

Bachelorthesis Informatik

Hyperlike: eine Plattform für Erfassung,
Analyse und Austausch ortsgebundener Daten

Autor Andreas Moor

Betreuer Beat Seeliger

Datum 25. Juni 2015



Zusammenfassung

Die Fülle und Vielfalt von durch Computersysteme erfassten Daten sowie die allgemeine Verfügbarkeit günstiger Rechenleistung haben zur Folge, dass Entscheidungen zunehmend auf Datenanalysen gestützt sind. Die fortschreitende Verbreitung von Smartphones und Tablets lädt dazu ein, Datenerfassung mittels Crowdsourcing durchzuführen zu lassen.

In dieser Arbeit wird eine wiederverwendbare Plattform entworfen für das Sammeln, Analysieren und Teilen von geolokalisierten Daten durch eine grosse Zahl unabhängiger Agenten. Durch eine Parametrisierung der Eingabewerte entsteht eine für eine Erhebung spezifische Oberfläche, die es den Teilnehmern erlaubt, numerische Beobachtungswerte einzugeben. Eine erweiterbare Analyse-Komponente untersucht die Eingabewerte auf räumliche und inhaltliche Zusammenhänge. Das Resultat der Analyse wird den Teilnehmern umgehend durch Visualisierung auf einer Karte zugänglich gemacht.

Die Arbeit besteht aus einer Formulierung der Anwendungsidee, der Durchführung einer Anforderungs- und einer Marktanalyse sowie der Erstellung eines Konzepts für die Umsetzung. Durch die Implementierung eines Prototypen und die Überprüfung von Akzeptanzkriterien wird ein Proof of Concept des Systems realisiert.

Inhaltsverzeichnis

1 Einleitung	1
1.1 Aufbau	1
1.2 Aufgabe	2
2 Motivation	4
2.1 Ausgangslage	4
2.2 Anwendungszweck	5
2.3 Zielgruppen	8
2.4 Anwendungsbeispiele	9
2.5 Fazit	14
3 Anforderungsanalyse	15
3.1 Einleitung	15
3.2 Allgemeine Übersicht	17
3.3 Systemfunktionalität	19
3.4 Anforderungen	24
3.5 Akzeptanzkriterien	31
4 Marktanalyse	34
4.1 Plattformen	35
4.2 Dienste	42
4.3 Komponenten	45
4.4 Ergebnis	47
4.5 Fazit	51
5 Konzept	52
5.1 Architektur	52
5.2 Nicht-funktionale Aspekte	57
5.3 Aufwandschätzung	58
5.4 Fazit	59

6 Proof of Concept	60
6.1 Anforderungen	60
6.2 Rahmenbedingungen	61
6.3 Umsetzung der Server-Komponente	62
6.4 Umsetzung der Client-Komponente	73
6.5 Installation und Deployment	81
6.6 Fazit	82
7 Testing	83
7.1 Testumgebung	83
7.2 Testprotokolle	84
7.3 Fazit	93
8 Schlussbemerkungen	94
8.1 Generelles Fazit	94
8.2 Herausforderungen	95
8.3 Persönliches Fazit	96
Glossar	97
Literaturverzeichnis	100
Bücher und Zeitschriften	100
Online-Dokumente	101
A Erklärung	102
B Projektplanung	103
Index	104

Abbildungsverzeichnis

1	Technologische und soziale Trends, die das System «Hyperlike» begünstigen	4
2	Bewertung von Orten in den Apps von «TripAdvisor» und «Google Maps»	6
3	Verarbeitungsschritte der Daten einer Erhebung in «Hyperlike»	7
4	Mögliche Teilnehmer der Erhebung aus Beispiel 1	9
5	Karte mit Resultat der Erhebung aus Beispiel 1	10
6	Mögliche Teilnehmer der Erhebung aus Beispiel 2	12
7	Karte mit Resultat der Erhebung mit zweistufiger Skala aus Beispiel 2	12
8	Karte mit Resultat der Erhebung aus Beispiel 3	13
9	Systemkontext	17
10	UML-Use-Case-Diagramm	19
11	Konfiguration von Gebiet, Raster und Segmenten der Analyse	25
12	Methode zur Bestimmung der Färbung eines Segments des Kartogramms	29
13	«ODK Collect» für Web bzw. Android und «ODK Aggregate»	36
14	Web-Client für Ushahidi: Eingabe (links) und Konfiguration (rechts)	38
15	Mobiler Ushahidi-Client für iOS	39
16	Formular-Builder und Administrations-Oberfläche von EpiCollectPlus	41
17	ODK Build (links) und Formhub-Backend (rechts)	43
18	Administrations-Oberfläche von Parse	44
19	Beispiel einer (mit Hilfe von Leaflet.js) angereicherten OpenStreetMap-Karte .	45
20	Komponenten	52
21	Datenfluss-Diagramm	53
22	Datenfluss Server-Komponente	55
23	Datenfluss Server-Komponente	55
24	Datenfluss Client-Komponente	56
25	Skalierung der Analyse von Segmenten durch <i>Sharding</i> der Datenbank	58
26	Überblick über den Technologie- <i>Stack</i> von Client- und Server-Komponente . .	61
27	Konzeptionelles Diagramm der umfangreichen Plattform Java SE	62
28	Von der Server-Komponente verwendete Frameworks und APIs	64

29	Diagramm der wichtigsten Schnittstellen der Server-Komponente	65
30	Java-IDE «JetBrains IntelliJ IDEA» mit Projekt- und Editor-Fenster	69
31	Graphische Oberfläche für die Datenbank MongoDB	70
32	Test Coverage Report der Server-Komponente	72
33	Von der Client-Komponente verwendete Frameworks und APIs	73
34	Rendering-Modell von React	74
35	Diagramm der wichtigsten UI-Komponenten des Clients	76
36	Hyperlike-Collector und -Reporter in Safari unter iOS	77
37	Entwicklungsumgebung für Client mit (im Uhrzeigersinn) Editor «Sublime Text», Web-Browser «Mobile Safari» in iOS Simulator, «Safari Web Inspector» und Terminal mit laufendem webpack-dev-server	78
38	Entwicklungsumgebung für React-Komponenten in Google Chrome	79
39	Test Coverage der Client-Komponente	80
40	Erfolgreiches Deployment der Applikation auf einem Linux-Host	81
41	Zusammenspiel von Client- und Server-Komponente während der Entwicklung	82
42	Konfiguration des Systems für AT-01	84
43	Gestartete Server-Applikation mit verbundenem Client	85
44	Ausgabe der Konfiguration der Erhebung	86
45	Anangepasste Client-Oberfläche mit rotem Hintergrund und weisser Schrift . . .	87
46	Dialoge und Resultat der automatischen Positionsbestimmung	88
47	UI-Elemente der Eingabemaske entsprechen der Konfiguration	89
48	Nach der Eingabe wird das Resultat angezeigt	90
49	Karte mit dem Resultat der Erhebung nach einer Eingabe	91
50	Ausgabe des Ergebnisses im Format JSON	92

Tabellenverzeichnis

1	Beispiel für Konfiguration der Eingabewerte	25
2	Beispiel für Konfiguration der Analyse	27
3	Beispiel für Konfiguration der Anzeige	29
4	Kategorien der Marktanalyse	34
5	Marktanalyse Plattformen	48
6	Bewertungsskala für Nutzwertanalyse	49
7	Nutzwertanalyse Dienste und Komponenten	50
8	Nutzwerte kombinierter Komponenten	50
9	Teilbereiche der zentralen Konfiguration	54
10	Arbeitspakete der Umsetzung mit Schätzung der relativen Komplexität	59
11	Erfüllung der erwarteten Ergebnisse der Akzeptanztests	93

1 Einleitung

1.1 Aufbau

Diese Arbeit ist in acht Teile gegliedert. In der **Einleitung** (dieser Abschnitt) wird die freigegebene Aufgabenstellung wiedergegeben und ein Überblick verschafft über die übrigen Abschnitte.

In der **Motivation** (Abschnitt 2) soll das Projekt «Hyperlike» motiviert und anhand konkreter (wenn auch fiktiver) Anwendungsbeispiele erläutert werden. Insbesondere soll in diesem Abschnitt das Problem vermittelt werden, welches das Resultat dieser Arbeit zu lösen beabsichtigt.

Die **Anforderungsanalyse** (Abschnitt 3) richtet ihr Auge auf die konkreten *Use Cases*, *Anforderungen* und *Akzeptanzkriterien* für ein System wie das in Abschnitt 2 beschriebene.

Im Rahmen der **Marktanalyse** (Abschnitt 4) wird abgeklärt, ob und wie bestehende Lösungen das gleiche oder ein ähnlich gelagertes Problem angehen und inwieweit sie diese Arbeit zu unterstützen vermögen. Darin werden Komplettlösungen ebenso untersucht wie Teillösungen, die bei der Umsetzung eingesetzt werden können.

Das **Konzept** (Abschnitt 5) verbindet die Anforderungen mit den Ergebnissen der Marktanalyse und legt dar, wie eine Umsetzung des angestrebten Systems gestaltet werden müsste.

Auf die analytischen Teile folgen Abschnitte, die sich mit der *Umsetzung* des Projekts befassen: Im Abschnitt **Proof of Concept** (Abschnitt 6) wird die Entwicklung eines funktionsfähigen Prototypen des Systems dokumentiert. Dieser durchläuft in Abschnitt 7 die Schritte des **Testing**, die notwendig sind, um die Funktionalität zu überprüfen.

Den Abschluss bilden die **Schlussbemerkungen** (Abschnitt 8), die die entstandenen Resultate kritisch beurteilen und die gewonnenen Erkenntnisse zusammenfassen.

Die letzten Teile sind formaler Natur und beinhalten ein **Glossar** mit mehrfach verwendeten oder erklärmgsbedürftigen Fachausdrücken sowie das **Literaturverzeichnis** und den **Index**.

Dieser Dokumentation liegt eine **Begleit-DVD** bei, auf der ein PDF-Dokument dieses Textes sowie der Programmcode und alle Artefakte des in Abschnitt 6 entwickelten Prototypen abgelegt sind.

1.2 Aufgabe

Die folgenden Abschnitte sind der Projektseite¹ im Einschreibe- und Bewertungssystem (EBS) der ZHAW entnommen. Sie werden an dieser Stelle nur zum Zwecke der Vollständigkeit wiedergibt.

1.2.1 Ziel der Arbeit

Das Ziel der Bachelor-Arbeit besteht in der Konzeption und Entwicklung einer Plattform für die Erfassung und Analyse ortsgebundener Daten mittels Smartphone. Die initiale Parametrisierung einer Erhebung bestimmt deren Wertemenge und Eingabeformen. Die Analyse der eingehenden Beurteilungen erfolgt mittels geeigneter Methoden, und das Resultat wird allen Benutzern mittels Visualisierung auf einer Karte zugänglich gemacht.

1.2.2 Aufgabenstellung

Im Rahmen dieser Bachelorarbeit werden vom Studenten folgende Aufgaben durchgeführt:

- Formulierung des Anwendungszwecks, detaillierte Erläuterung möglicher Einsatzmöglichkeiten
- Marktanalyse existierender Produkte und Lösungen (z.B. EpiCollect)
- Erstellung einer Anforderungsanalyse; diese beinhaltet folgende Aspekte:
 - Spektrum und Form der Eingabewerte
 - Auswertungsmethoden für eingehende Signale
 - Ausgabeform der Resultate
- Konzeption und Spezifikation eines Systems, das die gestellten Anforderungen erfüllt
- Implementierung des Prototyps
- Test des Prototyps und Protokollierung der Ergebnisse
- Vorbereitung und Durchführung einer Demonstration

1.2.3 Erwartete Resultate

- Formulierung des Anwendungszwecks, Liste von Fallbeispielen mit Erläuterungen für den möglichen Einsatz von «Hyperlike»
- Auflistung bestehender Produkte und Lösungen; Vergleich mit dem Anwendungszweck

¹ <https://ebs.zhaw.ch/projects/view/1629>

- Dokumentation des Anforderungskataloges an die Daten-Plattform
- Auswahl der für den Anwendungszweck geeigneten Technologien; Beschreibung der Konzeption und Spezifikation des Systems
- Lauffähige Prototypen für Plattform-Server und mobilen Client
- Dokumentation der Test-Szenarien und deren Ergebnisse
- Demonstration im Rahmen der Schlusspräsentation

1.2.4 Geplante Termine

- Kickoff: Februar 2014
- Design-Review: April 2015
- Präsentation: Juli 2015

2 Motivation

Dieser Abschnitt soll die vorliegende Arbeit motivieren und die in Abschnitt 1 formulierte Aufgabe erläutern. Nach einer gegenüber der Aufgabenstellung erweiterten Darlegung der Ausgangslage, die das Bedürfnis nach dem in dieser Arbeit entworfenen System begründet, folgt eine ausführliche Diskussion der Zielgruppen und des Anwendungszwecks. Eine anschließende Reihe von Anwendungsbeispielen soll den konkreten Nutzen des angestrebten Systems aufzeigen.

2.1 Ausgangslage

Eine Reihe gesellschaftlicher und technologischer Entwicklungen haben den Autor vom Bedürfnis nach einer Plattform für Erfassung, Analyse und Austausch ortsgebundener Daten überzeugt: zunehmend auf *Datenanalyse* gestützte Entscheidungen auf allen Ebenen der Gesellschaft, die fortschreitende Verbreitung von *Smartphones*, sowie die wachsende Bereitschaft, persönliche Einschätzungen mit der *Öffentlichkeit* zu teilen.

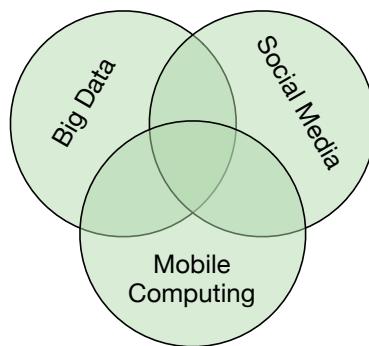


Abbildung 1: Technologische und soziale Trends, die das System «Hyperlike» begünstigen

Die Fülle und Vielfalt der durch Computersysteme erfassten Mess- und Logdaten sowie die allgemeine Verfügbarkeit günstiger Rechenleistung haben zur Folge, dass Entscheidungen von Firmen und Individuen zunehmend durch eine Datenanalyse gestützt oder motiviert sind: Datenjournalismus in den Medien, «Big Data» in Unternehmen, Anhänger der «Quantified Self»-Bewegung mit ihren Fitbit- und Jawbone-Geräten – sie alle sind *data-driven* und verlassen sich auf wachsende Datenarchive und Messstationen.

Die fortschreitende Verbreitung mobiler Taschencomputer wie *Smartphones* und *Tablets* mit ihren hochentwickelten Sensoren und Schnittstellen lädt dazu ein, Beobachtungswerte durch eine möglichst grossen Zahl von Agenten durchzuführen zu lassen. Dieses insbesondere durch [1] ins öffentliche Bewusstsein gerufene «*Crowdsourcing*» der Datensammlung kann objektive Messwerte des Geräts (wie Position und Bewegungsrichtung) mit Beobachtungswerten und

persönliche Beurteilungen (wie Stimmung oder Vorlieben) verbinden. Als Motivation zur Teilnahme an einer solchen Erhebung dient der individuelle Nutzen der gesammelten und analysierten Daten. Diese können den einzelnen bei seiner Entscheidungsfindung unterstützen. [2] beschreibt, wie die mobile Datensammlung in verschiedenen wissenschaftlichen Disziplinen zur Anwendung kommt.

Diese Auslagerung der Datenerhebung an die *Crowd* wird befördert durch eine fortschreitende Erosion der Privatsphäre durch den Siegeszug sozialer Medien in allen Gesellschaftsschichten und der damit einher gehenden Bereitschaft, persönliche Beurteilungen und Meinungen mit der Öffentlichkeit oder zumindest einem erweiterten Umfeld zu teilen.

Die vorliegende Arbeit versucht, diese in Abbildung 1 zu den plakativen *Buzzwords* verdichteten Trends «Big Data», «Social Media» und «Mobile Computing» nutzbar zu machen.

2.2 Anwendungszweck

Das in dieser Arbeit entworfene System mit dem Namen «Hyperlike» will die Durchführung von Erhebungen erleichtern, die sich durch folgende Eigenschaften auszeichnen:

- Sie werden durchgeführt durch eine *Vielzahl informeller Kollaborateure* (die «Crowd»), deren primäres Interesse darin liegt, selbst von den aus der Erhebung gewonnenen Erkenntnissen zu profitieren. Ein sekundäres Interesse kann das Bedürfnis des Ausdrucks des eigenen Standpunktes sein.
- Sie bestehen aus Fragestellungen, deren Antworten sich *quantitativ* ausdrücken lassen, also durch die Eingabe numerischer Werte.
- Die Antworten sind *ortsgebunden*, das heisst sie treffen eine Aussage über die momentane Position der Teilnehmer.

Erhebungen dieser Art sind trotz dieser auf den ersten Blick erheblichen Beschränkungen sehr vielseitig und verbreitet (siehe Anwendungsbeispiele in Abschnitt 2.4). In den meisten Bereichen müssen qualitative Aussagen als quantitative Signale strukturiert sein, um sich für eine Analyse irgendwelcher Art zu eignen.

Wie die fast flächendeckende Verbreitung von «Like-Buttons» in mobilen Apps und auf dem World Wide Web bezeugen, ist die Bewertung von Fotos und anderen Objekten durch die Benutzer eine der wichtigsten Funktionalitäten von Social-Media- und anderen Plattformen.

Auch das *Bewerten von Orten* durch die Crowd erfreut sich grosser Beliebtheit, wie die Applikationen «Google Maps» und «TripAdvisor» (Abbildung 2) oder das Facebook-Tool «Place Tips» zeigen. Zwar sind bei vielen Reise-Applikationen die Bewertungen nicht primär an den geografischen Ort gebunden, sondern betreffen eine an dieser Stelle erbrachte Dienstleistung

oder eine Sehenswürdigkeit – das Prinzip der Verbindung von Bewertung und Ort ist aber auch hier augenfällig.

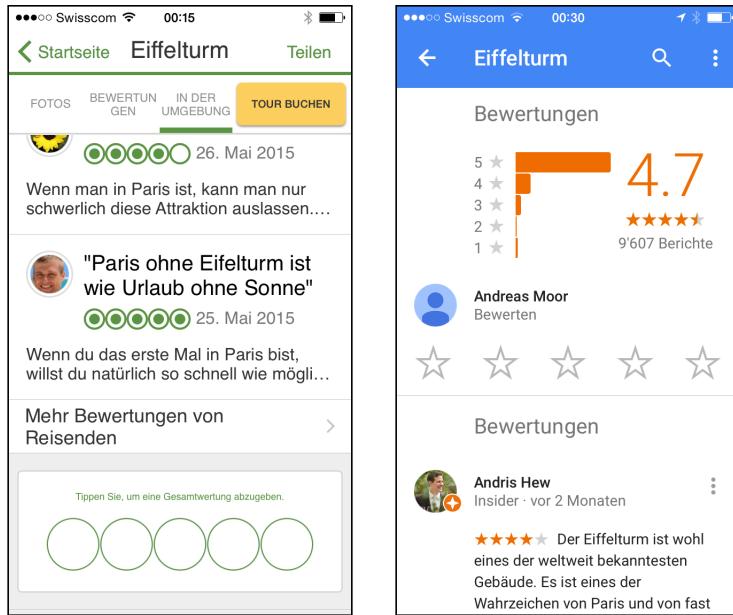


Abbildung 2: Bewertung von Orten in den Apps von «TripAdvisor» und «Google Maps»

«Hyperlike» soll Erhebungen mit den beschriebenen Eigenschaften unterstützen, indem die Schritte der Umsetzung, die wiederholbar sind, extrahiert und zu einem konfigurierbaren Dienst verpackt werden. Die Schritte der Spezifikation, Speicherung und Analyse von Signalen sowie das Design der Schnittstellen einer solchen kollaborativen Daten-Plattform sind bei vielen Probleminstanzen ähnlich und sind somit Kandidaten für die in dieser Arbeit vorgeschlagene Abstrahierung.

Das Resultat ist eine benutzerfreundliche Web-Applikation, die einerseits die Infrastruktur für zentralisierte Datenerfassung liefert und anderseits durch Parametrisierung den Kontext für die erfassten Werte schafft.

Konkret bedeutet dies, dass die Eingabe der gewünschten Fragestellungen und der Parameter der Darstellung ausreicht, um mit «Hyperlike» eine Erhebung zu starten. Die Teilnehmer der Umfrage sollen die konfigurierten Fragen beantworten und unmittelbar danach die Ergebnisse konsumieren können. Das Resultat der Erhebung wird dabei auf einer Karte visualisiert und laufend durch neu eintreffende Beobachtungswerte aktualisiert. Die Darstellung folgt dabei einem *Analyseschritt*, der Beobachtungswerte aus örtlich zusammengehörenden Gebieten zusammenfasst.

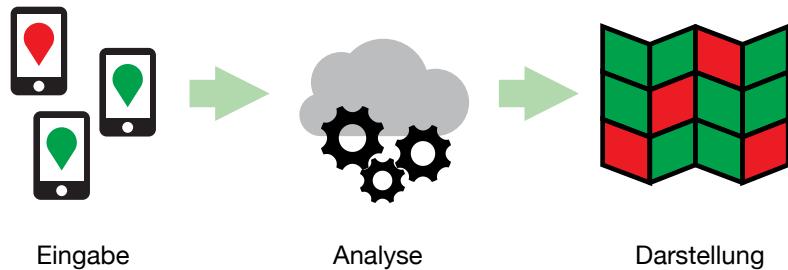


Abbildung 3: Verarbeitungsschritte der Daten einer Erhebung in «Hyperlike»

Die Bezeichnung «Hyperlike» soll einerseits die Mehrdimensionalität («hyper») der zu verarbeitenden Daten (räumlich, zeitlich, etc.) widerspiegeln, anderseits soll sie den Menschen («like») als primäre Datenquelle in den Vordergrund stellen. Zwar sind automatisierte Eingaben durchaus möglich, doch das Systems zielt in erster Linie darauf, durch relativ wenig Konfiguration eine Applikation bereitzustellen, die die gesamte Verwertungskette von der Dateneingabe durch Personen bis zur graphischen Ausgabe abdeckt.

2.2.1 Potenzial

Die Einsatzmöglichkeiten der «Hyperlike»-Plattform sind vielseitig. Überall dort, wo Menschen in einem geografisch begrenzten Gebiet zusammenkommen, entsteht das Bedürfnis nach *Überblick*. «Hyperlike» bietet die Möglichkeit, mit wenigen Handgriffen einen zentralen Dienst bereitzustellen, der es erlaubt, durch die Visualisierung von Beobachtungswerten eine *Vogelperspektive* über ein Gebiet einzunehmen.

Als Werkzeug für die Mitglieder einer technisch kompetenten, in der Interpretation von Infografiken geübten Gesellschaft hat «Hyperlike» das Potenzial, die Gestaltung der Freizeit zu beeinflussen. Durch die einfache Konfiguration einer Erhebung, die unkomplizierte Teilnahme daran und den unmittelbaren Konsum der Resultate ist die Plattform geeignet, ein selbstverständlicher Teil des Angebots von Unternehmen, Veranstaltungen und Vereinen zu werden.

Der für diese Arbeit abgesteckte Rahmen der Funktionalität lässt sich bei einer zukünftigen Iteration des Projekts erweitern. So ist es denkbar, den Funktionsumfang in den Bereichen Datenerfassung und Visualisierung zu vergrößern (etwa durch Anreicherung der Eingabewerte mit Fotos und Stichwörtern). Es ist nicht zuletzt mit Blick auf die vom Autor noch nicht in Betracht gezogenen Anwendungsmöglichkeiten, dass bei der Konzipierung des Systems viel Wert auf seine *Erweiterbarkeit* gelegt wurde.

2.2.2 Limitationen

«Hyperlike» hat, wie in Abschnitt 2.2 beschrieben, einen klaren Fokus auf die *Darstellung quantitativer, ortsgebundener Sachverhalte*. Es geht nicht darum, komplexe Formulare mit un-

strukturierten Eingaben (wie etwa Freitext oder *Emojis*) anzubieten. Durch die Ausrichtung auf Agilität und auf das Teilen einer grossen Zahl kleiner Informationspakete entsteht weniger ein Geoinformationssystem als ein klar auf soziale und örtliche Kontexte ausgerichtete *Data-Sharing*-Plattform, quasi ein «Twitter für Geodaten».

Natürlich sind durch die in dieser Arbeit erfassten *Use Cases* eine Vielzahl von Erhebungen nicht abgedeckt. Durch eine konfigurierbare Plattform können viele Spezialfälle nicht bedient werden. Ob und wie die hier gewählte Abstraktion die richtige ist, wird sich erst mit der Zeit zeigen. Auch hier vertraut der Autor auf die erweiterbare Ausgestaltung des Konzepts, um dessen Umfang bei Bedarf auszudehnen.

Zudem beschränkt sich die im Rahmen dieser Arbeit entworfene Funktionalität auf den Einsatz über das *World Wide Web* – und damit auf Gebiete, die über Zugang zum Internet verfügen. Da die Feststellung der Position nur von der Verfügbarkeit von GPS oder anderen Ortungsmechanismen abhängt, ist diese Beschränkung bloss eine *technische*, die durch aufwändigere Implementationen beseitigt werden kann.

2.3 Zielgruppen

«Hyperlike» hat als Plattform für die Bereitstellung von Erhebungen zweierlei Zielgruppen: den Anbieter einer Erhebung und den Betreiber des Dienstes. Beide sind insbesondere bei kleinen Installationen die gleiche Person. Dennoch seien sie hier einzeln skizziert. Die *Teilnehmer* einer Erhebung bilden hier keine Zielgruppe, da sie nicht direkte Konsumenten der Plattform – sondern nur von ihren Ergebnissen – sind.

2.3.1 Anbieter einer Erhebung

Als *Anbieter* kommt jedermann in Frage, der systematisch Beobachtungen über ein geografisches Gebiet sammeln und anbieten will. Wie die Anwendungsbeispiele im nächsten Abschnitt zeigen, sind die Profile der Anbieter sehr unterschiedlich.

Der Anbieter waltet als Mittler zwischen den Teilnehmern einer Erhebung, indem er das Angebot initiiert und Entscheidungen über dessen Umfang und Ausgestaltung fällt. Somit kommen primär offizielle Organe einer Gemeinschaft, Unternehmer und Organisatoren von öffentlichen Anlässen infrage. Ihre Motivation reicht vom altruistischen Antrieb bis zum kommerziellen Interesse an entweder den gesammelten Daten oder am (kommerziellen) Erfolg der Umgebung, in der die Erhebung durchgeführt wird.

2.3.2 Betreiber einer Erhebung

In technischen Belangen versucht das System, als *Betreiber* Personen im Bereich zwischen Informatiker und technisch versierten Laien anzusprechen. «Hyperlike» richtet sich also primär

an Personen, die zwar über eine hohe technische Kompetenz verfügen und fähig sind, Server in Betrieb zu nehmen und Dienste zu betreiben, jedoch mit Programmierung und Datenverarbeitung nicht oder nur wenig vertraut sind.

Wie als Beispiele die populären *Content-Management*-Systeme «Wordpress»² und Joomla³ oder der *Webshop*-Plattformen wie «Magento»⁴ zeigen, besteht ein Bedarf an solchen auf Implementierer ausgerichteten Systemen. Diese Plattformen verfügen im unkonfigurierten Zustand über keine oder nur wenig Funktionalität und werden erst durch kundenspezifische Anpassungen zum Leben erweckt. Diese Anpassungen beschränken sich auf Parameter und Modifikationen von deklarativen Elementen des Programms (z.B. Struktur der Site oder Templates und *Style Sheets* für einzelne Seiten).

2.4 Anwendungsbeispiele

2.4.1 Beispiel 1: Verhältnisse für Skitouren

Die Alpen sind ein Zentrum für einen modernen Trendsport: Ski- und Snowboard-Touren. Betreiber der Sportart besteigen mit Fellen oder Schneeschuhen die schneedeckten Gipfel der Berge, um anschliessend im (bestenfalls) unverspurten Schnee abzufahren. Das sportliche Spektrum der Anhänger ist sehr gross und reicht vom Leistungssportler bis hin zum Genusswanderer.



Abbildung 4: Mögliche Teilnehmer der Erhebung aus Beispiel 1

Als Outdoor-Betätigung in der (mehr oder weniger) unberührten Natur ist sie nicht ohne ihre Gefahren: neben den alpinistischen Risiken spielen die Gefahren des Skisports – wie Unfall-

² <https://wordpress.org>

³ <http://www.joomla.org>

⁴ <http://magento.com>

und Verletzungsgefahr – eine Rolle. Über das Ausmass der Gefahren bestimmen zu grossen Teilen die jeweils *aktuellen Schnee- und Wetterverhältnisse*.

Während über die Verhältnisse auf den Skipisten der Wintersport-Destinationen gute und aktuelle Informationen erhältlich sind, ist die Situation in nicht erschlossenen Gebieten oft unsicher oder nur verspätet feststellbar. Ein wichtiges Instrument dazu sind – neben den Wetterprognosen – die Schneekarten des Instituts für Schnee- und Lawinenforschung SLF⁵. Diese erscheinen allerdings oft nur mit Verzögerung und sind insbesondere in den voralpinen Gebieten zu wenig genau.

Für Skitourengänger sind *präzise und aktuelle Informationen* sehr wichtig: Ob die letzten 200 Höhenmeter einer Abfahrt fahrbar sind, kann über die Durchführung einer Tour entscheiden. Es gestaltet sich gerade zu Beginn und Ende der Saison schwierig, die Verhältnisse aus der Ferne (z.B. aus einem Büro im Mittelland) einzuschätzen.

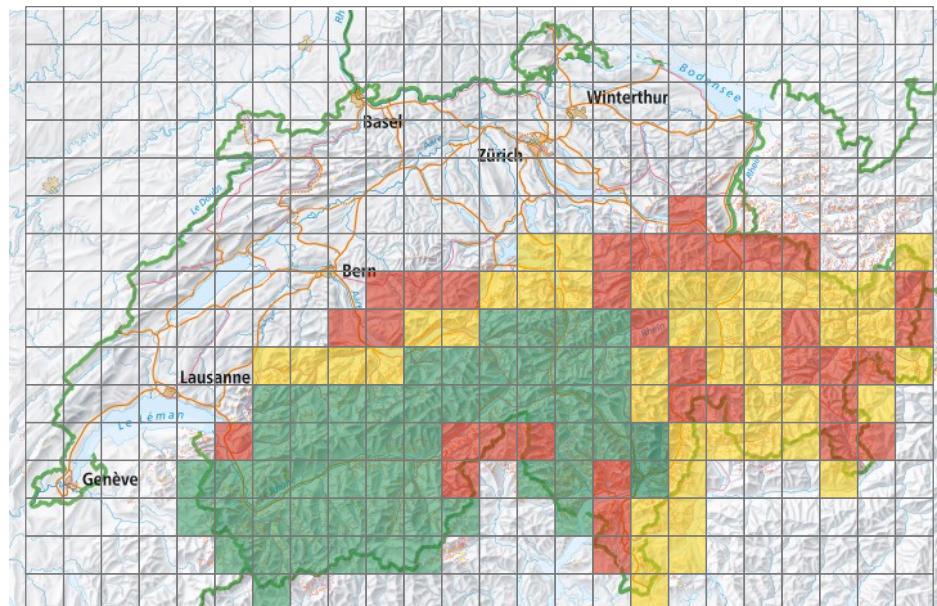


Abbildung 5: Karte mit Resultat der Erhebung aus Beispiel 1

Ein Dienst mit einer übersichtlichen Darstellung der Verhältnisse im gesamten schweizerischen Alpenraum wäre ein wertvolles Hilfsmittel für die Planung und Durchführung von Skitouren. Als solcher könnte eine Instanz von «Hyperlike» dienen. Während der Saison sind täglich viele Tourengänger unterwegs, die die Situation an mehreren Stationen ihrer Tour dokumentieren könnten. Durch eine zeitnahe Verarbeitung der Beobachtungen wären Sportler, die sich für einen Ausflug in ein bestimmtes Gebiet interessieren, nicht mehr auf veraltete Tourenberichte und verpixelte Bilder von Webcams angewiesen.

⁵ http://www.slf.ch/schneinfo/schneekarten/hnid/index_DE

Abbildung 5 zeigt, wie eine solche Darstellung aussehen könnte. Einerseits zeigt sie einen Überblick über die Gesamtsituation im Einzugsgebiet, andererseits geben einzelne Elemente des Rasters Aufschluss über die Situation in einem bestimmten Gebiet.

Als Anbieter der Erhebung könnte ein Sportverein auftreten, der über die Glaubwürdigkeit und das Know-how zur Ausgestaltung des Fragenkatalogs verfügt. Als Eingaben der Erhebung kommen Werte in Frage, die sich nicht aus der Ferne bestimmen lassen. Die möglichen Eingabe werden so festgelegt, dass durch die Unterschiedlichkeit der Wahrnehmung entstehende Missverständnisse minimiert werden können. Beispiele für Beobachtungswerte sind Schneehöhe, Neuschneemenge, Schneehärte, Temperatur und Wolkendichte. Wichtig für die Gestaltung des Fragenkatalogs ist, dass nur eine Handvoll Werte eingegeben werden müssen und die Werte auch durch Nicht-Experten festgestellt werden können. Damit kann die Qualität der Erhebung verbessert und die Partizipation erhöht werden.

Natürlich darf das Resultat der Erhebung *keinesfalls das einzige Kriterium* sein, mit Hilfe dessen über die Durchführung einer Tour entschieden wird. Skitouren erfordern Erfahrung, und für Entscheide müssen unbedingt auch andere Informationsquellen (Wetterbericht, Lawinenbulletin, SAC-Hüttenwarte) konsultiert werden. Doch für grobe Einschätzungen der regionalen Verhältnisse aus der Ferne kann «Hyperlike» wichtige Hinweise liefern.

2.4.2 Beispiel 2: Stadtfest

Veranstaltungen mit einem grösseren Einzugsgebiet sind mögliche Einsatzgebiete von «Hyperlike». Darunter fallen Anlässe wie Konzerte, Dorf- und Stadtfeste oder Jahrmärkte. Ein Stadtfest ist ein gutes Beispiel für diese Anwendung: Es ist geografisch und zeitlich klar begrenzt und spricht üblicherweise eine grosse Gruppe von Besuchern an. Eine Bewertung der einzelnen Lokalitäten, auf die das Fest verteilt ist, dient Besuchern und Touristen zum Auffinden der attraktivsten Orte und hätte zudem dokumentarischen und sozialen Nutzen für die Bewohner der Stadt.



Abbildung 6: Mögliche Teilnehmer der Erhebung aus Beispiel 2

Der Veranstalter des Festes tritt in diesem Beispiel als Anbieter auf. Er kann die Erhebung für Werbezwecke, als spielerischer Zusatz zum Anlass und für eigene statistische Zwecke benutzen. «Hyperlike» lässt sich in das bestehende Informationsangebot der Veranstaltung integrieren und mit wenig Zusatzaufwand betreiben.

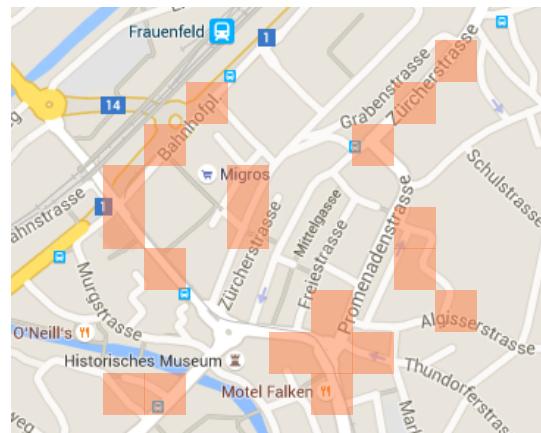


Abbildung 7: Karte mit Resultat der Erhebung mit zweistufiger Skala aus Beispiel 2

Ein so subjektive Beobachtung wie die Attraktivität der Lokalitäten eines Fests ruft nach einer einfachen Fragestellung. So würde eine vielleicht zweistufige Skala genügen, um das Ge- oder Missfallen der Besucher auszudrücken: «Like» oder «Don't Like». Die als zusammengehörend betrachteten Gebiete der Erhebung lassen sich vom Veranstalter so konfigurieren, dass sie die unterscheidbaren Bereiche des Festes widerspiegeln und als Ganzes bewertet und eingefärbt werden.

2.4.3 Beispiel 3: Naturkatastrophe

In der Folge des Erdbebens von Haiti im Jahr 2010 kam der *Social-Media-Dienst «Twitter»* zu Prominenz als wichtiges Hilfsmittel bei der Katastrophenhilfe. Insbesondere in amerikanischen Medien wurde die Rolle von Twitter unterstrichen. Das *Wall Street Journal* berichtete: “In the wake of the devastating earthquake in Haiti, Twitter has proven to be an important tool for fundraising and relief efforts to help the disaster’s victims.”⁶

Durch die Analyse von mit dem *Hashtag #Haiti* markierten *Tweets* gelang es einem Institut der Universität von Colorado⁷, die Nachrichten «systematischer und die enthaltenen Daten extrahierbar zu machen»⁸. Dadurch konnten die notwendigen Hilfsgüter besser geplant und Vermisste geortet werden.

«Hyperlike» würde die Erfassung und Weitergabe von strukturierter Information in einem solchen Fall stark vereinfachen. Mit Hilfe einer durch Helfer und Betroffene durchführbare Erhebung der lokalen Schadensausmasse könnte die Hilfeleistung in Notsituationen verbessert werden. Zudem entstünde eine übersichtliches, aus Berichten von erster Hand zusammengesetztes Bild der Situation. Dadurch, dass eine «Hyperlike»-Instanz schnell konfiguriert und in Betrieb genommen werden kann, eignet sich die Plattform besonders gut für dieses Szenario.

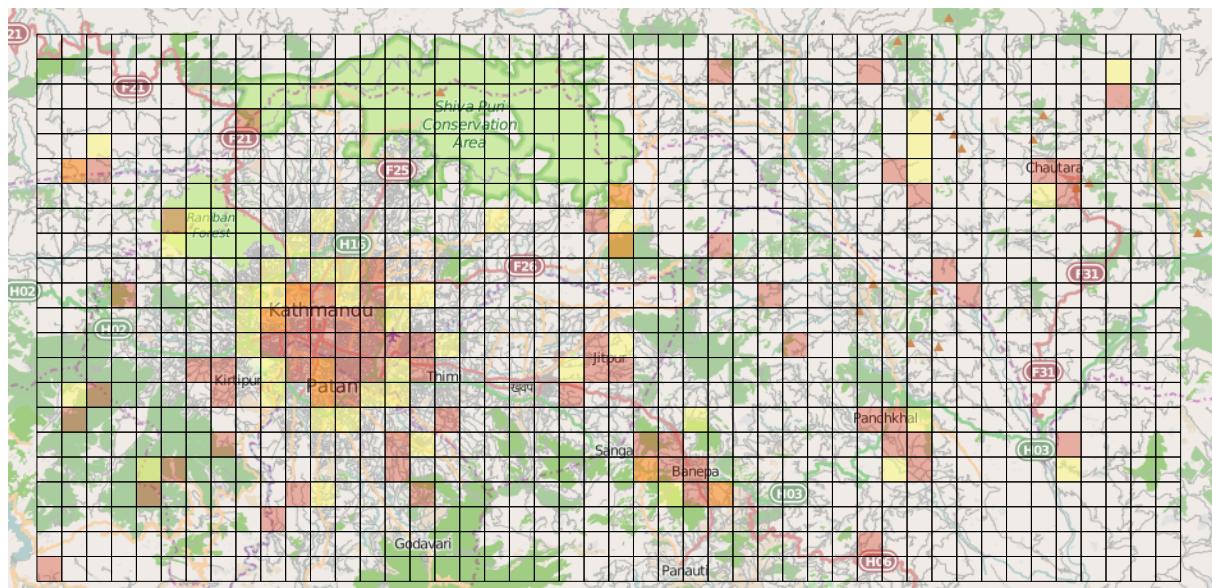


Abbildung 8: Karte mit Resultat der Erhebung aus Beispiel 3

Anbieter einer Erhebung über das Ausmass der Zerstörung in einem Katastrophengebiet können Regierungen und staatliche oder nichtstaatliche Organisationen sein. Die geografischen Grenzen der Erhebung ergeben sich aus dem betroffenen Gebiet, während die Fragestellungen sich

⁶ <http://blogs.wsj.com/digits/2010/01/14/twitter-helps-in-haiti-quake-coverage-aid/>

⁷ <http://epic.cs.colorado.edu/>

⁸ <http://www.pbs.org/newshour/rundown/haiti-quake-propels-twitter-community-mapping-efforts/>

aus Ausmass und Ursache der Katastrophe gewinnen lassen. Mögliche Fragestellungen bei einem Erdbeben sind etwa: Anzahl Toter und Verletzter, Prozentsatz zerstörter Häuser oder Wasserstand bei Überschwemmungen. Obwohl es sich bei allen Eingaben nur um Schätzungen handeln kann, entsteht bei genügend Teilnehmern und einer geschickten Wahl der Visualisierung ein gutes Bild der Lage.

2.5 Fazit

Bei dem zu entwerfenden System «Hyperlike» handelt es sich um eine zeitgemässse Plattform, die technologische und gesellschaftliche Trends nutzt, um Erhebungen anzubieten, die ortsgebundene Beobachtungswerte von Teilnehmern sammelt, analysiert und die Resultate anderen Teilnehmern zugänglich macht. Die Stärken eines solchen Systems liegen in der Einfachheit der Konfiguration und der Agilität des Deployments.

Der Einsatz von «Hyperlike» lohnt sich immer dann, wenn ortsgebundene, quantitative Beobachtungswerte von einer grossen Zahl informeller Kollaboratoren erfasst und konsumiert werden sollen. Es sind zahlreiche Anwendungen denkbar, beispielsweise für die Planung von Freizeitaktivitäten, den Besuch von Veranstaltungen oder die Hilfeleistung in Katastrophengebieten.

3 Anforderungsanalyse

3.1 Einleitung

3.1.1 Zweck

Diese Anforderungsanalyse soll den Grundstein legen für die Entwicklung des in Abschnitt 2 skizzierten Systems. Aus den in Abschnitt 3.3 formulierten *Use Cases* wird eine Liste von *Anforderungen* zusammengestellt. Der Abschnitt dient einerseits der Dokumentation und damit der Kontrolle der angestrebten Ergebnisse, anderseits soll er die dem Anwendungszweck zugrundeliegenden fachlichen Konzepte erläutern.

3.1.2 Stakeholder

Anbieter

Der *Anbieter* konfiguriert das System und gibt dessen Deployment und Betrieb in Auftrag. Er entscheidet über sämtliche Aspekte der Konfiguration einer Instanz und bietet diese den *Endbenutzern* zur Eingabe von Beobachtungswerten sowie zum Konsum der Resultate der Analyse an.

Die technischen Kenntnisse des Anbieters sind *gering*, d.h. er ist vertraut mit Zweck und grundlegender Funktionsweise des Systems, hat aber keinen Einblick in die Details der Umsetzung. Sein primäres Interesse am System ist die aus der Konfiguration hervorgehende Funktionalität.

Erwartungen:

- Zuverlässige Software
- Benutzerfreundlichkeit für die Teilnehmer der Erhebung
- Verständliche Dokumentation der Konfiguration
- Erweiterbarkeit

Entwickler

Der *Entwickler* nimmt durch die Erstellung von (Unter-)Programmen Einfluss auf die Funktionsweise des Systems. Die Ergebnisse seiner Tätigkeit kommen im Inneren des Systems durch den Zugriff auf öffentliche Schnittstellen zum Einsatz, und führen in der Regel dazu, dass der bestehende Funktionsumfang *erweitert* wird.

Der Entwickler hat ein naturgemäß *hohes technisches Niveau* – er kennt die Funktionsweise des Systems (zumindest in den für ihn relevanten Bereichen) sehr genau und ist in der Lage,

Fehlverhalten zu erkennen. Der Entwickler wird erst beigezogen, wenn das System die vom Anbieter gewünschte Funktion nicht durch Konfiguration erbringen kann.

Erwartungen:

- Stabile und mächtige Schnittstellen
- Umfassende Dokumentation der Schnittstellen

Betreiber

Der *Betreiber* der Instanz übernimmt das Deployment des Systems und stellt die laufende Instanz dem Anbieter und den Endbenutzern zur Verfügung. Sein technisches Know-how ist ungewiss und liegt irgendwo zwischen gering und hoch, zumal Betreiber und Anbieter oft die selbe Person sind. Verfügt der Betreiber über Know-how, dann ist dieses meist auf den Betrieb der Systems ausgerichtet, so dass er mit den technischen Innereien nicht notwendigerweise vertraut ist.

Erwartungen:

- Einfaches Deployment
- Einbindung in bestehende Deployment- und Betriebs-Prozesse
- Stabilität des Systems und kontrolliertes Verhalten im Fehlerfall
- Umfassende Dokumentation der Systemanforderungen und des Betriebs

Endbenutzer

Der *Endbenutzer* übernimmt die Eingabe von Beobachtungswerten in ein laufendes System und konsumiert die Ergebnisse der Datenanalyse. Er verfügt über *keine technischen Kenntnisse* und hat keinen Einblick in die Funktionsweise des Systems. Seine Möglichkeiten bei Eingabe und Konsum beschränken sich auf die durch den Anbieter konfigurierte Funktionalität.

Erwartungen:

- Intuitive Bedienung
- Verfügbarkeit des Systems
- Korrektheit der Analyse

3.1.3 Übersicht

Die Struktur dieses Abschnitts folgt den Empfehlungen bezüglich «Standardinhalte» von [3] (Seite 49f.), und damit teilweise [16]. Einige der dort beschriebenen, einleitenden Abschnitte fallen allerdings mit anderen Teilen dieser Dokumentation zusammen: So sind sämtliche *Definitionen* in Glossar zusammengefasst und alle *Referenzen* finden sich in der Literaturverzeichnis. Der *Systemumfang* und die *Zielgruppe* schliesslich wurden bereits in der Motivation (Abschnitt 2) erläutert.

Die folgenden Abschnitte betten das System in sein Umfeld ein, stellen Randbedingungen auf und erläutern die Systemfunktionalität anhand von Use Cases. Diese fliessen im Anschluss in die Formulierung von *funktionalen und nicht-funktionalen Anforderungen* und deren Akzeptanzkriterien ein.

3.2 Allgemeine Übersicht

3.2.1 Systemumfeld

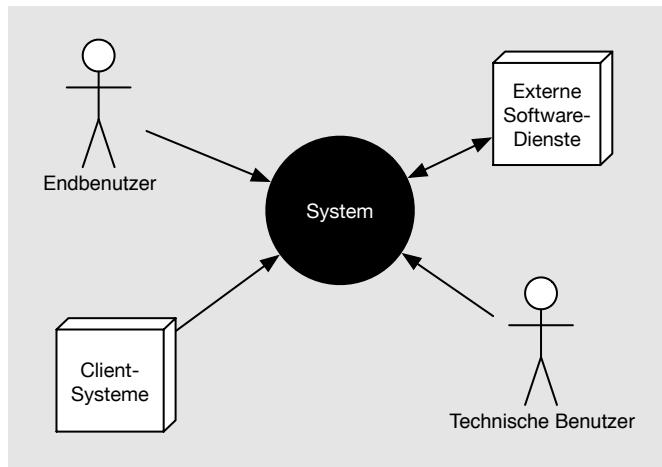


Abbildung 9: Systemkontext

Die Systemgrenze verläuft entlang dreierlei Achsen: Einerseits wird das System durch die *Schnittstellen für Eingabe und Ausgabe von Daten* einer Erhebung begrenzt. So bildet die Stakeholder-Gruppe der Endbenutzer die primäre Quelle und Senke des Systems: Sie nehmen die Dateneingabe vor und konsumieren die Ergebnisse der Analyse. Als zusätzliche Quelle des Systems können externe *Client-Systeme* für die automatisierten Dateneingabe fungieren, diese Möglichkeit wird jedoch im Rahmen dieser Arbeit nicht berücksichtigt.

Die zweite Grenze des Systems wird markiert durch die zur Gruppe der *technischen Benutzer* zusammengefassten Stakeholder: Anbieter, Entwickler und Betreiber. Sie beliefern das System

nicht mit Eingabewerten sondern schaffen mittels Konfigurations- und Programmier-Schnittstellen die Rahmenbedingungen für den Betrieb des Systems.

Die letzte Systemgrenze bildet die zu den *externen Software-Diensten*, die zur Umsetzung des Systems verwendet werden können. Welche dies sind und in welchem Verhältnis diese zum System stehen, kann erst eine in Abschnitt 4 folgende *Marktanalyse* zeigen. Es kann sich dabei um Dienste zur Analyse, Speicherung oder Anzeige der Daten einer Erhebung handeln – ihnen allen gemein ist, dass sie vom System über programmatiche Schnittstellen angesprochen werden und möglicherweise vom Anbieter zusätzlich zur Verfügung gestellt werden müssen.

3.2.2 Architekturbeschreibungen

Ebenso wie beim Systemumfeld werden die grundlegendsten Architekturentscheide auf Basis der fundamental unterschiedlichen Schnittstellen der drei Segmente der Systemkontextes gefällt.

1. Die Kommunikation mit Endbenutzern und Client-Systemen verläuft über die Protokolle des *World Wide Web*. Diese architektonische Weichenstellung ergibt sich unmittelbar aus dem Anwendungszweck des Systems: Nur das WWW bietet einen überall und zu jeder Zeit verfügbaren Kanal, über den Eingaben durch die Endbenutzer erfolgen können.
2. Die Schnittstelle für die Konfiguration des Systems bilden *Dokumente*, die beim Start des Systems gelesen werden. Durch diesen Mechanismus wird sichergestellt, dass die Konfiguration vor der Inbetriebnahme des Systems abgeschlossen ist und dass das Deployment und der Betrieb des Systems vereinfacht werden. Über die Methoden der Bearbeitung und Speicherung der Konfigurations-Dokumente wird bei der Umsetzung entschieden.
3. Über den Einfluss von externen Software-Diensten auf die Schnittstellen-Architektur lässt sich vor der Marktanalyse noch keine Aussage machen.

3.2.3 Randbedingungen und Annahmen

Viele Funktionen, die das angestrebte System für einen *öffentlichen Einsatz* zwingend benötigen würde, können im Rahmen dieser Arbeit nicht berücksichtigt werden. So müssen Aspekte der Sicherheit, des Betriebs und der Überwachung aussen vor bleiben. Es sei deshalb an dieser Stelle die Annahme getroffen, dass Funktionen dieser Art später hinzugefügt werden. Zudem können Aspekte der Fairness und Qualitätssicherung der Eigaben einer Erhebung in dieser Arbeit nicht behandelt werden und müssen auf eine spätere Iteration der Bearbeitung des Themas verschoben werden.

3.3 Systemfunktionalität

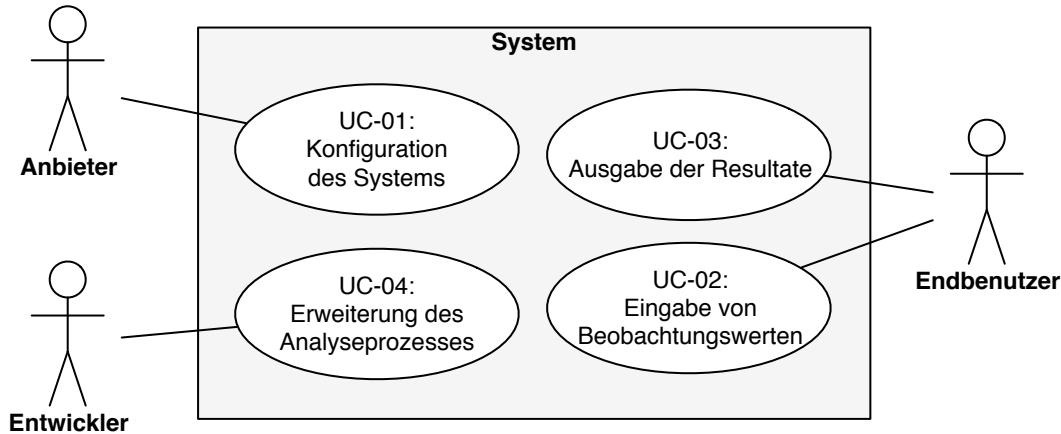


Abbildung 10: UML-Use-Case-Diagramm

3.3.1 Akteure

Von den in Abschnitt 3.1.2 beschriebenen Stakeholdern treten Anbieter, Endbenutzer und Entwickler in den Use Cases als Akteure auf. Der Betreiber tritt nicht als Akteur auf, da konkrete, als Use Case formulierbare Aspekte des Deployments und des Betriebs den Umfang des ange strebten Systems sprengen würden.

3.3.2 Priorität

Für alle Use Cases wird als Priorisierungstechnik die «Ein-Kriteriums-Klassifikation» eingesetzt (siehe [3] Seite 134). Diese unterscheidet die drei Prioritätsklassen

- *Mandatory* (unbedingt zu realisieren)
- *Optional* (nicht zwingend zu realisieren)
- *Nice-to-have* (Nichtberücksichtigung gefährdet Erfolg des Systems nicht)

Die Priorisierung der Use Cases hat Einfluss auf die spätere Gewichtung der daraus abgeleiteten Anforderungen: Höher priorisierte Use Cases führen zu stärker gewichteten Anforderungen.

3.3.3 Use Cases

UC-01: Konfiguration des Systems

Bezeichner	UC-01
Name	Konfiguration des Systems
Priorität	<i>Mandatory</i>
Beschreibung	Der Anbieter konfiguriert das System vor der Inbetriebnahme vollständig: Er definiert die Eingabewerte, den Prozess der Analyse und die Form der Ausgabe.
Auslösendes Ereignis	Anbieter will System bereitstellen.
Akteure	Anbieter
Vorbedingung	Das System ist gestoppt.
Nachbedingung	Das System ist gestartet.
Ergebnis	Gültige Konfiguration, die sich nach dem Start des Systems durch Ausgabe der aktiven Konfiguration überprüfen lässt.
Hauptszenario	<ol style="list-style-type: none"> 1. Der Anbieter definiert die von ihm gewünschte Konfiguration in einer vom System unterstützten Form. 2. Beim Start des Systems wird die Konfiguration gelesen und angewendet. 3. Nach dem Start des Systems wird die aktive Konfiguration über einen geeigneten Mechanismus ausgegeben und kann mit der Eingabe verglichen werden.
Ausnahmeszenarien	Bei einer fehlerhaften Konfiguration kann das System nicht oder nur teilweise gestartet werden.

UC-02: Eingabe von Beobachtungswerten

Bezeichner	UC-02
Name	Eingabe von Beobachtungswerten
Priorität	<i>Mandatory</i>
Beschreibung	Der Endbenutzer nimmt die Eingabe von Werten der Erhebung vor.
Auslösendes Ereignis	Der Endbenutzer will an der Erhebung, für die das System konfiguriert worden ist, teilnehmen.
Akteure	Endbenutzer
Vorbedingung	Das System ist so konfiguriert worden, dass die vom Endbenutzer gewünschte Eingabe möglich ist; das System ist gestartet.
Nachbedingung	–
Ergebnis	Erfolgreiche Eingabe und Speicherung der eingegebenen Werte und der damit assoziierten Metadaten, Bestätigung an den Endbenutzer.
Hauptszenario	<ol style="list-style-type: none">1. Der Endbenutzer öffnet die Eingabemaske des Systems.2. Der Endbenutzer nimmt die Eingabe mit Hilfe der konfigurierten Maske vor.3. Die Eingabe wird vom System übernommen.
Ausnahmeszenarien	–

UC-03: Ausgabe der Resultate

Bezeichner	UC-03
Name	Ausgabe der Resultate
Priorität	<i>Mandatory</i>
Beschreibung	Der Endbenutzer lässt sich die im System befindlichen, gemäss Konfiguration analysierten Werte der Erhebung anzeigen.
Auslösendes Ereignis	Der Endbenutzer will das aktuelle Ergebnis der Erhebung anschauen.
Akteure	Endbenutzer
Vorbedingung	Das System ist gestartet.
Nachbedingung	–
Ergebnis	Das vorläufige Ergebnis der Erhebung wird dem Endbenutzer angezeigt.
Hauptszenario	1. Der Endbenutzer öffnet die Ausgabemaske des Systems 2. Das Ergebnis der Erhebung wird dargestellt.
Ausnahmeszenarien	–

UC-04: Erweiterung des Analyseprozesses

Bezeichner	UC-04
Name	Erweiterung des Analyseprozesses
Priorität	<i>Optional</i>
Beschreibung	Der Prozess der Analyse der eingehenden Messwerte kann um zusätzliche Schritte oder Methoden erweitert werden. Diese müssen vor dem Start des Systems in angemessen codierter Form vorliegen.
Auslösendes Ereignis	Anbieter will eingehende Daten mit spezifischen Methoden analysieren.
Akteure	Entwickler
Vorbedingung	Das System ist gestoppt.
Nachbedingung	Das System ist gestartet.
Ergebnis	Das System aktiviert die Erweiterung des Analyseprozesses.
Hauptszenario	<ol style="list-style-type: none">Der Entwickler definiert die Schritte der Analyse in einer vom System unterstützen Form.Beim Start des Systems wird die Definition in den Analyseprozess eingebundenDas System bestätigt die Aktivierung während des Starts.
Ausnahmeszenarien	Bei einer fehlerhaften Definition kann das System nicht oder nur teilweise gestartet werden.

3.4 Anforderungen

Die funktionalen Anforderungen an das angestrebte System leiten sich direkt ab von den Use Cases. Dieser Umstand wird bei den Beschreibungen als *UC-Referenz* markiert. Jede Anforderung wird *gewichtet*, um ihre relative Wichtigkeit für den Erfolg des Systems auszudrücken. Die Gewichtung fließt ein in die spätere Marktanalyse und in die Priorisierung bei der Umsetzung.

3.4.1 Funktionale Anforderungen

FREQ-01: Konfiguration der Eingabewerte

UC-Referenz	UC-01, UC-02
Gewichtung	3
Beschreibung	Das System soll Typen von Eingabewerte verarbeiten, die in der Konfiguration vorgegeben werden.

Jede *Eingabe* einer Erhebung besteht aus einem oder mehreren Werten eines bestimmten Typs (Werte-Tupel). Um eine Erhebung durchführen zu können, muss vorgängig festgelegt sein, welche Typen von Eingabewerten erfasst werden sollen. Diese *Festlegung der Form der Eingabewerte* obliegt dem Anbieter. Eingabewerte sollen quantitativer Natur sein und bezüglich Wertebereich das gesamte im Rechner abbildbare Spektrum der Zahlen umfassen. Die Modellierung der Eingabewerte bestimmt direkt die Funktionalität des grafischen UIs, das dem Endbenutzer präsentiert wird.

Eine in UC-04 beschriebene Erweiterung der Analyse kann zur Folge haben, dass Eingabetypen weitere Eigenschaften benötigen, um den Analyse-Prozess durchlaufen zu können. Die einzelnen Typen werden mit *mindestens* folgenden Eigenschaften konfiguriert:

- Bezeichnung: Name der Variable, der bei Analyse und Ausgabe an den Typ gebunden ist
- Frage: Fragestellung, deren Antwort der Eingabewert darstellt
- Minimum: minimal möglicher Wert der Eingabe
- Maximum: maximal möglicher Wert der Eingabe
- Schrittweite: Anzahl möglicher Eingaben zwischen Minimum und Maximum

Beispiel

Tabelle 1 könnte die Konfiguration der Eingabewerte darstellen, die eine Erhebung über die Schneeverhältnisse in einem bestimmten Gebiet beschreibt. Eine mögliche Eingabe ist das Tupel [depth = 2.1, fresh = 15, snow = 4] bei einer Schneehöhe von 2.1 Metern, 15 Zentimetern Neuschnee und vereister Schneeoberfläche.

Tabelle 1: Beispiel für Konfiguration der Eingabewerte

Bezeichnung	Frage	Minimum	Maximum	Schrittweite
depth	Schneehöhe (in m)	0	15	0.1
fresh	Neuschnee (in cm)	0	200	1
snow	Schneehärte	1	4	1

FREQ-02: Analyse der Eingaben

UC-Referenz	UC-01, UC-03
Gewichtung	3
Beschreibung	Das System soll eingehende Werte gemäss der Konfiguration analysieren und die Resultate für die Ausgabe vorbereiten.

Die Ausgabe der analysierten Daten soll auf einer Karte erfolgen (siehe FREQ-05). Bei der Analyse soll die in einer einzelnen Eingabe enthaltene Information in einen grösseren Zusammenhang gestellt werden. Es liegt deshalb nahe, jede zusätzliche Eingabe mit den Werten zu verrechnen, die in einem *örtlichen Zusammenhang* mit ihr stehen. So entsteht ein *Kartogramm*, das die Ergebnisse der Erhebung den geografischen Bereichen zuweist.

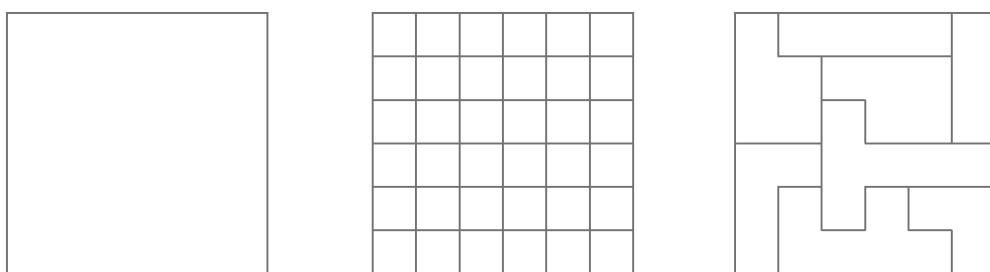


Abbildung 11: Konfiguration von Gebiet, Raster und Segmenten der Analyse

Die Konfiguration der Analyse beinhaltet somit folgende Parameter:

1. Grenze: Geographische Grenzen des durch die Analyse abgedeckten Bereichs
2. Raster: Aufteilung des durch die Analyse abgedeckten Bereichs in disjunkte Teilbereiche gleicher Fläche.
3. Segmente: Zusammenfassung der Teilbereiche zu zusammenhängenden Segmenten, die in einem örtlichen Zusammenhang stehen (siehe Abbildung 11 rechts)
4. Analyse-Methode: Methode, mit der ein eingehendes Werte-Tupel den aktuellen Analysewert für das betroffene Segment verändert (s.u.)
5. Beschreibung: Beschreibung der Aussage der Analyse

Analyse-Methode

Der Anbieter soll aus einer *Liste vordefinierter Analyse-Methoden* wählen können. Aufgabe einer Analyse-Methode ist die Berechnung eines skalaren Wertes, der einem Segment zugewiesen werden kann. Die Visualisierung auf der Karte in FREQ-05 basiert auf diesem Analyse-Wert. Dadurch soll dem Endbenutzer ermöglicht werden, einen visuellen Eindruck vom Resultat der bisher eingegangenen Werte zu bekommen. Diese *Verdichtung* der eingehenden Werte auf einen Wert schliesst natürlich weitergehende Analysen der Rohdaten zu einem späteren Zeitpunkt oder durch ein anderes System nicht aus.

Eine Analyse-Methoden, die sich für die Verarbeitung fortlaufend eintreffender Werte ebenso eignet wie für die Visualisierung auf der Karte, ist beispielsweise das *gewichtete Mittel aller Felder eines Tupels*. Die Gewichtung der Einzelwerte obliegt in diesem Fall dem Anbieter, der sein Fachwissen in dem Thema der Erhebung dafür einsetzt. Auch die *Anzahl und Regelmässigkeit* der eingegangenen Beobachtungswerte können interessant sein. Anschliessend an die Durchführung der Analyse einer einzelnen Eingabe wird der aktualisierte Wert des betroffenen Segments gespeichert und der grafischen Ausgabe gemäss FREQ-05 zugänglich gemacht.

Bei der Analyse darf auch die *zeitliche Dimension* nicht in Vergessenheit geraten. Bestimmte Erhebungen konzentrieren sich auf die jüngere Vergangenheit, so dass die Bedeutung älterer Werte für das Resultat mit der Zeit abnehmen sollte. Es ist das Ziel einer *Erweiterung des Analyseprozesses* (wie in UC-04 beschrieben), dieser Liste weitere Methoden hinzuzufügen. [4] bietet eine Übersicht über interessante Techniken der Analyse von durch Crowdsourcing entstandenen Daten.

Beim Entwurf einer Analyse-Methode ist zu beachten:

- Die Verarbeitung der eingehenden Werte soll in **Echtzeit** erfolgen. Da der Schwerpunkt von «Hyperlike» auf dem Teilen von Beobachtungswerten liegt, sollen neu eingetroffene Werte so schnell wie möglich in das vorläufige Resultat einfließen.

- Die Abfrage bestehender Werte kann unter Umständen sehr aufwändig sein, was die Leistungsfähigkeit des Systems beeinflusst. Für eine performante Analyse ist deshalb der **Lokalität der Daten** Beachtung zu schenken, also möglichst Werte zu verwenden, die schnell und in wenigen Schritten abgerufen werden können. Ein einfaches Beispiel für diesen Aspekt ist das Feststellen der Anzahl aller bisher eingegangenen Werte – dafür muss nur ein einzelner Wert gehalten werden, auf den für jedes eingehende Werte-Tupel nur einmal zugegriffen werden muss.

Beispiel

Tabelle 2 könnte die Konfiguration der Analyse darstellen, die eine Erhebung über die Verhältnisse für Skitouren in verschiedenen Gebieten der Schweiz beschreibt. Die darin beschriebene Analyse lässt sich auf Eingabe-Tupel aus dem Beispiel aus FREQ-01 in Abschnitt 3.4.1 anwenden. Dadurch entsteht ein Überblick über die Verhältnisse für Skitouren in den durch den Anbieter konfigurierten Regionen in den Schweizer Bergen.

Tabelle 2: Beispiel für Konfiguration der Analyse

Grenze	Bereich, der die Landesgrenzen der Schweiz umschliesst
Raster	Überdeckung der Grenze mit Flächen gleicher Grösse
Segmente	Zusammenhängenden Gruppierungen von Elementen des Rasters zu für Skitouren interessanten Gebieten
Analyse-Methode	Gewichtetes Mittel der Werte aller konfigurierten Felder

FREQ-03: Automatische Erfassung der Positionsdaten einer Eingabe

UC-Referenz	UC-02
Gewichtung	1
Beschreibung	Während der Eingabe soll das System die momentane Position des Benutzers automatisch erfassen und übermitteln.

FREQ-04: Grafische Benutzeroberfläche für die Dateneingabe

UC-Referenz	UC-02
Gewichtung	3
Beschreibung	Die Eingabe von Werten soll über eine grafische Benutzeroberfläche im Web-Browser erfolgen können, deren UI-Elemente von den in der Konfiguration definierten Typen von Eingabewerten gemäss FREQ-02 abgeleitet werden.

FREQ-05: Grafische Darstellung der Analyse-Resultate auf einer Karte

UC-Referenz	UC-03
Gewichtung	3
Beschreibung	Das System soll dem Benutzer das aktuelle Resultat der Analyse gemäss FREQ-02 als Felderkartogramm im Web-Browser grafisch darstellen.

Durch Kolorierung der über die Karte gelegten Segmente gemäss der in FREQ-02 konfigurierte Analyse-Methode entsteht ein Kartogramm, welches das Gebiet mit dem zu ihm gehörenden Analyse-Resultat verbindet (gelbe Färbung in Abbildung 12). Dadurch kann der Benutzer auf der Karte schnell Gebiete erkennen, die für ihn interessant sind. Zudem soll definierbar sein, unter welchen Bedingungen ein Segment überhaupt eingefärbt wird (und damit für den Benutzer sichtbar).

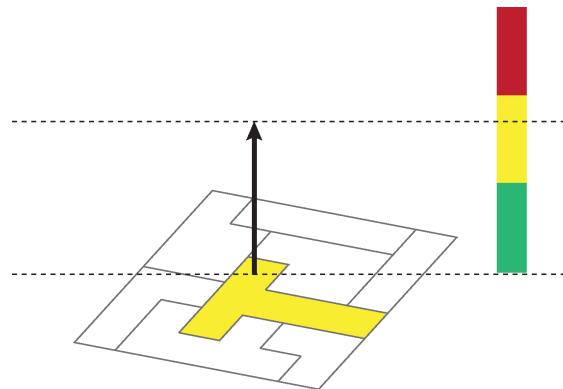


Abbildung 12: Methode zur Bestimmung der Färbung eines Segments des Kartogramms

Um die gewünschte Färbung zu erreichen, muss die Anzeige entsprechend konfiguriert werden. Die Konfiguration muss folgende Felder definieren:

- Färbe-Methode: Eine Liste von Kolorierungen, die einem Wertebereich des Resultats entspricht
- Quorum: Minimale Anzahl Eingabewerte, um ein Segment bei der Ausgabe zu berücksichtigen

So wie Tabelle 2 könnte die Konfiguration der Anzeige aus dem Beispiel in FREQ-02 aussehen. Um das Resultat für ein Segment sichtbar zu machen, sind 50 Eingabewerte nötig. Je ein Drittel der Segmente wird grün, braun oder weiss eingefärbt.

Tabelle 3: Beispiel für Konfiguration der Anzeige

Färbung	Grüne, braune oder weisse Einfärbung der Segmente gemäss Aufteilung aller der Resultate auf drei gleich grosse Wertebereiche
Quorum	50

FREQ-06: Grafische Web-Benutzeroberfläche ist anpassbar

UC-Referenz	UC-01
Gewichtung	2
Beschreibung	Das Aussehen der Benutzeroberflächen für Ein- und Ausgabe soll durch den Anbieter anpassbar sein.

3.4.2 Nicht-funktionale Anforderungen

NFREQ-01: Einfaches Deployment

Gewichtung	2
Beschreibung	Das System soll so angelegt sein, dass die Zahl der Bestandteile klein und deren Installation auch für <i>wenig Geübte</i> leicht möglich ist.

NFREQ-02: Skalierbarkeit

Gewichtung	1
Beschreibung	Das System soll so angelegt sein, dass der Analyse-Prozess nebenläufig erfolgen kann und somit eine horizontale Skalierbarkeit unterstützt.

NFREQ-03: Erweiterbarkeit

UC-Referenz	UC-04
Gewichtung	2
Beschreibung	Das System soll so angelegt sein, dass der Analyseprozess erweiterbar ist. So soll es möglich sein, zusätzliche Analyse-Methoden zu implementieren, die in diese Anforderungsanalyse noch nicht eingeflossen sind.

NFREQ-04: Open Source

Gewichtung	1
Beschreibung	Das System wird unter einer Open-Source-Lizenz publiziert und soll bei der Umsetzung ebenfalls ausschliesslich auf freier Software aufbauen.

3.5 Akzeptanzkriterien

Dieser Abschnitt enthält Abläufe, deren vollständige erfolgreiche Absolvierung Voraussetzung ist, um das System als funktionsfähig zu bezeichnen. Sie werden eingesetzt, um eine Implementierung auf Erfüllung der funktionalen Anforderungen zu überprüfen. Die beschriebenen Abläufe werden durch eine Testperson durchgeführt und unterscheiden sich so von den *Unit Tests* und *Integration Tests*, die spezifisch für eine Implementation sind und nur mechanische Überprüfung auf Programmier-Fehler zulassen.

Die einzelnen Kriterien bestehen aus zweierlei Anweisungen an die testende Person. *Nummerierte* Einträge sind *Handlungsanweisungen*, die wie beschrieben ausgeführt werden sollen. *Nicht-nummerierte* Einträge sind *Erwartungen*, denen das System gerecht werden muss, um das Kriterium zu erfüllen. Die Beschreibungen der Anweisungen sind allgemein gehalten, die detaillierte Durchführung muss aus der Dokumentation der Implementierung hervorgehen.

3.5.1 AT-01: System konfigurieren

UC-Referenz	UC-01
REQ-Referenz	FREQ-01
Vorbedingungen	<ul style="list-style-type: none">– Das System ist gestoppt– Es sind noch keine Messwerte erfasst worden.

Testablauf und erwartete Resultate

1. Konfiguration gemäss Dokumentation der Implementation vornehmen.
2. System starten.
3. Mit dem Web-Browser die URL der Konfigurations-Ressource ansteuern.
 - Die aktuelle Konfiguration wird geladen.
 - Die Einträge in der angezeigten Tabelle müssen mit den Angaben in der Konfigurationsdatei übereinstimmen.
4. Mit dem Web-Browser die URL der Erhebung ansteuern
 - Die Eingabemaske für eine neue Eingabe wird angezeigt.

3.5.2 AT-02: Anpassung der Benutzeroberfläche

UC-Referenz	UC-01
REQ-Referenz	FREQ-06
Vorbedingungen	AT-01 wurde durchgeführt.

Testablauf und erwartete Resultate

1. Die Anpassungen am Aussehen der Benutzeroberfläche gemäss Dokumentation der Implementation vornehmen.
2. Mit dem Web-Browser die URL der Applikation ansteuern.
 - Das Aussehen der Benutzeroberfläche muss die vorgenommenen Anpassungen widerspiegeln

3.5.3 AT-03: Erfassung der Positionsdaten einer Eingabe

UC-Referenz	UC-02
REQ-Referenz	FREQ-03
Vorbedingungen	AT-01 wurde durchgeführt.

Testablauf und erwartete Resultate

1. Mit dem Web-Browser eines Mobilgeräts die URL der Applikation ansteuern.
 - Die konfigurierte graphische Oberfläche für die Eingabe wird angezeigt.
2. Die Koordinaten der aktuellen Position werden ausgelesen.
 - Die aktuelle Position wird auf einer Karte angezeigt.

3.5.4 AT-04: Dateneingabe mittels grafischer Benutzeroberfläche

UC-Referenz	UC-02
REQ-Referenz	FREQ-04
Vorbedingungen	AT-03 wurde durchgeführt, die grafische Benutzeroberfläche wird angezeigt.

Testablauf und erwartete Resultate

1. Für jeden der konfigurierten Typen erscheint unterhalb der aktuellen Position ein entsprechendes UI-Element:
 - Der *Titel* des UI-Elements entspricht der konfigurierten *Frage*.
 - Die Eingabemaske des UI-Elements ist so gestaltet, dass sie die Konfiguration wider spiegelt bezüglich *Minimum*, *Maximum* und *Schrittweite*.
2. Die Eingabemasken der einzelnen Typen werden so bedient, dass sie den gewünschten Beobachtungswert abbilden.
3. Durch Bedienung des UI-Elements «Abschicken» wird die Übermittlung ausgelöst.
 - Der Messwert wird übermittelt und verarbeitet.
 - Die Maske für die Datenausgabe wird angezeigt (siehe AT-05).

3.5.5 AT-05: Ausgabe einer Resultatübersicht mittels grafischer Benutzeroberfläche

AT-04	Ausgabe einer Resultatübersicht mittels grafischer Benutzeroberfläche
UC-Referenz	UC-03
REQ-Referenz	FREQ-05
Vorbedingungen	AT-04 wurde durchgeführt

Testablauf und erwartete Resultate

1. Mit dem Web-Browser die Resultate-URL der Applikation ansteuern.
 - Ein Kartenausschnitt mit dem für die Erhebung konfigurierten geographischen Bereich wird angezeigt.
 - Der Kartenausschnitt ist so eingefärbt, dass er die analysierten Werte sichtbar macht.

4 Marktanalyse

Ziel der Marktanalyse ist, bestehende Software-Lösungen zu untersuchen, die die Funktionalität des zu erstellenden Systems unterstützen oder gar vollständig bzw. teilweise abdecken. Es wird Aufgabe des Konzepts in Abschnitt 5 sein, aus diesem Fundus ein System zusammenzustellen, das die in Abschnitt 3.4 formulierten Anforderungen erfüllt.

Im Fokus dieses Abschnitts stehen drei Kategorien von Software-Produkten. An erster Stelle befinden andere *Plattformen*, die die gleiche oder zumindest eine sehr ähnliche Absicht verfolgen wie die vorliegende Arbeit. Ihnen ist gemein, dass sie sich als *Komplettlösungen* verstehen, die alle Schritte in Verarbeitungskette der Daten einer Erhebung abdecken. Als *Teillösungen* hingegen sind die Kategorien *Dienste* und *Komponenten* zu verstehen: Sie umfassen externe bzw. interne Systeme, die einen Teil der Funktionalität erbringen und durch *Schnittstellen* eingebunden werden. Falls eine Plattform nicht alle Anforderungen erfüllen kann, jedoch geeignete Schnittstellen anbietet, kann sie (oder ein Teil von ihr) als Dienst in Betracht gezogen werden.

Tabelle 4: Kategorien der Marktanalyse

Kategorie	Beschreibung
<i>Plattformen</i>	Applikationen, die die gesamte oder eine ähnliche Funktionalität wie «Hyperlike» abdecken
<i>Dienste</i>	Externe Systeme, die durch Drittanbieter zur Verfügung gestellt werden und einen Teil der Funktionalität abdecken
<i>Komponenten</i>	Interne Systeme, die durch den <i>Betreiber</i> selbst zur Verfügung gestellt werden und einen Teil der Funktionalität abdecken

Dieser Abschnitt ist nach den genannten Kategorien gegliedert und widerspiegelt so den in Abschnitt 3.2.1 beschriebenen Systemkontext: Plattformen umschließen das Gesamtsystem, Dienste bilden die externen Software-Dienste, und Komponenten befinden sich innerhalb der Systemgrenze.

Die Marktanalyse erhebt keinen Anspruch auf Vollständigkeit. Vielmehr soll sie einen Überblick verschaffen über das technologische Umfeld, in dem diese Arbeit entsteht. Das Gebiet der (mobilen) Datensammlung ist sehr gross, mit zahlreichen Anbietern, die ähnliche Probleme mit unterschiedlichen Methoden und Schwerpunkten angehen. Es wird deshalb in der Kategorie *Plattformen* nur auf die wichtigsten und vielversprechendsten Vertreter detailliert eingegangen. Die Auswahl potenzieller *Dienste* und *Komponenten* beschränkt sich auf solche, die das angestrebte System in einem oder in mehreren der drei Verarbeitungsschritte für Messwerte – Eingabe, Analyse, Ausgabe – zu unterstützen vermag.

Eine detaillierte Betrachtung eines Produkts, bzw. eines Repräsentanten einer Kategorie, besteht aus einführenden Bemerkungen über seine Herkunft und seinen Kontext. Darauf folgt eine Erläuterung seiner Funktionsweise und eine Einschätzung seiner Verwendbarkeit für die vorliegende Arbeit. Die Beschreibungen der untersuchten Software-Produkte dienen der *Illustration der Funktionsweise*, während die Tabellen im *Ergebnis* (Abschnitt 4.4) eine systematische Untersuchung der Kompatibilität mit den Anforderung liefern. Die Marktanalyse mündet in ein *Fazit*, welches als Grundlage dient für das spätere Konzept.

Da Plattformen als Komplettlösungen antreten, müssen sie zwingend alle funktionalen Anforderungen an das Zielsystem erfüllen – nur so kann sichergestellt werden, dass die gewünschte Funktionalität erbracht werden kann. *Damit sind für Plattformen alle funktionalen Anforderungen K.O.-Kriterien*. Aus diesem Grund sind in der Übersichts-Darstellung in Tabelle 5 funktionale Anforderungen nur entweder als erfüllt (grün) oder nicht erfüllt (rot) eingetragen: Eine rote Markierung genügt, um die Plattform als taugliche Implementierung des Zielsystems auszuschliessen. Die Nichterfüllung einer nichtfunktionalen Anforderung wird gelb markiert und führt nicht zum sofortigen Ausschluss des Produkts.

Bei der Untersuchung von Diensten und Komponenten wird eine *Nutzwertanalyse* durchgeführt. Diese verbindet die *Gewichtung* einer Anforderung mit der Bewertung, wie gut ein untersuchtes Produkt diese zu erfüllen vermag (siehe Tabelle 7). So wird sichergestellt, dass bei der Umsetzung Teilsysteme zum Einsatz kommen, die mit dem *geringsten Zusatzaufwand* die Erfüllung der Anforderungen ermöglichen.

4.1 Plattformen

Es gibt ein paar Applikationen, die das Problem der mobilen Datensammlung lösen wollen. [14] bietet eine gute (wenn auch ziemlich veraltete) Übersicht mit detaillierten Anleitungen für die wichtigsten Produkte. Die Lösungen in dieser Kategorie treten als Komplettlösungen an, die in sich geschlossen installiert und betrieben werden wollen. Als solche werden sie auch auf Verträglichkeit mit den Anforderungen geprüft.

4.1.1 OpenDataKit

Die Idee von **OpenDataKit** ODK kommt derjenigen, die dieser Arbeit zugrunde liegt wohl am nächsten. Nach eigenen Angaben handelt es sich dabei um ein *_free and open-source set of tools which help organizations author, field, and manage mobile data collection solutions_*⁹. Das Produkt besteht aus verschiedenen Komponenten, die für verschiedene Aspekte der Datensammlung zuständig sind. Um deren Zusammenwirken zu veranschaulichen, ist es hilfreich, die einzelnen Verarbeitungsschritte im Geiste zu durchlaufen:

⁹ <https://opendatakit.org/about/>

1. Als erster Schritt muss das *Formular* der Erhebung entworfen werden. Dieses besteht aus einer XML-Datei, die mit verschiedenen Werkzeugen erzeugt werden kann: durch Web-Applikation ODK Build¹⁰ oder durch eine nach XLSForm-Spezifikation¹¹ gestaltete Excel-Datei. Der Schwerpunkt bei der Gestaltung liegt auf der Erstellung eines «Wizard»-ähnlichen Formulars, das Schritt für Schritt bearbeitet wird.
2. Als Server-Applikation kommt «ODK Aggregate» zum Einsatz, eines in Java geschriebenen Dienstes, der für Speicherung und Ausgabe der Messwerte zuständig ist. «ODK Aggregate» lässt sich herunterladen und installieren, ist aber auch auf dem Google App Engine lauffähig. Als Alternative zu einer eigenen Instanz von «ODK Aggregate» kann das Portal «Formhub» als Backend benutzt werden (mehr dazu in Abschnitt 4.2.1).
3. Als Client wird entweder eine native Android-Applikation namens «ODK Collect» verwendet oder eine für mobile Web-Browser optimierte Formular-Applikation namens «Enketo Smart Paper».
4. Die eingehenden Daten lassen sich innerhalb von «ODK Aggregate» auf einer Karte oder als Grafiken visualisieren bzw. in verschiedene Formate exportieren. Zudem kann man alle Werte auf «Google Fusion Tables» publizieren und so über «Google Maps» der Öffentlichkeit zugänglich machen. Letztere Funktionalität bietet auch die Option, eingehende Daten laufend zu aktualisieren, also neu eingehende Eingabewerte zu «streamen».

Einen detailliereren Einblick in die Funktionsweise und Bedienung von OpenDataKit geben die «Open Data Kit Tutorials» von Google Earth Outreach¹².

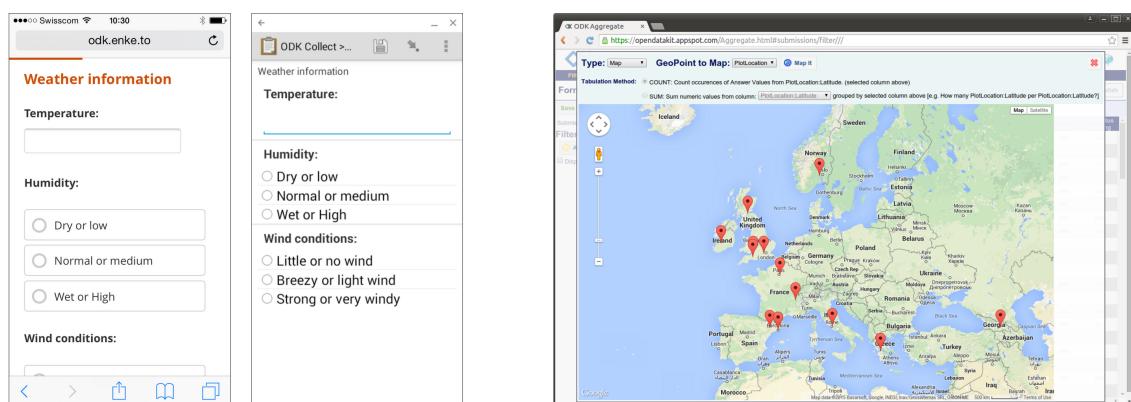


Abbildung 13: «ODK Collect» für Web bzw. Android und «ODK Aggregate»

¹⁰ <http://build.opendatakit.org>

¹¹ <http://xlsform.org/>

¹² <http://www.google.com/earth/outreach/tutorials/#odk>

Insgesamt hinterlässt «OpenDataKit» einen gemischten Eindruck. Zwar verfügt die Plattform über viele der gewünschten Features von «Hyperlike», dennoch vermag sie im ganzen nicht zu überzeugen. Auch hier ist vieles umständlich, unfertig und veraltet: komplizierte Bedienung von Formularen, verwaiste Repositories und instabile Implementierungen. Zudem bietet auch OpenDataKit keine eingebaute Möglichkeit, die eingehenden Messwerte laufend zu analysieren oder zusammenzufassen – eine solche müsste bei Verwendung von «ODK Aggregate» als Backend auf den in Google Fusion Tables abgelegten Daten erfolgen. Die Spezifikation von Formularen mittels Excel bzw. «ODK Builder» funktioniert hingegen recht gut. Leider lässt sich die Benutzeroberfläche nicht anpassen, so dass eine Integration in andere Systeme schwierig ist.

Obwohl die Plattform die meisten Anforderungen zu erfüllen scheint, leidet die Benutzerfreundlichkeit des Gesamtsystems arg unter den mangelhaften Konfigurations- und Bedienungskonzepten. Die monolithische Architektur dieser Gesamtlösung lässt zudem Fragen der Skalierbarkeit offen. Die fehlende automatische Erfassung des Standort eines Benutzers bei dessen Eingabe zeigt, dass OpenDataKit für die mobile Datensammlung in einem zu allgemeinen Sinne angelegt ist, um die Funktionen von «Hyperlike» zu implementieren. Das auf die Sammlung von Geodaten spezialisierte, auf OpenDataKit aufbauende Projekt **GeoODK**¹³ wiederum ist zu stark spezialisiert und auf das Zusammenspiel mit Geoinformationssystemen ausgerichtet. Ob einzelne Teile der OpenDataKit-Plattform als Dienste oder Komponenten verwendet werden können, werden die folgenden Abschnitte zeigen.

¹³ <http://geoodk.com/>

4.1.2 Ushahidi

Ushahidi Platform¹⁴ ist ein Open-Source-Projekt¹⁵ aus Kenia, das auf den ersten Blick ein sehr ähnliches Ziel verfolgt wie diese Arbeit: Definieren von Fragebögen, Sammeln von Daten über mobile Kanäle sowie deren Visualisierung auf einer Karte. Die Organisation hinter Ushahidi entstand nach Ausschreitungen bei Wahlen in 2008 und hat sich dem «Bürger-Journalismus» verschrieben.

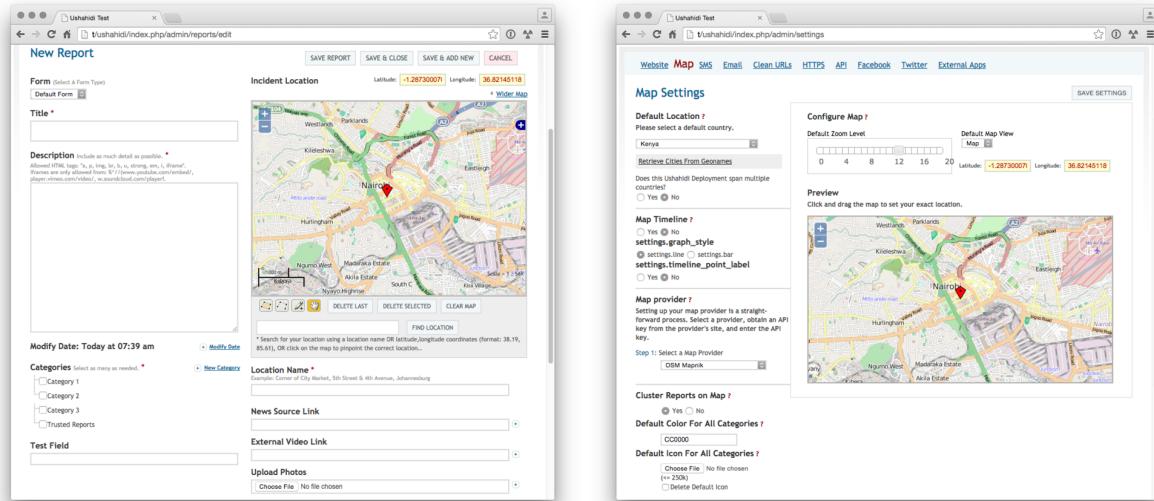


Abbildung 14: Web-Client für Ushahidi: Eingabe (links) und Konfiguration (rechts)

Der aktuelle Version 2 von Ushahidi ist eine Web-Applikation, die serverseitig auf PHP und MySQL aufbaut. Als klassische LAMP-Applikation lässt sie sich durch den geübten Benutzer leicht installieren und starten. Für die Definition einer Erhebung steht ein Web-Interface zur Verfügung, das etwas veraltet scheint. Zwar stehen verschiedene Typen von Eingabewerten zur Auswahl, doch lassen sich diese nur sehr rudimentär parametrisieren: Zwar lassen sich numerische Werte von Text unterscheiden, doch damit erschöpft sich die Konfigurierbarkeit bereits.

¹⁴ <http://www.ushahidi.com/product/ushahidi/>

¹⁵ <https://github.com/ushahidi/platform>

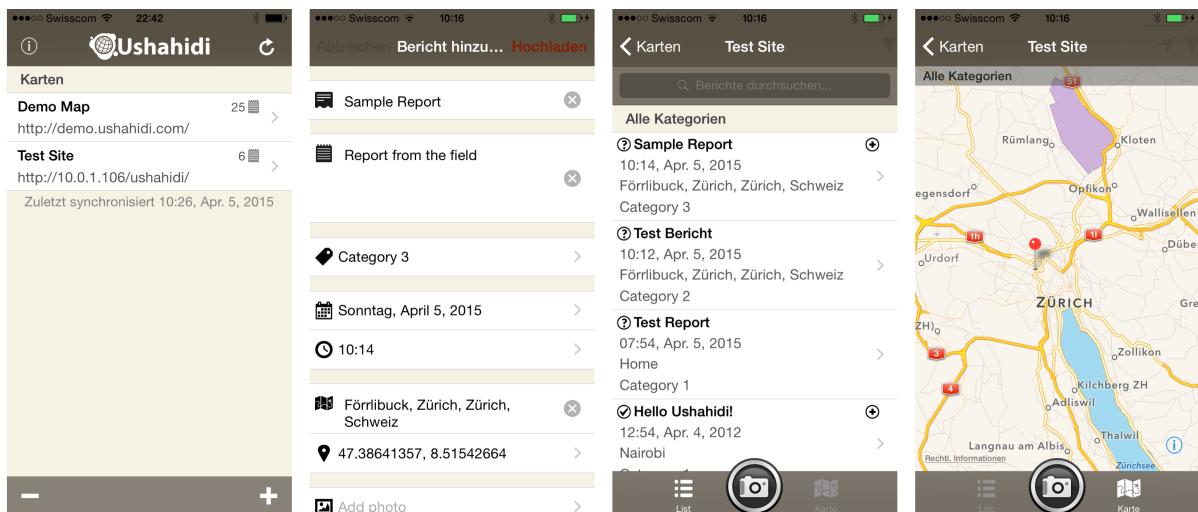


Abbildung 15: Mobiler Ushahidi-Client für iOS

Als Clients kommen hauptsächlich *native Clients* für iOS und Android zum Einsatz. Das *Web-Formular* ist nicht für mobile Browser optimiert und ist auf diesen Geräten schwierig zu bedienen. Die nativen Clients werden über den jeweiligen App-Store als Clients der *Plattform* ausgeliefert (siehe Abbildung 15). Über eine anpassbare Web-Benutzeroberfläche verfügt Ushahidi nicht.

«Ushahidi Plattform» bietet keine Analyse der Eingabewerte: Die Daten werden lediglich erfasst, gespeichert und angezeigt. Dadurch eignet sich das Produkt zwar für das Crowdsourcing der Datensammlung – als Hilfsmittel für darauf basierende Entscheidungen erweist es sich jedoch als wenig nützlich. Um Analyseschritte zur Berechnung und Gruppierung von Resultaten einführen zu können, müsste der Kern der Applikation modifiziert werden. Daher eignet sich Ushahidi auch nicht dafür, als Komponente von «Hyperlike» eingesetzt zu werden, die einen Teil der Anforderungen abdeckt.

Version 2 von Ushahidi hinterlässt einen angestaubten Eindruck: Installation, Konfiguration und Bedienung wirken veraltet und *buggy*. Dies verwundert nicht weiter, schliesslich läuft schon seit geraumer Zeit die Entwicklung von Version 3 der Plattform. Diese trägt sogar bereits die Bezeichnung «Beta», doch scheint die Entwicklung ins Stocken geraten zu sein. Vom Einsatz für neue Projekte wird im Wiki der Plattform ausdrücklich abgeraten¹⁶.

Die Organisation hinter Ushahidi betreibt auch das Portal **Crowdmap**¹⁷, auf dem registrierte Benutzer eine Karte anlegen, die dann für andere Benutzer zur Kommentierung zur Verfügung steht – ähnlich den Profilen in sozialen Netzwerken. Allerdings lassen sich die Eingabewerte der Crowdmap-Karten nicht festlegen; eine Eingabe besteht lediglich aus der Position des Benutzers sowie einem Textfeld versehen mit der Aufforderung «Write something...». Das Portal

¹⁶ <https://wiki.ushahidi.com/display/WIKI/Ushahidi+Platform+v3.x>

¹⁷ <http://www.crowdmap.com>

reagiert zudem langsam und ist nur bedingt funktionstüchtig; die Abwesenheit von Benutzern und Karten kommt deshalb wohl nicht von ungefähr.

4.1.3 EpiCollect

EpiCollect¹⁸ ist eine recht prominente Applikation für die mobile Datensammlung. EpiCollect entstammt den Bedürfnissen der Feldforschung der Epidemiologie, die Funktionsweise wird in [5] wie folgt beschrieben:

[...] a generic framework, consisting of mobile phone software, EpiCollect, and a web application [...]. Data collected by multiple field workers can be submitted by phone, together with GPS data, to a common web database and can be displayed and analysed, along with previously collected data, using Google Maps (or Google Earth).

Der Link zur Website von EpiCollect ist das erste Resultat einer Google-Suche nach «Mobile Data Collection», die Funktionsweise wird in [14] ausführlich beschrieben. Die ursprüngliche Plattform ist mittlerweile durch ihren Nachfolger «EpiCollectPlus» abgelöst worden. Dieser ist neu geschrieben worden und verfügt über einen erweiterten Funktionsumfang.

Die Installation und der Setup der LAMP-Applikation ist ohne Schwierigkeit. Auch die Konfiguration der Eingabewerte durch den Formular-Editor funktioniert. Allerdings liegt der Schwerpunkt von EpiCollectPlus eindeutig auf der Sammlung von Daten: Die passene Android-App «EpiCollectPlus» erlaubt eine schrittweise Bearbeitung des auf dem Server definierten Formulars.

Eine Web-Benutzeroberfläche für die Dateneingabe ist nicht auffindbar. Die im obigen Zitat beschriebenen Analyse-Funktionen beschränken sich auf die Visualisierung von Verteilungen auf Charts und von Clustern auf einer Karte – alles nur im Backend: EpiCollectPlus bietet keine direkte Möglichkeit zur *Veröffentlichung* der Analyse-Resultate.

¹⁸ <http://www.epicollect.net>

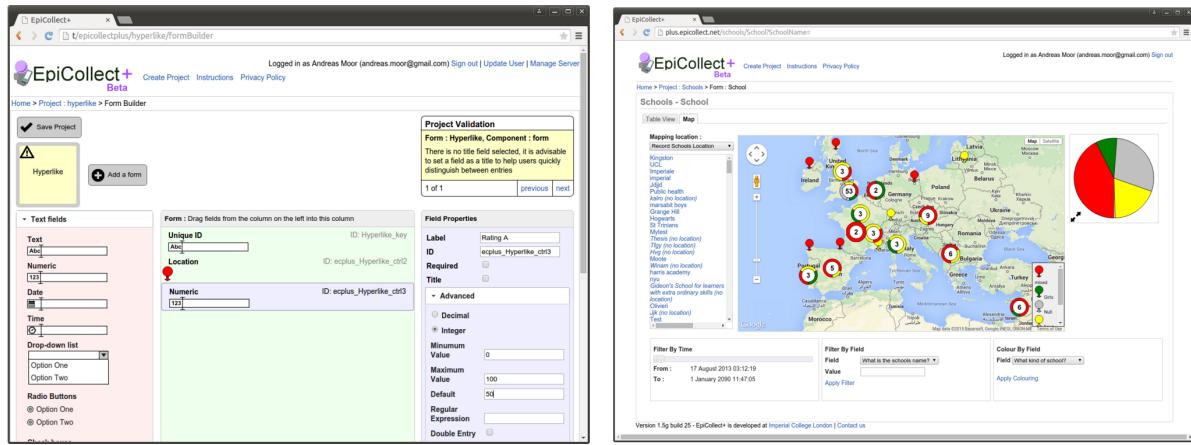


Abbildung 16: Formular-BUILDER und Administrations-Oberfläche von EpiCollectPlus

Es scheint fast so, als sei das Projekt von seinen Urhebern aufgegeben worden. Die Bedienung der Plattform über das Web ist stark veraltet, das Projekt wird auf Github¹⁹ kaum mehr gepflegt. An den eingesetzten Technologien nagt der Zahn der Zeit.

4.1.4 Weitere Plattformen

Die in diesem Teil einzeln behandelten Produkte sind nur einige Teilnehmer neben vielen anderen in diesem Markt. **ArcGIS** ist die führende kommerzielle Plattform für Erfassung und Analyse von geo-lokalisierten Daten. «ArcGIS brings together maps, apps, data, and people to make smarter decisions and enable innovation in your organization or community», heisst es auf der Homepage von ArcGIS²⁰. Der Schwerpunkt liegt hier eindeutig auf dem *Enterprise*-Umfeld. Zwar unternimmt das Unternehmen auch Anstrengungen in Richtung «OpenData», doch ist die Ausrichtung auf eine integrierte Lösung für grosse Unternehmen klar erkennbar. ArcGIS bietet umfangreiche Werkzeuge zur Erfassung, Speicherung und Analyse geolokalizierter Daten – sie alle sind jedoch den *professionellen Einsatz durch Aussendienstmitarbeiter* ausgelegt.

Andere kommerzielle Anbieter, wie z.B. **Fulcrum**, **GIS Cloud**, **Form.com** oder **Poimapper**, haben ähnliche Angebote – ihnen ist gemein, dass sie *kommerziell* sind und auf *Unternehmen mit komplexen Prozessen und Formularen* ausgerichtet sind. Da sie sich dadurch als Implementierung oder Komponente von «Hyperlike» (deren Fokus auf *Crowdsourcing* und *Data Sharing* liegt) nicht eignen, werden sie an dieser Stelle nicht weiter berücksichtigt.

Neben den genannten gibt es insbesondere aus dem wissenschaftlichen Bereich einige Angebote (z.B. **GISKey Field Assets**, **ArcPad** oder **Geopaparazzi**), die in eine ähnliche Richtung gehen, aufgrund ihrer akademischen Ausrichtung und oft unfertigen Umsetzung für das vorliegende Projekt nicht in Frage kommen.

¹⁹ <https://github.com/ImperialCollegeLondon/EpiCollectplus>

²⁰ <http://www.esri.com/software/arcgis>

4.2 Dienste

Die funktionalen Anforderungen an «Hyperlike» lassen sich grob in drei Gruppen aufteilen: FREQ-03 und FREQ-04 befassen sich mit der *Erfassung* von Werten, FREQ-02 betrifft die *Analyse* der so entstandenen Rohdaten, während FREQ-05 die *Ausgabe* der durch Analyse gewonnenen Resultate abdeckt. Die in diesem und dem folgenden Abschnitt betrachteten *Dienste* und *Komponenten* treten, anders als die im vorherigen Abschnitt betrachteten *Plattformen*, nicht als Komplettlösungen für alle Anforderungen an, sondern beschränken sich darauf, einen Teil davon abzudecken. Aus diesem Grund werden sie nur in den Bereichen auf Kompatibilität mit den Anforderungen überprüft, in denen sie eine Funktionalität für diese anbieten.

4.2.1 Formular-Dienste

Den Aspekt der Eingabe von Werten einer Erhebung übernimmt das *Formular*, so wie es in FREQ-04 verlangt wird. Die Erstellung eines Formulars gehört zur Konfiguration des Systems. Für die Bearbeitung und Benutzung gibt es eine Reihe von Online-Diensten wie etwa **Google Forms**²¹, **Formstack**²² oder **Wufoo**²³ – sie alle sind für sehr allgemeine Erhebungen mit beliebigen Fragen und komplexer Logik ausgelegt und für die Zwecke von «Hyperlike» ungeeignet.

Vielversprechender sind Dienste aus dem Umfeld von OpenRosa²⁴. Dabei handelt es sich um standardisiertes Open-Source-Format zur Beschreibung von Formularen, bzw. zur Übermittlung von damit erfassten Daten. OpenDataKit nutzt die OpenRosa-Spezifikation, um die Schnittstellen des Backends «ODK Aggregate» zu betreiben. Wie in Abschnitt 4.1.1 beschrieben, können Formulare in Excel oder «ODK Build» bearbeitet und danach zu XML konvertiert und in einer kompatiblen Applikation verwendet werden.

Formhub ist, neben «ODK Aggregate», der wohl prominenteste Vertreter im OpenRosa-Ökosystem. Formhub ist ein Backend-Dienst, der es Benutzern erlaubt, Formulare zu definieren und zu veröffentlichen. Wie die Bezeichnung «Hub» im Namen andeutet, verfolgt Formhub eine ähnliche Idee wie Github, das Portal, auf dem Programmierer die Quelltexte von Software veröffentlichen können. Auf Formhub können individuelle Benutzer ihre Formulare publizieren und anderen zugänglich machen.

Formhub lässt sich als Backend für OpenDataKit einsetzen und unterstützt neben diversen nativen Clients ebenfalls die Browser-Applikation «Enketo». Formhub ist die Basis für andere, ähnliche Formular-Dienste wie **Ona.io**²⁵ oder **KoBoToolbox**²⁶. Der Dienst bietet allerdings keine Funktion für die Analyse eingehender Resultate (und damit auch nicht für deren Darstellung). Zudem ist der Dienst unzuverlässig – die Website formhub.org ist sehr oft überlastet

²¹ <https://www.google.com/forms/about/>

²² <https://www.formstack.com/>

²³ <http://www.wufoo.com/>

²⁴ <http://www.dimagi.com/collaborate/openrosa/>

²⁵ <https://ona.io>

²⁶ <http://www.kobotoolbox.org/>

und nicht zu erreichen. Inwieweit sich Formhub in Form einer lokalen Instanz als *Komponente* betreiben lässt, kommt in Abschnitt 4.3.1 zur Sprache.

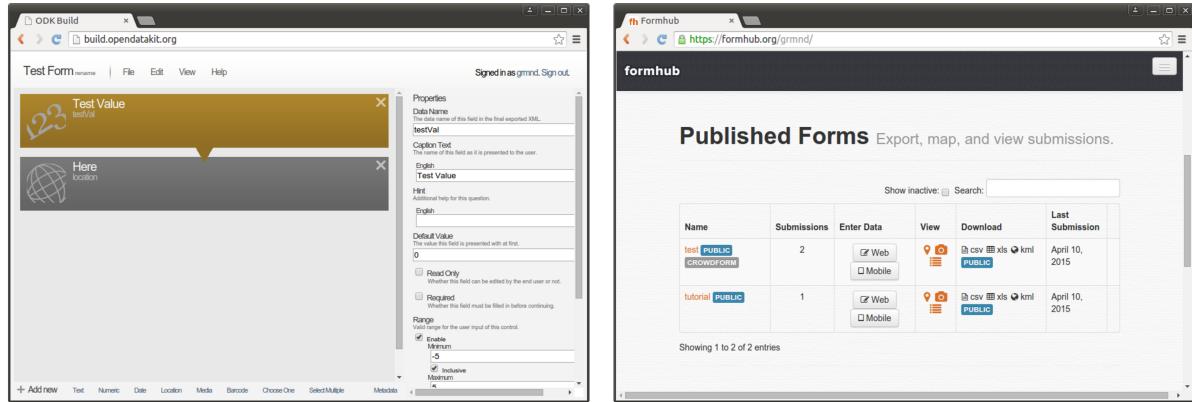


Abbildung 17: ODK Build (links) und Formhub-Backend (rechts)

Trotz der Vielfalt des Angebots lässt sich kein Produkt der Kategorie Formular-Dienste finden, das für die Umsetzung von Hyperlike interessant wäre. Die Kandidaten sind entweder zu weit von den Anforderungen entfernt oder zu unzuverlässig, um in Betracht gezogen zu werden.

4.2.2 Backend-as-a-Service

Der Begriff «Backend-as-a-Service» (BaaS) beschreibt eine Familie von Diensten, die Software-Anbietern die Infrastruktur («Backend») für ihre Applikationen als Dienst anbieten. Sie übernehmen Bereiche wie etwa Datenspeicherung, Datenverarbeitung, Benutzer-Notifikation oder Sicherheit und machen sie über SDKs einzubindende und leicht zu bediendene APIs zugänglich. Dadurch entfällt für den Software-Anbieter viel Entwicklungs-, Administrations- und Unterhalts-Aufwand. Die grosse Beliebtheit von *Apps* für Mobilgeräte sorgt für einen wachsenden BaaS-Markt. Der zu Facebook gehörende Dienst **Parse**²⁷ gehört wohl zu den bekanntesten. Weitere Anbieter sind **Appcelerator**, **Firebase** oder **Backendless**.

Parse bietet komfortable APIs für Datenspeicherung, Datenverarbeitung und Nutzungsanalyse. Für verschiedene Client-Plattformen stehen SDKs bereit, etwa für REST, JavaScript, Android sowie iOS. Die APIs sind jedoch alle spezifisch für Parse und sind ausschliesslich als Cloud-Dienst erhältlich. Als Backend-Service bietet Parse keine Funktionalität für Benutzeroberflächen.

²⁷ <http://parse.com>

The screenshot shows the Parse.com administration interface for a collection named 'TestObject'. The interface includes a sidebar with options like Data, Cloud Code, Webhooks, Jobs, Logs, and Config. The main area displays a table with columns: objectId, test, createdAt, updatedAt, ACL, and foo. There are 1001 rows listed, with the first few entries being 29maN3N3XN, Vyt9nyB50a, S2IYK7PJ7u, oRdv1MT9f5, IpuXX9W8NM, q2lC0V0Mxb6, BwArFyDs19, YnIHLZGNxm, O2n0nAzruqi, YoUsJD9kBV, KhN9g1PrvW, LncKZoZQTY, Dx1kJnbCZB, LBBxJBPyuV, 97FfmjMfHA, N0ts88JRV9, Pciw3YDNH9, MmwvtLvcS6, lEyhZ6V3Ab, and IRbac2vARs. Each entry shows its creation and update dates, the ACL (Public Read and Write), and the value of the 'foo' field (all undefined). Navigation controls at the bottom indicate 1 - 20 of 1001 rows.

Abbildung 18: Administrations-Oberfläche von Parse

Die Bedienung von Parse erfolgt einerseits über eine Web-Oberfläche für Einstellungen und Administration (Abbildung 18), anderseits über ein Kommandozeilen-Programm für das Deployment der Applikation. Parse bietet viele Funktionen, die über die Anforderungen von «Hyperlike» weit hinaus gehen.

4.2.3 Karten-Dienste

Dienste für die Darstellung und Anreicherung von Karten haben in den letzten Jahren an Beliebtheit gewonnen und sind aus dem Alltag des Internauten kaum mehr wegzudenken. Ob **Google Maps**, **OpenStreetMap**, **Bing Maps** oder **Here Maps** – alle sind in ihren Grundfunktionen sehr ähnlich: Die Karten können im Web-Browser oder in einer App dargestellt werden, die eingezeichneten Elemente sind detailliert beschriftet, und der sichtbare Ausschnitt lässt sich vom User einstellen. Das Tool «Map Compare»²⁸ bietet einen anschaulichen Vergleich der wichtigsten Karten-Dienste, einen guten Überblick liefert auch [7] (S. 44ff.). Neben diesen kostenlosen Angeboten gibt es kommerzielle Karten-Dienste wie **MapBox**, **CartoDB** oder **ArcGIS** mit zum Teil umfangreichen Visualisierungsoptionen, ausgefeilten Schnittstellen und eigener Datenhaltung. Allen Angeboten ist zudem gemein, dass sie unter Verwendung der API bzw. durch den Einsatz geeigneter Bibliotheken (wie z.B. Leaflet.js) fast beliebig anreichern lassen.

²⁸ <http://tools.geofabrik.de/mc>

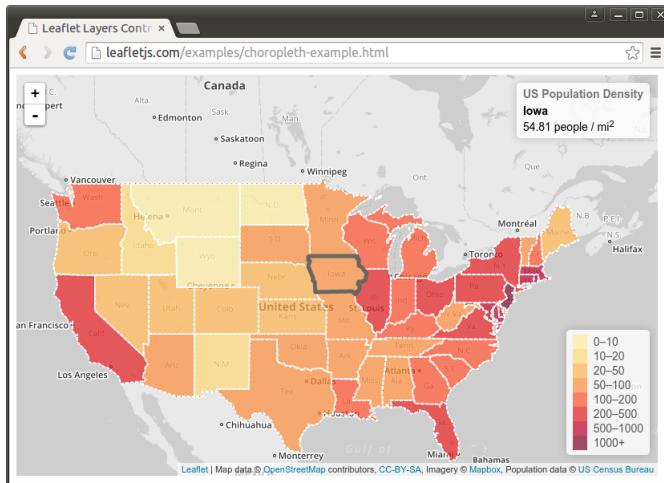


Abbildung 19: Beispiel einer (mit Hilfe von Leaflet.js) angereicherten OpenStreetMap-Karte

Für diese Arbeit soll *OpenStreetMap* diese Kategorie von externen Systemen in der Marktanalyse vertreten. OpenStreetMap ist ein etablierter, von namhaften Applikationen verwendeteter Dienst, der die in ihm enthaltene Information als *Open Data* lizenziert und die grafischen Inhalte unter einer *Creative-Commons-Lizenz* veröffentlicht²⁹. Somit liesse sich OpenStreetMap auch als *Komponente* von Hyperlike betreiben – natürlich nur durch erhöhten Installationsaufwand. Die kartographischen Informationen von OpenStreetMap werden laufend durch die Community verbessert, und eine ausführliche Dokumentation hilft bei der Entwicklung des Systems. Der Karten-Dienst soll im übrigen als *Implementation Detail* betrachtet werden, der ohne tiefgreifende Anpassung der Applikation ausgetauscht werden kann.

4.3 Komponenten

4.3.1 Formular-Server

Die Möglichkeit zur Eingabe von Werten ist eine kritische Anforderung, so dass in diesem Bereich die Abhängigkeit von externen System nicht angeraten ist. Da es sich bei **Formhub** (siehe Abschnitt 4.2.1) um freie Software handelt³⁰, lässt sich der unter formhub.org nur sehr schlecht erreichbare Dienst auch *lokal installieren*. Die Applikation ist in Python geschrieben, läuft innerhalb des Django Web Frameworks und benutzt PostgreSQL sowie MongoDB zur Datenspeicherung. Durch die lokale Instanz steigt einerseits die Verfügbarkeit der Applikation, andererseits nimmt auch der Aufwand für Installation und Administration zu.

Formhub bietet zudem keine laufende Analyse der Eingaben, diese müsste vollständig durch eine externe Applikation umgesetzt werden. Zwar lassen sich die Eingabewerte mittels Excel

²⁹ <https://www.openstreetmap.org/copyright>

³⁰ <https://github.com/SEL-Columbia/formhub>

oder XML konfigurieren, Eingabe und Ausgabe sind jedoch kaum anpassbar. Als komplexes, aus zahlreichen Komponenten bestehendes und doch in sich geschlossenes System kann Formhub die Anforderungen zu Skalierbarkeit und Erweiterbarkeit nicht erfüllen.

Auch **ODK Aggregate** (siehe Abschnitt 4.1.1) ist als lokaler Dienst lauffähig, der unabhängig von anderen OpenDataKit-Komponenten betrieben werden kann. Doch wie in der detaillierten Betrachtung in Abschnitt 4.1.1 bereits erwähnt, eignet sich dieser aus verschiedenen Gründen nicht für den in dieser Arbeit verfolgten Zweck.

4.3.2 Geodatenbanken

Die eingegebenen Werte, sowie möglicherweise Zwischenresultate der Analyse, wollen in einer Datenbank gespeichert werden. Für die Analyse steht neben der zuverlässigen Persistierung zudem im Vordergrund, dass die Datenbank über die Eigenschaften einer Geodatenbank verfügt. Als solche unterstützt sie *geografische Objekte und Funktionen*, insbesondere die räumliche Indexierung von Feldern. Diese erlaubt Abfragen nach geografischen Merkmalen eines Datensatzes – beispielweise, ob sich ein Punkt *in der Nähe* eines andern befindet, oder ob sich ein Punkt *innerhalb eines bestimmten geografischen Bereichs* befindet.

Für viele Datenbank-Produkte gibt es Erweiterungen, um räumliche Indexierung zu ermöglichen. Die etabliertesten Produkte in diesem Bereich sind sicherlich **PostGIS** und **MongoDB**. PostGIS ist eine Erweiterung der beliebten relationalen SQL-Datenbank PostgreSQL, die dieser durch eine grosse Sammlung von geografischen Funktionen hinzufügt. PostGIS bildet die Grundlage für viele kommerzielle und freie Software-Produkte im Bereich der Geoinformation.

Die NoSQL-Datenbank MongoDB bietet einen kleineren Funktionsumfang als PostGIS. So werden auf der Query-Seite auf räumlich indexierten Feldern nur die Operatoren `within`, `int` `ersects` und `near` unterstützt³¹. Diese genügen jedoch, um die Funktionalität von Anforderung FREQ-02 abzudecken. Eine Mongo-Datenbank ist dokumentenbasiert und benötigt deshalb kein Schema. Davon profitiert ein System wie «Hyperlike», bei dem die gespeicherten Werte bei jeder Konfiguration anders sind. Ein weiterer Vorteil von MongoDB ist die Performance von Schreiboperationen und die Möglichkeit, einfache verteilte Systeme zu konfigurieren. [8] beschreibt die Überlegenheit von NoSQL-Systemen bei der Analyse grosser Datenmengen. Ein weiterer Vorteil von MongoDB sind die vielseitigen Analyse-Funktionen für eingehende Daten. Deshalb soll MongoDB als das aussichtsreichere Produkt diese Kategorie von Komponenten vertreten.

³¹ <http://docs.mongodb.org/manual/reference/operator/query-geospatial/>

4.4 Ergebnis

4.4.1 Plattformen

Im Rahmen dieser Marktanalyse wurden Plattformen als mögliche Kandidaten für die Verwendung für das Ziel dieser Arbeit untersucht. Die ausgewählten Plattformen traten als Gesamtlösungen für die formulierten Anforderungen an und wurden auch als solche untersucht. Sie müssen deshalb zumindest *alle funktionalen* Anforderungen erfüllen.

Das Ziel dieser Arbeit ist die Erstellung eines wiederverwendbaren, konfigurierbaren Systems, das eine bestimmte Funktionalität bieten soll. Während der Verzicht auf die Erfüllung bestimmter funktionaler Anforderungen für den *Anbieter einer einzelnen «Hyperlike»-Instanz* sicher verkraftbar wäre, würde das Fehlen von Funktionalität der Plattform zu einem unvollständigen Produkt führen. Die in Abschnitt 3.4 formulierten Anforderungen behandeln die gesamte Verarbeitungskette einer Erhebung, die keine Lücke aufweisen darf. Andernfalls wäre der Anspruch als Komplettlösung für die definierten Use Cases hinfällig. Deshalb kommt den funktionalen Anforderungen die Rolle von *K.O.-Kriterien* zu.

Dagegen sind die nicht-funktionalen Anforderungen naturgemäß nicht zwingend für die Erbringung der gewünschten Funktionalität. Vielmehr verbessert ihre Erfüllung den Nutzen des Produkts. Eine ungenügende Erfüllung disqualifiziert die Lösung nicht zwingend, sondern macht weitere Abklärungen nötig.

In Tabelle 5 sind erfüllte funktionale Anforderungen beim entsprechenden Produkt grün markiert; rote Markierungen stehen für eine Nichterfüllung einer funktionalen Anforderungen, gelbe für die Nichterfüllung einer nicht-funktionalen Anforderung. Eine rote Markierung eines Produkts genügt, um die gesamte Plattform zu disqualifizieren.

Schnell wird klar, dass keine der Plattformen die funktionalen Anforderungen von «Hyperlike» erfüllt. Die untersuchten Plattformen haben die Gemeinsamkeit, dass sie primär auf die Sammlung komplex strukturierter Beobachtungswerte ausgelegt sind. Dem durch «Hyperlike» verfolgten Ziel des *Crowdsourcing* und *Data Sharing* vermögen sie nicht gerecht zu werden – die Eingabemethoden sind nicht auf räumliche Werte und nicht auf Anpassbarkeit der Benutzeroberfläche ausgelegt, Funktionen für die Analyse und die Veröffentlichung der entstandenen Resultate fehlen weitgehend, und die Skalierbarkeit der monolithischen Systeme ist nicht gewährleistet.

Einzig bei der Konfigurierbarkeit der Eingabewerte können alle Plattformen punkten – die Konfigurationen sind allerdings von aussen kaum zugänglich. Dieser Umstand sowie die Tatsache, dass alle Plattformen nur wenig erweiterbar sind, sorgt dafür, dass sie sich auch nicht für die Verwendung als *Komponenten* eignen.

Tabelle 5: Marktanalyse Plattformen

	<i>FREQ-01: Konfiguration der Eingabewerte</i>	<i>FREQ-02: Analyse der Eingaben</i>	<i>FREQ-03: Automatische Erfassung der Positionsdaten einer Eingabe</i>	<i>FREQ-04: Grafische Web-Benutzeroberfläche für die Dateneingabe</i>	<i>FREQ-05: Grafische Darstellung der Analyse-Resultate auf einer Karte</i>	<i>FREQ-06: Grafische Web-Benutzeroberfläche ist anpassbar</i>	<i>NFREQ-01: Einfaches Deployment</i>	<i>NFREQ-02: Skalierbarkeit</i>	<i>NFREQ-03: Erweiterbarkeit</i>	<i>NFREQ-04: Open Source</i>
OpenDataKit	●	●	●	●	●	●	●	●	●	●
Ushahidi	●	●	●	●	●	●	●	●	●	●
EpiCollect	●	●	●	●	●	●	●	●	●	●

- Produkt erfüllt die Anforderung weitgehend oder vollständig
- Produkt erfüllt die *funktionale* Anforderung nicht oder nur ungenügend
- Produkt erfüllt die *nicht-funktionale* Anforderung nicht oder nur ungenügend

4.4.2 Dienste und Komponenten

In den Kategorien Dienste bzw. Komponenten traten die Kandidaten nicht als Gesamtlösungen an, sondern wurden darauf untersucht, ob sie *einen Teil der Funktionalität* von «Hyperlike» abdecken. Bei der Untersuchung von Produkten in dieser Kategorie wurde eine Nutzwertanalyse durchgeführt. Der Repräsentant jeder untersuchten Produktekategorie erhält je nach Erfüllungsgrad einer Anforderung eine Bewertung zwischen 0 und 3 (gemäss den Kriterien in Tabelle 6). Diese Bewertung wird in Tabelle 7 mit der *Gewichtung* der Anforderung (mögliche Werte 1 bis 3, vgl. Abschnitt 3.4) multipliziert, um die Wertigkeit der Produktekategorie für die Umsetzung zu bestimmen. Die *Summen der Wertigkeiten* schliesslich bilden das Ergebnis der Nutzwertanalyse.

Tabelle 6: Bewertungsskala für Nutzwertanalyse

Grad der Erfüllung einer Anforderung	Bewertung
Produkt bietet keine Funktionalität für Anforderung	0
Produkt kann für Umsetzung der Anforderung verwendet werden, bietet dafür aber keine direkte Unterstützung	1
Produkt bietet Unterstützung (z.B. durch APIs) für die Anforderung	2
Produkt bietet teilweise oder vollständige Umsetzung der Anforderung	3

Der Karten-Dienst *OpenStreetMap* bietet sich an, um bei der Benutzeroberfläche eine tragende Rolle zu spielen. Er lässt sich mit den anderen kombinieren und erfüllt alle nicht-funktionalen Kriterien.

Obwohl *Parse.com* als Repräsentant der Backend-Dienste viele Funktionalitäten bietet, eignet sich der Dienst aufgrund seiner Lizenzierung und seiner Eigenschaft als proprietärer, kommerzieller Cloud-Dienst nicht für die Erfüllung der Anforderungen. Der Formular-Server *Formhub* bietet als Komponente einiges im Bereich Datensammlung, überzeugt jedoch bei Anpassbarkeit, Skalierbarkeit, Deployment und Analyse nicht.

Die Geodatenbank *MongoDB* schliesslich vermag bei allen nicht-funktionalen Anforderungen zu überzeugen und eignet sich als Hilfsmittel zur Erfüllung der funktionalen Anforderungen durch ihre mit einer gut dokumentierten API zugänglichen Analyse-Funktionen.

Tabelle 7: Nutzwertanalyse Dienste und Komponenten

	<i>FREQ-01: Konfiguration der Eingabewerte</i>	<i>FREQ-02: Analyse der Eingaben</i>	<i>FREQ-03: Automatische Erfassung der Positionsdaten einer Eingabe</i>	<i>FREQ-04: Grafische Web-Benutzeroberfläche für die Dateneingabe</i>	<i>FREQ-05: Grafische Darstellung der Analyse-Resultate auf einer Karte</i>	<i>FREQ-06: Grafische Web-Benutzeroberfläche ist anpassbar</i>	<i>NFREQ-01: Einfaches Deployment</i>	<i>NFREQ-02: Skalierbarkeit</i>	<i>NFREQ-03: Erweiterbarkeit</i>	<i>NFREQ-04: Open Source</i>	<i>Ergebnis</i>
Gewichtung	3	3	1	3	3	2	2	1	2	1	19
BaaS: Parse.com	0	2	0	0	0	0	3	3	2	0	19
<i>Formular-Server:</i> Formhub	2	0	0	3	0	0	0	0	2	3	22
<i>Karten-Dienst:</i> OpenStreetMap	0	0	1	1	2	2	3	3	3	3	32
<i>Geodatenbank:</i> MongoDB	0	2	0	0	0	0	3	3	3	3	24

Tabelle 8: Nutzwerte kombinierter Komponenten

<i>Karten-Dienst + BaaS:</i>	0	2	1	1	2	2	6	6	4	3	51
<i>Karten-Dienst + Formular-Server:</i>	2	0	1	4	2	2	3	3	5	6	54
<i>Karten-Dienst + Geodatenbank:</i>	0	2	1	1	2	2	6	6	6	6	56

4.5 Fazit

Tabelle 8 enthält die möglichen Kombinationen von Komponenten: eine Backend-Lösung, jeweils kombiniert mit dem Karten-Dienst. Die Verbindung von Geodatenbank MongoDB mit dem Karten-Dienst OpenStreetMap deckt die meisten funktionalen und alle nicht-funktionalen Anforderungen zumindest teilweise ab und erzielt das höchste Ergebnis in der Nutzwertanalyse.

Die beiden anderen Kombinationen schneiden schlechter ab, wenn auch knapp. Dies ist im Falle von *Parse.com* vor allem auf das kommerzielle Lizenz-Modell zurückzuführen. Zudem ist zwar Parse sehr mächtig und bietet über umfangreiche APIs viel Möglichkeiten zur Erweiterung, doch ist sie durch ihre Eigenschaft als Cloud-Dienst weniger gut integrierbar: die Anbindung läuft über ein öffentliches Netz, die Entwicklung geschieht aus der Distanz. Bei Formhub geben das aufwändige Deployment, die mangelhafte Erweiterbarkeit sowie die fehlende Analyse-Funktionalität den Ausschlag.

Es ist Aufgabe des *Konzepts* im nächsten Abschnitt, das Zusammenspiel dieser Teile sowie die Orchestrierung des Gesamtsystems zu entwerfen. Die Komponenten MongoDB und OpenStreetMap sind beide mächtig, erweiterbar und gut dokumentiert, so dass die fehlende Funktionalität in den Bereichen Konfiguration, Analyse und grafische Ein-/Ausgabe mit nur wenig Aufwand abgedeckt werden kann.

5 Konzept

Die Marktanalyse in Abschnitt 4 hat ergeben, dass zur Erfüllung der Anforderungen an «Hyper-like» ein *Karten-Dienst* und eine *Geodatenbank* eingesetzt werden sollen. Da mit diesen beiden Produkten die Anforderungen nur zum Teil erfüllt werden können, muss für die durch diese beiden Teilsysteme nicht oder nur teilweise erfüllten Anforderungen eine Lösung gefunden werden. Dies betrifft die Schritte der Analyse, die Konfiguration der Eingabewerte sowie teilweise die grafische Benutzeroberfläche für die Eingabe der Beobachtungswerte und die Ausgabe der Resultate. Darüber hinaus muss ein Mechanismus entwickelt werden, der die verschiedenen Teillösungen zur einer Komplettlösung zusammenschweisst, so dass das resultierende System eine *integrierte, wiederverwendbare Plattform* bildet.

Das in diesem Abschnitt formulierte *Konzept* soll Antworten auf diese Fragen liefern. Zuerst soll eine Architektur definiert werden, welche das angestrebte System aus der Vogelperspektive betrachtet. Darauf folgt ein genauerer Blick auf die einzelnen Teile und Subsysteme sowie eine Zusammenstellung der zur Umsetzung nötigen Schritte. Diese werden anschliessend auf konkrete Arbeitspakete aufgeteilt, was eine grobe Aufwandschätzung ermöglicht.

5.1 Architektur

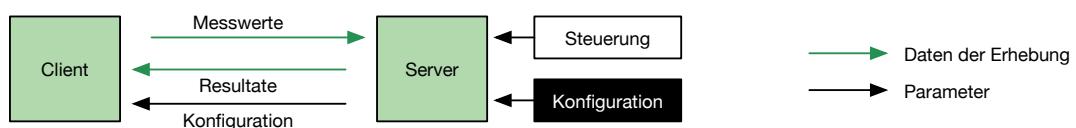


Abbildung 20: Komponenten

«Hyperlike» besteht aus einer *Client*- und aus einer *Server-Komponente* (siehe Abbildung 20). Diese Aufteilung scheint dem Autor als *alternativlos* – durch seine verteilten Satelliten für die Datensammlung und seinem zentralen Mechanismus für die Analyse erfordert das angestrebte System einen solchen Aufbau.

Abzuwagen bleibt jedoch, wie die zu erbringende Funktionalität zwischen Client- und Server-Komponente aufgeteilt wird. So könnte die Client-Komponente so konzipiert werden, dass sie lediglich die *Anzeige* übernimmt, während die Server-Komponente für die gesamte Verarbeitungs- und Bedienlogik verantwortlich ist. Mit Blick auf zukünftige weitere Eingabemechanismen (wie beispielsweise native Apps oder automatisierte Clients) liegt es nahe, möglichst wenig Bedienlogik auf der Server-Komponente zu belassen. Die Vorteile der Entwicklung eines *Rich Clients* liegen auf der Hand:

- Die klare Trennung von Ein-/Ausgabe- und Verarbeitungslogik führt zur **Entkoppelung der Teilsysteme** und damit einer Vereinfachung der Entwicklung und zu besserer Wartbarkeit.

- Durch Kommunikation über öffentliche Schnittstellen bleibt **Erweiterbarkeit** durch weitere Eingabemethoden erhalten.
- Client- und Server-Komponente können die für ihre Anwendung optimalen Technologien einsetzen; dies ermöglicht eine höhere **Zufriedenheit des Benutzers**.
- Durch die serverseitige Berechnung der Resultate kann der Client entlastet werden, was eine Steigerung der **Effizienz** bedeutet.

Der Client ist zuständig für die Benutzeroberfläche und die Datenübermittlung zum Server. Während die Server-Komponente nur einmal pro Erhebung existiert, wird der Client für jede Eingabe im Web-Browser jedes Benutzers einzeln ausgeführt. Die Server-Komponente enthält Mechanismen für folgende Funktionen:

- Management der Konfiguration des Systems
- Entgegennahme von neu eintreffenden Eingabewerten
- Analyse der Eingaben und Speicherung der Resultate
- Ausgabe von Resultaten der Analyse
- Steuerung des Gesamtsystems
- Auslieferung und Parametrisierung der Client-Komponente

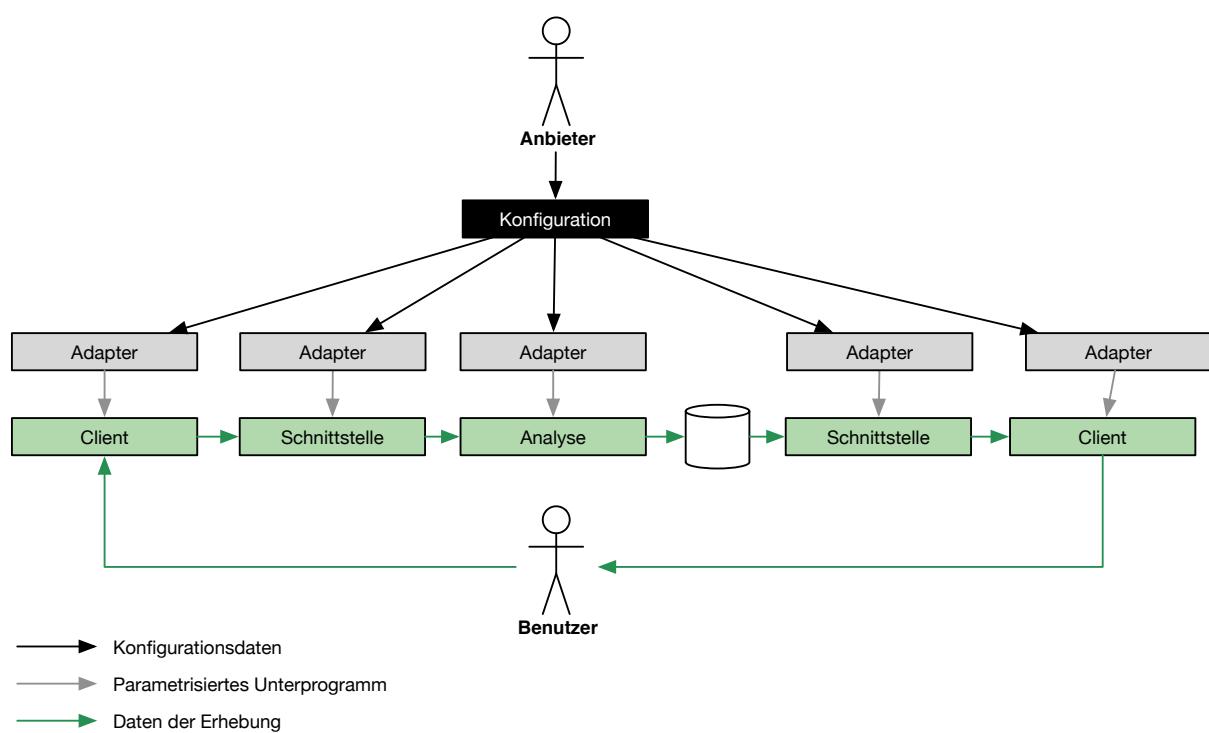


Abbildung 21: Datenfluss-Diagramm

Die detailliertere Sicht auf das Zusammenspiel von Client- und Server-Komponente liefert das Datenfluss-Diagramm in Abbildung 21. Die Kreisläufe der Datenverarbeitung bei der System-Initialisierung (schwarze und graue Pfeile) und während des Betriebs (grüne Pfeile) sind getrennt und unidirektional. Dies fördert das Verständnis des Zustands des Systems bei der Entwicklung und im Betrieb.

Der zentrale Konfigurationsmechanismus übergibt die Konfiguration einer Erhebung an *Adapter-Module*. Diese setzen die Konfiguration zu einer Applikation um, die die gewünschte Funktionalität bietet. Die Adapter haben gleiche Schnittstellen, sind unabhängig voneinander und werden alle durch die übergebene Konfiguration vollständig definiert.

Die *Konfiguration* soll als eine *Single Source of Truth* dienen – ein *atomarer* Wert, der bei der Initialisierung in einem Schritt vollständig ersetzt wird. In diesem Verhalten folgt die Architektur von «Hyperlike» der stetig an Popularität gewinnenden Schule der *reaktiven Systeme*. Diese zeichnen sich aus durch klare Datenflüsse von der Umgebung in das System hinein sowie durch Subsysteme, die auf neue Daten umgehend reagieren. Das «Reactive Manifesto»³² namhafter Vertreter der Software-Industrie erklärt die Vorteile solcher Systeme, während verschiedene neuere funktionale Programmiersprachen (Clojure, Erlang) und Frameworks (Akka, React) die Umsetzung demonstrieren. Bei «Hyperlike» bewirken eine neue Konfiguration oder neue Eingabedaten eine Zustandsänderung des Systems, der sich, ausgehend von den Schnittstellen, durch alle Teile fortpflanzt.

Aus den Anforderungen aus Abschnitt 3.4 lassen sich verschiedene Teilbereiche der Konfiguration ableiten (Tabelle 9). Diese entstammen den Anforderungen und markieren auch die Bedürfnisse der einzelnen Komponenten.

Tabelle 9: Teilbereiche der zentralen Konfiguration

Eingabewerte	Bezeichnung und zulässige Wertebereiche der Eingabedaten (gemäss FREQ-01)
Segmente	Definition des geografischen Bereichs der Erhebung und dessen Aufteilung in zusammenhängende Segmente (gemäss FREQ-02)
Analyse	Methode, mit der das Ergebnis für einzelne Segmente bestimmt wird (gemäss FREQ-02)
Anzeige	Spektrum der Kolorierung der Segmente in der Resultate-Karte (gemäss FREQ-05); Anpassung der Oberfläche (gemäss FREQ-06)

³² <http://www.reactivemanifesto.org/>

5.1.1 Server-Komponente

Die Server-Komponente besteht aus drei Subsystemen: der Input-Schnittstelle, dem Analyse-Subsystem und der Output-Schnittstelle. Abbildung 22 illustriert den Datenfluss innerhalb der Server-Komponente: Die Adapter (grau) setzen die Werte der Konfiguration (schwarz) zu Teilkomponenten (grün) um.

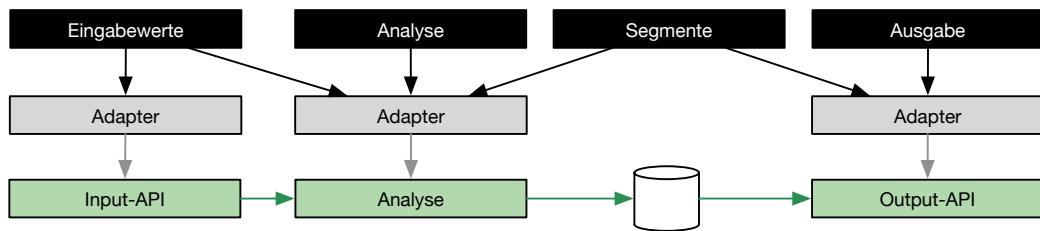


Abbildung 22: Datenfluss Server-Komponente

Die *Input-Schnittstelle* nimmt neue Eingabewerte entgegen und validiert diese gegenüber der Konfiguration. Im Interesse der Datensicherheit werden die neuen Werte möglicherweise schon hier in der Datenbank gespeichert. Danach werden die neuen Werte an das Analyse-Subsystem übergeben (s.u.). Die *Output-Schnittstelle* bietet Zugriff auf die Resultate der Analyse, damit diese vom Client auf deiner Karte dargestellt werden können.

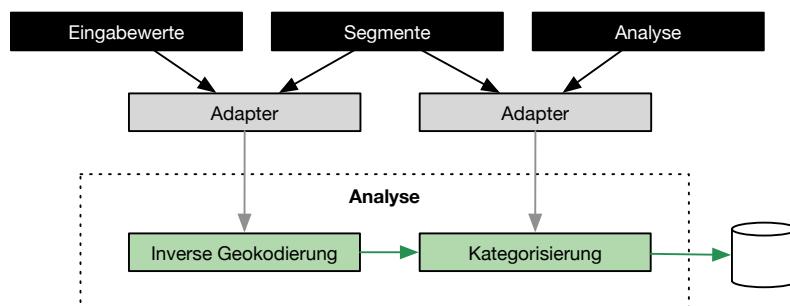


Abbildung 23: Datenfluss Server-Komponente

Das Analyse-Subsystem (Abbildung 23) führt zuerst die *inverse Geokodierung* der eingehenden Werte durch. Danach erfolgt die Bestimmung des neuen Werts des Segments gemäss der konfigurierten Analyse-Methode, also die *Abbildung* in den Wertebereich der Visualisierung.

Gemäss FREQ-02 soll die Analyse so konfigurierbar sein, dass vor dem Deployment aus einer Liste verfügbarer Methoden ausgewählt werden kann. Es liegt also nahe, die Methoden als in sich geschlossene *Module* zu implementieren, die über ihre Bezeichnung in der Konfiguration referenziert werden. Die einzelnen Module sollen im gleichen Kontext laufen, so dass ein Modul auf sämtliche Ressourcen der Server-Komponenten zugreifen kann.

5.1.2 Client-Komponente

Die Client-Komponente behandelt ausschliesslich die UI-Logik, also sämtliche Schritte, die notwendig sind, um Eingabewerte vom Benutzer zum Server zu übermitteln sowie Analyse-Resultate auf einer Karte darzustellen. Die Kommunikation mit der Server-Komponenten erfolgt ausschliesslich über die entsprechenden Schnittstellen (vgl. Abbildung 20).

Für Eingabe und Ausgabe werden zwei separate Subkomponenten erzeugt. Der *Input-Client* deckt die Anforderungen der Eingabe ab. Er enthält Eingabe-Elemente für die konfigurierten Werte-Typen (FREQ-04) und führt die Bestimmung der Position einer Eingabe durch (FREQ-03). Input- und Output-Client haben wenig Überschneidungen und lassen sich fast unabhängig implementieren.

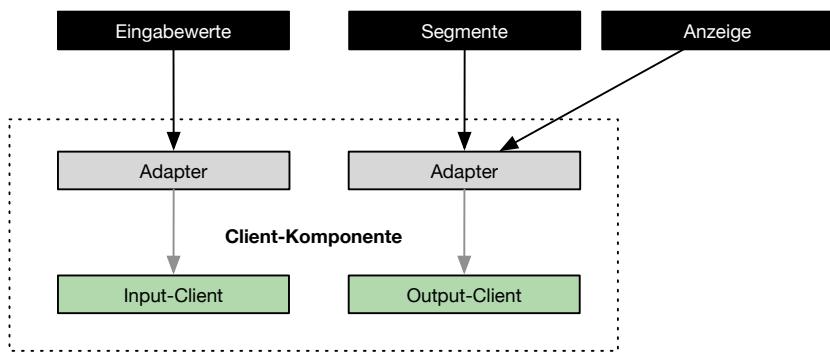


Abbildung 24: Datenfluss Client-Komponente

Der *Output-Client* bezieht die analysierten Zwischenergebnisse der Erhebung vom Server und stellt sie auf einer Karte dar. Dazu teilt er den für die Erhebung konfigurierten Kartenausschnitt in die definierten Segmente auf und färbt diese gemäss der konfigurierten Aufteilung des Wertebereichs ein.

5.1.3 Datenbank

Sämtliche eingehenden Werte werden in der Datenbank als Rohdaten persistiert. Dadurch können sie zu einem späteren Zeitpunkt in weitere Analysen einfließen oder direkt den Nutzern zugänglich gemacht werden. Neben der Datenspeicherung kommt der Datenbank auch eine wichtige Rolle im *Analyseprozess* zu, da dieser primär Anweisungen generiert, die die Datenbank auf bestehende Resultate anwendet, um neue Zwischenresultate zu erzeugen.

5.2 Nicht-funktionale Aspekte

Neben den funktionalen Anforderungen muss der Entwurf der Architektur auch nichtfunktionale Aspekte berücksichtigen, welche beschreiben, auf welche Art und Weise die Funktionalität erbracht wird. Nichtfunktionale Anforderungen beziehen sich auf das System als ganzes und werden komponentenübergreifend betrachtet.

5.2.1 Einfaches Deployment

Der Grad der Einfachheit des Deployment wird letztlich bestimmt durch die Umsetzung. Erst sie produziert Artefakte für die in diesem Abschnitt definierten Komponenten. Durch das zentrale Management der Konfiguration, die Verwendung bekannter Software-Produkte als Teillösungen (Karten-Dienst und Geodatenbank) wird ein einfaches Deployment und Betrieb begünstigt.

5.2.2 Skalierbarkeit

Die Skalierbarkeit des Systems hängt ab von der Ausgestaltung der möglichen *Bottlenecks*, also Stationen in der Verarbeitungskette, die besonders anfällig sind auf Ressourcen-Engpässe. Im hier konzipierten System sind dies einerseits die Input- bzw. Output-Schnittstellen, die als zentrale Endpunkte dienen, sowie andererseits das Analyse-Subsystem der Server-Komponente, die alle eingehenden Werte in Echtzeit zu einem Zwischenresultat pro Segment zusammenführen muss.

Da alle Ein- und Ausgaben unabhängig voneinander erfolgen und deren Reihenfolge für das Resultat unerheblich ist, kann einem Engpass bei der *Schnittstellen* durch *Network Load Balancing* begegnet werden. Dabei werden Verbindungen an verschiedene Instanzen der Schnittstellen geschickt, abhängig von der Auslastung des Systems.

Engpässen im *Analyse-Subsystem* kann begegnet werden, indem die Berechnung der Resultate auf verschiedenen Rechnern ausgeführt wird. Da die einzelnen Segmente der Konfiguration sich nicht überschneiden und ihre Definitionen konstant sind, kann die Berechnung auf mehrere Prozesse verteilt werden. Dazu wird die Datenbank entlang der geografischen Grenzen der Segmente *gesharded*, also auf verschiedene unabhängige Instanzen aufgeteilt; die gewählte Geodatenbank MongoDB unterstützt diese Funktionalität. [15] liefert Hinweise für den Entwurf einer verteilten Architektur.

Ein neu eintreffender Wert kann so der Instanz zugewiesen werden, die für das entsprechende Segment zugewiesen wird (siehe Abbildung 25). Die Granularität dieser Aufteilung ist das Segment. Dass dessen Grenzen zu Beginn der Erhebung statisch konfiguriert werden, vereinfacht die Aufteilung, da so nur ein zentraler Lookup-Mechanismus für Datenbank-Instanzen existieren muss.

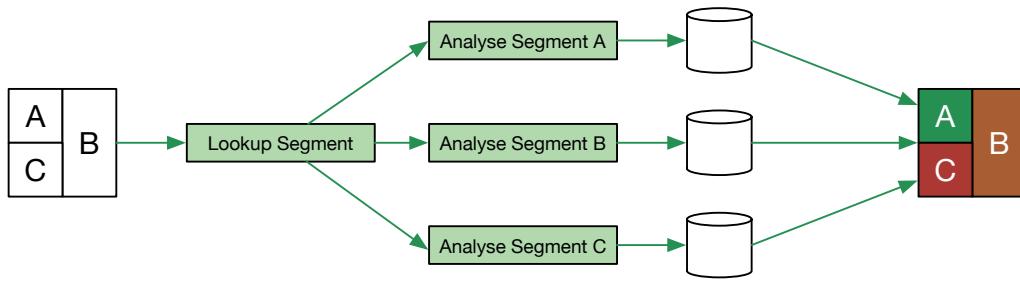


Abbildung 25: Skalierung der Analyse von Segmenten durch *Sharding* der Datenbank

5.2.3 Erweiterbarkeit

Durch die Auslagerung der gesamten UI-Logik in den Client werden die Aspekte Eingabe und Analyse entkoppelt und nur durch geeignete Schnittstellen verbunden. Dies fördert die Erweiterbarkeit durch mögliche weitere Eingabe-Methoden. Ebenso hilft die modulare Architektur durch Adapter der zentralen Konfiguration bei der Erstellung von neuer Funktionalität oder zusätzlichen Komponenten.

5.3 Aufwandschätzung

Die klare Trennung der Zuständigkeiten der Subsysteme ermöglichen eine einfache Aufteilung der zur Umsetzung nötigen Schritte auf Arbeitspakete: Für die beiden Teile der Client-Komponente benötigt es je einen Adapter; einen Adapter braucht das Analyse-Modul; zusätzlich muss ein Mechanismus für das Management der Konfiguration erstellt werden. Die Datenstruktur der Konfiguration muss unter Berücksichtigung des Zusammenspiels der Teilsysteme definiert werden.

Tabelle 10 gibt eine Übersicht über die Arbeitspakete. Der Aufwand für die einzelnen Punkte misst deren Komplexität im Vergleich zu anderen. Die Aufwandschätzung ist somit als Richtwert für die Umsetzung zu verstehen: Während die Implementierung eines Pakets für den Prototypen um vieles geringer ausfällt als für ein produktives System, so bleibt doch die relative Komplexität gegenüber anderen Arbeitspaketen erhalten. Die Aufwandschätzung beschränkt sich auf die Umsetzung der *Use Cases* mit funktionalen Anforderungen, die Implementierung der nichtfunktionalen Anforderungen wird nicht geschätzt.

Tabelle 10: Arbeitspakete der Umsetzung mit Schätzung der relativen Komplexität

Use Case	Arbeitspaket	Komplexität
<i>UC-01: Konfiguration des Systems</i>	Datenstruktur für die Konfiguration	3
	Konfigurations-Management	1
	Installation und Deployment	2
	Analyse-Adapter	5
<i>Eingabe von Beobachtungswerten</i>	Adapter für Client-Schnittstellen	2
	Adapter für Input-Client	2
	Speicherung der eingehenden Werte	1
<i>UC-03: Ausgabe der Resultate</i>	Adapter für Output-Client	5

5.4 Fazit

Das Resultat dieses Konzepts ist eine *Client-/Server-Architektur* mit einem zentralen Konfigurationsmechanismus (Abbildung 20). Die einzelnen Komponenten sind implementiert als modulare Adapter, die die Einstellungen der Konfiguration umsetzen (Abbildung 21). Diese Architektur unterstützt moderne Konzepte des Software-Designs und berücksichtigt die nicht-funktionalen Anforderungen bezüglich Skalierbarkeit und Erweiterbarkeit.

Der nächste Schritt dieser Arbeit besteht aus der Implementierung eines Prototypen, der als *Proof of Concept* (PoC) die hier formulierten Ideen *praktisch umsetzen* soll. Neben Design und Programmierung des PoC steht das Testing auf Grundlage der Akzeptanzkriterien im Vordergrund.

6 Proof of Concept

Dieser Abschnitt dokumentiert die Entwicklung eines Prototypen von Hyperlike, der als *Proof of Concept* des in dieser Arbeit entworfenen Systems dienen soll. Dabei wird die Funktionalität von Hyperlike in ihrer gesamten *Breite* abgebildet, d.h. es sollen alle wesentlichen Aspekte der funktionalen Anforderungen so implementiert werden, dass die Funktionsweise der Plattform als Ganzes sichtbar wird.

Zuerst werden die Anforderungen aus Abschnitt 3.4 ausgewählt und konkretisiert, dann wird die Design- und Entwicklungsarbeit der im Konzept entworfenen Software dokumentiert. Die Dokumentation soll einen Einblick in die verwendeten Methoden und Technologien verschaffen, ohne zu stark ins Detail zu gehen.

Sämtliche Resultate der Umsetzung sind auf der dieser Dokumentation beigelegten DVD gespeichert. Darauf befindet sich neben dem Quelltext sämtlicher Komponenten und den Resultaten der *Unit Tests* auch ein lauffähiges Artefakt der Applikation.

6.1 Anforderungen

Der Prototyp soll sämtliche *funktionalen Anforderungen* im Wesentlichen erfüllen. Dadurch wird die gesamte Verarbeitungskette der Eingabewerte einer Erhebung abgebildet, und es entsteht ein zumindest schematischer Überblick über alle beteiligten Prozesse. Das Resultat soll eine Applikation sein, die die als *Mandatory* markierten *Use Cases* in Abschnitt 3.3 abdeckt.

Obwohl alle Anforderungen aus Abschnitt 3.4.1 umgesetzt werden, sollen diese *gewissen Einschränkungen* unterliegen, um den Rahmen dieser Arbeit nicht zu sprengen. Durch die reduzierte Funktionalität kann einerseits der Entwicklungsaufwand verkleinert werden, andererseits kann der Autor sich bei der Entwicklung auf diejenigen Aspekte konzentrieren, die das Konzept am besten anhand einer konkreten Umsetzung illustrieren.

Anforderung *FREQ-02* (vgl. Abschnitt 3.4.1) betrifft die «Analyse der Eingaben» und soll für den Prototypen so eingeschränkt werden, dass Punkt 3 der Spezifikation *nicht umgesetzt wird*. Der Prototyp wird also nicht die Möglichkeit bieten, Elemente des konfigurierten Rasters zu *Segmenten* zusammenzufassen. So wird ein Erhebungswert jeweils nur einen der Teilbereiche des für die Erhebung gültigen Gebietes beeinflussen. Zudem wird *nur eine Analyse-Methode* implementiert: Der Mittelwert aller Felder eines Tupels und aller Eingaben. Es wird also nicht möglich sein – wie in Punkt 4 der Spezifikation beschrieben – aus einer Liste von Analyse-Methoden auszuwählen.

Die übrigen Anforderungen sollen weitgehend der Spezifikation entsprechen, jedoch ohne Anspruch, den für eine *produktive Applikation* geltenden Kriterien zu genügen. So werden Schnittstellen und Bedienelemente aller Komponenten nur gerade den für die Erfüllung notwendigen Funktionsumfang aufweisen.

Trotz diesen Einschränkungen wird der Prototyp keine Lücke in der Grundfunktionalität aufweisen – es wird also möglich sein, eine Erhebung zu konfigurieren, Werte einzugeben und die Resultate der Analyse auf einer Karte zu betrachten. Somit wird die resultierende Applikation geeignet sein, die in Abschnitt 7 folgenden *Tests der Anforderungskriterien* durchzuführen.

6.2 Rahmenbedingungen

Für Server- und Client-Komponente bestehen unterschiedliche technologische Rahmenbedingungen: Während die Server-Komponente als langlebiger, zentraler Dienst existiert, wird die Client-Komponente pro Benutzer einmal instanziert und nach Verwendung wieder geschlossen; während beim Deployment auf dem Server mit einer von System to System unterschiedlichen Umgebung gerechnet werden muss, ist das technologische Umfeld für den Client mehr oder weniger uniform: ein moderner Web-Browser mit einer grafischen Oberfläche.

Die im Konzept in Abschnitt 5 gewählte Architektur lässt sich durch verschiedene Technologien implementieren. Während beim Client, die *Lauffähigkeit im (mobilen) Web-Browser* im Vordergrund steht, sind die Hauptkriterien für die Wahl der Technologien für die Server-Komponente *Plattformunabhängigkeit, Stabilität* und *Integrierbarkeit* mit der in der Marktanalyse gewählten Geodatenbank MongoDB. Die Kommunikation zwischen Client und Server erfolgt ausschließlich über das Internet-Protokoll HTTP, für das Implementierungen in allen Umgebungen leicht verfügbar sind.

Für die Umsetzung des Prototypen wurden für diese Arbeit die beiden Plattformen *JavaScript* für den Client und *Java* für die Server-Komponente gewählt. Die folgenden Abschnitte werden diese Auswahl begründen und die Arbeit mit ihnen und mit den verwendeten Hilfsmitteln dokumentieren.

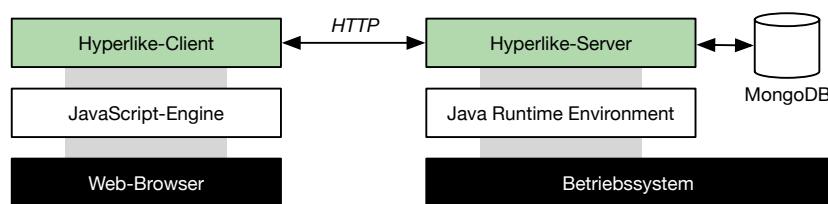


Abbildung 26: Überblick über den Technologie-Stack von Client- und Server-Komponente

Abbildung 26 zeigt die beiden «Technologie-Stapel», also die Schichten von Technologien, auf denen die beiden Komponenten aufsetzen. Die unterste Schicht zeigt jeweils die Grundlage, von der bei der Entwicklung ausgegangen werden kann. Die weiss gefärbten Elemente repräsentieren die für die Umsetzung des Prototypen *gewählten Technologien*.

6.3 Umsetzung der Server-Komponente

Die *Server-Komponente* soll selbstständig ausführbar sein und zur Laufzeit nur gegenüber der *Geodatenbank* eine Abhängigkeit haben. Neben dem Anbieten der Schnittstellen für die Client-Komponente und der Durchführung der Analyse kommt dieser Komponente die Aufgabe zu, die *Artefakte der Client-Komponente* an die Endbenutzer auszuliefern.

6.3.1 Technologien

Die Server-Komponente wurde mit Programmiersprache Java von Oracle³³ umgesetzt. Zum einen ist der Autor mir dieser vertraut, so dass die Implementierung ohne Umwege angegangen werden konnte, zum andern erfreut sich Java grosser Beliebtheit und ist so weit verbreitet, dass ihre Verfügbarkeit bei potenziellen Betreibern einer Hyperlike-Instanz fast vorausgesetzt werden kann. Ihre direkte Konkurrentin aus dem Hause Microsoft – die .NET-Plattform – bietet weitgehend die gleiche Funktionalität, ohne jedoch bezüglich Plattformunabhängigkeit mithalten zu können, da sie (bis jetzt) ausschliesslich im Windows-Umfeld zuhause ist.

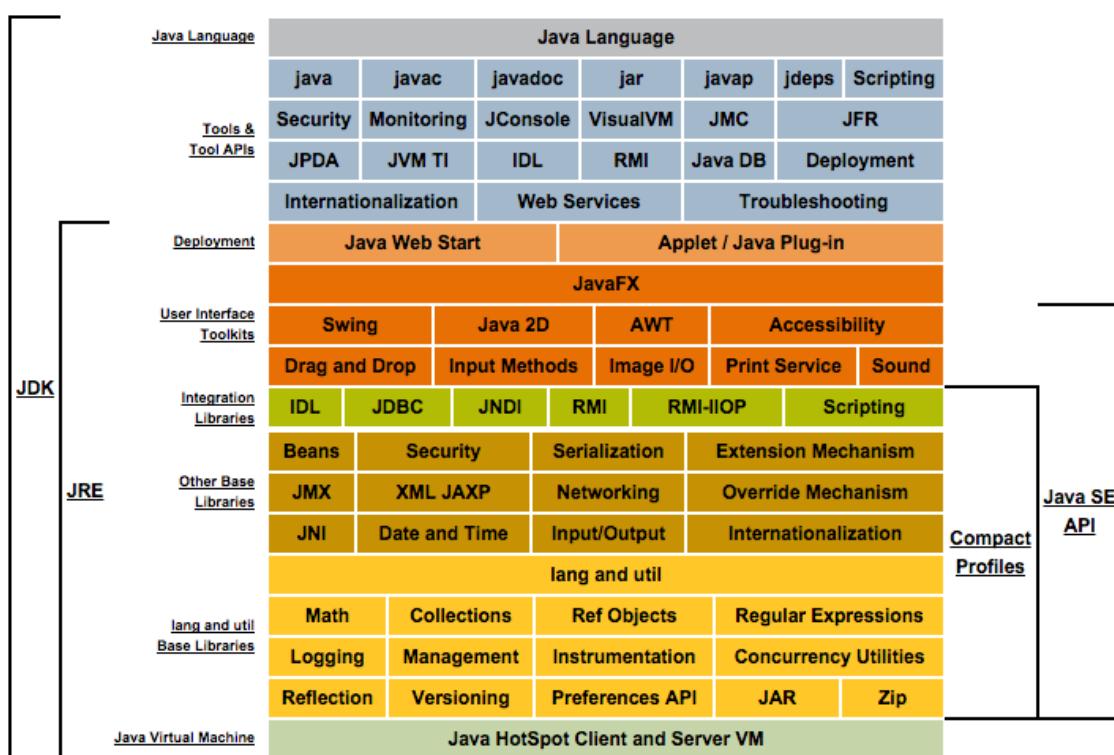


Abbildung 27: Konzeptionelles Diagramm der umfangreichen Plattform Java SE³⁴

³³ <http://www.oracle.com/technetwork/java/index.html>

Die **Java-Plattform** bietet in der *Standard Edition* Java SE ein *Runtime Environment* mit der *Java Virtual Machine*, die kompilierten *Bytecode* ausführt, sowie mit einem Kompilierer für die objektorientierte Programmiersprache Java. Hinzu kommt eine umfangreiche Standardbibliothek mit einer grossen Auswahl an Datentypen und Hilfsklassen. Zusätzlich zu Java SE gibt es die Variante *Enterprise Edition* J2EE mit einer Vielzahl zusätzlicher Standards für den Einsatz bei klassischen Applikationen im Unternehmensumfeld. Der Prototyp von Hyperlike verwendet *Java SE 7*, den Vorgänger der aktuellen Version 8, um ein möglichst grosses Publikum anzusprechen.

Während die Java-Plattform die Grundlage bildet für die Ausführung der Applikation, benötigt Hyperlike eine beträchtliche Menge an zusätzlicher Funktionalität – wie etwa die Anbindung an die Geodatenbank, die Übermittlung der Eingabewerte sowie Mechanismen für Start und Initialisierung der Applikation. Um diese mit der *Infrastruktur* beauftragten Programmteile nicht selbst implementieren zu müssen, griff der Autor auf die Dienste der **Spring-Plattform**³⁵ des zu VMware gehörenden Anbieters Pivotal zurück.

Das Spring Framework trat im Jahr 2004 an, um als leichtgewichtiger Ersatz für die schwerfällige *Java Enterprise Edition* zu dienen. Die Enterprise-Entwicklung sollte stark vereinfacht und durch moderne Paradigmen wie *Inversion of Control* und *Aspect-Oriented Programming* ergänzt werden. Mittlerweile hat sich Spring als dominierende Plattform für grössere Java-Applikationen etabliert. Die Plattform ist zudem stark gewachsen und hat andere Mitglieder des Java-Ökosystems durch Abstraktion über deren Schnittstellen (z.B. zu Datenbanken oder Message-Brokern) absorbiert. Als *Plattform auf der Plattform* ist Spring heute zwar nicht mehr leichtgewichtig, dafür bezüglich Funktionsumfang *umfassend* und in vielen Projekten *unausweichlich*.

Von den überaus zahlreichen unterschiedlichen zum Spring Framework gehörenden Modulen³⁶ sollten für den Prototypen folgende zum Einsatz kommen:

- *Spring Context* für den Start und die Initialisierung aller Dienste. Spring Context übernimmt zusätzlich durch *Dependency Injection* von als *Services* markierten Klassen auch die *Verdrahtung* aller Komponenten. Diese Zusammensetzung der Instanzen von Klassen wird deklarativ festgelegt – früher in XML-Dateien, heutzutage hauptsächlich durch *Konfigurations-Klassen* und durch Annotationen direkt im Code.
- *Spring Web* für die Bereitstellung von HTTP-Schnittstellen. Spring Web ist ein auf der Erstellen von Web-Applikationen spezialisiertes Modul der Spring-Plattform. Das Modul ist vielseitig anwendbar – von grafischen, serverseitigen Interfaces mit einer MVC-Architektur bis hin zu REST-Schnittstellen.
- *Spring Data MongoDB* für die Anbindung der Datenbank. Spring Data ist ein zentrales Spring-Projekt, das es Entwicklern erlaubt, mit wenigen Handgriffen so genannte «NoSQL»-Daten-

³⁵ <http://spring.io>

³⁶ Für eine Übersicht über die Module von Spring, vgl. [11], Seite 26

banken in eine grössere Applikation einzubinden. Während der Zugriff auf *relationalen SQL-Datenbanken* von Anfang an durch das Spring Framework gut unterstützt wurde, konnte auch der Zugriff auf NoSQL- Datenbanken durch Spring Data weitgehend vereinheitlicht werden, ohne die Stärken der einzelnen Produkte durch eine gemeinsame Schnittstelle zu verlieren (siehe [10], Seite 77f).

Neben den Spring-Modulen seien hier zwei weitere wesentliche Komponenten namentlich erwähnt:

- **Jackson**, eine Bibliothek für den Umgang mit JSON³⁷. Diese populäre Bibliothek hat unter anderem die Funktion, als JSON gespeicherte Daten zu entsprechenden Java-Datenstrukturen zu *deserialisieren*. Jackson übernimmt oft wichtige Aufgaben in vielen Applikationen, ist aber durch seine indirekte Einbindung via Spring oft für den Entwickler unsichtbar.
- **Java Servlet 3.0**, eine vom Java Community Process (JCP) erstellte Spezifikation³⁸ für Web-Schnittstellen. Die Servlet-Spezifikation dient dazu, Web-Services unter Java zu standardisieren: Ein kompatibler Application Server (z.B. Apache Tomcat³⁹ oder Eclipse Jetty⁴⁰) dient als *Servlet Container* und reicht HTTP-Anfragen an registrierte Servlet-Instanzen weiter. Die aus der Spezifikation abgeleiteten Java-Interfaces erscheinen dem Entwickler wiederum wegen der Absorption durch Spring versteckt.

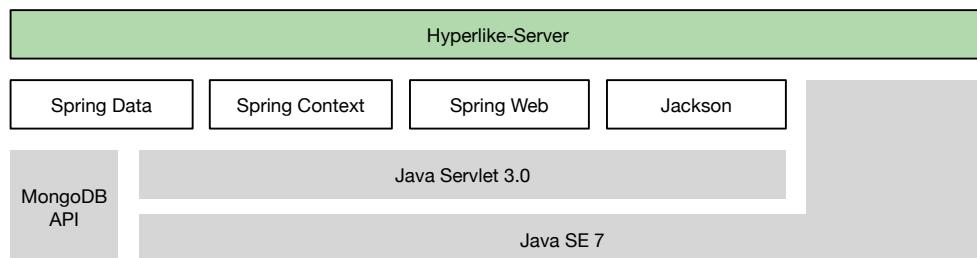


Abbildung 28: Von der Server-Komponente verwendete Frameworks und APIs

Abbildung 28 zeigt die von der Server-Komponenten verwendeten Frameworks (weiss) und Schnittstellen (grau) sowie deren Verhältnis zueinander. Spring Data MongoDB läuft als Spring-Modul innerhalb des Servlet Containers, greift jedoch über die vom MongoDB-Dienst exponierte Netzwerk-Schnittstelle auf die Datenbank zu.

³⁷ <https://github.com/FasterXML/jackson>

³⁸ JSR 315, siehe <https://www.jcp.org/en/jsr/detail?id=315>

³⁹ <http://tomcat.apache.org>

⁴⁰ <http://www.eclipse.org/jetty>

6.3.2 Design

Das Design der Server-Komponente hält sich an etablierte *Best Practices* (vgl. z.B. [12]) des objektorientierten Designs und an im Java-Umfeld existierende Konventionen. So wird das *Verhalten* der Applikation durch Interface-Typen modelliert, deren Bezeichnung den üblichen Namenskonventionen («Service» für Dienste oder «Repository» für Sammlungen von persistierten Objekten) folgt.

Beim Design der Server-Komponente orientierte sich der Autor weitgehend am Konzept aus Abschnitt 5.1.1. Die dort beschriebenen Subsysteme für Input, Analyse sowie Output werden direkt auf entsprechende Teilprogramme abgebildet. Zusätzlich bietet ein Mechanismus für das Management der Konfiguration Funktionen für das Starten und Initialisieren der einzelnen Dienste.

In Abbildung 29 sind die *Java-Interfaces* der wichtigsten Dienste abgebildet und die Abhängigkeiten zwischen ihnen durch Pfeile eingezeichnet. Eine Reihe von *Model-Klassen* dient der Kommunikation der Schnittstellen untereinander. Die *Adapter* werden gebildet von Klassen, die direkt oder indirekt auf die Konfiguration der Erhebung zugreifen.

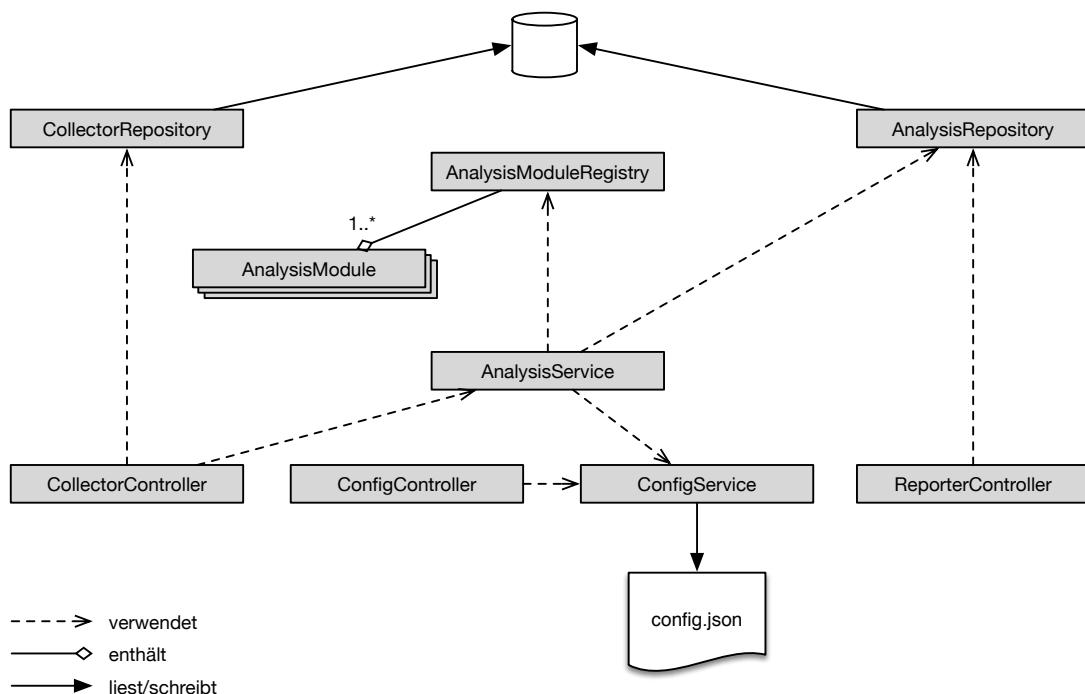


Abbildung 29: Diagramm der wichtigsten Schnittstellen der Server-Komponente

Für den Input zuständig sind die Schnittstellen **CollectorController** und **CollectorRepository**. Erstere repräsentiert eine REST-Schnittstelle, an welche mit dem HTTP-Kommando POST neue Erhebungswerte im Format JSON übermittelt werden können. Das Interface **CollectorR**

pository bietet Methoden zum Speichern und Lesen von Eingabewerten der Erhebung.

Für die Analyse zuständig sind die Schnittstellen `AnalysisService`, `AnalysisModuleRegistry`, `AnalysisModule` sowie `AnalysisRepository`. Beim Systemstart ruft der `AnalysisService` die in einem `AnalysisModule` gekapselte Analyse-Methode aus dem `AnalysisModule Registry`-Dienst gemäss Konfiguration der Erhebung ab. Ein neu eintreffender Eingabewert wird vom `CollectorController` an den `AnalysisService` weitergegeben, von diesem zur Berechnung des Analyse-Werts an die `AnalysisModule`-Instanz übergeben und das Resultat mittels `AnalysisRepository` in der Datenbankpersistiert.

Bei der Modellierung der Datenbank muss berücksichtigt werden, dass MongoDB nicht – wie SQL-Datenbanken – tabellenorientiert ist, sondern auf *Dokumenten* aufbaut. MongoDB-Dokumente sind zu *Collections* gruppiert. Eine Collection erzwingt nicht ein fixes Schema für alle enthaltenen Dokumente, sondern erlaubt die Speicherung völlig unterschiedlicher Strukturen – trotzdem ist es natürlich im vorliegenden Fall sinnvoll, ähnliche Datenstrukturen zu einer Collection zusammenzufassen. Hyperlike benutzt zwei Collections: die durch das Interface `CollectorRepository` repräsentierte enthält pro Eingabe-Tupel je ein Dokument; im `AnalysisRepository` befindet sich je ein Dokument pro Karten-Segment.

Die durch das Interface `ReporterController` definierte REST-Schnittstelle bedient die HTTP GET-Abfragen der Client-Komponente mit dem gesamten Inhalt des `AnalysisRepository`-Dienstes. Diese enthält für jedes Element des konfigurierten Rasters der Analyse ein Objekt mit dem entsprechenden Resultat – so entsteht eine Liste von Resultaten, die dem Client zur Darstellung übergeben werden kann.

Der Dienst `ConfigService` ermöglicht den Zugriff auf die Konfiguration der Erhebung durch alle Klassen welche die Rolle eines Adapters übernehmen. Der Dienst verfügt über die Methode `ConfigService.getConfig`, die ein Objekt der Datenstruktur `Config` zurückgibt, die sämtliche durch den Anbieter vorgenommenen Einstellungen umfasst.

Die HTTP-Methoden und Ressourcen-Pfade der REST-Schnittstelle orientieren sich an den Empfehlungen von [6]. Sie umfasst die folgenden Ressourcen:

- GET `/api/config`: Abfrage der aktuellen Konfiguration des Systems. Sie dient einerseits zur Überprüfung des vorgenommenen Einstellungen sowie anderseits zur Übermittlung der Konfiguration an die Client-Komponente.
- POST `/api/collect`: Erfassung eines Eingabewertes, der als JSON im *Body* des Anfrage enthalten sein muss.
- GET `/api/report`: Abfrage des aktuellen Resultats der Erhebung im Format JSON. Die Antwort besteht aus einem Array von Objekten, von denen jedes dem Analyse-Wert eines Segments der Karte mit den Ergebnissen entspricht.

6.3.3 Umsetzung

Die Umsetzung der Server-Applikation besteht hauptsächlich aus je einer Implementierung der obigen Interface-Typen. So entsteht aus dem Interface `AnalysisService` eine Klasse `AnalysisServiceImpl` mit der Logik zur initialen Abfrage der konfigurierten Analyse-Methode sowie zur Verarbeitung neu eintreffender Eingabewerte.

Die im Prototypen verfügbare Analyse-Methode des Mittelwerts aller Felder einer Eingabe wird umgesetzt in einer Klasse `MeanAnalysisModule`, die das Interface `AnalysisModule` implementiert. Die Analyse selbst besteht aus zwei Schritten:

1. Zuerst erfolgt eine Abfrage der Datenbank nach den Dokumenten im `AnalysisRepository`, in dessen geografischen Grenzen der neu eintreffende Wert liegt. Dies sind immer zwei: ein Dokument für das entsprechende Kartensegment und ein Dokument für das gesamte konfigurierte Gebiet. Letzteres enthält somit die «Meta-Analyse», die bei der Ausgabe benötigt wird, um etwa das globale Maximum aller Analyse-Werte zu bestimmen.
2. Anschliessend werden die Felder der Dokumente durch den neu berechneten Wert aktualisiert und das Dokument in der Datenbank gespeichert.

Die Aufteilung der Dokumente folgt den Empfehlungen in der Dokumentation von MongoDB⁴¹, eine mögliche Optimierung besteht jedoch sicherlich darin, beide Schritte zusammenzufassen und in einem *Upsert* auszuführen.

Die Konfiguration der Erhebung erfolgt beim Prototypen über eine *JSON-Datei*, die per Konvention den Namen `config.json` tragen und im Arbeitsverzeichnis der Server-Applikation liegen muss. Beim Systemstart wird sie gelesen und den Adaptern zur Verfügung gestellt.

Die Verknüpfung der einzelnen Dienste geschieht durch den von Spring angebotenen Mechanismus der *Dependency Injection*. Eine Abhängigkeit zur Instanz eines bestimmten Typs wird durch die Annotation `@Autowired` markiert. Beim Systemstart sorgt die durch den Spring Context durchgeführte Initialisierung für eine korrekte Erstellung und Verdrahtung der notwendigen Objekte. Die Registrierung der REST-Endpunkte im Servlet-Container verläuft nach einem ähnlichen Prinzip: Durch die Spring-Annotation `@RestController` werden die Controller-Klassen beim Systemstart durch Spring Web als Java Servlets registriert.

6.3.4 Werkzeuge

Ein für die Entwicklung sehr wertvolles Hilfsmittel war **Spring Boot**, ein *Rapid Application Development Toolkit* für die Spring-Plattform⁴². Spring Boot besteht aus einer Reihe vorbereiteter *Starter Kits*, mit denen sich sehr schnell lauffähige Applikationen zusammenstellen lassen.

⁴¹ <http://docs.mongodb.org/manual/core/data-modeling-introduction>

⁴² <http://projects.spring.io/spring-boot>

Dabei übernimmt Spring Boot durch *Autokonfiguration* nach dem Prinzip *Konvention vor Konfiguration* die Initialisierung sämtlicher Komponenten. Gerade bei der Entwicklung von Prototypen ist dieses Werkzeug von grosser Nützlichkeit, da es schnelle Fortschritte ermöglicht, ohne dass der Entwickler zuerst viel Zeit mit einem detaillierten Setup verbringen muss. Die Auswahl der Starter Kits erfolgt über die Website „Spring Initializr“⁴³, die die Zusammenstellung der benötigten Starter Kits zu einer Projekt-Definition auf übersichtliche Weise gestattet.

Die im Verlaufe der Entwicklung entstehenden spezifischen Bedürfnisse und Ansprüche können durch schrittweise Übersteuerung der automatischen Konfiguration befriedigt werden. Bei der Entwicklung von Hyperlike führte beispielsweise die Einbindung des Starter Kits `spring-boot-starter-data-mongodb` dazu, dass die Applikation beim Systemstart selbstständig die notwendigen Treiber installierte und Verbindung zur lokalen MongoDB-Instanz aufnahm. Erst als der Wunsch aufkam, die Entwicklungs-Datenbank in einer *Virtual Machine* zu betreiben, wurden die automatischen Einstellungen in einer Konfigurations-Datei übersteuert. Ein ähnliches Vorgehen ist möglich beim Setup des Application Servers Jetty und dem Spring Context selbst, der durch die Klassen-Annotation `@SpringBootApplication` schnell eine selbstständig lauffähigen Applikation erzeugt.

Aus der professionellen Java-Entwicklung nicht mehr wegzudenken ist natürlich das Build-Tool **Apache Maven**. Neben der Kompilierung des Codes und der Verpackung der Artefakte übernimmt Maven eine Reihe anderer Aufgaben, wie etwa die Verwaltung von Abhängigkeiten zu Software von Drittanbietern oder die Einbindung statischer Ressourcen. So lässt sich mit Maven der gesamte Prozess der Herstellung aller Artefakte eines Projekts steuern: von der Vorbereitung des Quelltextes über das automatische Durchführen aller Unit Tests bis hin zur Erstellung der Dokumentation.

Seine Vielseitigkeit gewinnt Maven nicht zuletzt aus einer grossen Sammlung von *Plugins*, die die Funktionsweise des Werkzeugs erweitern. Durch das von Spring erstellte `spring-boot-maven-plugin` kann am Ende des Build-Prozesses aus einem Projekt, das Spring Boot einsetzt, ein in sich geschlossenes JAR-Archiv erstellt werden, das durch Übergabe an das Kommandozeilen-Programm `java` gestartet werden kann.

Für die Bearbeitung des Programm-Codes verwendete der Autor das *Integrated Development Environment* (IDE) **IntelliJ IDEA** von JetBrains. IntelliJ IDEA ist eine mächtige IDE, die neben hoher Bedienungsfreundlichkeit bei der Programmierung einen sehr leistungsfähigen Debugger und Werkzeuge zur Code-Analyse bietet. Dazu verfügt die Software über eine gute Unterstützung für diverse Java-Frameworks sowie eine elegante Anbindung externer Software-Produkte – etwa die Versionsverwaltung Git oder die *Application Server* Tomcat und Jetty.

⁴³ <https://start.spring.io>

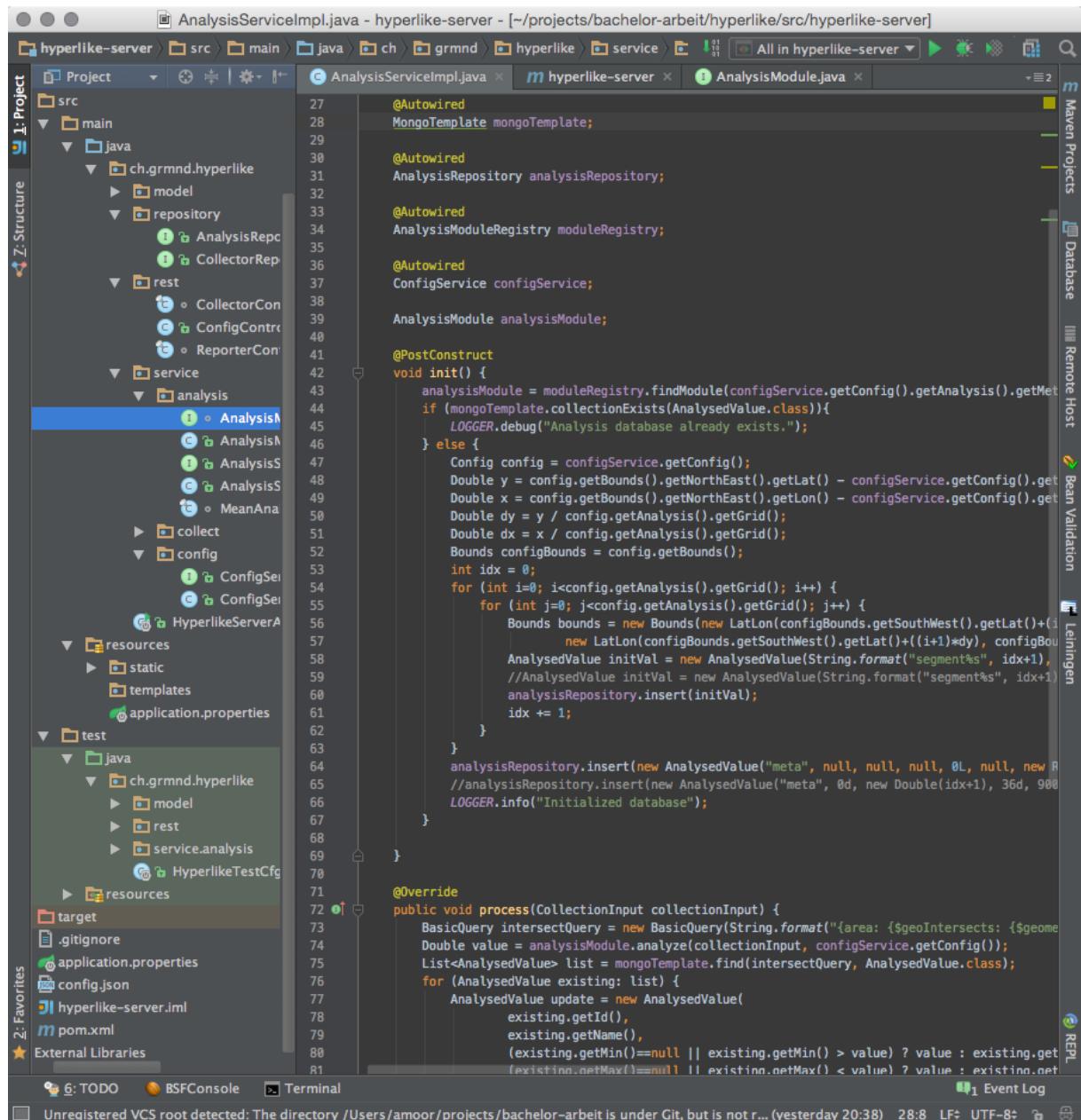


Abbildung 30: Java-IDE «JetBrains IntelliJ IDEA» mit Projekt- und Editor-Fenster

Durch diese Verbindung von Programmier-Umgebung und Java-Ökosystem ist es möglich, Server-Applikationen in einem Application Server zu starten und mit der IDE zu überwachen und inspizieren. So kann – beispielsweise durch das Setzen von Haltepunkten – eine Web-Applikation während der Entwicklung in ihrem produktiven Umfeld genau untersucht und beobachtet werden.

Für die Arbeit mit der Datenbank MongoDB wurde das grafische Werkzeug **Robomongo**⁴⁴ ver-

⁴⁴ <http://robomongo.org>

wendet. Zwar bietet die Datenbank mit mongo ein leistungsfähiges Programm für die Kommandozeile, doch die Arbeit in der Mongo-Shell ist gerade bei strukturierten Daten oft mühselig. In Robomongo lassen sich die vorhandenen Datensätze gut überfliegen und erkunden. Gleichzeitig bietet das Werkzeug vollen Zugriff auf die Mongo-Shell, so dass auf die Vielseitigkeit der Query-Möglichkeiten nicht verzichtet werden muss.

Key	Type
(7) ObjectId("556cb12ffbea4f7fd358fb45")	{ 6 fields }
(8) ObjectId("556cb12ffbea4f7fd358fb46")	{ 6 fields }
(9) ObjectId("556cb12ffbea4f7fd358fb47")	{ 6 fields }
(10) ObjectId("556cb12ffbea4f7fd358fb48")	{ 6 fields }
(11) ObjectId("556cb12ffbea4f7fd358fb49")	{ 6 fields }
(12) ObjectId("556cb12ffbea4f7fd358fb4a")	{ 6 fields }
(13) ObjectId("556cb12ffbea4f7fd358fb4b")	{ 6 fields }
(14) ObjectId("556cb12ffbea4f7fd358fb4c")	{ 6 fields }
(15) ObjectId("556cb12ffbea4f7fd358fb4d")	{ 6 fields }
(16) ObjectId("556cb12ffbea4f7fd358fb4e")	{ 6 fields }
(17) ObjectId("556cb12ffbea4f7fd358fb4f")	{ 6 fields }
(18) ObjectId("556cb12ffbea4f7fd358fb50")	{ 10 fields }
_id	ObjectId("556cb12ffbea4f7fd358fb50")
_class	ch.grmnd.hyperlike.model.analy...
name	segment18
min	42.666667
max	196.333333
sum	1127.000000
count	10
computed	{ 1 fields }
last	2015-06-01 19:37:17.505Z
area	{ 2 fields }
(19) ObjectId("556cb12ffbea4f7fd358fb51")	{ 6 fields }
(20) ObjectId("556cb12ffbea4f7fd358fb52")	{ 6 fields }
(21) ObjectId("556cb12ffbea4f7fd358fb53")	{ 6 fields }

Abbildung 31: Graphische Oberfläche für die Datenbank MongoDB

6.3.5 Automatisierte Tests

Bei der Entwicklung der Server-Komponente ging der Autor von Anfang an nach den Prinzipien des *Test-Driven Development* vor: Von den wesentlichen Schnittstellen der Software wurden zuerst *Unit Tests* erstellt, die durch die entsprechende Implementation zu erfüllen waren. Ebenso wurden die *Model*-Klassen auf ihre Tauglichkeit als *Werte-Typen* geprüft durch Tests der Gleichheit und Serialisierbarkeit von Instanzen. Die Unit Tests wurden mit **JUnit** ausgeführt, dem Standard-Werkzeug für automatisierte Tests in Java.

Da die *Applikationslogik* hauptsächlich in den Analyse-Modulen umgesetzt wird, wurde dieser Teil als möglichst *zustandslos* implementiert, d.h. ohne Datenhaltung, die zu einem von Lauf zu Lauf unterschiedlichen Verhalten der Programms führen könnte. Dies führt natürlich auch

zu einer besseren Testbarkeit dieses Teils der Software: Da bei gleichen Parametern immer mit dem gleichen Rückgabewert gerechnet werden kann, muss vor einem Test bei solchen *Pure Functions* keine aufwändige Initialisierung durchgeführt werden.

Die *Dienste* der Applikation hingegen sind mit dem *Verwalten des Zustands der Applikation* vertraut und müssen deshalb auch unter diesem Aspekt geprüft werden. Da es zu umständlich wäre, bei jedem Testlauf eine Datenbank zu starten oder eine bestehende in einen definierten Zustand zu versetzen, griff der Autor zu einer *In-Memory-Implementierung* von MongoDB: Beim Projekt «Fongo»⁴⁵ handelt es sich um eine in Java-Implementierung der MongoDB-Schnittstellen, die keine Werte dauerhaft persistiert, sondern nur für die Laufzeit des Tests im Speicher vorhält. Dies führt einerseits zu einer Vereinfachung der Test-Umgebung; andererseits zwingt sie den Entwickler dazu, die Applikation von Anfang an so zu programmieren, dass diese in der Lage ist, die Datenbank beim Systemstart selbstständig zu initialisieren – ein Verhalten, das den späteren Installationsaufwand klein hält. Trotzdem erscheinen die Schnittstellen der Datenbank gegenüber den darauf zugreifenden Diensten der Applikation als voll funktionsfähig mit dem gleichen Verhalten wie eine produktive Umgebung.

Um das Verhalten der *Web-Schnittstellen* zu überprüfen, wurde für diese jeweils ein *Integrations-Test* vorbereitet, der die korrekte Verarbeitung der Ein- und Ausgaben der Controller-Klassen verifizierte. In diesem Unterfangen war das Spring-Projekt `spring-boot-starter-test` sehr hilfreich, das eine Reihe von Hilfsmitteln mitbringt, die das Schreiben von Tests für Spring Boot unterstützen. So führt beispielsweise die Klassen-Annotation `@WebIntegrationTest` dazu, dass der Test-Runner beim Durchlaufen der Tests einen Applikation-Server startet, der dem Unit Test für Abfragen zu Verfügung steht. Zusammen mit der In-Memory-Datebank kann so die gesamte Verarbeitungskette aller Daten integriert getestet werden, ohne vorher fehleranfällige Skripte ausführen zu müssen.

Die Unit- wie die Integrations-Tests wurden bei jedem Durchlauf des Build-Prozesses durch Apache Maven automatisiert ausgeführt. Ein Nichtbestehen eines Tests führt dabei stets zu einem Abbruch des Prozesses, was einer umgehenden Behebung von Fehlern zuträglich war.

⁴⁵ <https://github.com/fakemongo/fongo>

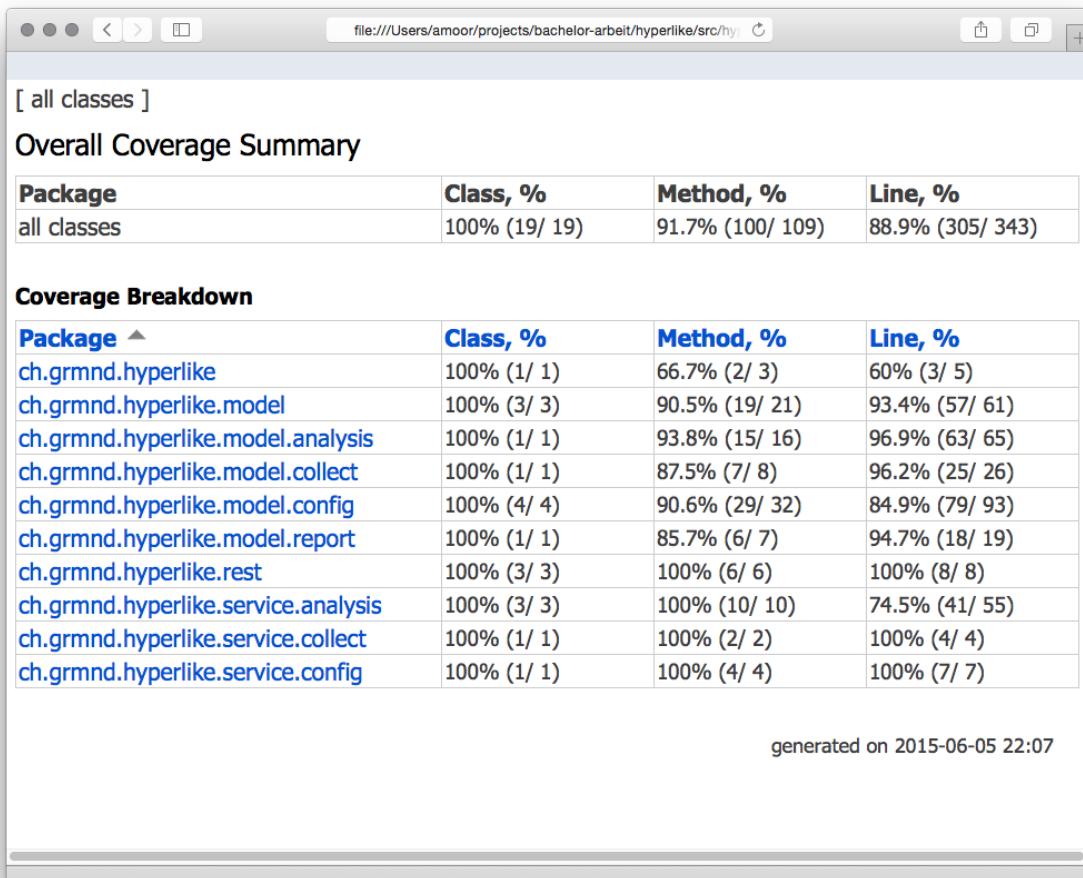


Abbildung 32: Test Coverage Report der Server-Komponente

Die IDE IntelliJ erlaubt den Export eines «Coverage Reports» (siehe Abbildung 32) der Unit- und Integrations-Tests. Dieser dokumentiert, wie viel Programmier-Code des Projekts durch automatisierte Tests durchlaufen werden. Die gute Integration von Frameworks durch IntelliJ IDEA stellt sicher dass für den Report die gleichen Tests durchlaufen werden wie in einem Maven-Build.

Aus Abbildung 32 geht hervor, dass die Server-Komponente von Hyperlike sehr gut durch Tests abgedeckt ist: alle Klassen, 91.7% der darin enthaltenen Methoden und 88.9% aller Code-Zeilen werden von den Tests durchlaufen. Die nicht hundertprozentige Abdeckung ist hauptsächlich auf *Boilerplate*-Code zurückzuführen, also einer Aufblähung des Quelltextes durch formell oder technisch bedingte Zeilen: Zum Beispiel brauchen Klassen, deren Instanzen serialisiert werden müssen (etwa zu JSON), zwingend einen Konstruktor ohne Argumente – dieser ist jedoch in der Regel als `private` markiert, also für eine Instanzierung in Tests gar nicht erreichbar.

6.4 Umsetzung der Client-Komponente

Für den Client wurden viele technologische Details schon in Abschnitt 3.4.1 durch FREQ-04 und FREQ-05 und im Konzept durch die Architektur des Gesamtsystems vorweggenommen: eine im Web-Browser laufende Applikation, die über eine API mit der Server-Komponente kommuniziert. Als solche werden ihre Artefakte nach ihrer Erstellung von der Server-Komponente als *statische Ressourcen* an den Benutzer ausgeliefert und in dessen Web-Browser ausgeführt.

6.4.1 Technologien

Die Umsetzung der Client-Komponente erfolgte in JavaScript. Diese Programmiersprache aus der Anfangszeit des World Wide Web würde früher gerne belächelt (vgl. [13], Seite 101ff.). Sie hat jedoch in den letzten Jahren die Web-Programmierung im Sturm erobert, nicht zuletzt dank den Bemühungen der Browser-Hersteller Mozilla und Google um effiziente Laufzeit-Umgebungen.

JavaScript ist einerseits sicherlich die am weitesten verbreitete Laufzeitumgebung für Applikationen im Web, und zwar so sehr, dass sie gerne als Ziel-Plattform für andere Programmiersprachen dient (z.B. ClojureScript oder Googles Dart) – ein Umstand, der ihr das Etikett «Assembly of the Web»⁴⁶ eingebracht hat. Anderseits ist die mittlerweile auch die mächtigste: Zahlreiche neue, unter das Etikett «HTML5» zusammengefasste JavaScript-APIs im Browser haben die Möglichkeiten der Plattform wachsen lassen. [9] bietet eine detaillierte Einführung in die moderne Web-Programmierung.

Alternative Umgebungen für *Rich Clients* im Web, wie etwa «Adobe Flash» oder «Java Applets», verlieren zunehmend an Bedeutung. Durch immer neue Sicherheitslücken und umständliche Installation büßen diese Plattformen immer mehr ihrer Marktanteile ein.

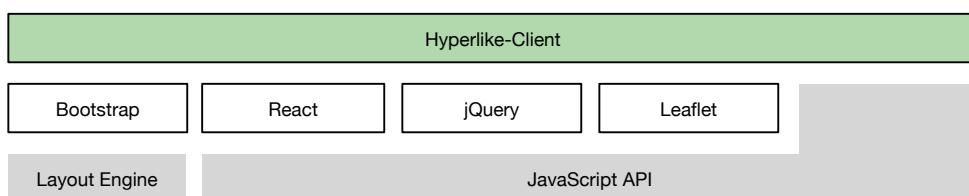


Abbildung 33: Von der Client-Komponente verwendete Frameworks und APIs

Zwar unterscheiden sich die JavaScript-APIs der Web-Browser je nach Hersteller recht stark⁴⁷. Doch für die Zwecke des Hyperlike-Clients kann davon ausgegangen werden, dass die Laufzeit-Umgebung bei allen Implementierungen einheitlich ist. Gleches gilt für die *Layout Engine*, welche mit *Cascading Style Sheets* (CSS) gesteuert wird.

⁴⁶ <http://hanselminutes.com/274/javascript-is-assembly-language-for-the-web-semantic-markup-is-dead-clean-vs-machine-coded>

⁴⁷ Für eine detaillierten Vergleich der Implementationen siehe z.B. <http://caniuse.com>

Laufzeit-Umgebung und Layout-Engine bilden natürlich bloss das Fundament zur Ausführung des Programms – um die gewünschte Funktionalität anbieten zu können, wurde bei der Entwicklung auf eine Reihe von Komponenten von Drittanbietern zurückgegriffen. Abbildung 33 zeigt wiederum den *Stack* der eingesetzten Frameworks und Bibliotheken.

React⁴⁸ ist eine JavaScript-Bibliothek für den Bau von User-Interfaces. Diese von Facebook als Open-Source-Software angebotene Bibliothek ist recht jung, erfreut sich aber schon grosser Beliebtheit im Frontend-Bereich. Sie unterscheidet sich von anderen Bibliotheken in diesem Umfeld dadurch, dass der Entwickler mit ihr beim Rendering nicht versucht, den Zustand des Browsers durch *DOM-Operationen* punktuell zu verändern, sondern bei jeder Änderung der Ansicht ein *komplettes Rerendering der ganzen Seite* vornimmt. Dies tut die Bibliothek allerdings nur konzeptionell – tatsächlich werden bei jedem Rendering im Hintergrund die Unterschiede zwischen den aufeinander folgenden Zuständen der Ansicht in DOM-Operationen übersetzt, so dass die Darstellung der Seite auf effiziente Weise erfolgt.

React verfolgt ein *Komponenten-Modell*: Einzelne Teile einer Seite werden zu *React Components* zusammengefasst, die ihr Rendering zu jedem Zeitpunkt vollständig beschreiben. Durch hierarchische Schachtelung solcher Komponenten entsteht eine baumförmige Struktur, die bei der Darstellung in das *Document Object Model* (DOM) des Web-Browsers übersetzt wird. Ändert der Benutzer durch Interaktion den Zustand des Baumes (Übergang von v_0 zu v_1 auf der Zeitachse t in Abbildung 34), dann bewirkt dies ein Rerendering der betroffenen Teile – und damit einen Satz von Operationen, die den DOM-Baum im Browser modifizieren (rechter Teil von Abbildung 34).

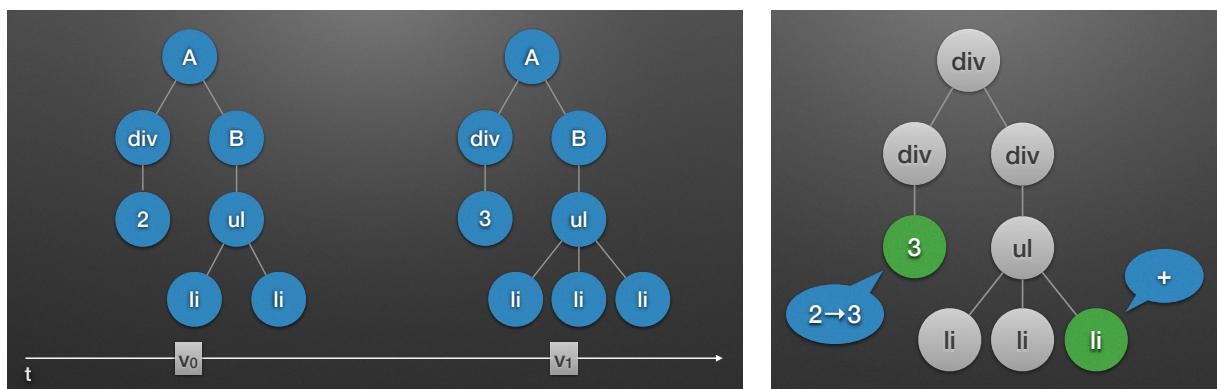


Abbildung 34: Rendering-Modell von React

Die Bibliothek **jQuery**⁴⁹ gehört wohl zu den bekanntesten und etabliertesten Bibliotheken im JavaScript-Ökosystem. Sie ist ein erweiterbares Universal-Tool und bietet eine reiche Sammlung an Hilfsmitteln und Funktionen für die Erstellung von JavaScript-Applikationen im Web-

⁴⁸ <http://facebook.github.io/react/index.html>

⁴⁹ <https://jquery.com>

Browser. Da der Hyperlike-Client beim Rendering fast ausschliesslich auf React aufbaut, kommt jQuery nur bei der Kommunikation mit der REST-Schnittstelle des Backends zum Einsatz. Eine direkte Modifikation des DOM-Baumes des Web-Browsers durch jQuery würde der Implementation durch React zuwiderlaufen und könnte zu einem undefinierten Zustand der Applikation führen.

Die Darstellung und Interaktion mit dem Karten-Dienst OpenStreetMap übernimmt die Bibliothek **Leaflet**⁵⁰ des Karten-Dienstleisters MapBox. Sie ist ebenfalls Open-Source und bietet eine reichhaltige API für die Darstellung von interaktiven Karten. So kann mit Leaflet ein aus den Kacheln von OpenStreetMap bestehende Grundebene mit programmatisch erstellten Ebenen überlegt werden. Diese wiederum können auf Aktionen des Benutzers reagieren und mittels *Handler* registrierte Funktionen aufrufen.

Wie bereits erwähnt, kann eine Modifikation des DOM-Baumes durch eine Bibliothek beim gleichzeitigen Einsatz von React zu ungewollten Nebeneffekten führen. Deshalb war es wichtig, die DOM-Bereiche von React und Leaflet strikte zu trennen. Dies geschieht am einfachsten, indem eine React-Komponente einen einzelnen DOM-Knoten als *Rendering-Target* für die externe Bibliothek vorsieht, diesen nach dem initialen Rendering nicht mehr modifiziert und vom Rerendering ausschliesst.

Für die grafische Gestaltung des Clients wurde die CSS-Bibliothek **Bootstrap**⁵¹ eingesetzt. Dieses von Twitter gegründete Projekt bietet eine reichhaltige und flexibel einsetzbare Sammlung von CSS-Klassen für das Styling von HTML. Zusätzlich verfügt Bootstrap über zusätzliche, auf jQuery aufbauende JavaScript-Komponenten, die jedoch wegen der bereits erwähnten möglichen Inkompabilität mit React nicht verwendet wurden. Neben ansprechender Typographie glänzt Bootstrap in erster Linie durch ein einfach bedienbares *Grid*-System zur Erstellung klar strukturierter Layouts.

6.4.2 Design und Umsetzung

Das Design unterscheidet – wie im Konzept in Abschnitt 5 beschrieben – klar zwischen dem Mechanismus zur Eingabe neuer Werte einer Erhebung und dem Mechanismus zur Darstellung der Resultate auf der Karte. Beim Entwurf der für die Umsetzung benötigten React-Komponenten bedeutet dies eine Aufteilung in *Collector*- und *Reporter*-Ansichten. Diese Namensgebung widerspiegelt die Implementation der Server-Komponente, die ebenfalls die Namespaces `collect` und `report` aufweist. Es sind diese beiden Komponenten, die die Rolle von *Adaptoren* der Konfiguration übernehmen, diese also zu UI-Komponenten für die Ein- und Ausgabe der Erhebung umsetzen.

⁵⁰ <http://leafletjs.com>

⁵¹ <http://getbootstrap.com>

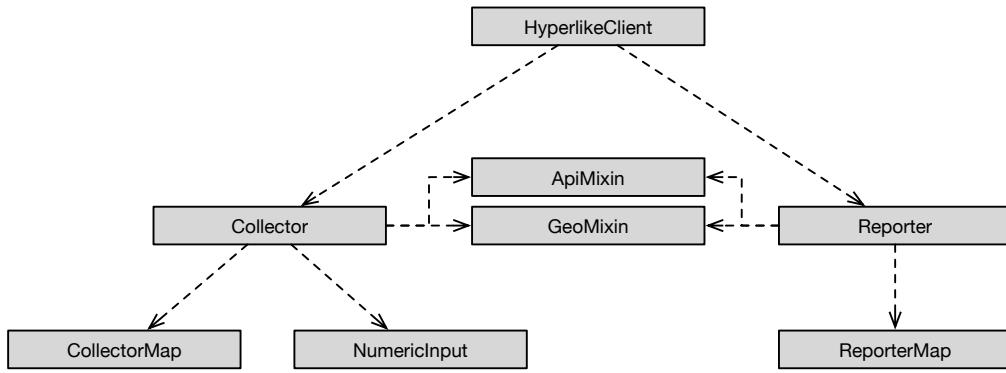


Abbildung 35: Diagramm der wichtigsten UI-Komponenten des Clients

Abbildung 35 zeigt die wichtigsten Komponenten des Clients. Die Komponente `HyperlikeClient` fasst die beiden Teile zu einer Applikation zusammen und übernimmt das *Routing*, also die Entscheidung, welche der beiden Ansichten dem Benutzer präsentiert wird. Sie tut dies aufgrund des *Pfades der URL* im Browser: Bei `/#/collect` wird der `Collector` angezeigt; dessen Bedienung durch den Benutzer führt zum Pfad `/#/report` – und damit zur Anzeige des `Reporters`.

Für die Bedienung der `Collector`-Ansicht werden zwei weitere Komponenten benötigt: `CollectorMap` zeigt einen kleinen Kartenausschnitt mit der aktuellen Position des Benutzers an. Pro konfiguriertes Feld der Eingabe-Tupel wird je eine Instanz von `NumericInput` eingeblendet. Dabei handelt es sich um einen Regler (HTML-Input vom Typ `range`) mit den konfigurierten Parametern für *Maximum*, *Minimum* und *Schrittweite* (siehe Abbildung 36). Bei der Entwicklung der Input-Komponente kam die *Wiederverwendbarkeit* von React-Komponenten zum Tragen, die die Programmierarbeit stark vereinfachte.

Der `Reporter` benötigt nur eine weitere Komponente: `ReporterMap`. Diese stellt einen Kartenausschnitt mit dem gesamten Gebiets der Erhebung dar und platziert darauf das *Raster* der Elemente, denen durch die Analyse ein Wert zugewiesen wurde. Die Rechtecke werden so eingefärbt wie durch der Konfiguration festgelegt, so dass ein übersichtliches Bild des Ergebnisses der Erhebung entsteht.

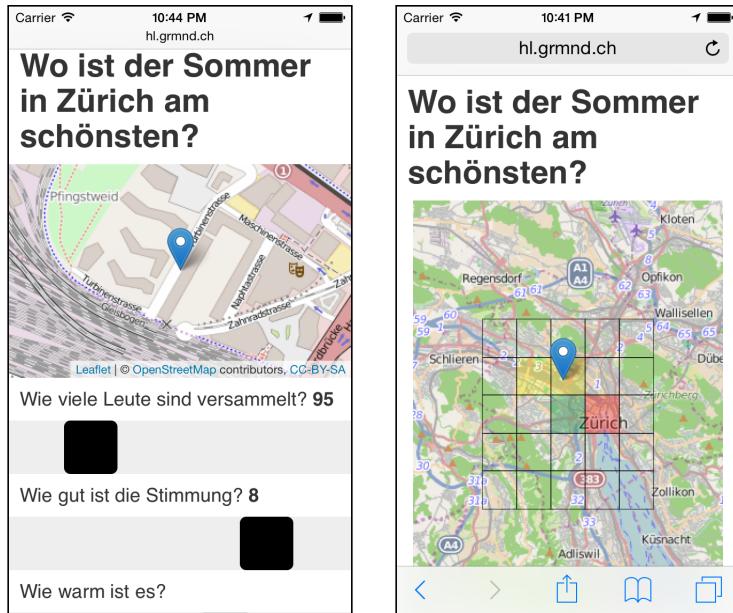


Abbildung 36: Hyperlike-Collector und -Reporter in Safari unter iOS

Funktionen, die von den beiden Hauptkomponenten benötigt werden, wurden nach dem DRY-Prinzip⁵² in so genannte *Mixins* ausgelagert. Diese übernehmen die Aufgabe von *abstrakten Komponenten*, wobei eine konkrete Komponente von mehreren Mixins ableiten kann. Bei React sind Mixins JavaScript-Objekte, auf deren Methoden über der Referenz `this` zugegriffen werden kann.

Das Mixin `GeoMixin` ermöglicht einer Komponente, über eine Abfrage der `navigator`-API des Web-Browsers die Koordinaten der momentanen Position des Geräts zu bestimmen. Das `ApiMixin` bietet Zugriff auf die REST-Schnittstellen der Server-Komponente. Damit können neue Eingaben, die Daten des Ergebnisses sowie die Konfiguration der Erhebung zwischen Client und Server ausgetauscht werden.

6.4.3 Werkzeuge

Während die Ausführung der Client-Komponente von der JavaScript-Engine des Web-Browsers übernommen wird, kommt als Laufzeitumgebung der Entwicklungswerkzeuge **Node**⁵³ zum Einsatz. Dabei handelt es sich um die aus dem Web-Browser «Chrome» herausgelöste JavaScript-Engine «V8» von Google⁵⁴. Node ist als Laufzeitumgebung für Server-Applikationen ebenso beliebt wie für Desktop-Programme und Kommandozeilen-Tools. Die Renaissance von JavaScript als dominante Programmiersprache des World Wide Web ist nicht zuletzt auf die Populärität und Performanz von Node zurückzuführen.

⁵² «Don't Repeat Yourself»

⁵³ <https://nodejs.org>

⁵⁴ <https://developers.google.com/v8>

Für das Management der Abhängigkeiten zu externen Bibliotheken kommt bei Node der **Node Package Manager** NPM zum Einsatz, der durch eine JSON-Datei konfiguriert wird und festlegbare Versionen von JavaScript-Komponenten herunterlädt und dem Projekt zugänglich macht. Bibliotheken, die sich nicht für die Verwendung mit NPM eignen, wurden als statische Ressourcen eingebunden.

Der Autor setzte das Build-Tool **Webpack** ein, um den Quelltext der Client-Komponente zusammen mit den gewünschten Bibliotheken zu einer Datei `bundle.js` zu verschmelzen. Diese Datei wird anschliessend komprimiert und zusammen mit einer minimalen HTML-Datei von der Server-Komponente an die Benutzer ausgeliefert mit dem Ziel, die Applikation im Web-Browser auszuführen.

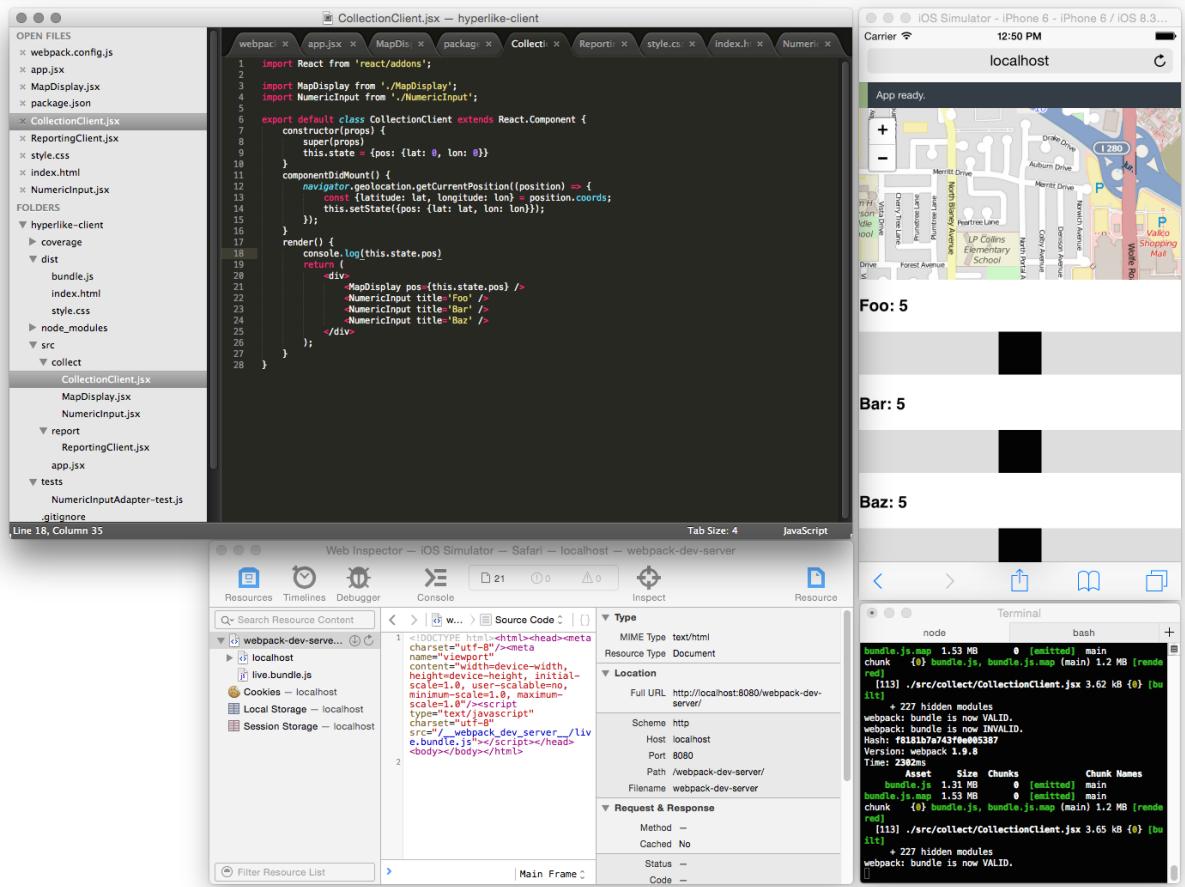


Abbildung 37: Entwicklungsumgebung für Client mit (im Uhrzeigersinn) Editor «Sublime Text», Web-Browser «Mobile Safari» in iOS Simulator, «Safari Web Inspector» und Terminal mit laufendem `webpack-dev-server`

Neben den Build-Tools setzte der Autor eine Reihe von Werkzeugen für die Entwicklung mobiler Web-Applikationen ein, darunter die Applikation **iOS Simulator**, die ein beliebiges Mobilgerät aus dem Hause Apple simuliert. Einerseits ermöglicht diese Software das bequeme

Entwickeln für den mobilen Browser Safari am Desktop, anderseits erlaubt sie eine Verbindung zum Debugger der Desktop-Variante des Browsers, so dass das eine Inspektion des laufenden Programms möglich ist.

Die Mehrheit der Entwicklungsarbeit erfolgte allerdings im Web-Browser **Google Chrome**, der mit mächtigen Hilfsmitteln für die Web-Programmierung ausgestattet ist. Als Server für die Artefakte des Clients diente die Applikation **webpack-dev-server**, welche die Quelltext-Dateien auf Änderungen zu überwachen vermag, die Applikation bei Bedarf neu baut und sie im verbundenen Web-Browser automatisch neu lädt.

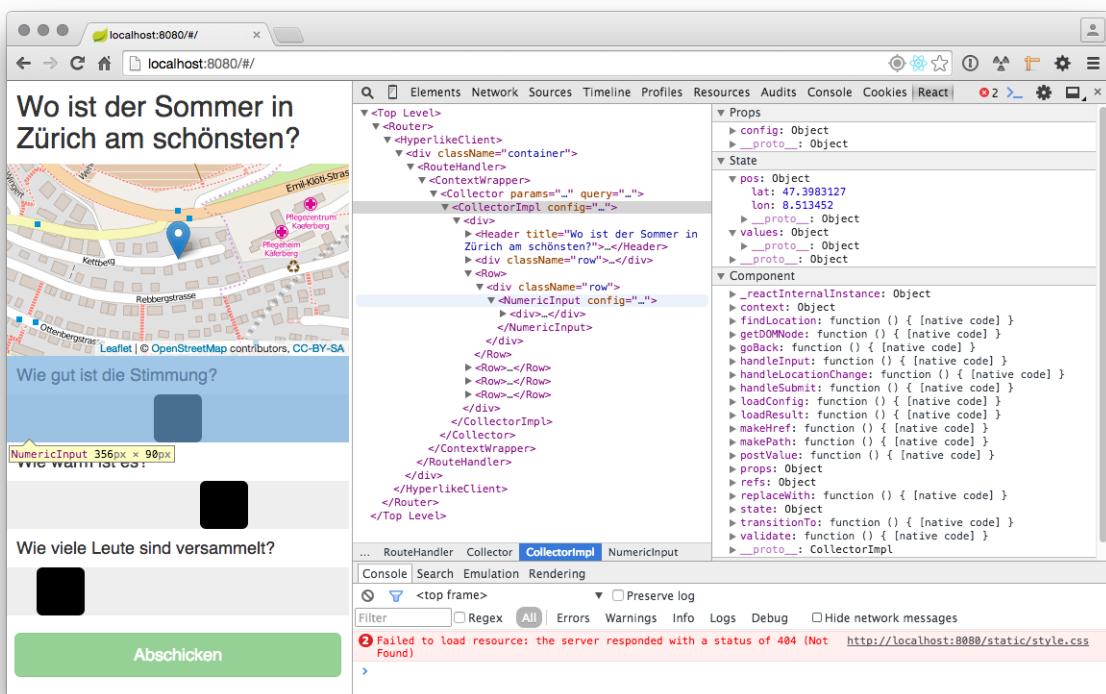


Abbildung 38: Entwicklungsumgebung für React-Komponenten in Google Chrome

6.4.4 Automatisierte Tests

Auch bei der Entwicklung der Client-Komponente kamen automatisierte Tests zum Einsatz, die bei jedem Lauf des Build-Prozesses ausgeführt werden und so einen gewissen Schutz vor Fehlern in der Programmlogik bieten. Allerdings war die Vorgehensweise umgekehrt: Zuerst wurden die React-Klassen erstellt und erst *danach* die Test formuliert, die das programmierte Verhalten prüfen. Da es sich bei den Komponenten des Clients um UI-Elemente handelt, die nicht von Anfang an genau spezifiziert waren, sondern bei der Entwicklung schrittweise aufgebaut wurden, erschien dieses Vorgehen dem Autor als das zweckmässigere.

Für das automatisierte Testing kam das Test-Framework **Jest** zum Einsatz, das ebenfalls von Facebook stammt und von React empfohlen wird. React-Klassen eignen sich besonders gut für verlässliches Testing: Eine Komponente muss nicht unbedingt in das Fenster eines Web-Browsers gerendert werden, sondern sie kann innerhalb des mit React ausgelieferten Klasse TestUtils instanziert und auf Korrektheit untersucht werden. Die erspart die oft umständlichen Test-Setups, die bei anderen JavaScript-Bibliotheken für die Durchführung des Testings notwendig sind.

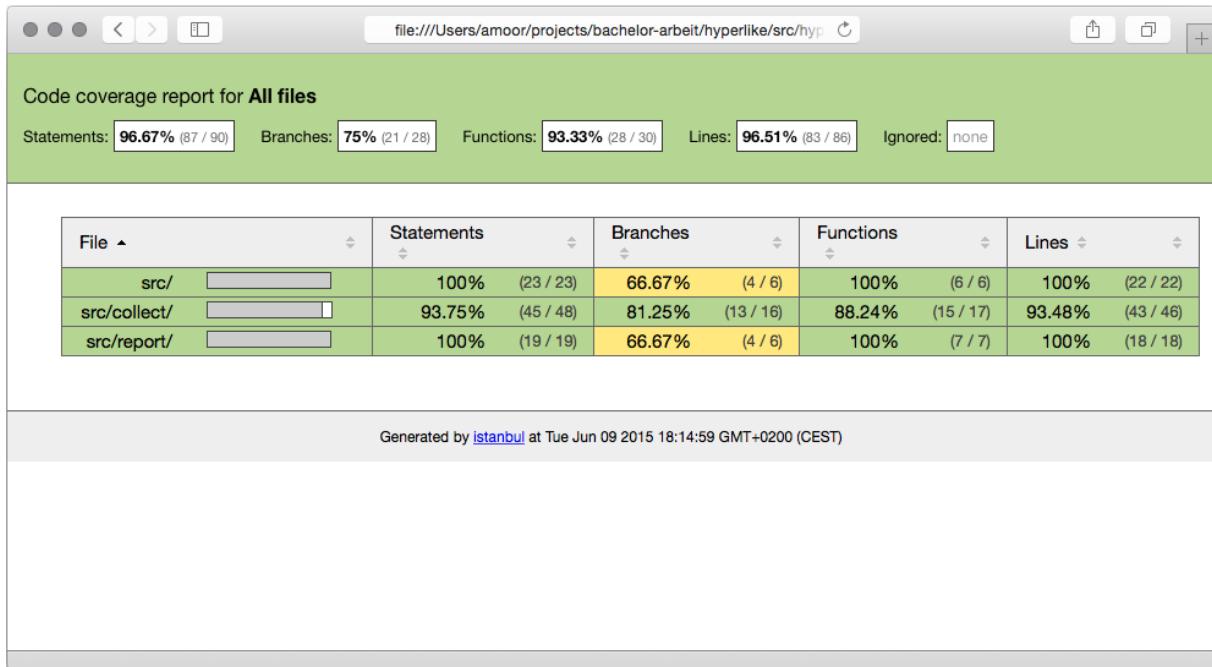


Abbildung 39: Test Coverage der Client-Komponente

Jest exportiert auf Wunsch einen Bericht über die Test-Abdeckung als HTML-Datei. Abbildung 39 zeigt das Resultat für den Hyperlike-Client. Die gelb markierte, nicht vollständige Abdeckung ist darauf zurückzuführen, dass der für die Client-Komponente geschriebene Quelltext vor der Erstellung des bundle.js-Artefakts transformiert werden muss – dies führt zu zusätzlichen Codezeilen, die durch den anschliessenden Test nicht erfasst werden. Ebenfalls nicht getestet wurde die Kommunikation mit der REST-Schnittstelle des Backends, da diese schon im Rahmen der Integrations-Tests der Server-Komponente untersucht wurde.

6.5 Installation und Deployment

Wie in Abschnitt 6.3.4 beschrieben, produziert das Maven-Plugin von Spring Boot ein in sich geschlossenes, lauffähiges JAR-Archiv, das direkt ausgeführt werden kann. Einzige Voraussetzungen sind ein installiertes Java Runtime Environment sowie eine laufende MongoDB-Instanz.

Detaillierte Anleitungen für Build und Deployment sind auf der beigelegten DVD in einer *Read-Me*-Datei enthalten. Da der im Rahmen dieser Arbeit entstehende Prototyp nicht für den produktiven Einsatz gedacht ist, verzichtete der Autor auf die Installationsskripte, welche für eine Integration mit dem Betriebssystem in Form eines *Dienstes* notwendig wären.

```
Terminal
bash mongod java

.
.
.
:: Spring Boot :: (v1.2.3.RELEASE)

2015-06-02 20:51:50.443 INFO 62334 — [           main] c.g.h.HyperlikeServerApplication      : Starting HyperlikeServerApplication v1.0.0-SNAPSHOT
on macbook.local with PID 62334 (/Users/amoer/projects/bachelor-arbeit/hyperlike/src/hyperlike-server/target/hyperlike-server-1.0.0-SNAPSHOT.jar started by amoer in /Users/amoer/projects/bachelor-arbeit/hyperlike/src/hyperlike-server/target)
2015-06-02 20:51:50.512 INFO 62334 — [           main] s.AnnotationConfigEmbeddedWebApplicationContext : Refreshing org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@7b90a264: startup date [Tue Jun 02 20:51:50 CEST 2015]; root of context hierarchy
2015-06-02 20:51:51.348 INFO 62334 — [           main] o.s.b.f.s.DefaultListableBeanFactory : Overriding bean definition for bean 'beanNameViewResolver': replacing [Root bean: class [null]; scope: abstract=false; lazyInit=false; autowireMode=3; dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=org.springframework.boot.autoconfigure.web.ErrorMvcAutoConfiguration$WhitelabelErrorViewConfiguration; factoryMethodName=beanNameViewResolver; initMethodName=null; destroyMethodName=(inferred); defined in class path resource [org/springframework/boot/autoconfigure/web/ErrorMvcAutoConfiguration.class]] with [Root bean: class [null]; scope: abstract=false; lazyInit=false; autowireMode=3; dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=org.springframework.boot.autoconfigure.web.WebMvcAutoConfiguration$WebMvcAutoConfigurationOnAdapter; factoryMethodName=beanNameViewResolver; initMethodName=null; destroyMethodName=(inferred); defined in class path resource [org/springframework/boot/autoconfigure/web/WebMvcAutoConfiguration$WebMvcAutoConfigurationAdapter.class]]]
2015-06-02 20:51:52.362 INFO 62334 — [           main] e.j.JettyEmbeddedServletContainerFactory : Server initialized with port: 8080
2015-06-02 20:51:52.366 INFO 62334 — [           main] org.eclipse.jetty.server.Server       : jetty-9.2.9.v20150224
2015-06-02 20:51:52.576 INFO 62334 — [           main] /                                : Initializing Spring embedded WebApplicationContext
2015-06-02 20:51:54.228 INFO 62334 — [           main] o.s.b.c.e.ServletRegistrationBean    : Mapping servlet: 'dispatcherServlet' to [/]
2015-06-02 20:51:54.232 INFO 62334 — [           main] o.s.b.c.embedded.FilterRegistrationBean : Mapping filter: 'characterEncodingFilter' to: [//*]
2015-06-02 20:51:54.233 INFO 62334 — [           main] o.s.b.c.embedded.FilterRegistrationBean : Mapping filter: 'hiddenHttpMethodFilter' to: [//*]
2015-06-02 20:51:54.590 INFO 62334 — [           main] o.e.jetty.server.handler.ContextHandler : Started o.s.b.c.e.JettyEmbeddedWebAppContext@6c4ee6d{/null,AVAILABLE}
2015-06-02 20:51:54.590 INFO 62334 — [           main] org.eclipse.jetty.server.Server       : Started @6193ms
2015-06-02 20:51:55.027 INFO 62334 — [           main] c.g.h.service.config.ConfigServiceImpl : Config[title='Wo ist der Sommer in Zürich am schönen?', bounds=Bounds{southWest=Position{lat=47.330937, lon=-8.462729}, northEast=Position{lat=47.41371, lon=0.572936}}, analysis=AnalysisConfig{grid=5, method='mean'}, input=[InputConfig{question='Wie hoch ist der Frauenanteil?', min=0.0, max=100.0, step=5.0, initial=50.0}, InputConfig{question='Wie warm ist es?', min=0.0, max=35.0, step=1.0, initial=23.0}], InputConfig{question='Wie viele Leute sind versammelt?', min=0.0, max=500.0, step=5.0, initial=50.0}], display=DisplayConfig{metrics='mean', quorum=1, colors=[#009933, #FFFF00, #FF0000]}}
2015-06-02 20:51:56.195 INFO 62334 — [           main] o.s.j.e.a.AnnotationMBeanExporter     : Registering beans for JMX exposure on startup
2015-06-02 20:51:56.216 INFO 62334 — [           main] /                                : Initializing Spring FrameworkServlet 'dispatcherServlet'
2015-06-02 20:51:56.388 INFO 62334 — [           main] o.eclipse.jetty.server.ServerConnector : Started ServerConnector@2fe22120[HTTP/1.1]{0.0.0.0:8080}
2015-06-02 20:51:56.398 INFO 62334 — [           main] .s.b.c.e.JettyEmbeddedServletContainer : Jetty started on port(s) 8080 (http/1.1)
2015-06-02 20:51:56.400 INFO 62334 — [           main] c.g.h.HyperlikeServerApplication      : Started HyperlikeServerApplication in 6.49 seconds
(JVM running for 7.003)
```

Abbildung 40: Erfolgreiches Deployment der Applikation auf einem Linux-Host

6.6 Fazit

Durch den Einsatz etablierter Technologien ist es gelungen, einen Prototypen von Hyperlike zu erstellen. Dieser erfüllt alle funktionalen Anforderungen aus Abschnitt 3.4.1 mit nur geringen Einschränkungen und dient als *Proof of Concept* des in dieser Arbeit entworfenen Systems.

Die im Konzept definierte Client-/Server-Architektur wurde auf die gewählten Technologien übertragen. Der Client ist eine im Web-Browser ausgeführte JavaScript-Applikation, die auf dem «React»-Framework aufbaut und Komponenten für Dateneingabe und Visualisierung auf einer Karte umsetzt. Bei der Server-Komponente handelt es sich um eine Java-Applikation, die mit Hilfe von «Spring Boot» entwickelt wurde.

Die gesamte Entwicklungsarbeit wurde unterstützt durch Werkzeuge für das Management von Abhängigkeiten und das Durchführen eines stabilen Build-Prozesses. Um die Qualität der Software sicherzustellen, wurde grossen Wert gelegt auf einen breiten Einsatz automatisierter Unit Tests.

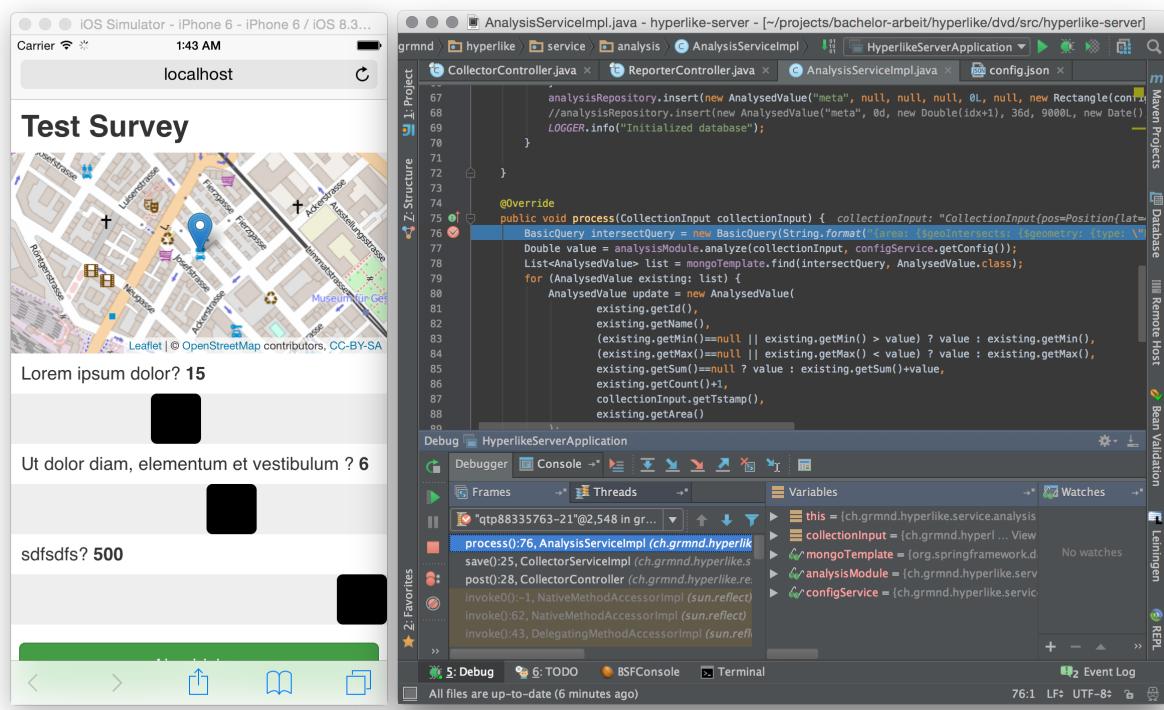


Abbildung 41: Zusammenspiel von Client- und Server-Komponente während der Entwicklung

7 Testing

Die Entwicklung des Prototypen wurde begleitet von ausführlichen *Unit Tests*. Diese werden bei der Herstellung der Artefakte von Client- und Server-Komponente automatisiert ausgeführt und prüfen die Applikation auf Programmierfehler, indem sie für einzelne Teile der Logik vordefinierte Ein- und Ausgaben vergleichen. Durch die Durchführung von Unit Tests können jedoch lediglich *technische Fehler* vermieden werden – ob die Applikation *inhaltlich* die Erwartungen erfüllt, kann durch sie nicht festgestellt werden.

In diesem Teil der Arbeit geht es darum, den Prototypen auf Erfüllung der *Akzeptanzkriterien* zu überprüfen. Dies geschieht durch die Ausführung der in Abschnitt 3.5 festgelegten, von den Anforderungen abgeleiteten *Abnahmetests*. Über die Durchführung der Tests wird ein *Testprotokoll* geführt, das als Beleg dient für die erfolgreiche Absolvierung.

7.1 Testumgebung

Vor der Durchführung der Abnahmetests wird die Testumgebung vorbereitet und alle Komponenten installiert. Die detaillierten Schritte der Installation sind auf der dieser Arbeit beiliegenden DVD enthalten. Folgende Systeme bilden die *Testumgebung* des Systems:

- Für die Server-Komponente wird in einer virtuellen Maschine das Betriebssystem «Ubuntu Server 14.04» aufgesetzt und darauf die notwendigen Debian-Pakete installiert. Der aktuelle Build des Prototypen wird in das Heimverzeichnis eines Benutzers ohne besondere Systemprivilegien kopiert.
- Für die Client-Komponente wird im für Mac OS erhältlichen «iOS Simulator» ein iPhone 5s mit der App «Mobile Safari» gestartet.

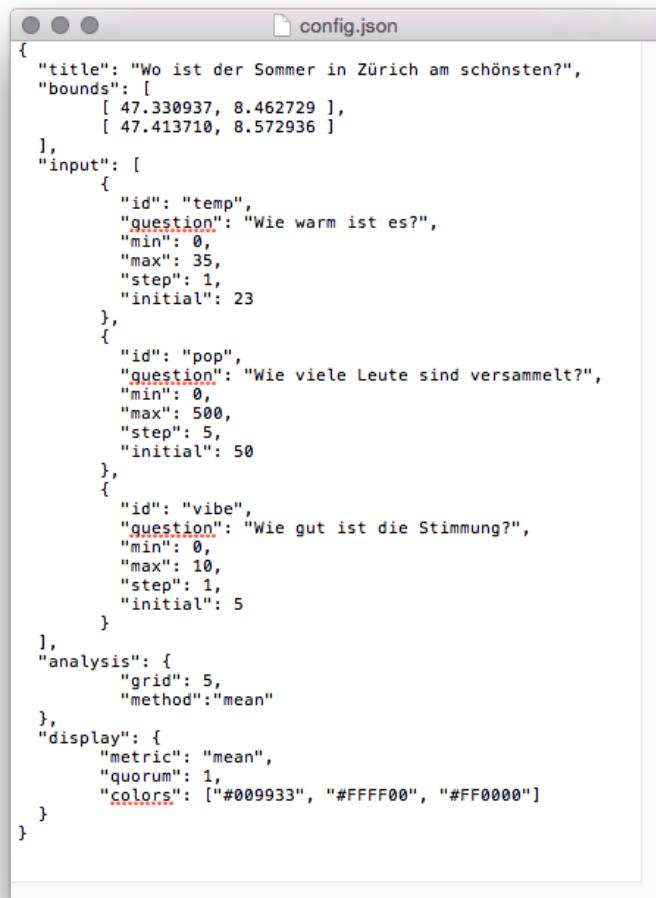
7.2 Testprotokolle

Die einzelnen Teile des Testprotokolls enthalten alternierend eine Wiederholung der in den Akzeptanzkriterien festgelegten Schritte und die bei der Durchführung tatsächlich entstandenen Resultate.

7.2.1 AT-01: System konfigurieren

Konfiguration gemäss Dokumentation der Implementation vornehmen.

Im Verzeichnis des angemeldeten Benutzers wird eine Konfigurationsdatei mit dem Namen config.json angelegt. Das Format des Inhalts ist JSON, als Dokumentation des Inhalts muss für den Prototypen ein Beispiel genügen (dieses befindet sich ebenfalls auf der beiliegenden DVD). Die Konfiguration (siehe) enthält folgende Teile: Titel, Grenzen, Raster, Input und Anzeige.



```
{  
    "title": "Wo ist der Sommer in Zürich am schönsten?",  
    "bounds": [  
        [ 47.330937, 8.462729 ],  
        [ 47.413710, 8.572936 ]  
    ],  
    "input": [  
        {  
            "id": "temp",  
            "question": "Wie warm ist es?",  
            "min": 0,  
            "max": 35,  
            "step": 1,  
            "initial": 23  
        },  
        {  
            "id": "pop",  
            "question": "Wie viele Leute sind versammelt?",  
            "min": 0,  
            "max": 500,  
            "step": 5,  
            "initial": 50  
        },  
        {  
            "id": "vibe",  
            "question": "Wie gut ist die Stimmung?",  
            "min": 0,  
            "max": 10,  
            "step": 1,  
            "initial": 5  
        }  
    ],  
    "analysis": {  
        "grid": 5,  
        "method": "mean"  
    },  
    "display": {  
        "metric": "mean",  
        "quorum": 1,  
        "colors": ["#009933", "#FFFF00", "#FF0000"]  
    }  
}
```

Abbildung 42: Konfiguration des Systems für AT-01

Vorbedingungen: System gestoppt, noch keine Messwerte erfasst

Diese sind für die Testumgebung erfüllt.

System starten.

Das Kommando `java -jar hyperlike-server-1.0.0-SNAPSHOT.jar` wird ausgeführt, die Applikation startet (siehe Abbildung 43 links).

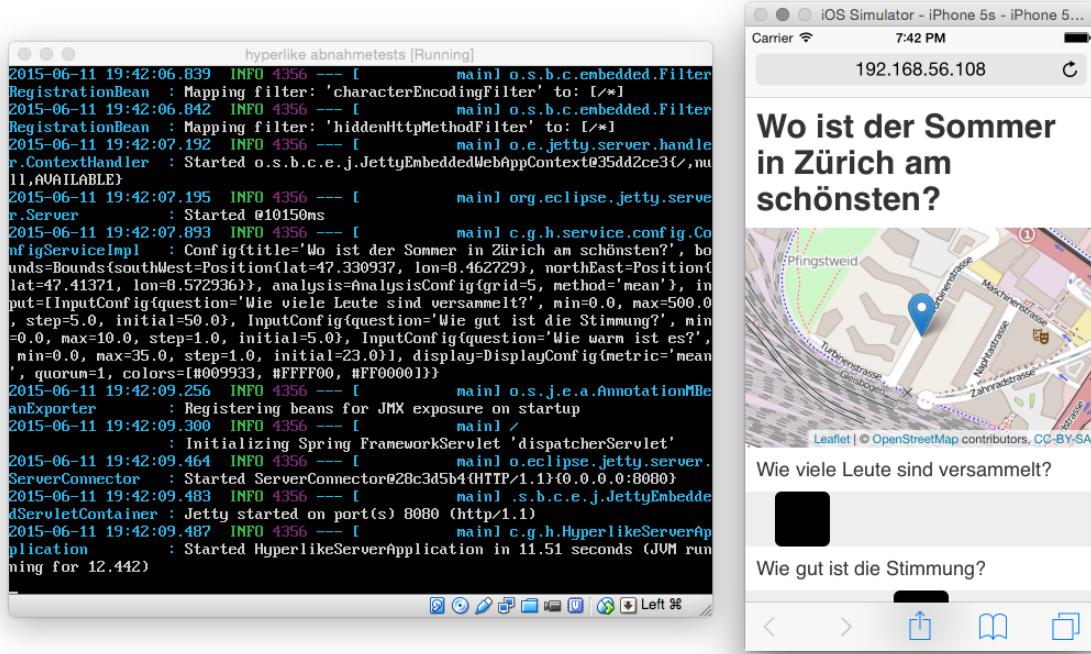
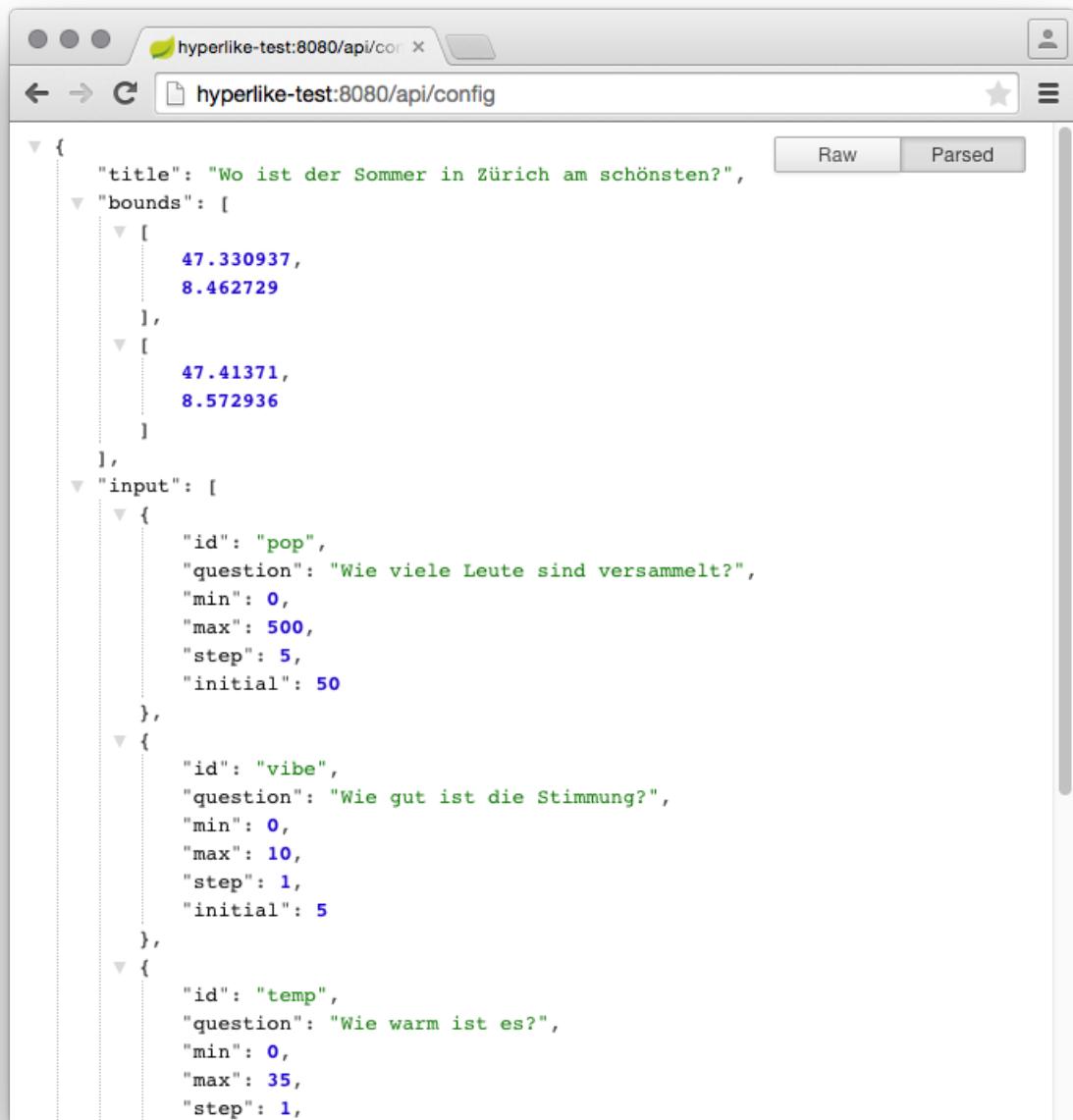


Abbildung 43: Gestartete Server-Applikation mit verbundenem Client

Mit dem Web-Browser die URL der Konfigurations-Ressource ansteuern.

Die URL der Konfigurationsressource lautet in diesem Fall `http://hyperlike-test:8080/api/config`. Durch Aufrufen dieser Adresse wird die Konfiguration der Erhebung wie in Abbildung 44 ausgegeben. Es bleibt anzumerken, dass es sich bei der Ausgabe nicht einfach um den Inhalt der Datei `config.json` handelt, sondern um die serialisierte Form des Model-Objekts, das alle Adapter zur Umsetzung verwenden. Dennoch stimmen die Inhalte von Datei und Ausgabe überein.



The screenshot shows a web browser window with the URL `hyperlike-test:8080/api/config`. The page displays a JSON object representing a survey configuration. The JSON structure includes a title, bounds (geographical coordinates), and input fields (questions with ranges and steps). The browser interface has tabs for 'Raw' and 'Parsed' data.

```
{
  "title": "Wo ist der Sommer in Zürich am schönsten?",
  "bounds": [
    [
      [47.330937, 8.462729],
      [47.41371, 8.572936]
    ],
    [
      [47.330937, 8.462729],
      [47.41371, 8.572936]
    ]
  ],
  "input": [
    {
      "id": "pop",
      "question": "Wie viele Leute sind versammelt?",
      "min": 0,
      "max": 500,
      "step": 5,
      "initial": 50
    },
    {
      "id": "vibe",
      "question": "Wie gut ist die Stimmung?",
      "min": 0,
      "max": 10,
      "step": 1,
      "initial": 5
    },
    {
      "id": "temp",
      "question": "Wie warm ist es?",
      "min": 0,
      "max": 35,
      "step": 1,
      "initial": 5
    }
  ]
}
```

Abbildung 44: Ausgabe der Konfiguration der Erhebung

Mit dem Web-Browser die URL der Erhebung ansteuern.

Die URL der Erhebung lautet in diesem Fall `http://hyperlike-test:8080`. Beim Aufruf dieser URL durch «Mobile Safari» erscheint die Eingabemaske für einen neuen Beobachtungswert (Abbildung 43 rechts).

Damit sind alle Schritte des Tests **erfüllt**.

7.2.2 AT-02: Anpassung der Benutzeroberfläche

Vorbedingung: AT-01 wurde durchgeführt.

Erfüllt.

Die Anpassungen am Aussehen der Benutzeroberfläche gemäss Dokumentation der Implementation vornehmen.

Im Verzeichnis des angemeldeten Benutzers wird ein Verzeichnis `static` angelegt. Die Server-Komponente ist so programmiert, dass darin nach zusätzlichen statischen Ressourcen für das *Styling* der Applikation gesucht wird. Die Client-Komponente bindet automatisch den Pfad `static/style.css` als Quelle für CSS-Informationen ein. So kann das Aussehen des Clients durch den Anbieter angepasst werden.

Für den Abnahmetest soll der Hintergrund der Client-Oberfläche rot und die Schrift weiss eingefärbt werden. Im erstellen Verzeichnis wird also die Datei `style.css` mit folgendem Inhalt angelegt:

```
body {  
    background-color: red;  
    color: white;  
}
```

Eine erneute Abfrage der URL der Hyperlike-Instanz führt dazu, dass die Ausgabe gegenüber AT-01 tatsächlich verändert erfolgt: roter Hintergrund, weisse Schrift (siehe Abbildung 45). Damit sind alle Schritte des Tests **erfüllt**.



Abbildung 45: Angepasste Client-Oberfläche mit rotem Hintergrund und weisser Schrift

7.2.3 AT-03: Erfassung der Positionsdaten einer Eingabe

Vorbedingung: AT-01 wurde durchgeführt.

Erfüllt.

Mit dem Web-Browser eines Mobilgeräts die URL der Applikation ansteuern.

Erneut wird mit «Mobile Safari» die URL <http://hyperlike-test:8080> aufgerufen. Die Eingabemaske erscheint.

Die Koordinaten der aktuellen Position werden ausgelesen.

Der Browser bittet den Benutzer um Erlaubnis, dessen aktuelle Position auslesen zu dürfen. Wird diese erteilt, dann wird diese danach auf der Karte der Eingabemaske angezeigt.

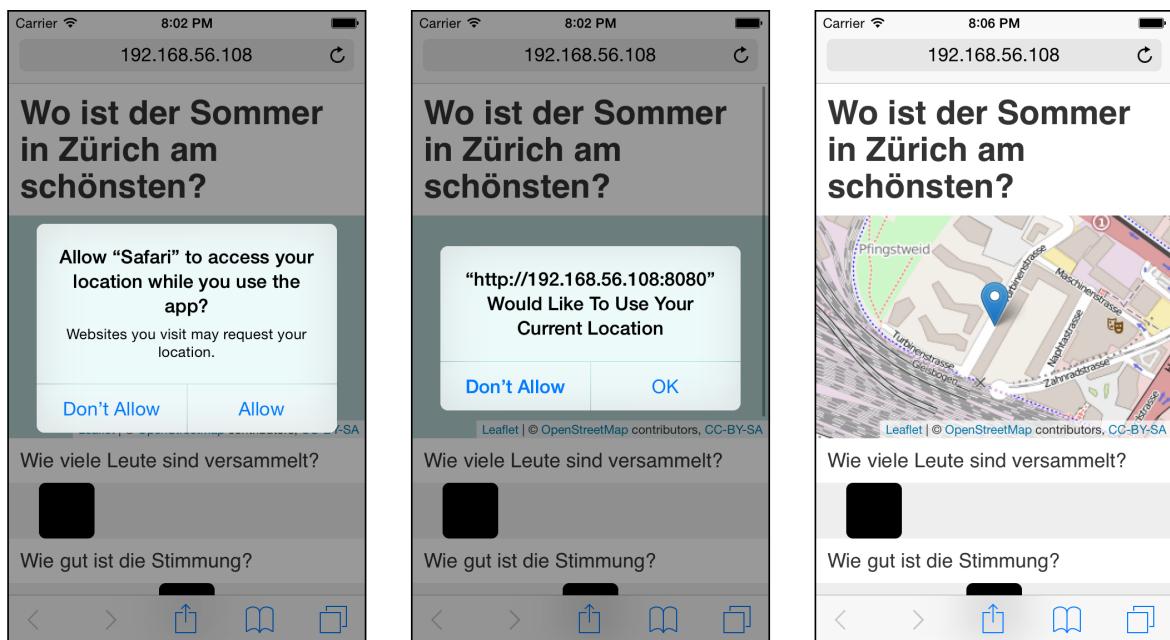


Abbildung 46: Dialoge und Resultat der automatischen Positionsbestimmung

Damit sind alle Schritte des Tests **erfüllt**.

7.2.4 AT-04: Dateneingabe mittels grafischer Benutzeroberfläche

Vorbedingung: AT-03 wurde durchgeführt, die grafische Benutzeroberfläche wird angezeigt.

Erfüllt.

Für jeden der konfigurierten Typen erscheint unterhalb der aktuellen Position ein entsprechendes UI-Element.

In der Konfiguration wurden drei Eingabe-Typen definiert (siehe Abbildung 42). Diese erscheinen als entsprechende UI-Elemente in der Eingabemaske (siehe Abbildung 47). Durch Herumziehen der Regler wird klar, dass Maximum, Minimum und Schrittweite tatsächlich mit den Parametern der Konfiguration übereinstimmen.

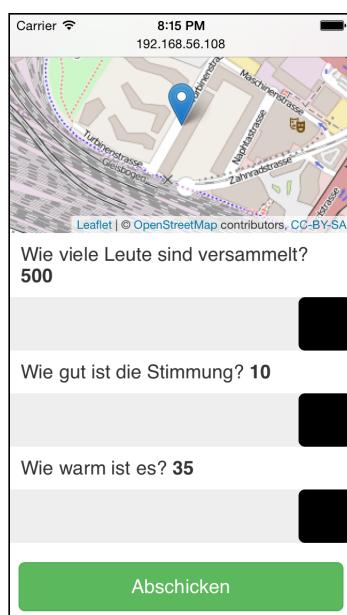


Abbildung 47: UI-Elemente der Eingabemaske entsprechen der Konfiguration

Die Eingabemasken der einzelnen Typen werden so bedient, dass sie den gewünschten Beobachtungswert abbilden.

Erneut werden die Regler bedient, so dass sie einen Beobachtungswert widerspiegeln (Abbildung 48 links).

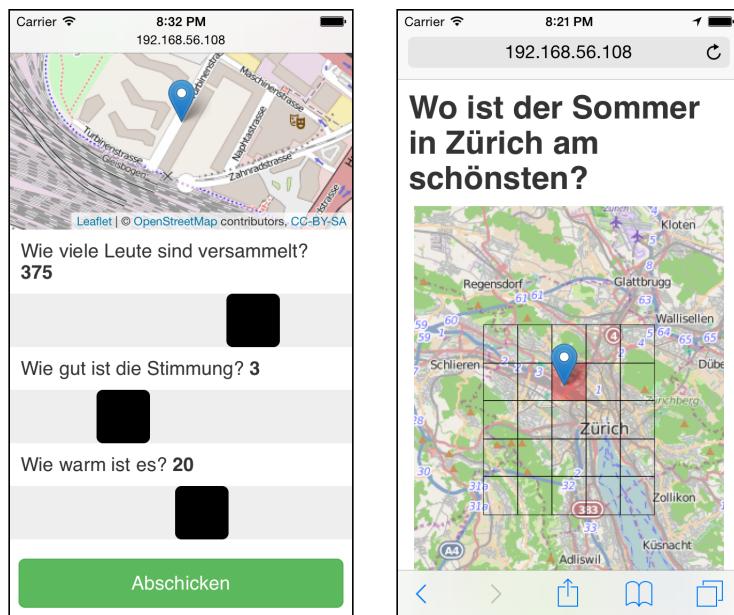


Abbildung 48: Nach der Eingabe wird das Resultat angezeigt

Durch Bedienung des UI-Elements «Abschicken» wird die Übermittlung ausgelöst.

Das erfasste Tupel wird übermittelt, und die Karte der Ansicht für die Datenausgabe erscheint (Abbildung 48 rechts). Dass der Wert tatsächlich verarbeitet wurde, zeigt ein Blick in die Datenbank, die jetzt in der Collection `collectionInput` einen einzigen, den Eingaben entsprechenden Wert enthält.

Damit sind alle Schritte des Tests **erfüllt**.

7.2.5 AT-05: Ausgabe einer Resultatübersicht mittels grafischer Benutzeroberfläche

Vorbedingung: AT-04 wurde durchgeführt, die grafische Benutzeroberfläche wird angezeigt.
Erfüllt.

Mit dem Web-Browser die Resultate-URL der Applikation ansteuern.

Die Resultate-URL der Erhebung lautet in diesem Fall <http://hyperlike-test:8080/#/report>. Eine Abfrage der URL führt dazu, dass die Karte mit dem vorläufigen Ergebnis der Erhebung angezeigt wird.



Abbildung 49: Karte mit dem Resultat der Erhebung nach einer Eingabe

Das Segment, in dessen Grenzen die Eingabe in AT-04 erfolgte, ist als einziges eingefärbt – dies bedeutet der letzte (und einzige) Eingabewert ist in die Bestimmung des Wertes dieses Segments eingeflossen.

Ein Blick auf die Färbung der Karte zugrundeliegenden Datensätze zeigt, dass die Färbung korrekt ist. Durch Aufruf der URL <http://hyperlike-test:8080/api/report> erfolgt der Zugriff auf die Rohdaten des Ergebnisses im Format JSON (siehe Abbildung 50).

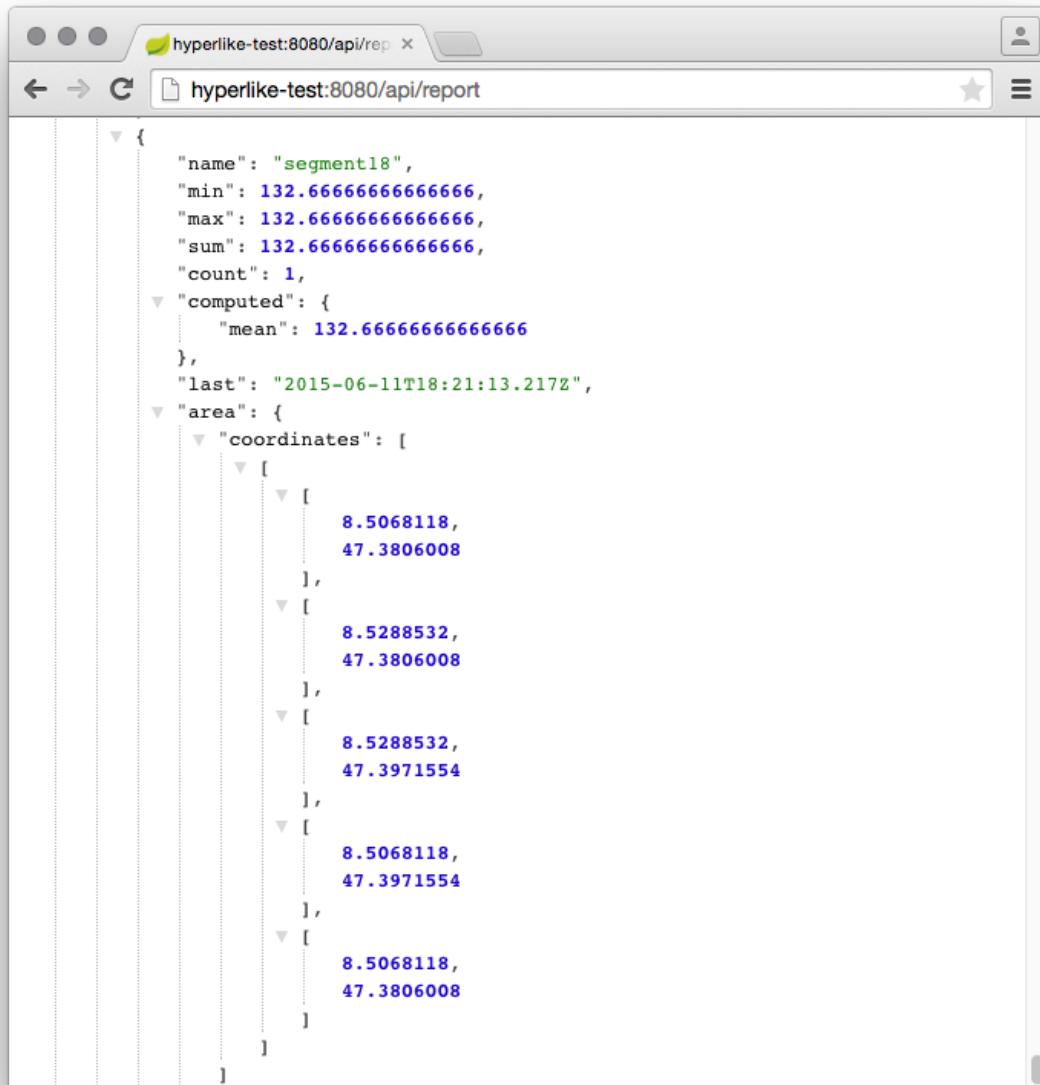


Abbildung 50: Ausgabe des Ergebnisses im Format JSON

Für das auf der Karte eingefärbte Segment Nummer 18 enthält das Resultat den Mittelwert 132.666 – dies entspricht dem Mittelwert der vorgenommenen Eingabe $(375 + 3 + 20) / 3 = 132.666$. Die Konfiguration bestimmt eine Kolorierung der Segment durch drei Farben: Grün, Gelb und Rot. Da der eingegebene Wert der einzige ist, wird er richtigerweise dem oberen Drittel der Werte zugerechnet, und das Segment wird rot eingefärbt.

Damit sind alle Schritte des Tests **erfüllt**.

7.3 Fazit

Die Abnahmetests waren alle erfolgreich. Somit erfüllt der Prototyp alle funktionalen Anforderungen mit den in Abschnitt 6 erwähnten Einschränkungen. Die nichtfunktionalen Anforderungen wurden nicht umgesetzt und deshalb auch nicht überprüft.

Tabelle 11 fasst die Ergebnisse der Abnahmetests zusammen: Die in Abschnitt 3.5 formulierten *erwarteten Resultate* werden alle als erfüllt markiert.

Tabelle 11: Erfüllung der erwarteten Ergebnisse der Akzeptanztests

AT-01: System konfigurieren	
Die aktuelle Konfiguration wird geladen.	●
Die Einträge in der angezeigten Tabelle müssen mit den Angaben in der Konfigurationsdatei übereinstimmen	●
Die Eingabemaske für eine neue Eingabe wird angezeigt	●
AT-02: Anpassung der Benutzeroberfläche	
Das Aussehen der Benutzeroberfläche muss die vorgenommenen Anpassungen widerspiegeln	●
AT-03: Erfassung der Positionsdaten einer Eingabe	
Die konfigurierte graphische Oberfläche für die Eingabe wird angezeigt	●
Die aktuelle Position wird auf einer Karte angezeigt	●
AT-04: Dateneingabe mittels grafischer Benutzeroberfläche	
Der <i>Titel</i> des UI-Elements entspricht der konfigurierten Frage	●
Die Eingabemaske des UI-Elements ist so gestaltet, dass sie die Konfiguration widerspiegelt bezüglich <i>Minimum</i> , <i>Maximum</i> und <i>Schrittweite</i>	●
Der Messwert wird übermittelt und verarbeitet	●
Die Maske für die Datenausgabe wird angezeigt	●
AT-05: Ausgabe einer Resultatübersicht mittels grafischer Benutzeroberfläche	
Ein Kartenausschnitt mit dem für die Erhebung konfigurierten geographischen Bereich wird angezeigt	●
Der Kartenausschnitt ist so eingefärbt, dass er die analysierten Werte sichtbar macht	●

8 Schlussbemerkungen

Die Erarbeitung der vorliegenden Bachelor-Arbeit erstreckte sich über die erste Hälfte des Jahres 2015: vom Kick-Off-Meeting am 25. Februar bis zur Abgabe am 25. Juni. In dieser Zeit war der Autor intensiv mit der Konzeption, der Formulierung des Konzepts und mit der Ausprogrammierung des Prototypen beschäftigt.

In diesem Abschnitt sollen der Ablauf und die Ergebnisse dieses Projekts zusammengefasst und kritisch gewürdigt werden. Er bildet den Abschluss des Hauptteils dieser Dokumentation.

8.1 Generelles Fazit

Das Projekt «Hyperlike» ist erfolgreich verlaufen. Die ursprüngliche Idee einer wiederverwendbaren, einfach zu installierenden Applikation für das Sammeln, Analysieren und Teilen von Beobachtungswerten konnte auf Tauglichkeit untersucht, konzipiert und durch einen funktionierenden Prototypen in der Praxis bestätigt werden.

Die Durchführung des Projekts umfasste eine sachlich begründete Reihe von aufeinander aufbauenden Etappen. Ausgehend von einem bestehenden Bedürfnis wurde die *Idee der Plattform* formuliert, welche die Lösung eines konkreten Problems zu einem anpassbaren, wiederverwendbaren Werkzeug verallgemeinert. Diese wurde durch weitere denkbare Anwendungsbeispiele erläutert. Danach wurde die Idee durch eine Reihe von *Use Cases* und daraus abgeleiteten *Anforderungen* konkretisiert. Anschliessend wurde das Feld *bestehender Gesamt- und Teillösungen* erkundet und in einer Marktanalyse den Anforderungen gegenübergestellt.

Keine der bestehenden Lösungen vermochte der Idee alleine gerecht zu werden; eine Nutzwertanalyse der Teillösungen führte zum Ergebnis einer Kombination aus einer Geodatenbank und einem Karten-Dienst. Auf dieser Basis entstand das *Konzept* eines parametrisierbaren Dienstes, der aus einer Client- und einer Server-Komponenten besteht. Die Formulierung des Konzepts mündete in das Design eines Prototypen, der sämtliche funktionalen Anforderungen erfüllen und damit als *Proof of Concept* die Ausgangsidee greifbar machen sollte.

Bei der Umsetzung des Prototypen lag der Schwerpunkt auf dem *handwerklichen Aspekt* der Softwareentwicklung. Neben der Auswahl von konkreten Technologien und Werkzeugen mussten Entscheidungen gefällt werden über das Design der einzelnen Komponenten. Die ursprünglichen Anforderungen an die Plattform wurden eingeschränkt, um den Entwicklungsaufwand zu begrenzen, ohne jedoch die Vollständigkeit der Grundfunktionalität zu gefährden. Die im Konzept definierten Grundsätze konnten ohne Abstriche in die Praxis übersetzt werden.

Der fertige Prototyp durchlief eine Reihe von Test, die während der Anforderungsanalyse als Abnahmekriterien definiert worden waren. Die Tests prüften die Funktionalität mittels detaillierter Anleitungen, die Resultate wurden in *Testprotokollen* festgehalten. Die Software erfüllte sämtliche Tests zur Zufriedenheit des Autors.

8.2 Herausforderungen

Das Projekt stand unter einem hohen *Zeitdruck*, da der Abgabetermin vom Reglement vorgegeben war. Dies führte zu, dass der Autor diszipliniert und zielgerichtet arbeiten musste. Trotz des erhöhten zeitlichen Drucks konnten die Etappenziele erreicht werden, ohne dass die Qualität der Resultate beeinträchtigt wurde.

Obwohl der Autor sein längerer Zeit über eine vage Vorstellung der Idee von «Hyperlike» verfügte, gestaltete sich die Formulierung des einleitenden Teils (Abschnitt 2) schwieriger als erwartet. Insbesondere das Herausarbeiten des Zusammenhangs von Eingabe, Analyse und Ausgabe der Eingabewerte schien aus der Ferne einfacher als es beim Zusammenstellen der Anwendungsbeispiele tatsächlich war.

Die *Anforderungsanalyse* stellte verschiedene Herausforderungen an den Autor. Eine Schwierigkeit lag in der formalen Präsentation des Anforderungsanalyse. Die Literatur kennt hier keine einheitliches Vorgehen, sondern lediglich eine Reihe von Empfehlungen und sehr umrisshaften Anleitungen. Der Autor entschied sich hier dafür, den Ideen von [3] möglichst genau zu folgen und sich von anderen Bachelor-Arbeiten inspirieren zu lassen.

Das Verhältnis der Anforderungen an die *Plattform* zu den Anforderungen an eine *Instanz* (also die Umsetzung einer konkreten Erhebung) war nicht von Anfang an klar. *Benutzer* und *Anbieter* haben aus unterschiedlicher Perspektive ähnlich gelagerte und gleichberechtigte Ansprüche an das Produkt. Gleichzeitig verwischt der Fokus auf die einfache Konfigurierbarkeit des Systems die Grenzen zwischen *Betreiber* und *Anbieter*. Trotz dieser in der Realität unklaren Abgrenzungen wurden die Rollen bei der Definition der Stakeholder deutlich von einander getrennt.

Bei der Durchführung *Marktanalyse* wirkte in erster Linie die Vielfalt des Angebots als fortschrittshemmend. Die Themen Datensammlung und Crowdsourcing sind sehr in Mode, so dass der Markt für Lösungen in diesem Gebiet entsprechend mit Anbietern gefüllt ist. Es war sehr zeitintensiv, die bestehenden Anbieter zu sichten und einen Überblick über das Gebiet zu gewinnen.

Die Auswahl der Methode, mit der die kandidierenden Gesamt- und Teillösungen auf Erfüllung der Anforderungen geprüft wurden, erforderte mehrere Anläufe. Eine Aufteilung aller Produkte in erfüllende und nicht-erfüllende Lösungen durch die ausschliessliche Anwendung von K.O.-Kriterien erwies sich als zu grob. Gleichzeitig war der Anforderungskatalog – dem Anwendungszweck entsprechend – sehr spezifisch und eng formuliert, so dass eine Nichterfüllung in nur einem Punkt das gesamte System in Frage gestellt hätte. Der Kompromiss bestand am Ende darin, die Erfüllung der funktionalen Anforderungen bei Gesamtlösungen als zwingend vorzusetzen und bei Teillösungen im Rahmen einer Nutzwertanalyse nach Priorität gewichtet miteinander zu verrechnen.

Zwar konnte der Autor bei der *Umsetzung* des Prototypen auf bestehendes Technologie-Know-how aufbauen, so dass insbesondere im Java-Bereich in der Anlauf-Phase des Projekts wenig

Zeit verloren ging. Dennoch hielt das Zusammenspiel von Client- und Server-Komponente einige Knacknüsse bereit: Das UI-Framework React ist noch recht jung und verlangt ein Umdenken des Entwicklers; das Zusammenspiel der Entwicklungsumgebungen für Client und Server musste zuerst aufgebaut werden gestaltete sich oft als zu wenig robust; die Build-Prozesse der beiden Komponenten – mit Apache Maven bzw. NPM – mussten aufeinander abgestimmt werden.

Obwohl ein Prototyp entstanden ist, der die Grundfunktionen der Plattform abdeckt, ist der Autor von der vollständigen Umsetzung seiner ursprünglichen Idee noch entfernt. Einerseits verfügt Hyperlike noch nicht über genügend Analyse-Methoden, um für die Durchführung von realen Erhebungen nützlich zu sein, anderseits ist der Weg vom Prototypen zu einer produktiv einsetzbaren Plattform noch weit. Dennoch dürfen die Anforderungs- und Marktanalyse sowie das erarbeitete Konzept als wichtige Etappenziele verbucht werden.

8.3 Persönliches Fazit

Das Projekt Hyperlike war für den Autor lehrreich und interessant. Die Beschreibung des gesamten Weges von einer vagen Idee über die Marktanalyse und die Konzeption bis hin zum lauffähigen Prototypen war eine Erfahrung, die für die berufliche Zukunft sicherlich nützlich sein wird. Besonders genoss der Autor die Freiheit, frei von Zwängen des beruflichen Alltags einer Software-Idee auf den Grund gehen zu können.

Auf der technologischen Seite war insbesondere die Arbeit an der grafischen Web-Oberfläche interessant. Hier konnte der Autor dazulernen und neue Werkzeuge und Methoden kennenlernen. Insbesondere von der React-Bibliothek ist der Autor nachgerade begeistert, aber auch die Sprache JavaScript als ganzes hat an Statur gewonnen.

Die Aufgabenstellung war sehr weit gefasst, und der Autor hätte gerne den einen oder anderen Aspekt etwas mehr vertieft (etwa die Erweiterung der Plattform um zusätzliche Analyse-Methoden), während andere sich rückblickend weniger Aufmerksamkeit verdient hätten (etwa die Untersuchung bestehender Gesamtlösungen, von denen sich keine als nützlich herausstellte). Doch insgesamt ist ein gutes Bild der Möglichkeiten der Idee entstanden, und die Betrachtung der bestehenden Produkte hat den Bedarf nach ihr aufgezeigt.

Abschliessend ist die Bilanz eine positive: Der Autor fühlt sich in seiner ursprünglichen Idee bestätigt, es wurde eine gründliche Analyse eines mit sehr ähnlichen Produkten gefüllten Marktes durchgeführt, es entstand ein klares Konzept des Gesamtsystems, und der entstandene Prototyp erfüllt die Anforderungen.

Glossar

In diesen Abschnitt sollen einerseits Fachbegriffe erläutert werden, die im Text mehrfach vorkommen und deren Bedeutung nicht als allgemein bekannt vorausgesetzt werden darf (z.B. Geodatenbank). Einige davon werden an der Stelle ihrer ersten Verwendung definiert und werden hier zum Zwecke der Übersicht wiederholt. Anderseits finden sich in diesem Glossar Begriffe, die der Autor aufgrund ihrer spezifischen Verwendung als *erklärungsbedürftig* erachtet (z.B. Plattform).

Beobachtungswert

Als Beobachtungswerte werden die *Eingabewerte von «Hyperlike»* bezeichnet. Da es sich bei «Hyperlike» nicht um eine Plattform für Messwerte handelt, sondern der Mensch als primäre Datenquelle dienen soll, fallen unter den Begriff der Beobachtung ebenso *persönliche Einschätzungen* wie *Bewertungen* und *Meinungsäusserungen*. Beobachtungswerte sind somit *qualitative* Äusserungen, die zwecks Verarbeitung durch «Hyperlike» als *quantitative* Werte codiert werden.

Crowdsourcing

Crowdsourcing bezeichnet die Auslagerung von Arbeitsschritten an eine Gruppe freiwilliger Personen. Das Vertrauen in die Qualität der Resultate von Crowdsourcing wird gestützt durch die in [1] beschriebene «Weisheit der Vielen», die den Einschätzungen von Experten oft überlegen ist. Der Begriff ist abgeleitet vom Wort *Outsourcing* und wurde im Jahr 2006 geprägt vom *Wired*-Journalisten Jeff Howe⁵⁵.

Data-driven

Als *data-driven* werden in einem nicht-technischen Kontext Prozesse und Verhalten bezeichnet, die durch die Interpretation von Daten bestimmt werden, nicht durch Erfahrung oder Intuition. Die Entscheidungsfindung folgt dabei dem Vorbild der Naturwissenschaften, die neue Erkenntnisse stets auf die Resultate vorangehender, empirischer Untersuchungen stützt.

Erhebung

Eine Erhebung bezeichnet eine Befragung der einer Gruppe von Personen mittels einer Instanz von «Hyperlike». Um eine Erhebung durchführen zu können muss die Instanz konfiguriert und bereitgestellt werden.

⁵⁵ <http://archive.wired.com/wired/archive/14.06/crowds.html>

Geodatenbank

Eine Geodatenbank ist eine relationale oder andere *Datenbank*, die über Zusatzfunktionen verfügt, die auf geografische Anwendungen ausgelegt sind. Sie erlaubt die Speicherung geographischer Objekte (z.B. durch Koordinaten definierte Polygone) und Abfragen mittels räumlicher Funktionen (z.B. Abstand oder Fläche) und Operatoren (z.B. Umschließung oder Überschneidung). Dies ermöglicht einen *High-level*-Umgang mit räumlichen Daten, da die geometrischen und geographischen Operationen abstrahiert wurden.

Geoinformationssystem

Gemäss dem Bundesamt für Landestopografie wird der Begriff wie folgt definiert: «Ein Geoinformationssystem (Kurzform GIS) oder Geografisches Informationssystem ist ein rechnergestütztes Informationssystem, das aus Hardware, Software, Daten und Anwendungen besteht. Mit ihm können raumbezogene Daten digital erfasst und redigiert, gespeichert und reorganisiert, modelliert und analysiert sowie tabellarisch und grafisch präsentiert werden.»⁵⁶

Karten-Dienst

Ein Karten-Dienst ist ein Internet-Dienst, der über eine öffentliche API kartographische Daten anbietet. In der Regel liefert er bei der Eingabe von geographischen Koordinaten die entsprechenden *Kacheln* der Karte als Bilder, die bei der Anzeige zu einem Kartenausschnitt zusammengesetzt werden.

Mobile Data Collection

Der englische Begriff «Mobile Data Collection» (etwa «Datensammlung von unterwegs») bezeichnet die Tätigkeit, mittels tragbarer Geräte dauernd oder an gewissen Stationen einer Bewegung gewisse Aspekte der Umgebung zu messen oder zu beobachten und als *ortsgebundene Daten* zu speichern oder zwecks Weiterverarbeitung an ein zentrales System zu übermitteln.

NoSQL-Datenbank

Der oft zu *Not only SQL* expandierte Begriff ist ein Sammelbegriff für eine Klasse von Datenbanken, deren Datenhaltung nicht als Tabellen modelliert ist. Die Bezeichnung trifft auf so unterschiedliche Produkte wie MongoDB (Dokumenten-orientiert) und Redis (Key/Value-Store) zu.

Ortsgebundene Daten

Ortsgebundene Daten sind durch Beobachtung oder Messung gewonnene Werte, die sich auf eine durch geographische Koordinaten bestimmte Position beziehen und für diese gültig sind. Die geographische Information ist dabei Teil der zu dem Datensatz gehörigen *Metainformation*.

⁵⁶ <http://www.geo.admin.ch/internet/geoportal/de/tools/glossar.html>

Plattform

Der Begriff der Plattform ist ein weit gefasster. Im Kontext dieser Arbeit soll er im Sinne einer «Basis zur Entwicklung und Ausführung von Programmen» verstanden werden. Unter «Entwicklung» fällt hier die Konfiguration der Instanz, unter «Ausführung» fällt das Bereitstellen des Dienstes, mit dem die Erhebung durchgeführt wird.

Werte-Tupel

Jede *Eingabe* einer Erhebung kann aus mehreren Werten zusammengesetzt sein – je ein Wert für jeden konfigurierten Typ. Die Eingabe besteht somit aus einem *Tupel von Einzelwerten*.

Literaturverzeichnis

Bücher und Zeitschriften

- [1] James Surowiecki. *The Wisdom of Crowds*. Knopf Doubleday Publishing Group. 2005. ISBN 978-0-30727-505-9.
- [2] Roberta Kwok. *Phoning in Data*. Nature 458, 959-961. 2009.
- [3] Klaus Pohl, Chris Rupp. *Basiswissen Requirements Engineering: Aus- und Weiterbildung zum «Certified Professional for Requirements Engineering»*. Dpunkt.Verlag. 2011. ISBN 978-3-89864-771-7.
- [4] Toby Segaran. *Programming Collective Intelligence*. O'Reilly & Associates. 2007. ISBN 0-596-52932-5.
- [5] Aanensen DM, Huntley DM, Feil EJ, al-Owain F, Spratt BG. *Epi-Collect: Linking Smartphones to Web Applications for Epidemiology, Ecology and Community Data Collection*. PLoS ONE 4(9): e6968. doi:10.1371/journal.pone.0006968
- [6] Leonard Richardson, Sam Ruby. *RESTful Web Services*. O'Reilly & Associates. 2007. ISBN 978-0-59652-926-0.
- [7] Richard Ferraro, Murat Aktihanoglu. *Location-Aware Applications*. Manning. 2011. ISBN 978-1-935182-33-7.
- [8] Nathan Marz. *Big Data: Principles and best practices of scalable realtime data systems*. Manning. April 2015. ISBN 9781617290343.
- [9] Estelle Weyl. *Mobile HTML5*. O'Reilly & Associates. 2013. ISBN 978-1449311414.
- [10] Mark Pollack, et al. *Spring Data: Modern Data Access for Enterprise Java*. O'Reilly & Associates. Oktober 2012. 978-1-449-32395-0.
- [11] Marten Deinum et al. *Pro Spring MVC: With Web Flow*. Apress. Juni 2012. ISBN 978-1430241553.
- [12] Joshua Bloch. *Effective Java*. Addison-Wesley. Mai 2008. ISBN 978-0321356680.
- [13] Douglas Crockford. *JavaScript: The Good Parts*. O'Reilly & Associates. Mai 2008. ISBN 978-0596517748.

Online-Dokumente

- [14] Doug Browning. Mobile Solutions for GIS Data Collection and Display 2011.
<https://sites.google.com/site/dougbrowningportfolio/Resources/mobile-gis>
- [15] MongoDB Architecture Guide. 2015. https://s3.amazonaws.com/info-mongodb-com/MongoDB_Architecture_Guide.pdf
- [16] IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications. The Institute of Electrical and Electronics Engineers 1998.
<http://www.math.uaa.alaska.edu/~afkjm/cs401/IEEE830.pdf>

A Erklärung

Hiermit bestätigt der Unterzeichnende, dass die Bachelorarbeit mit dem Thema «*Hyperlike: eine Plattform für Erfassung, Analyse und Austausch ortsgebundener Daten*» gemäss freigegebener Aufgabenstellung mit Freigabe vom 18. Januar 2015 ohne jede fremde Hilfe im Rahmen des gültigen Reglements selbstständig ausgeführt wurde.

Andreas Moor

B Projektplanung

Index

A

Akzeptanzkriterien, 31
Analyse-Methode, 26
Analyseprozess, 23
Anbieter, 8, 15
Anforderungen, 15
Anforderungsanalyse, 15
Anwendungsbeispiele, 9
ArcGIS, 41
Architekturentscheide, 18

B

Backend-as-a-Service, 43
Benutzeroberflächen, 29
Beobachtungswert, 97
Beobachtungswerte, 4
Betreiber, 8, 16
Big Data, 4, 5

C

Content-Management-Systeme, 9
Crowd, 5
Crowdsourcing, 4, 97

D

Data-driven, 97
data-driven, 4
Data-Sharing, 8
Datenbank, 46
Datenerfassung, 6
Datensammlung, 4
Datenverarbeitung, 9

E

Endbenutzer, 16
Entwickler, 15
EpiCollect, 40
Erhebung, 97

Erweiterbarkeit, 30

F

Formhub, 42, 45

G

Geodaten, 8
Geodatenbank, 98
Geodatenbanken, 46
Geoinformationssystem, 8, 98
Google Maps, 44

J

J2EE, 63
Java, 62
Java Enterprise Edition, 63

K

Karten-Dienst, 98
Karten-Dienste, 44
Kartogramm, 25, 28
Komplettlösung, 34
Konsumenten, 8

M

Marktanalyse, 34
Messwerte, 4
Mobile Computing, 5
Mobile Data Collection, 98
MongoDB, 46, 51

N

NoSQL, 46
NoSQL-Datenbank, 98
Nutzwertanalyse, 49

O

ODK, 35
Open Source, 30
OpenDataKit, 35

OpenStreetMap, 44, 51
Ortsgebundene Daten, 98

P

Parse, 43
Plattform, 99
PostGIS, 46
PostgreSQL, 46
Programmierung, 9

Q

Quantified Self, 4

R

Raster, 26

S

Schema, 46
Schnittstellen, 18
Segmente, 26
Skalierbarkeit, 30
Social Media, 5
Spring Framework, 63
SQL, 46
Stakeholder, 15
Systemgrenze, 17, 34
Systemumfeld, 17

T

Teillösung, 34
Teilnehmer einer Erhebung, 8
Twitter, 8, 13

U

Use Cases, 15
Ushahidi, 38

W

Werte-Tupel, 26, 99
World Wide Web, 8, 18

Z

Zielgruppe, 8
