adapter rein konzeptionell. Daher muss bei der Implementierung der Adapter-Klasse darauf geachtet werden, dass die Funktionen der Schnittstelle mit der korrekten Signatur implementiert werden.

Dieses Vorgehen entspricht dem Strategy-Pattern[18]. Damit wird erreicht, dass der Datenlieferant ohne grossen Wartungsaufwand gewechselt werden kann. Somit wird das nicht-funktionale Requirement REQ-NF-2 (Erweiterbarkeit) erfüllt.

[17] Pro Klasse existiert ein File
[18] Entwurfsmuster in der Softwareentwicklung, welches eine Familie austauschbarer Algorithmen definiert

7 Risiken

Im folgenden Abschnitt sind die Risiken aufgeführt, welche bei Projektbeginn erkannt wurden. Die Beschreibung umfasst die Bewertung und den Umgang mit dem entsprechenden Risiko.

7.1 Risiko 1: Abhängigkeit von Google und Limitierung der Nutzung

Zu Projektbeginn war Google sowohl Lieferant der Kartendaten als auch ausschliesslicher Lieferant der Höhendaten. Es bestand ein gewisses „Klumpenrisiko“ aufgrund der starken Abhängigkeit von Google.

Google überwacht die Nutzung der Elevations API mit einem App-Key, welcher bei jeder Anfrage mitgegeben werden muss. Das Risiko bestand darin, dass Google meine App (bzw. meinen App-Key) bei Überschreitung des kostenlosen Nutzungskontingentes jederzeit sperren konnte und ich keine Höhendaten mehr beziehen konnte. Somit wäre auch die Berechnung eines Höhenprofils unmöglich geworden.

7.1.1 Risikobewertung

Die Limitierung der *Elevation* API durch Google ist zwar ärgerlich, eine Sperrung war aber eher unwahrscheinlich, da ich meine Web-App ausschliesslich in privatem Rahmen verwendete. Ausserdem sperrt Google einen App-Key gemäss AGB[19] erst nach Überschreiten des Nutzungslimits an 90 Tagen in Folge und nur nach vorgängiger Kontaktaufnahme mit dem Besitzer. Falls mein App-Key dennoch gesperrt worden wäre, hätte ich verschiedene Möglichkeiten gehabt:

1. Erstellen eines neuen App-Keys (dies ist kostenlos möglich)
2. Bezahlung der Nutzung, welche über das kostenlose Kontingent hinausgeht
3. Anbindung eines alternativer Datenlieferanten

Insbesondere den letzten Punkt habe ich bei der Wahl der Architektur berücksichtigt, indem alternative Datenlieferanten per Adapter einfach in *Maps3D* eingebunden werden können. Alles in allem war die Limitierung der *Google Elevation* API zwar mit Zusatzaufwand verbunden, stellte aber kein unüberwindbares Hindernis dar, welches das Projekt gefährdete.

7.2 Risiko 2: Innovationsgrad

Da ich zu Projektbeginn noch wenig Erfahrung mit Web-Technologien hatte, bestand das Risiko, dass der Aufwand unrealistisch geschätzt wurde oder die Einarbeitungszeit zu hoch ist. Zudem bestand bei der Umsetzung der 3D-Darstellung Unklarheit bezüglich der zu verwendenden Technologie (s. Kapitel *Umsetzung*): Einerseits konnte ich auf eine mir vertraute Technologie setzen, welche aber zu einer

[19] vgl. https://developers.google.com/maps/documentation/javascript/usage

umständlichen Architektur geführt hätte oder ich konnte mich für eine besser geeignete Technologie entscheiden, welche aber noch relativ jung ist und dementsprechend wenig dokumentiert ist.

7.2.1 Risikobewertung

Das Risiko bestand vor allem darin, dass sich der Aufwand aufgrund der fehlenden Erfahrung schlecht abschätzen liess, so dass das Projekt möglicherweise nicht innerhalb der vorgesehenen Zeit fertiggestellt werden konnte. Da ich mich persönlich aber im Bereich Webtechnologien weiterentwickeln wollte, nahm ich dieses Risiko in Kauf. Ich versuchte, ihm mit einer sorgfältigen Planung und frühzeitigen Informationsbeschaffung zu begegnen. Ausserdem hielt ich mir die Möglichkeit offen, beim Design noch Abstriche am Projektumfang zu machen.

Teil 3: Realisierung

8 Umsetzung

In diesem Kapitel wird beschrieben, wie *Maps3D* realisiert wurde. Der Fokus liegt auf der technischen Umsetzung und umfasst sowohl die Wahl der Technologien, als auch die innere Struktur von *Maps3D*. Zum Schluss des Kapitels wird exemplarisch auf zwei aufgetretene Probleme eingegangen.

8.1 Wahl der Technologie zur 3D-Modellierung

Zu Beginn der Implementierungsphase stand lediglich fest, dass *Maps3D* als Web-Applikation realisiert würde (vergleiche Kapitel *Wahl der Technologie*). Noch unklar war, wie das Höhenprofil visualisiert wurde. Zu diesem Zweck verglich ich zwei Technologien miteinander: SVG und WebGL. Der Vergleich erfolgte anhand der folgenden Kriterien:

Kriterium	Beschreibung	Bedeutung für Maps3D
Tauglichkeit	Wie einfach lassen sich Objekte im dreidimensionalen Raum mit der Technologie darstellen?	Hoch, da diese Funktionalität von zentraler Bedeutung für den Hauptanwendungsfall von *Maps3D* ist.
Integrierbarkeit	Wie einfach lässt sich die Technologie in die Web-Applikation integrieren?	Hoch, da aus einer schlechten Integrierbarkeit nebst zusätzlichem Integrationsaufwand auch Architekturänderungen oder ein schlechtes Benutzererlebnis folgen
Manipulierbarkeit	Wie einfach können Änderungen der Ansicht (Rotation, Zoom) im Modell umgesetzt werden?	Mittel, da die Daten nicht kontinuierlich ändern, sondern lediglich anders dargestellt werden.
Performance	Wie gut skaliert die Technologie bei Höhenprofilen mit vielen Punkten?	Hoch, da auch ein Höhenprofil mit wenigen Polygonen viele Punkte enthält.
Dokumentation/ persönliches Vorwissen	Wie viel Vorwissen ist zur Technologie bereits vorhanden? Kann fehlendes Wissen einfach angeeignet werden? Wie ist die Technologie dokumentiert?	Niedrig, da kein direkter Bezug zu den Anforderungen. Der Punkt wurde jedoch als Risiko erfasst.
Unterstützung (REQ-NF-3)	Wie gut wird die Technologie von aktuellen und älteren Browsern unterstützt?	Niedrig, da alle Browser in der aktuellsten Version beide Technologien unterstützen.

Tabelle 36: Kriterien zur Beurteilung der Technologie zur Darstellung des Höhenprofils

8.1.1 SVG

SVG ist die Abkürzung für „Scalable Vector Graphics“ und ein weit verbreitetes Format für die Darstellung vektorbasierter Grafiken im Web. Der Hauptvorteil von Vektorgrafiken gegenüber Rastergrafiken ist die bessere Skalierbarkeit. Im Gegensatz zu rasterbasierten Formaten wie GIF, JPG oder PNG entsteht beim Skalieren oder Zoomen von SVG-Grafiken kein Qualitätsverlust. Ausserdem entspricht jedes Element einer SVG-Grafik einer XML-Node und kann somit wie ein normales DOM-Element selektiert und manipuliert werden. Beides sind wichtige Voraussetzungen für die Animation des Höhenprofils (Rotation und Zoom).

SVG-Grafiken werden in XML definiert, wie die folgende Grafik aufzeigt.

```
<svg height="210" width="500">
      <polygon points="100,10 40,180 190,60 10,60 160,180"
      style="fill:lime;stroke:purple;stroke-width:5;fill-rule:nonzero;"/>
</svg>
```

Abbildung 12[20]: Beispiel für eine SVG-Grafik mit zugehöriger Repräsentation in XML

Der Vorteil bei dieser Technologie ist, dass SVG auf praktisch allen Browsern ohne Plugins unterstützt wird, eine Grundvoraussetzung gemäss REQ-NF-3. Ausserdem existiert die Technologie zur Darstellung von SVG schon sehr lange[21] und ist entsprechend gut im Internet dokumentiert.

Der Nachteil bei der Verwendung von SVG liegt bei der schlechte Abschätzbarkeit der Performance. SVG wurde ursprünglich zur Darstellung zweidimensionaler Grafiken spezifiziert, kann aber über entsprechende JavaScript-Libraires auch zur Darstellung dreidimensionaler Inhalte verwendet werden. Im Web existieren einige Beispiele für animierte 3D-Vektorgrafiken[22]. Diese sind aber meist als *„Proof of Concept“* (POC) gedacht und animieren deshalb meist sehr primitive Grundformen. Da für *Maps3D* aber auch bei geringer Auflösung des Höhenprofils eine beträchtliche Anzahl Polygone zu erwarten ist, bleibt die Tauglichkeit aus Performance-Sicht unsicher.

[20] Quelle: http://www.w3schools.com/svg/tryit.asp?filename=trysvg_polygon4
[21] Die Spezifikation von SVG wurde erstmals 2001 veröffentlicht. Aktuelle Spezifikation siehe http://www.w3.org/TR/2003/REC-SVG11-20030114/
[22] Vgl. http://www.petercollingridge.co.uk/blog/3d-animation-svg oder http://debeissat.nicolas.free.fr/svg3d.php

8.1.2 WebGL

Der Begriff *WebGL* steht für *Web Graphics Library* und ist ein Bestandteil von modernen Webbrowsern, mit dessen Hilfe hardwarebeschleunigte 3D-Grafiken ohne zusätzliche Erweiterungen dargestellt werden können.

Der Vorteil bei der Verwendung von WebGL zur Darstellung des Höhenprofils ist die sehr gute Performance, da die Berechnungen direkt durch die Grafikkarte durchgeführt werden. Es existieren WebGL-Anwendungen, deren Komplexität weit über diejenige von *Maps3D* hinausgeht und die trotzdem flüssig dargestellt werden[23]. Da WebGL ausserdem explizit für die Darstellung dreidimensionaler Inhalte entwickelt wurde, eignet sie sich hervorragend für die Darstellung eines Höhenprofils wie es von *Maps3D* vorgesehen ist.

Nachteilig wirkt sich aus, dass WebGL eine noch relativ junge Technologie ist[24] und nicht auf allen Browserversionen unterstützt wird. Firefox, Chrome, Safari und Opera unterstützen WebGL zwar schon seit längerem[25], Internet Explorer aber erst seit Version 11 (welches beim Verfassen dieser Arbeit die aktuellste Version ist). Da auf Desktop-Plattformen aber für jeden der fünf meistverwendeten Browsern eine Version existiert, in welche WebGL unterstützt wird, ist die Plattformunabhängigkeit trotzdem gewährleistet.

WebGL wurde auf der Basis von OpenGL entwickelt und enthält viele Konzepte, die auch in der OpenGL-Programmierung verwendet werden. So müssen beispielsweise zum Erstellen eines Würfels die acht Eckpunkte als *Vertices*[26] definiert werden. Anschliessend müssen die Vertices in einen *Array-Buffer* geladen werden und die Reihenfolge bestimmt werden, in der sie wieder ausgelesen werde. Die Reihenfolge bestimmt die Interpretation der Vertices als *Polygone*. Zum Rendern der Polygone müssen *Shader* definiert werden, welche die Oberfläche der Polygone (Farbe, Spiegelung, Transparenz, …) bestimmen. Dies können *Vertex-Shader* sein, welche die Attribute pro Vertex definieren, oder *Fragment-Shader*, welche die entsprechenden Werte zwischen den Vertices berechnen (z.B. für Schattenwurf). Die Position der Vertices kann mittels *Matrixmanipulationen* verändert werden, was die Aktualisierung der Darstellung in einem *Framebuffer* bedingt.

Wie aus oben dargestelltem Abriss der Funktionsweise von WebGL deutlich wird, ist die Programmierung mit WebGL sehr aufwändig. Glücklicherweise existiert jedoch eine Vielzahl an JavaScript-Frameworks wie *Three.js* oder *GLGE*, welche diese Vorgänge abstrahieren und den Programmieraufwand enorm reduzieren.

---

[23] vgl. http://www.zygotebody.com/, eine Webapplikation zur Darstellung der menschlichen Anatomie
[24] Version 1.0 der WebGL-Spezifikation wurde im März 2011 veröffentlicht
[25] Sie Liste der unterstützten Browser: http://caniuse.com/webgl
[26] Als Vertex wird in der Geometrie die Ecke eines Polygons bezeichnet.

8.1.3 Nutzwertanalyse

Die folgende Tabelle zeigt die Bewertung der zwei Technologien anhand der einleitend beschrieben Kriterien.

Kriterium	Gewicht	SVG		WebGL	
		Einfach	Gewichtet	Einfach	gewichtet
Tauglichkeit	2	2	4	4	8
Integrierbarkeit	1.8	5	9	3	5.4
Manipulierbarkeit	1.5	2	3	5	7.5
Performance	1.7	1	1.7	5	8.5
Dokumentation/Vorwissen	1.4	5	7	2	2.8
Total		**14**	**24.7**	**19**	**32.2**

Tabelle 37: Nutzwertanalyse für die Wahl der Technologie zur Darstellung des Höhenprofils

8.1.3.1 Fazit

Aufgrund des relativ deutlichen Ergebnisses in der Gesamtwertung werde ich WebGL als Technologie zur Darstellung des Höhenprofils verwenden. Ausschlaggebend waren vor allem die Unsicherheit bei der Performance von SVG sowie die bessere Tauglichkeit von WebGL für die Darstellung dreidimensionaler Inhalte.

8.2 Technologie-Stack

Um die Projektzeit möglichst effizient zu nutzen und mir die Arbeit zu erleichtern, habe ich folgende Libraries und Frameworks verwendet.

8.2.1 Architektur: Backbone.js/Underscore.js

Objekte in JavaScript unterscheiden sich von den Objekten kompilierter Sprache vor allem durch die Art und Weise, mit der Klassen definiert werden sowie die fehlende Type-Safety[27]. Schnittstellen von Objekten und Funktionen können auch zur Laufzeit beliebig erstellt und verändert werden, was in der Praxis schnell zu einer unübersichtlichen Sammlung von Scripts und Funktionen führt, die über die gesamte Applikation verstreut sind.

Um dies zu vermeiden und dem Programmcode eine Struktur zu geben entschied ich mich dafür, ein Framework zu verwenden. Backbone.js ist ein bekanntes JavaScript-Framework, welche die objektorientierte Implementierung von MVC-Anwendungen im Web erleichtert. Es unterstützt die Implementierung der Model-, View- und Controller-Klassen, indem es für jeden Typ Basisklassen

[27] Fehlende oder falsch implementierte Schnittstellen manifestieren sich bei JavaScript erst zur Laufzeit.

bereitstellt und damit einen Grossteil des *Boilerplate*[28]-Code eliminiert. Ausserdem ist Backbone einzig von Underscore.js abhängig, einer JavaScript-Hilfsbibliothek für oft verwendete Aufgaben, was den Einarbeitnungsaufwand reduziert.

8.2.2 Dependency-Management mit Require.js

Die objektorientierte Programmierung mit JavaScript ist im Vergleich zu kompilierten Sprachen relativ aufwändig. JavaScript enthält von Haus aus kein Dependency-Management[29]. In kompilierten Sprachen löst der Compiler die Abhängigkeiten zwischen den Klassen auf. Abhängige Klassen können danach meist automatisch durch Import-Statements deklariert werden. Das Dependency-Management in einer Scriptsprache wie JavaScript kann dagegen sehr mühsam sein, da die Abhängigkeiten zwar implizit gegeben sind, aber nicht automatisch aufgelöst werden. Es muss darauf geachtet werden, dass die entsprechenden Files vollständig und in der richtigen Reihenfolge geladen werden. Dies kann bei vielen Klassen sehr umständlich werden, da bei der Instanziierung einer Klasse sichergestellt sein muss, dass die Files abhängiger Klassen bereits geladen worden sind.

Aus diesem Grund entschied ich mich dafür, Require.js zu nutzen. Require.js ist eine Hilfsbibliothek, mit welcher die Abhängigkeiten in einer separaten Datei deklariert werden. Die benötigten Klassen (resp. Files) werden danach zur Laufzeit und je nach Bedarf automatisch (nach-)geladen.

8.2.3 WebGL: Three.js

Da die Programmierung von WebGL-Applikationen ohne Hilfsmittel aber sehr zeitaufwändig ist, habe ich mich auch bei der Darstellung von 3D-Inhalten für ein Framework entschieden. Als Quasi-Standard im Bereich WebGL wird Three.js gerne mit jQuery verglichen[30], wobei Three.js für WebGL das ist, was jQuery für JavaScript ist: Eine Abstraktion der low-level Funktionalitäten zum Erstellen und Manipulieren visueller Elemente.

Wie bei der Verwendung von jQuery ist es auch bei der Verwendung von Three.js wichtig, die darunterliegenden Konzepte zu kennen. Bei WebGL sind dies aus der OpenGL-Technologie übernommene Techniken wie Pixel- oder Fragment-Shader und Buffers sowie mathematische Grundlagen zur Matrixmanipulation (v.a. Normalenberechnung, Einheitsmatrix und Matrixmultiplikation).

[28] Code oder Codefragmente, die in ähnlicher Form immer wieder repetitiv geschrieben werden müssen, um eine gewünschte Funktionalität zu erhalten und somit die Entwicklungszeit verlängert
[29] Abhängigkeiten von einem Objekt zu einem anderen sind implizit gegeben, die entsprechenden Files müssen aber bei Auflösen der Abhängigkeit bereits durch den Browser geladen worden sein, was manuell sichergestellt werden muss.
[30] Vgl. *Beginning WebGL for HTML5* Kapitel 7 und *WebGL Beginner's Guide* Kapitel 1

Three.js abstrahiert einen Grossteil der Komplexität, indem es einige Grundformen (sog. *Meshes*) zu Verfügung stellt und oft benötigte Dinge wie Shader, Beleuchtung und Matrix-Manipulationen in einer JavaScript-API zu Verfügung stellt.

8.2.4 UI-Design: jQuery, jQuery UI, Twitter Bootstrap

Um die Manipulation von DOM-Elementen möglichst einfach zu realisieren, habe ich *jQuery* inklusive der Erweiterung *jQuery UI* verwendet. Einzelne UI-Elemente wurden ausserdem mithilfe des Open-Source Framework *Twitter Bootstrap* realisiert.

8.2.5 Unit-Testing: QUnit und Sinon.js

Um die Funktionsfähigkeit der einzelnen Klassen sicherzustellen, habe ich *QUnit* als Testframework verwendet. Abhängigkeiten zu externen Klassen der Google-Maps API habe ich mittels Mocks aufgelöst. Dazu habe ich *Sinon.JS* als Mock-Framework verwendete.

8.2.6 Version Control: Git/GitHub

Der Code ist mithilfe von Git versioniert und auf einem GitHub-Repository öffentlich zugänglich.

8.3 Klassendiagramm

Die folgende Abbildung zeigt die wichtigsten Klassen in der Übersicht. Der Einfachheit halber werden nur die für das Verständnis der Funktionsweise relevanten Attribute und Funktionen angezeigt. Hilfsfunktionen sowie vom der Framework-Klasse geerbte Members wurden bewusst weggelassen.

8.3.1 Model-Klassen

Die Model-Klassen sind (mit einer Ausnahme) alle von der Backbone-Model-Klasse abgeleitet und lassen sich somit sehr einfach ins Backbone-Framework integrieren.

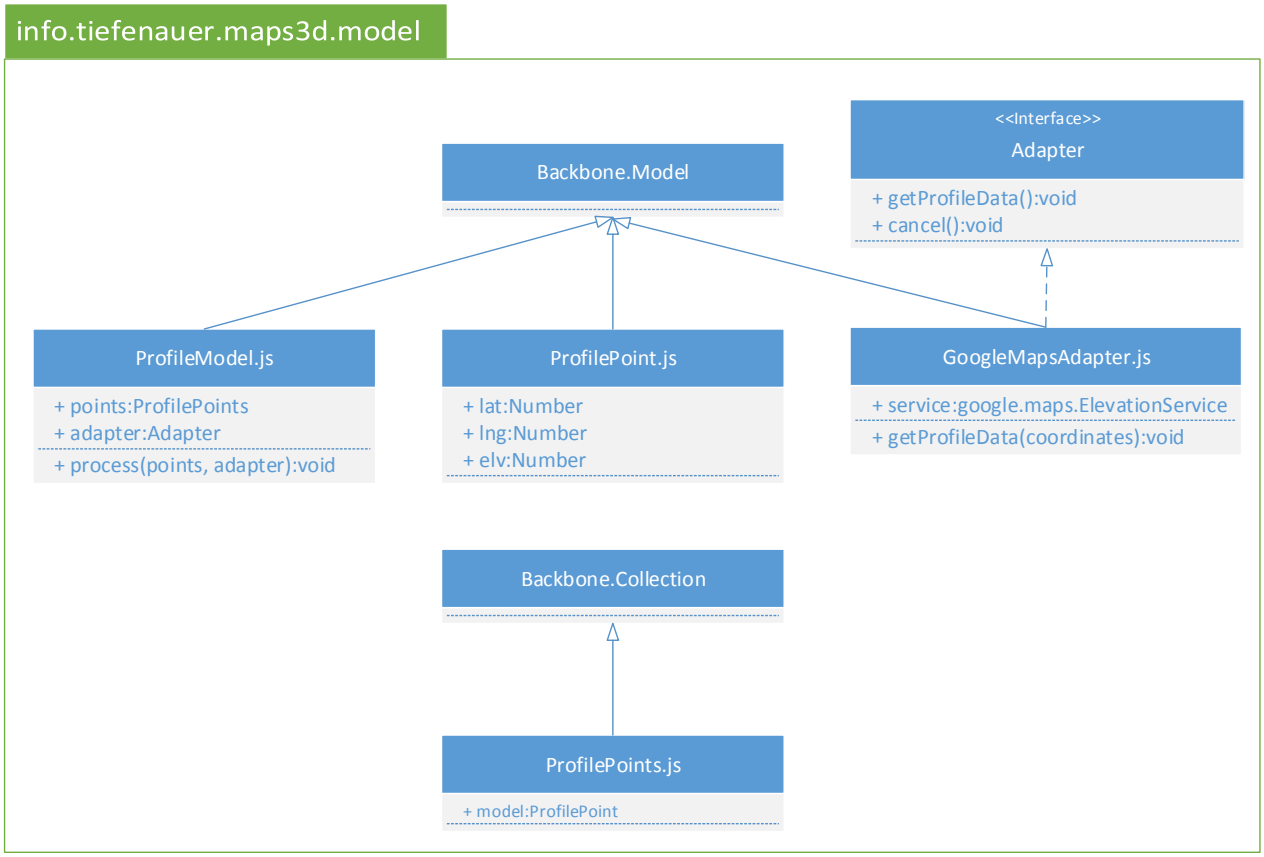

Abbildung 13: Klassendiagramm der Model-Klassen

Die Model-Klasse kommunizieren ihre Statusänderungen gegenüber anderen Framework-Klassen mithilfe von Events. Andere Klassen können auf diese Events reagieren, indem sie diese abhören und entsprechende Event-Handler registrieren.

8.3.1.1 ProfileModel.js

Die *ProfileModel*-Klasse repräsentiert das Modell des Höhenprofils und enthält alle für dessen Darstellung notwendigen Informationen. Sie kapselt intern die Logik zur Ermittlung der Höhendaten sowie deren Zusammenführung mit den dazugehörigen Koordinaten.

Die *ProfileModel*-Klasse benachrichtigt andere Klassen mittels Events über Statusangaben während dem Ermitteln der Höhendaten.

	Name	Beschreibung
Attribute	`points`	Punkte des Höhenprofils inkl. Koordinaten und Höheninformationen. Falls die Höheninformationen noch nicht ermittelt wurden, ist die Höhe für jeden Punkt 0.
	`adapter`	Adapter, der zur Ermittlung der Höhendaten verwendet wird.
Funktionen	`process(points,adapter)`	Funktion zum Auslösen der Ermittlung der Höhendaten durch einen Adapter. Der Methode können zwei optionale Parameter mitgegeben werden: - `points`: Koordinaten, für die die Höhe ermittelt werden soll. Falls *null*, werden die beim Instanziieren verwendeten Punkte genommen - `adapter`: Adapter, der zur Ermittlung der Höhendaten für jede Koordinate verwendet werden soll. Falls *null*, wird der *GoogleMapsAdapter* verwendet.
Events	`processing:start`	Event, der beim Start der Ermittlung der Höhendaten einmal ausgelöst wird.
	`processing:progress`	Event, der während der Ermittlung der Höhendaten mehrfach ausgelöst wird und Informationen zum Fortschritt enthält
	`processing:end`	Event, der einmal ausgelöst wird, nachdem die Höhendaten ermittelt wurden.

Tabelle 38: Members der ProfileModel-Klassen

8.3.1.2 ProfilePoint.js

Die *ProfilePoint*-Klasse repräsentiert einen einzelnen Punkt im Höhenmodell und enthält alle zu dessen Darstellung benötigten Informationen. Die *ProfilePoint*-Klasse ist eine reine Datenklasse und enthält keine Applikationslogik. Sie ist deshalb sehr einfach und erlaubt die Repräsentation eines Vertex als so genanntes Value Object[31].

	Name	Beschreibung
Attribute	`lat`	Breitengrad (*Latitude*)
	`lng`	Längengrad (*Longitude*)
	`elv`	Höhe über Meer in Metern (*Elevation*)

Tabelle 39: Members der ProfilePoint-Klasse

8.3.1.3 Adapter-Interface

Eine Adapter-Klasse repräsentiert die Anbindung an eine Quelle von Höhendaten und muss Funktionen mit bestimmten Signaturen implementieren. Der Status der Verarbeitung muss dem Aufrufer über bestimmte Events mitgeteilt werden.

	Name	Beschreibung
Funktionen	`getProfileData(points)`	Funktion zum Auslösen des Bezugs der Höhendaten. Der Methode muss ein Parameter mitgegeben werden: - `points`: Koordinaten, für die die Höhe ermittelt werden soll.
	`cancel()`	Abbruch des Bezugs der Höhendaten
Events	`adapter:start`	Event, der beim Start des Bezugs der Höhendaten einmal ausgelöst wird.
	`adapter:progress`	Event, der während des Bezugs der Höhendaten mehrfach ausgelöst wird und Informationen zum Fortschritt enthält
	`adapter:end`	Event, der nach dem Bezug der Höhendaten einmal ausgelöst wird.

Tabelle 40: Members des Adapter-Interfaces

[31] ValueObjects sind kleine Objekte ohne eigene Identität zur Repräsentation einer einfachen Entität

8.3.1.4 GoogleMapsAdapter.js

Die *GoogleMapsAdapter*-Klasse implementiert das Adapter-Interface und ist gleichzeitig Standard-Adapter für den Bezug von Höhendaten. Wie der Name sagt bezieht er Höhendaten von Google über die *Google Maps* API. Er kapselt intern die Logik zum Instanziieren und Aufrufen entsprechender Services inklusive Error-Handling.

	Name	Beschreibung
Attribute	`service`	Service der *Google Elevations* API, welcher zum Bezug von Höhendaten verwendet wird
Funktionen	`getProfileData(points)`	Implementierung der Interface-Funktion
Events	`adapter:start`	Event, der beim Start des Bezugs der Höhendaten einmal ausgelöst wird.
	`adapter:progress`	Event, der während des Bezugs der Höhendaten mehrfach ausgelöst wird und Informationen zum Fortschritt enthält
	`adapter:end`	Event, der beim Beenden des Bezugs der Höhendaten einmal ausgelöst wird.

Tabelle 41: Members der GoogleMapsAdapter-Klasse

8.3.1.5 ProfilePoints.js

Die *ProfilePoints*-Klasse ist eine Subklasse der Backbone-*Collection*-Klasse und stellt eine Liste von *ProfilePoint*-Objekten dar. Als reine Aggregator-Klasse erbt sie sämtliche von der Backbone-*Collection*-Klasse zu Verfügung gestellten Funktionen zur Manipulation der Liste und enthält keine weiteren Funktionen. Sie existiert einzig aus dem Grund, um von den von Backbone (bzw. Underscore) bereitgestellten Funktionalitäten[32] zu profitieren.

[32] Underscore.js ist die einzige Abhängigkeit von Backbone.js und stellt sehr viele nützliche Funktionen zur Manipulation von Listen zu Verfügung, wie zum Beispiel die pluck()-Funktion, welche von jedem Listenelement jeweils ein bestimmtes Attribut ermittelt. Dies ist eine Funktion, welche sehr nützlich bei der Verarbeitung der Ermittelten Koordinaten und Höhendaten ist, indem sehr einfach Maximum und Minimum von Längengrad, Breitengrad und Elevation ermittelt werden können, eine Information die zum Berechnen der Länge, Breite und Höhe des Höhenprofils benötigt wird.

8.3.2 View-Klassen

Eine View-Klasse steht für einen sichtbaren Teilbereich des GUI und repräsentiert die Schnittstelle zwischen sichtbarem GUI und Applikationslogik. Sie übernimmt meist die Versorgung eines DOM-Elements mit aktuellen (Model-)Daten, die durch das DOM-Element dargestellt werden. Ausserdem sind View-Objekte für die Umformung von Benutzeraktionen in Framework-Events zuständig. Sämtliche View-Klassen erben von der Backbone-*View*-Klasse und lassen sich somit problemlos ins Framework integrieren.

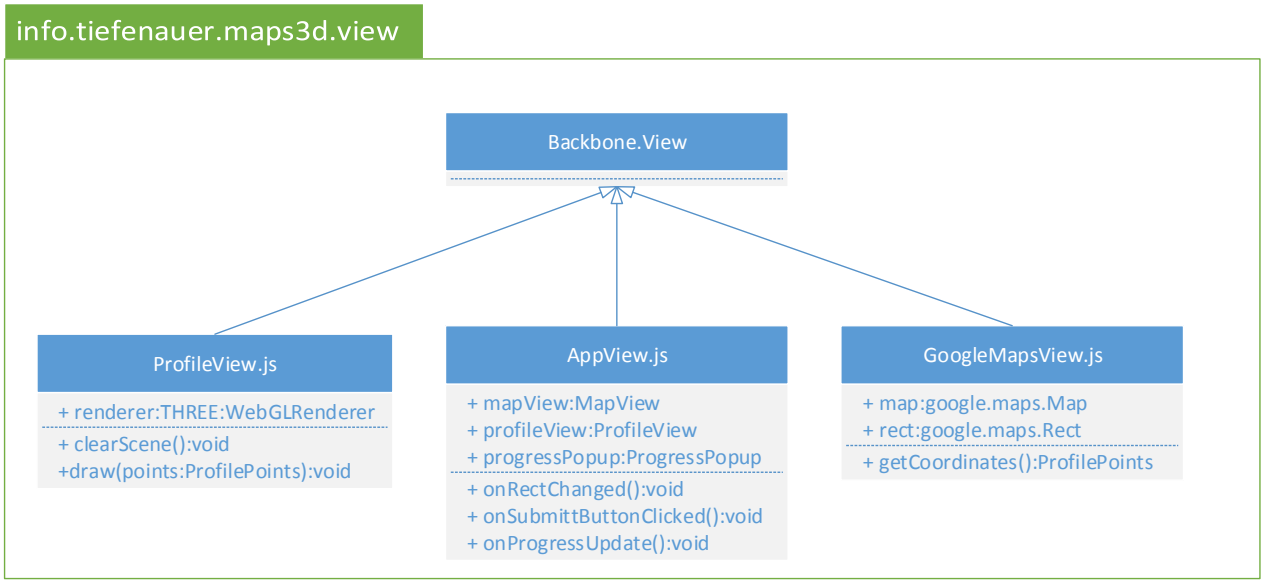

Abbildung 14: Klassendiagramm der View-Klassen

8.3.2.1 AppView.js

Die *AppView*-Klasse repräsentiert die App als Ganzes, verwaltet jedoch keine eigenen DOM-Elemente. Diese werden durch die *ProfileView* bzw. durch die *GoogleMapsView* verwaltet. Trotzdem ist die *AppView* von zentraler Bedeutung, da sie die von den View-Klassen oder DOM-Elementen ausgelösten Events mit den entsprechenden Callback-Funktionen anderer View-Klassen verbindet. Somit realisiert die *AppView*-Klasse die Benutzerinteraktion mit dem System und ermöglicht das Zusammenspiel aller GUI-Elemente.

Da die AppView rein verbindenden Charakter hat und keine eigene Logik besitzt, stellt sie nach aussen auch keine eigenen Funktionen oder Events zu Verfügung.

	Name	Beschreibung
Attribute	`mapView`	Instanz einer *GoogleMapsView*, welche die Karte repräsentiert
	`profileView`	Instanz einer *ProfileView*, welche das Höhenprofil repräsentiert
	`progressPopup`	Popup zur Darstellung eines Fortschritts

Tabelle 42: Members der AppView-Klasse

8.3.2.2 GoogleMapsView.js

Die *GoogleMapsView*-Klasse repräsentiert die Karte im GUI, auf welcher der Benutzer ein Rechteck für das Höhenprofil zeichnen kann. Sie enthält sämtliche Logik für die Rasterung des dargestellten Rechtecks. Durch die Rasterung werden die durch das Auswahlrechteck definierten Koordinaten der Punkte für das Höhenprofil ermittelt. Somit enthält die *GoogleMapsView*-Klasse die Liste der zu verarbeitenden Koordinaten.

Die *GoogleMapsView* benachrichtigt ausserdem andere Komponenten über Veränderungen des dargestellten Rechtecks mithilfe entsprechender Events.

	Name	Beschreibung
Attribute	`map`	Model des DOM-Element, in welchem die Karte dargestellt wird
	`rect`	Modell des Rechtecks, welches über der Karte angezeigt wird
Funktionen	`getCoordinates()`	Funktion ohne Parameter, welche das Rechteck rastert und die einzelnen Punkte als Koordinaten zurückgibt.
Events	`rect:bounds:changing`	Event, der mehrfach ausgelöst wird, wenn sich die Grösse oder Position des Rechtecks ändert
	`rect:bounds:changed`	Event, der einmal ausgelöst wird, nachdem sich die Grösse oder Position des Rechtecks ändert

Tabelle 43: Members der GoogleMapsView-Klasse

8.3.2.3 ProfileView.js

Die *ProfileView*-Klasse repräsentiert das Höhenprofil, welches die verarbeiteten Höhendaten dreidimensional darstellt und vom Benutzer gedreht und gezoomt werden kann. Sie enthält sämtliche Logik zur Darstellung dieser Daten als 3D-Modell mithilfe des *Three.js*-Frameworks.

Da sie als passiver Akteur lediglich Daten entgegennimmt und keine Benutzerinteraktionen ermöglicht, welche Auswirkungen auf andere GUI-Elemente hätten, gehen von ihr keine Events aus.

	Name	Beschreibung
Attribute	`renderer`	Instanz der *WebGLRenderer*-Klasse (Three.js-Klasse), welche das Rendering der Three.js-Meshes übernimmt
Funktionen	`draw(points)`	Erstellt mit den in **`points`** mitgegebenen Punkten das Höhenprofil. • points: *ProfilePoints*-Objekt, welches die Koordinaten inkl. Höhendaten für das Höhenprofil enthält

Tabelle 44: Members der ProfileView-Klasse

8.3.3 Controller-Klassen

Da Backbone kein klassisches MCV-Framework ist (es wird oft als MV*-Framework bezeichnet[33]), existiert keine explizite Controller-Basisklasse, von welcher geerbt werden könnte. Stattdessen wird der Controller implizit über einen Event-Bus realisiert. Die in klassischen Controller-Klassen enthaltene Logik ist zum Teil in den View- und zum Teil in den Model-Klassen enthalten.

[33] Ausprägungen eines MVC-Framework ohne dedizierten Controller mit zwei Hauptvertretern: „Model View Presenter“ und „Model View ViewModel“ (vgl. „Beginning Backbone.js“ Kapitel 1)

8.4 Unit Testing

Um die Funktionalität der Model-Klassen sicherzustellen, wurden Unit Tests erstellt. Pro Model-Klasse existiert eine Testklasse, welche die verschiedenen Funktionen der Model-Klasse aufruft und den Rückgabewert oder die ausgelösten Events überprüft. Um die unter Test stehenden Klassen möglichst isoliert zu testen, wurden Abhängigkeiten zu anderen Klassen mit Mocks aufgelöst. Auf diese Weise können Fehler minimiert werden, deren Ursache in fehlerhaft implementierten abhängigen Klassen liegt. Ausserdem erlauben die Mocks die Simulation von fremden Klassen, auf deren Quellcode ich keinen Zugriff habe. So wurde beispielsweise in der Testklasse für die *GoogleMapsAdapter*-Klasse der *ElevationService* durch einen Mock ersetzt.

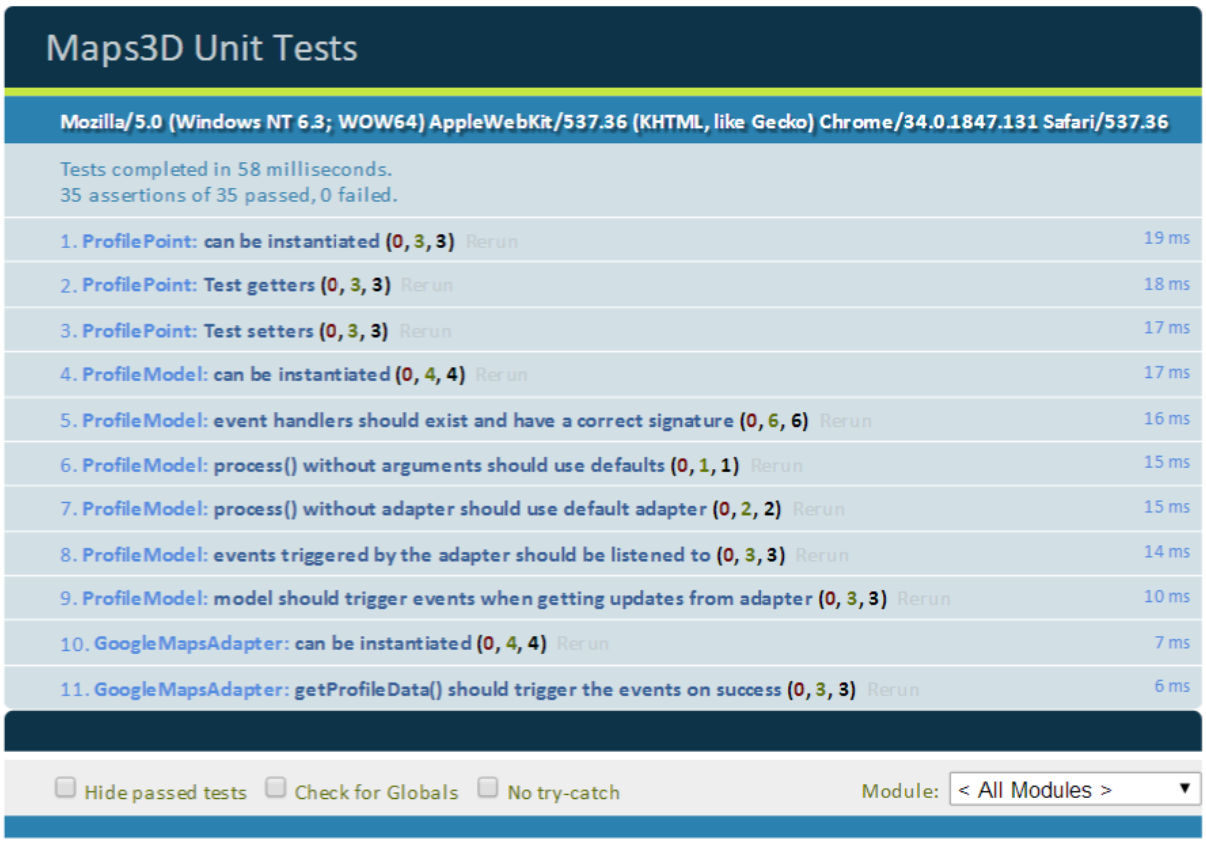

Abbildung 15: Screenshot des Resultats der Unit Tests

8.5 Ablauf bei Ausführung des Haupt-Use Cases

Das folgende Diagramm zeigt den Ablauf und die Interaktion der beteiligten Klassen, wenn der Benutzer das Höhenprofil für ein definiertes Rechteck berechnen lässt. Die einzelnen Schritte sind mit Nummern markiert und werden in der Legende darunter erklärt.

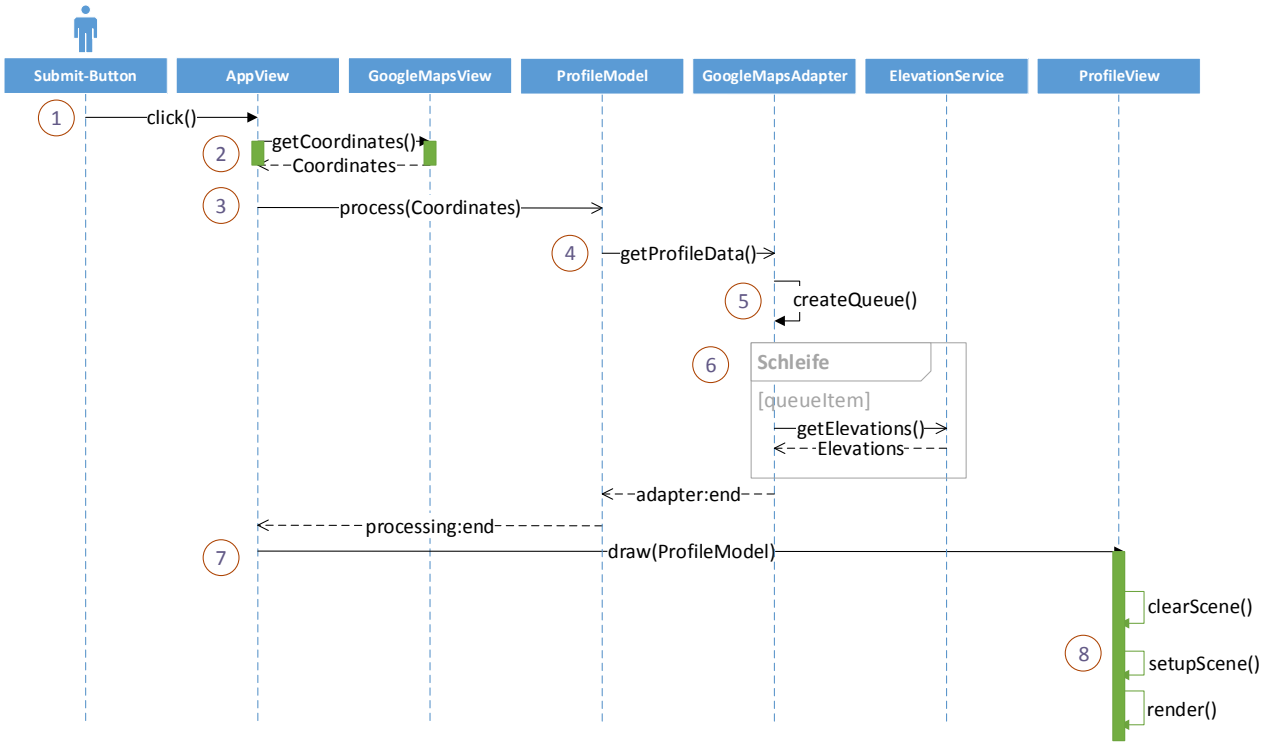

Abbildung 16: Sequenzdiagramm für die Berechnung eines Höhenprofils

1. Beim Klicken des Buttons zum Berechnen des Höhenprofils wird in der *AppView* der entsprechende Event-Handler aufgerufen
2. Die *AppView* ermittelt die Liste der zu verarbeitenden Koordinaten aus der *GoogleMapsView*. Dazu wird das definierte Rechteck gerastert.
3. Die *AppView* ruft auf dem *ProfileModel* die Funktion auf, um für die jede Koordinate die zugehörige Höhe in Meter über Meer zu ermitteln. Der Aufruf erfolgt asynchron.
4. Das *ProfileModel* verwendet den ihm übergebenen Adapter (hier *GoogleMapsAdapter*), um für die einzelnen Koordinaten die Höhe zu ermitteln.
5. Der *GoogleMapsAdapter* unterteilt die Liste von Koordinaten in Teile gleicher Länge. Dies aufgrund der Beschränkung der *GoogleMaps*-API[34].
6. Für jedes Teilstück wird eine Anfrage über den *google.maps.ElevationService* gemacht, indem die Funktion *getElevationsForLocations()* aufgerufen wird.
7. Nachdem für alle Koordinaten die Höhe ermittelt wurde, werden die Punkte des *ProfileModels* der *ProfileView* übergeben, welche das Höhenprofil zeichnet.
8. Um das Höhenprofil in der Canvas zu zeichnen werden zuerst allenfalls vorhandene Elemente entfernt, das Höhenprofil vorbereitet und dann gerendert.

[34] vgl. *Problem 2: Limitierung der GoogleMaps API* im folgenden Abschnitt

8.6 Aufgetretene Probleme

Im folgenden Abschnitt gehe ich auf zwei Problem ein, die bei der Implementierung aufgetreten sind und erkläre, wie diese gelöst wurden.

8.6.1 Problem 1: Rasterung des Rechtecks

Google Maps stellt Koordinaten mithilfe von *Latitude* (Breitengrad) und *Longitude* (Längengrad) dar. Diese Werte werden üblicherweise in Grad, Minuten und Sekunden unterteilt. Untenstehende Grafik visualisiert die Repräsentation von Längen- und Breitengrad in einem *geographischen* Koordinatensystem.

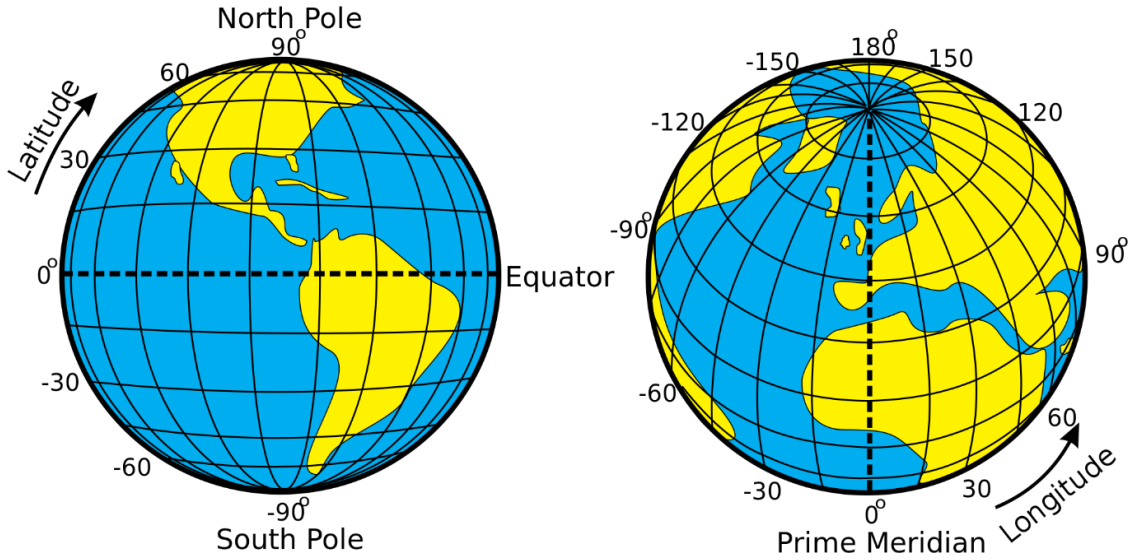

Abbildung 17[35]: Latitude (Breitengrad) und Longitude (Längengrad)

Statt mit Minuten und Sekunden stellt Google Winkelmasse jedoch mit Dezimalbrüchen dar. Zudem wird die Höhe über Meer in Metern angegeben. Diese Darstellung wird konsistent über alle Services der GoogleMaps-API verwendet. So werden beispielsweise die Koordinaten des ZHAW-Gebäudes am Standort Zürich wie folgt ausgewiesen:

Adresse	**Koordinaten/Höhe**		**Darstellung in Google Maps**

Lagerstrasse 41, 8004 Zürich	Breitengrad:
	Längengrad:
	Höhe:
		47° 22' 44.295"
		8° 31' 50.7138"
		408.283 M.ü.M.
			<pre>{
			"location":{
			"lat":"47.378971",
			"lng":"8.530754"
			},
			"elevation":“408.283“
			}</pre>

Tabelle 45: Beispiel für die Darstellung von Geodaten durch Google

[35] Quelle: http://en.wikipedia.org/wiki/File:Latitude_and_Longitude_of_the_Earth.svg

Zur Darstellung des Höhenprofils wird das durch den Benutzer festgelegte Auswahlrechteck gerastert, d.h. in rechteckige Bereiche gleicher Grösse unterteilt. Da Three.js aber intern ein *kartesisches* Koordinatensystem verwendet, können die von Google übermittelten Daten nicht einfach übernommen werden. Die durch das gerasterte Auswahlrechteck repräsentierten Geodaten müssen zuerst für die Verwendung in einem kartesischen Koordinatensystem umgerechnet werden.

Folgende Abbildung illustriert den Sachverhalt:

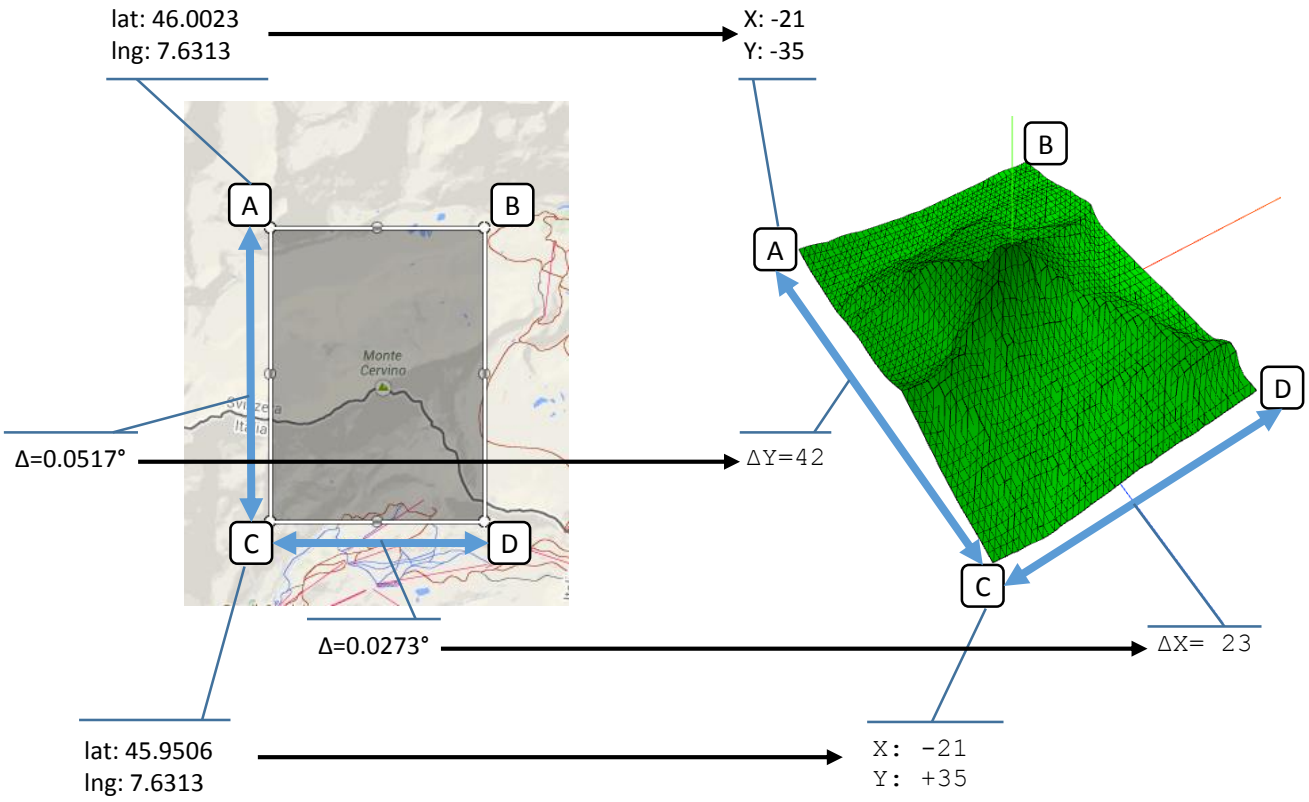

Abbildung 18: Darstellung der Fläche (links) und des zugehörigen Profils (rechts)

Für die Umrechnung sind folgende Schritte notwendig:

1. Ermitteln von Breite und Höhe des Rechtecks in Metern
2. Normalisieren der Höhen
3. Umrechnen der Koordinaten und Höhen in Vertices

Im Folgenden ist beschrieben, wie diese Schritte umgesetzt wurden.

8.6.1.1 Ermitteln von Breite und Höhe des Rechtecks in Metern

Der Abstand zweier Punkte auf der Erdkugel kann mithilfe der so genannten *Haversine*[36]-Formel (Deutsch: *Semiversus*) berechnet werden. Dieser Formel bestimmt den Abstand *d* zweier Punkte $p_1 = (\phi_1, \lambda_1)$ und $p_2 = (\phi_2, \lambda_2)$ auf einer Kugel mit Durchmesser r wie folgt:

$$d = 2 \cdot r \cdot \sin^{-1}\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1) \cdot \cos(\phi_2) \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

Formel 1: Berechnung des Abstandes zweier Punkte auf einer Kugel mit der Haversine-Formel

Φ steht hierbei für den Längengrad und λ für den Breitengrad als Winkel in Radiant. Da Winkelmasse von Google in Grad und nicht in Radiant angegeben werden, muss das Winkelmass zuerst mithilfe der folgenden Formel umgerechnet werden.

$$toRad = f(winkel) = \frac{winkel \cdot \pi}{180}$$

Formel 2: Umrechnung des Winkelmasses von Grad in Radiant

Für Formel 2 wurde der der *Number*-Prototyp in JavaScript um eine Funktion erweitert:

```
Number.prototype.toRad = function () { return this * Math.PI / 180; }
```

Script 1: Erweiterung des Number-Prototyps um die Funktion zur Umrechnung von Grad in Radiant

Die *Haversine*-Formel wurde als Klassenfunktion in der *GoogleMapsUtil*-Hilfsklasse implementiert[37].

```
degreeToMeter: function(p1, p2)
{
    var R = 6371; // Earth radius in km
    var dLat = Number(Math.max(p1.lat, p2.lat)-Math.min(p1.lat, p2.lat)).toRad();
    var dLng = Number(Math.max(p1.lng, p2.lng)-Math.min(p1.lng, p2.lng)).toRad();
    var lat1 = Number(p1.lat).toRad();
    var lat2 = Number(p2.lat).toRad();

    var a = Math.sin(dLat/2) * Math.sin(dLat/2) +
            Math.sin(dLng/2) * Math.sin(dLng/2) * Math.cos(lat1) * Math.cos(lat2);
    var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    var d = R * c;
    return d*1000;
}
```

Script 2: Implementierung der Haversine-Formel mit JavaScript

[36] Vgl. http://en.wikipedia.org/wiki/Haversine_formula

[37] Die Berechnung der Distanz mit der *Haversine*-Formel scheint auf den ersten Blick etwas umständlich, da die Seiten des Rechtecks auch über die Verrechnung der Differenz der Längen- oder Breitengrade mit dem Erdumfang realisiert werden könnte. Dies bedingt aber, dass die Seiten parallel zu den Längen- bzw. Breitengraden verlaufen. Im Gegensatz dazu erlaubt die *Haversine*-Formel aber die Berechnung des Abstands zweier *beliebiger* Punkte, was hilfreich sein könnte, falls in Zukunft das Höhenprofil für andere Formen als Rechtecke berechnet werden soll.

8.6.1.2 Normalisierung der Höhen

Damit das Höhenprofil unabhängig von der durchschnittlichen Höhe des gewählten Kartenausschnittes immer innerhalb des sichtbaren Bereichs liegt, wird der maximale Höhenunterschied $diff$ einer Menge $elevations$ von Höhenangaben wie folgt berechnet:

$$diff = \max(elevations) - \min(elevations)$$

Formel 3: Berechnung des maximalen Höhenunterschieds aller Punkte

Die Berechnung wurde mithilfe von Underscore wie folgt in JavaScript implementiert:

```
var minElv = _.min(elevationPoints.pluck('elv'));
var maxElv = _.max(elevationPoints.pluck('elv'));
var diff = (maxElv - minElv);
```

Script 3: Berechnung der maximalen Höhendifferenz mit JavaScript

Mit dem für $diff$ berechneten Wert können anschliessend die Höhen normalisiert werden, so dass der tiefste Punkt des gewählten Kartenausschnittes im Koordinatensystem den Wert $z = 0$ hat und der höchste Punkt den Wert $z = diff$ hat. Ohne diese Normalisierung bestünde die Gefahr, dass das Höhenprofil nicht angezeigt wird, weil die z-Koordinate des tiefsten Punktes ausserhalb des sichtbaren Bereichs liegt.

8.6.1.3 Umrechnen der Koordinaten und Höhen in Vertices

Mit der in Metern berechneten Breite und Höhe kann anschliessend mit Three.js eine Fläche gezeichnet werden.

```
var plane = new THREE.Mesh(new THREE.PlaneGeometry(width, height, segments,
segments),
    new THREE.MeshLambertMaterial({
        color: 0x00ff00,
        dynamic: true
    })
);
```

Script 4: Zeichnen einer Ebene mit Three.js

Die Ebene wird durch Three.js automatisch mittig über dem Ursprung positioniert. Abbildung 19 zeigt ein Beispiel einer von Three.js gezeichneten Ebene

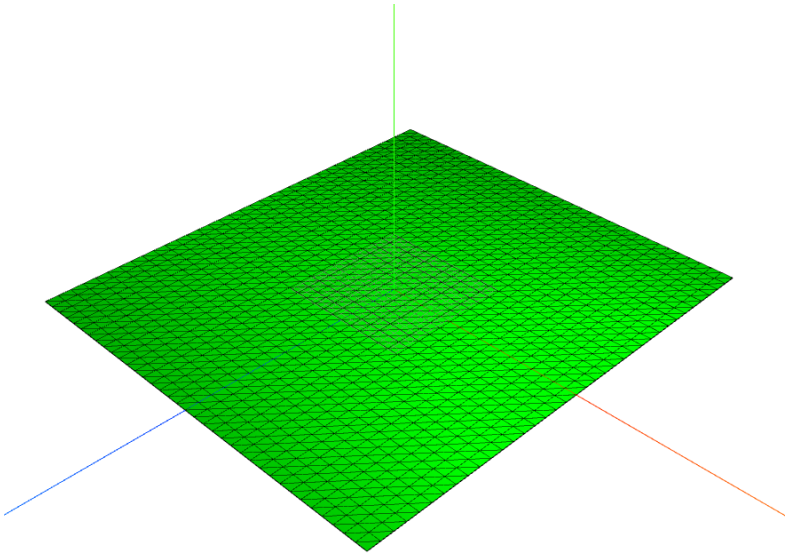

Abbildung 19: Ebene ohne Höhenangaben

Das Profil kann anschliessend erstellt werden, indem die einzelnen Punkte der Ebene um die normalisierte Höhe der zugehörigen Koordinate verschoben wird.

```
for (var i = 0; i < plane.geometry.vertices.length; i++) {
    diff = elevationPoints.models[i].get('elv') - minElv;
    plane.geometry.vertices[i].z += diff / 100;
}
```

Script 5: Setzen der Höhe

Abbildung 20 zeigt dieselbe Ebene wie Abbildung 19 um den Betrag der Höhenangabe verschobenen Punkten.

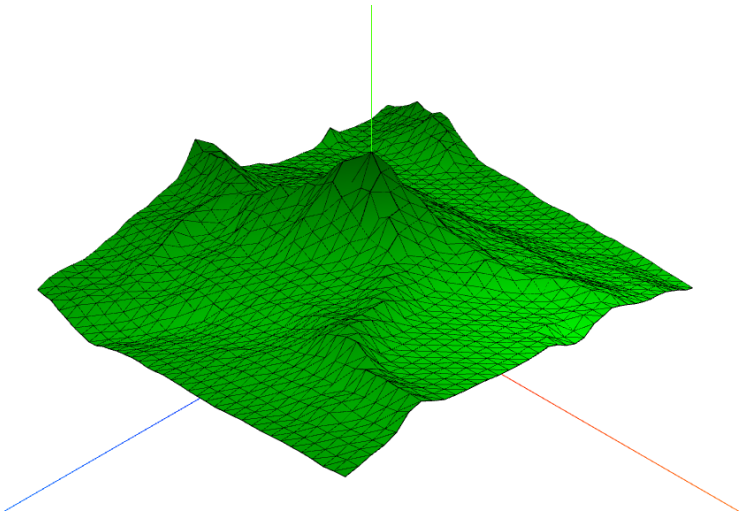

Abbildung 20: Ebene mit Höhenangaben

8.6.2 Problem 2: Limitierung der GoogleMaps API

Es hat sich herausgestellt, dass zusätzlich zur dokumentierten zeitlichen und quantitativen Limitierungen die maximale Frequenz zum Aufrufen der API bei ungefähr 10 Anfragen pro Sekunde liegt. Falls die Frequenz höher ist, wird ein Fehlercode als Antwort zurückgegeben. Aus diesem Grund musste die Liste der zu verarbeitenden Koordinaten in Stücke gleicher Länge unterteilt werden. Pro Anfrage werden jeweils die Höhendaten eines solchen Teilstücks ermittelt. Nach jeder Anfrage wird überprüft, ob von der *GoogleMaps*-API ein Fehler zurückgegeben wird. Falls dies der eingetreten ist, wird die Anfrage mit einer Sekunde Verzögerung wiederholt.

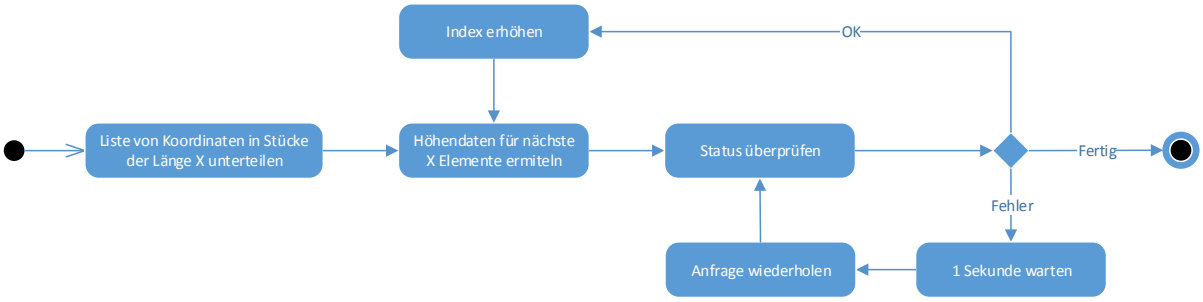

Abbildung 21: Aktivitätsdiagramm für die Ermittlung der Höhendaten

9 Fazit

Ich habe beim Umsetzen dieser Semesterarbeit sowohl technisch als auch konzeptionell enorm viel gelernt. Im technischen Bereich konnte ich meine eher rudimentären Kenntnisse im Bereich Webprogrammierung vertiefen und gleichzeitig drei neue Frameworks (Backbone.js, Three.js, Qunit/Sinon.js) kennen lernen. Im konzeptionellen Bereich kam mir vor allem der Besuch des Kurses „Requirements Engineering“ im Herbstsemester 2013 zugute. Die dort gelernten Grundlagen für das Erfassen und Dokumentieren von Anforderungen sowie die Planung der Architektur waren sehr hilfreich beim strukturieren Durchführen des Projektes.

Sämtliche im EBS erfassten Ziele wurden erreicht, die Resultate sind in der vorliegenden Arbeit oder in Form von Source-Code ersichtlich:

- ✓ Vergleich und Analyse von Unzulänglichkeiten bestehender, ähnlicher Produkte im zweidimensionalen Bereich
- ✓ Evaluieren von Möglichkeiten zur Darstellung dreidimensionaler Inhalte im Web
- ✓ Einarbeitung in die Schnittstellendefinition und Funktionsweise der *Google Maps* API und Google Elevation API
- ✓ Implementieren einer Applikation, mit welcher eine Fläche über einem Kartenausschnitt definiert werden kann.
- ✓ Anbindung der *Google Maps*/Google Elevation API an die Applikation zum Bezug von Höhendaten
- ✓ Implementieren einer Applikation, welche aus den von Google bezogenen Daten verarbeitet und in ein für die Weiterverarbeitung optimiertes Format bringt
- ✓ Dreidimensionale Darstellung der verarbeiteten Daten

Eine besondere Herausforderung war das Kennenlernen einer mir bisher unbekannten Technologie (WebGL/Three.js), welche zudem noch sehr jung und dementsprechend wenig dokumentiert ist. Ich bin aber zufrieden mit der erreichten Lösung, welche Raum für Erweiterungen lässt. Aufgrund der sauberen, modularen Architektur könnte *Maps3D* in Zukunft problemlos um folgende Features erweitert werden.

- Manuelles Einstellen der Auflösung des Höhenprofils
- Überlagern des Höhenprofils mit Texturen
- Caching von Höhendaten oder Speicherung in einer Datenbank
- Anbindung weiterer Datenlieferanten
- Entwickeln einer Mobile-Version der Web-App
- Definition der Grundfläche für andere Formen (Kreise, Dreiecke, Ortsgrenzen)

10 Glossar

	Begriff	Erklärung
A	Adapter-Pattern	Entwurfsmuster zur Übersetzung einer Schnittstelle in eine andere, welche die Kommunikation zweier inkompatiblen Klassen ermöglicht. http://en.wikipedia.org/wiki/Adapter_pattern
	API	*Application Programming Interface*: Programmierschnittstelle zu einem System
B	Boilerplate	Codefragmente, die an vielen Stellen in mehr oder weniger unveränderter Form benötigt werden.
C	Canvas	HTML-Element bzw. Bereich, in den mit JavaScript gezeichnet werden kann.
D	DOM	*Document Object Model*. Spezifikation einer Schnittstelle für den Zugriff auf HTML- oder XML-Dokumente. Webseiten bestehen im Wesentlichen aus DOM-Elementen.
E	EBS	Einschreibe- und Bewertungssystem der ZHAW
	Elevations API	API von *Google Maps* für den Bezug von Höhendaten
	Event-Handler	Funktion, welche den Programmcode zur Behandlung eines Ereignisses enthält
G	GIS	*Geographic Information System*. Informationssystem zur Erfassung, Bearbeitung Organisation Analyse und Präsentation räumlicher Daten. http://en.wikipedia.org/wiki/Geographic_information_system
	GUI	*Graphical User Interface*. Grafische Benutzerschnittstelle, welche die Kommunikation zwischen Anwender und System ermöglicht.
I	IDE	*Integrated Development Environment*. Entwicklungsumgebung/Programm zur Erstellung von Software
K	KISS	Keep It Simple and Stupid
L	Latitude	Englischer Begriff für Breitengrad
	Longitude	Englischer Begriff für Längengrad
M	MVC	Model View Controller. Muster zur Strukturierung von Software-Entwicklung in die Einheiten Datenmodell (Model), Präsentation (View) und Programmsteuerung (Controller).
	Mock (-Objekt)	Platzhalter für eine echtes Objekt, dessen Verhalten und Wert gesteuert werden kann
	(GUI-)Mockup	Attrappe, schematische Darstellung eines GUI ohne Funktionalität
N	NAS	*Network Attached Storage*
O	OpenGL	Plattform- und programmiersprachenunabhängige Programmierschnittstelle zur Entwicklung von 2D und 3D-Computergrafikanwendungen
P	POC	*Proof Of Concept:* Machbarkeitsnachweis, anhand dessen die prinzipielle Durchführbarkeit eines Projektes gezeigt wird
	Polygon	Geometrische Figur aus mindestens 3 Punkten
R	Runtime	Laufzeitumgebung, also ein Computerprogramm, welches

		einem Ausführungsprogramm die Kommunikation mit dem Betriebssystem ermöglicht.
S	SVG	*Scalable Vector Graphics:* Spezifikation von Vektorgrafiken mit XML.
	Single-Page-Application	Web-Applikation, welche nur aus einem einzigen HTML-File besteht. Die einzelnen DOM-Elemente werden durch JavaScript erzeugt und/oder manipuliert.
T	Type Safety	Ausmass, zu welcher eine Programmiersprache in der Lage ist Fehler aufgrund falscher Typen zu erkennen und verhindern (Type Errors)
V	VCS	*Version Control System:* System zur Versionsverwaltung, welche Änderungen an Dateien protokolliert.
	Vertex	Ecke eines Polygons im dreidimensionalen Raum
W	WebGL	*Web Graphics Library:* Bestandteil von modernen Webbrowsern zur Darstellung von dreidimensionalen Inhalten auf Webseiten ohne zusätzliche Erweiterungen.
	Wireframe	Gitternetzdarstellung eines dreidimensionalen Objekts, bei dem lediglich die Körperkanten und Hilfslinien gezeichnet werden

11 Links

#	Beschreibung	Link
1	**Maps3D** Lauffähige Web-Applikation	http://maps3d.tiefenauer.info/
2	**Repository** Link zu GitHub Repository mit dem Code	https://github.com/tiefenauer/maps3D
3	**Projekthomepage** Übersicht über das Projekt	http://www.tiefenauer.info/projects/maps3d/
4	**Backbone.js** Von *Maps3D* verwendetes Framework	http://backbonejs.org/
5	**Require.js** Von Für *Maps3D* verwendete JavaScript-Library für das Dependency Management	http://requirejs.org/
6	**THREE.js** Von *Maps3D* verwendete JavaScript-Library zur Darstellung von WebGL	http://threejs.org/
7	**QUnit** Von *Maps3D* verwendetes Framework für Unit Tests	https://qunitjs.com/
8	**Sinon.JS** Von *Maps3D* verwendete Library zum Erstellen von Mocks und Stubs	http://sinonjs.org/
9	**Learning WebGL** Tutorial zur Programmierung mit WebGL	http://learningwebgl.com/blog/?page_id=1217

12 Abbildungsverzeichnis

Abbildung 1: Grobe Visualisierung des Hauptanwendungsfalls der fertigen Applikation 6
Abbildung 2: 3D-Darstellung des Matterhorns mit Google Maps 13
Abbildung 3: Darstellung des Matterhorns in Google Earth (Windows) 15
Abbildung 4: Screenshot der App "3D Outdoor Guides" 17
Abbildung 5: Darstellung des Matterhorns mit HERE Maps 18
Abbildung 6: Use Case-Diagramm 26
Abbildung 7: Natürlichsprachliche Satzschablone gemäss Pohl/Rupp Kap. 5.2 30
Abbildung 8: Mockup der Benutzerschnittstelle von Maps3D 44
Abbildung 9: Systemkontext 46
Abbildung 10: Grobkonzept von Maps3D 47
Abbildung 11: Komponentendiagramm von Maps3D 48
Abbildung 12: Beispiel für eine SVG-Grafik mit zugehöriger Repräsentation in XML 54
Abbildung 13: Klassendiagramm der Model-Klassen 59
Abbildung 14: Klassendiagramm der View-Klassen 63
Abbildung 15: Screenshot des Resultats der Unit Tests 67
Abbildung 16: Sequenzdiagramm für die Berechnung eines Höhenprofils 68
Abbildung 17: Latitude (Breitengrad) und Longitude (Längengrad) 69
Abbildung 18: Darstellung der Fläche (links) und des zugehörigen Profils (rechts) 70
Abbildung 19: Ebene ohne Höhenangaben 73
Abbildung 20: Ebene mit Höhenangaben 73
Abbildung 21: Aktivitätsdiagramm für die Ermittlung der Höhendaten 74

13 Tabellenverzeichnis

Tabelle 1: Versionsprotokoll 1
Tabelle 2: Beschreibung der Projektphasen 9
Tabelle 3: Projektplan 10
Tabelle 4: Vergleich von ähnlichen Produkten mit Maps3D 20
Tabelle 5: Nutzwertanalyse für Technologieentscheid 24
Tabelle 6: Schema der Use Case Spezifikation 26
Tabelle 7: Spezifikation Use Case UC-1 27
Tabelle 8: Spezifikation Use Case UC-2 28
Tabelle 9: Spezifikation Use Case UC-3 29
Tabelle 10: Schema für die Beschreibung der Anforderungen 30
Tabelle 11: Beschreibung der Anforderung REQ-F-1: "Bereich definieren" 31
Tabelle 12: Beschreibung der Anforderung REQ-F-2 "Position des Auswahlrechtecks ändern" 31
Tabelle 13: Beschreibung der Anforderung REQ-F-3 "Grösse des Auswahlrechtecks ändern" 32
Tabelle 14: Beschreibung der Anforderung REQ-F-4: "Ort suchen" 32
Tabelle 15: Beschreibung der Anforderung REQ-F-5: "Höhenmodell berechnen und darstellen" 33
Tabelle 16: Beschreibung der Anforderung REQ-F-6: Daten beziehen 33
Tabelle 17: Beschreibung der Anforderung REQ-F-7: Höhenprofil rotieren 34
Tabelle 18: Beschreibung der Anforderung REQ-F-8: Höhenprofil zoomen 34
Tabelle 19: Beschreibung der Anforderung REQ-NF-1: Limitierung der Anfragen 35
Tabelle 20: Beschreibung der Anforderung REQ-NF-2: Erweiterbarkeit 35
Tabelle 21: Beschreibung der Anforderung REQ-NF-3: Plattformunabhängigkeit 36
Tabelle 22: Rahmenbedingungen 36
Tabelle 23: Akzeptanztest T-01 37
Tabelle 24: Akzeptanztest T-02 37
Tabelle 25: Akzeptanztest T-03 38
Tabelle 26: Akzeptanztest T-04 38
Tabelle 27: Akzeptanztest T-05 39
Tabelle 28: Akzeptanztest T-06 39
Tabelle 29: Akzeptanztest T-07 40
Tabelle 30: Akzeptanztest T-08 41
Tabelle 31: Beschreibung des Akzeptanztests T-09 41
Tabelle 32: Beschreibung des Akzeptanztests T-10 42
Tabelle 33: Beschreibung des Akzeptanztests T-11 43

Tabelle 34: Beschreibung des Akzeptanztests T-12 43
Tabelle 35: UI-Elemente 45
Tabelle 36: Kriterien zur Beurteilung der Technologie zur Darstellung des Höhenprofils 53
Tabelle 37: Nutzwertanalyse für die Wahl der Technologie zur Darstellung des Höhenprofils 56
Tabelle 38: Members der ProfileModel-Klassen 60
Tabelle 39: Members der ProfilePoint-Klasse 61
Tabelle 40: Members des Adapter-Interfaces 61
Tabelle 41: Members der GoogleMapsAdapter-Klasse 62
Tabelle 42: Members der AppView-Klasse 64
Tabelle 43: Members der GoogleMapsView-Klasse 65
Tabelle 44: Members der ProfileView-Klasse 66
Tabelle 45: Beispiel für die Darstellung von Geodaten durch Google 69

14 Formelverzeichnis

Formel 1: Berechnung des Abstandes zweier Punkte auf einer Kugel mit der Haversine-Formel 71
Formel 2: Umrechnung des Winkelmasses von Grad in Radiant 71
Formel 3: Berechnung des maximalen Höhenunterschieds aller Punkte 72

15 Scriptverzeichnis

Script 1: Erweiterung des Number-Prototyps um die Funktion zur Umrechnung von Grad in Radiant 71
Script 2: Implementierung der Haversine-Formel mit JavaScript 71
Script 3: Berechnung der maximalen Höhendifferenz mit JavaScript 72
Script 4: Zeichnen einer Ebene mit Three.js 72
Script 5: Setzen der Höhe 73

16 Literaturverzeichnis

- **POHL Klaus, RUPP Chris.** ***Basiswissen Requirements Engineering***. 3. Korrigierte Auflage. dPunkt-Verlag GmbH. 2011. Heidelberg
  ISBN: 978-3-89864-771-7
- **SUGRUE, James.** ***Beginning Backbone.js***. Apress/Springer. 2013. New York.
  ISBN: 978-1-4302-6334-0
- **DANCHILLA, Brian.** ***Beginning WebGL for HTML5***. Apress/Springer. 2012. New York.
  ISBN: 978-1-4302-3996-3
- **CANTOR Diego, JONES Brandon.** ***WebGL Beginner's Guide***. Packt Publishing Ltd. 2012. Birmingham.
  ISBN: 978-1-84969-172-7