Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Torben Wallbaum, B.Sc.

Konzeption und Realisierung eines mobilen Crowdsourcing-Dienstes

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Torben Wallbaum, B.Sc.

Konzeption und Realisierung eines mobilen Crowdsourcing-Dienstes

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science — Verteilte Systeme
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Olaf Zukunft
Zweitgutachter: Prof. Dr. Bettina Buth

Eingereicht am: 20. Februar 2012

Torben Wallbaum, B.Sc.

Thema der Arbeit
Konzeption und Realisierung eines mobilen Crowdsourcing-Dienstes

Stichworte
Crowdsourcing, Schwarmintelligenz, Android, standortbasierte Dienste

Kurzzusammenfassung
Reiseführer, ob analog oder digital, sind zumeist veraltet, zu kostspielig oder bieten dem Reisenden keine Informationen, die seinen Interessen entsprechen. Die Vernetzung der heutigen Gesellschaft und das Aufkommen des Crowdsourcing — als Meta-Konzept für eine moderne Informationsaggregation — bieten die Möglichkeit, Informationen über Städte und ihre Sehenswürdigkeiten durch Ortsansässige zusammenzutragen. Innerhalb dieser Masterarbeit wird eine mobile Android-Applikation konzipiert und entwickelt, die einen solchen Dienst zur Verfügung stellt. Der Einsatz von Gamification-Elementen und die Entwicklung einer ansprechenden Nutzerschnittstelle sollen zur Verwendung der Applikation ermutigen.

Torben Wallbaum, B.Sc.

Title of the paper
Design and Implementation of a Mobile Crowdsourcing-Service

Keywords
Crowdsourcing, swarm intelligence, android, location-based services

Abstract
For most traditional and digital travel guides it has proven impossible to keep up to date and to provide information tailor-made for individual users. Additionally, consumers have to pay for those guides. Due to digital networking and crowdsourcing, society nowadays has additional means to gather information about places and sights - both tourists and locals can post it online. The subject of this thesis is the development of a mobile Android application, which offers a respective innovative travel guide. The use of gamification elements and an attractive user interface are designed to encourage the use of the application.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1. Einleitung

Kollektive Intelligenz ist seit über 20 Jahren ein gut untersuchtes Forschungsfeld der Informatik. Als Teilgebiet der Künstlichen Intelligenz wird hier das kooperative Verhalten von Individuen zur Bildung eines „Superorganismus“ untersucht. Die Vernetzung von Menschen durch das Internet sowie die Entwicklungen im Bereich „Ubiquitous Computing“ (Rechnerallgegenwart) [Weiser, 1991] ermöglichen den Zusammenschluss von leistungsfähigen, menschlichen Schwärmen. Das junge Gebiet des Crowdsourcing [Howe, 2006] nutzt die Fähigkeiten dieser Gruppen, um komplexe und umfangreiche Probleme zu lösen.

1.1. Motivation

Für den interessierten Reisenden bietet der Markt eine große Anzahl an Reiseführern und City-Guides. Erhältlich sind diese in Form eines Buches bzw. einer Broschüre oder als digitale Ausgabe auf mobilen Geräten. Bekannte Hersteller von Reiseführern bieten zudem einfach zu handhabende Anwendungen auf Smartphones an (siehe [Polo, 2012], [Falk, 2012], [Planet, 2012a]). Nutzer eines digitalen Reiseführers können je nach Interessenlage die Angebote filtern und passende Einträge auswählen.

Nachteilhaft bei all diesen Reiseführervarianten ist jedoch, dass Informationen zu Sehenswürdigkeiten, Restaurants u.a. veraltet sein können oder nachgekauft werden müssen (siehe beispielsweise [Planet, 2012b]). Auch handelt es sich bei den empfohlenen Orten um die Selektion der Guide-Ersteller, welche nicht notwendigerweise mit den Interessen des Konsumenten übereinstimmt. Die besten Hinweise für Sehenswertes in einer Stadt bzw. einer Region erhält man zudem meist von Einheimischen. Leider haben Reisende jedoch oft kaum Kontakte zu Anwohnern. Die Hauptmotivation dieser Masterarbeit ist, eine Alternative zu obig thematisierten Reiseführern zu entwickeln, welche kostenfrei und stets aktuell eine Vielzahl unterschiedlicher Interessenlagen abdeckt.

1.2. Ziele

Im Rahmen dieser Masterthesis soll ein Reiseführer und City-Guide entwickelt werden, der als Smartphone-Anwendung die Mechanismen des Crowdsourcing nutzt. Informationen über Sehenswürdigkeiten werden dementsprechend durch die Nutzer selber generiert. Ein Vorteil des Crowdsourcing ist dabei, dass Einheimische potentiellen Besuchern Angaben zu den Sehenswürdigkeiten ihrer Heimat zur Verfügung stellen können.

Ein weiteres Ziel ist zudem, die vorhandenen Daten zur Erhebung von Metainformationen über bestimmte Stadtteile und Gegenden zu nutzen. So werden zum Beispiel Heatmaps generiert, die Demographien anhand von Schlagwörtern ermöglichen. Auch ist es möglich, dem Nutzer Orte zu empfehlen, die auf seine Interessen zugeschnitten sind. Um die Erzeugung von Content zu begünstigen und Anwender zu animieren, möglichst viele ihrer Tipps und Hinweise in das System einzubringen, sollen Spielmechaniken genutzt werden (Gamification). Diese fördern die Motivation bei der Generierung von Inhalten und können so die Partizipation der Nutzer am System erhöhen.

Um ein solches System prototypisch umzusetzen, sollen anhand von konkreten Szenarien Anwendungsfälle erhoben werden, welche anschließend durch funktionale, technische und nicht-funktionale Anforderungen an eine mobile Crowdsourcing-Anwendung weiter konkretisiert werden. Diese dienen dann einem Entwurf und der software-technischen Architektur. Zuletzt werden die identifizierten Anforderungen implementiert und getestet.

1.3. Abgrenzung

Bezogen auf die identifizierten Ziele wird im Rahmen dieser Arbeit eine prototypische Umsetzung eines mobilen Crowdsourcing-Dienstes angestrebt. Bei der Realisierung wird sich auf die identifizierten Anforderungen beschränkt, welche die Kern-Funktionalitäten des Systems darstellen. Die Festlegung von nicht-funktionalen Anforderungen an Skalierbarkeit und Performance — vor allem in Bezug auf die server-seitige Implementierung — werden nur teilweise berücksichtigt.

Die client-seitige Realisierung beschränkt sich auf die Android-Plattform. Eine weitere Umsetzung auf anderen mobilen Systemen ist nicht geplant. Die Entwicklung einer „Cross-Platform"-Lösung ist ebenfalls kein Ziel dieser Arbeit.

Um die Akzeptanz durch tatsächliche Nutzer zu überprüfen und Untersuchungen im Bereich der Usability und User Experience durchzuführen, müssten umfangreiche Feldversuche erfolgen, die den zeitlichen Rahmen dieser Arbeit überschreiten würden.

Eine Anbindung an weitere Dienste oder die Erarbeitung einer öffentlichen Schnittstelle, welche von externen Applikationen genutzt werden könnte, ist wünschenswert jedoch für die zeitlichen Vorgaben ebenfalls zu umfangreich.

1.4. Gliederung

In Kapitel 2 werden die Begrifflichkeiten der Hauptkomponenten dieser Arbeit vorgestellt. Hierzu gehören beispielsweise das Gebiet der Kollektiven Intelligenz und der Bereich des Crowdsourcing. Weiterhin werden Besonderheiten bei der Entwicklung von mobilen Applikationen erläutert — sowohl bezüglich der Android-Plattform als auch im Allgemeinen. Da spielerische Elemente ebenfalls Teil dieser Arbeit sind, wird weiterhin das junge Gebiet der Gamification definiert.

Um die Entwicklung des geplanten Dienstes einzugrenzen, wird in Kapitel 3 ein Überblick über den Bereich der standortbezogenen Dienste gegeben. Dieser Überblick wird anschließend mittels eines konkreten Anwendungsszenarios spezifiziert; auch wird der Kontext beschrieben, in welchem sich ein Nutzer typischerweise befindet. Das Kapitel wird mit einer groben Skizzierung der Kern-Funkionalitäten abgeschlossen.

Kapitel 4 gibt zunächst einen beispielhaften Überblick über artverwandte Arbeiten in diesem Bereich. Danach werden die Funktionalitäten anderer Arbeiten den Funktionalitäten der hier entwickelten Smartphone-Applikation in einer Tabelle gegenübergestellt. Es werden allgemeine Anforderungen an mobiles Crowdsourcing aufgezeigt und erläutert. Die Bestandteile des Systems werden danach durch die Erstellung von Anwendungsfällen, sowie hieraus abgeleitete funktionale, technische und nicht-funktionale Anforderungen identifiziert.

Die Anforderungen dienen der in Kapitel 5 aufgezeigten Architektur und der konkreten, technischen Planung der client- und server-seitigen Module. Neben einer Systemübersicht und der Konzeption von Systemkomponenten und ihrer Architektur wird in diesem Kapitel auch auf die Konzeption der geplanten Nutzerschnittstellen eingegangen. Hierfür werden Interface-Skizzen erstellt und ihre Funktionalitäten erläutert.

Die Realisierung der konzipierten Module — sowohl client- als auch server-seitig — werden in Kapitel 6 im Detail erläutert. Anhand von Beispielen wird die Umsetzung erklärt und die Kommunikation zwischen Client- und Server-Anwendung vorgestellt. Die server-seitige Implementierung der erstellten Anforderungen wird anhand von Beispielen aufgezeigt. Danach wird die prototypische Umsetzung der Heatmap-Ansicht erklärt, und bereits existierende Systeme werden vorgestellt.

Die Durchführung von Tests ist für die Beurteilung und die Sicherstellung der Qualität der entwickelten Software essentiell. Kapitel 7 gibt eine kurze Einführung in die Thematik und beschreibt anschließend die in dieser Arbeit verwendeten Methoden für den Software-Test.

In Kapitel 8 werden die Anforderungen des entwickelten Entwurfs mit der Realisierung der Komponenten abgeglichen. Es werden mögliche Erweiterungen für das System aufgezeigt und ihre Vor- und Nachteile diskutiert. Abschließend werden mögliche Erweiterungen für die server-seitige Datenaggregation sowie die Datenanalyse gegeben.

2. Grundlagen

Im folgenden Kapitel werden die Grundlagen der in dieser Masterarbeit verwendeten Komponenten dargelegt. Zunächst wird das Gebiet der Kollektiven Intelligenz in seiner Entstehung und seinem aktuellen Forschungsstand umrissen. Anschließend wird der Bereich des Crowdsourcing — als Grundlage der vorliegenden Masterarbeit — erläutert und exemplarisch erklärt.

Die praktische Umsetzung der Arbeit basiert client-seitig auf der Android-Plattform, welche in ihrer Geschichte und Architektur vorgestellt wird. Ergänzend wird im Abschnitt Mobile Computing 2.4 beschrieben, welche Besonderheiten der Entwicklung eines mobilen Dienstes anhaften und welche Auswirkungen diese Entwicklungsarbeit hat.

Abschließend wird das junge Gebiet der Gamification definiert und ansatzweise vorgestellt.

2.1. Kollektive Intelligenz

Kollektive Intelligenz — hier gleichgesetzt mit Schwarmintelligenz — bezeichnet die Möglichkeit, durch Kooperation von Individuen eine Gruppenintelligenz entstehen zu lassen. Dieser „Superorganismus“ ist durch seine Entscheidungen und sein intelligentes Verhalten dazu in der Lage, die Intelligenz von einzelnen Experten zu übertreffen.

Die Bedeutung des Begriffs Schwarmintelligenz lässt sich durch die Definition seiner Bestandteile erklären: Das Langston Santa Fe Institute definiert den Begriff Schwarm als „...use the word „swarm“ to mean any loosely structured collection of agents that interact with one another.“ [Eberhart und Kennedy, 2001]. Die hier genannten Agenten können dabei Lebewesen sein — wie zum Beispiel Menschen, Ameisen, Vögel — oder auch anorganische Agenten wie Roboter, die durch ein kollektives Verhalten bestimmte Ziele erreichen.

Die Definition des Begriffs Intelligenz ist dahingegen nicht so eindeutig — sinnvoll ist es jedoch, die verschiedenen Erklärungen aus dem Gebiet der Psychologie zu Rate zu ziehen: In der Psychologie kann gemessene Intelligenz je nach Art des Messverfahrens und Aufgabengebietes unterschiedlich bewertet werden. Eine angemessene Interpretation, die die Ambivalenz des Begriffs aufgreift, wurde von E. G. Boring im Jahr 1923 aufgebracht. Er beschreibt Intelligenz als einen Wert, der von einem Intelligenz-Test abhängig ist „intelligence is whatever it is that an intelligence test measures.“ [Eberhart und Kennedy, 2001].

Eine frühe Erwähnung der Idee einer kollektiven Intelligenz ist schon in Aristoteles politikwissenschaftlichem Werk „Politik“ unter dem Abschnitt „Summierungsthese“ zu finden:

> „Daß aber die Entscheidung eher bei der Menge als bei der geringeren Zahl der Besten [den aristoi] zu liegen habe, das scheint zu bestehen und sich verteidigen zu lassen, ja vielleicht sogar wahr zu sein. Denn die Menge, von der der einzelne kein tüchtiger Mann ist, scheint doch in ihrer Gesamtheit besser sein zu können als jene Besten; nicht jeder Einzelne für sich, sondern die Gesamtheit, so wie die Speisungen, zu denen viele beigetragen haben, besser sein können als jene, die ein Einzelner veranstaltet. Denn es sind viele, und jeder hat einen Teil an Tugend und Einsicht. Wie sie zusammenkommen, so wird die Menge wie ein einziger Mensch, der viele Füße, Hände und Wahrnehmungsorgane hat und ebenso, was den Charakter und den Intellekt betrifft. So beurteilt auch die Menge die Werke der Musik und der Dichter besser; der eine beurteilt diese, der andere jene Seite, und so urteilen alle über das Ganze.“[Aristoteles, 2003]

Aristoteles beschreibt also, dass die kollektive Gesellschaft in der Gesamtleistung mehr erreichen kann als der einzelne Experte. Ebenfalls wird der erhöhte Intellekt einer aus vielen einzelnen Individuen bestehenden Gesamtheit thematisiert.

Innerhalb der Informatik findet der Begriff Schwarmintelligenz seine erste Erwähnung im Jahr 1989. Gerardo Beni und Jing Wang erläutern seine Bedeutung in einer Arbeit über zellulare Roboter-Systeme: Die für sich genommenen Roboter sind nicht besonders intelligent, als Kollektiv finden sie jedoch sinnvolle Lösungen zu vorgegebenen Problemstellungen.

> „systems of non-intelligent robots exhibiting collectively intelligent behaviour evident in the ability to unpredictably produce ’specific’ ([i.e.] not in a statistical sense) ordered patterns of matter in the external environment“ [Beni und Wang, 1989]

Die Entstehung des Internets und die damit verbundene Entwicklung von modernen Informations- und Kommunikationssystemen fördert die Bildung von kollektiver Intelligenz maßgeblich. Die mögliche Vernetzung von Menschen gleicher Interessen oder Ideale lässt insbesondere innerhalb der letzten zehn Jahre eine Tendenz zu vermehrter Nutzung von Gruppenintelligenz erkennen. Hierzu zählen unter anderem:

- Plattformen wie Wikipedia, die eine nahezu umfassende Informationsaggregation ermöglichen.
- Das Entstehen der Open Source-Gemeinde, welche umfangreiche Software-Projekte auf Basis einer freiwilligen Kooperation umsetzen.

- Kartendienste wie Google Maps oder Open Street Maps innerhalb derer Menschen Einträge zu Orten, Geschäften, u.a. vornehmen können.

Howard Rheingold erklärt in seinem Buch „Smart Mobs“ so bereits 2003: „The „Killer-Apps“ of tomorrow’s mobile infocom industry won’t be hardware devices or software programs but social practices.“ [Rheingold, 2003]

Die These, dass kollektive Intelligenz einen Mehrwert gegenüber Spezialistenwissen bietet, wird weiterhin durch die Aussagen von James Surowiecki gestützt. In „Die Weisheit der vielen“ [Surowiecki, 2005] erläutert er „Unter den richtigen Umständen sind Gruppen bemerkenswert intelligent — und oft klüger als die Gescheitesten in ihrer Mitte.“ Auch müssen laut Surowiecki Gruppen von Individuen nicht unbedingt aus Experten bestehen, um vernünftige Einschätzungen abzugeben: „...Auch wenn die allermeisten Angehörigen einer Gruppe weder sonderlich informiert noch zu rationalem Denken imstande sind, vermögen sie als Kollektiv gleichwohl vernünftige Entscheide zu treffen.“. Folgendes Zitat Surowieckis untermauert die Sinnhaftigkeit des in dieser Masterarbeit vorgestellten Dienstes zur Aggregation von interessanten Plätzen und Sehenswürdigkeiten: „Statt darauf aus zu sein, die jeweils beste aller möglichen Lösungen zu finden, geben wir uns häufig mit einer passablen Variante zufrieden.“. Das gesammelte Wissen eines Kollektivs über bestimmte Orte kann dementsprechend laut Surowiecki vorteilhafter sein, als die optimale Reisgestaltung eines versierten Individuums.

2.2. Crowdsourcing

Angelehnt an die in Abschnitt 2.1 thematisierte Kollektive Intelligenz oder auch Schwarmintelligenz soll im Folgenden das Gebiet des Crowdsourcing erläutert werden. Hierbei handelt es sich — im Gegensatz zum Outsourcing einer Aufgabe an ein anderes Unternehmen — um einen alternativen Ansatz zur Aufgabenbewältigung. Erstmalig von Jeff Howe innerhalb eines Artikels im Jahr 2006 erwähnt [Howe, 2006], versteht man unter Crowdsourcing die Auslagerung von Aufgaben an die Intelligenz und Arbeitskraft der Vielen.

Surowiecki [Surowiecki, 2005] zeigt einige Beispiele auf, bei denen die Masse komplexe Aufgaben löst oder unbekannte Situationen vorhersagt: 1968 etwa, verschwand ein U-Boot der US-Marine nach einer Routinetour im Nordatlantik. Viele Experten versuchten, jeder für sich, die mögliche Position des U-Bootes anhand weniger Informationen zu ermitteln. Nachdem keine dieser Berechnungen zur Auffindung des U-Bootes führte, stellte der Marineoffizier John Craven einen Versuch auf. Er ließ alle Experten eine Schätzung über die Position abgeben und verwendete die Bayessche Regel, um das U-Boot zu lokalisieren. Fünf Monate nach dem Verschwinden wurde

das U-Boot entdeckt. Die tatsächliche Position war lediglich 75 m von der Stelle entfernt, welche die Aggregation der Schätzungen ergab.

Crowdsourcing als Konzept wurde bereits viele Male erfolgreich eingesetzt. Bei Plattformen wie Amazons Mechanical Turk lassen Aufgabensteller ihre Probleme erfolgreich von einer Vielzahl von Arbeitern erledigen [Ross u. a., 2010]. Anhand der Zahl von verfügbaren Aufgaben (HITs) innerhalb einer Größenordnung von 180.000 [Mechanical-Turk, 2011] ist zudem ersichtlich, dass sie sowohl von Auftraggebern als auch von Auftragnehmern gut angenommen werden. Mechanical Turk ermöglicht es dem Auftraggeber zudem, eine Aufgabe zu definieren. Ein Beispiel hierfür ist das Hinzufügen von Tags zu Bildern oder die Markierung von Objekten in Bildern. Die Auftragnehmer erledigen diese Aufgabe und erhalten im Gegenzug einen kleinen Geldbetrag als Vergütung; umfangreiche Aufgaben werden so zeitnah bearbeitet. Die Plattform ist insbesondere für Aufgaben in den Bereichen Bildverarbeitung und Identifizierung interessant, da ein Algorithmus hier an seine Leistungsgrenzen stößt.

Auch im wissenschaftlichen Kontext gibt es gute Beispiele für die erfolgreiche Anwendung des Crowdsourcing. Hierzu zählt der Captcha-Dienst ReCaptcha [von Ahn, 2009], welcher sowohl bei der Identifizierung von Individuen bei Anmeldeprozessen als auch bei der Digitalisierung von Büchern hilft. Auch „fold it“ wäre zu nennen [Cooper u. a., 2010]: Nutzer dieses Diensten falten bei der Lösung von Puzzle-Aufgaben automatisch Proteine und helfen somit dabei, die Struktur und Arbeitsweise von Proteinen zu erforschen.

Die Kernidee, die mit der Anwendung von Crowdsourcing verbunden ist — nämlich das Erreichen von Zielen durch eine große Gruppe von Menschen — stellt die Grundfunktionalität des praktischen Teils der vorliegenden Masterarbeit dar. Das Konzept des Crowdsourcing wird hierbei dazu verwendet, eine möglichst umfangreiche und diverse Menge an Orten und Sehenswürdigkeiten zusammenzutragen, um eine Vielzahl an unterschiedlichen Interessen abdecken zu können. Im späteren Verlauf der Arbeit wird der Einsatz des Crowdsourcing noch weiter verdeutlicht.

2.3. Gamification

Der Begriff Gamification (dt. Gamifizierung) bezeichnet nach der Definition von Deterding et al. „...the use of game design elements in non-game contexts.“ — also die Verwendung von Spielelementen außerhalb von Spielen [Deterding u. a., 2011]. Aufgekommen etwa im Jahr 2008, wurde Gamification innerhalb des Jahres 2010 populär. Heute werden die Techniken und Ideen der Gamification in einigen Softwareanwendungen eingesetzt. Insbesondere in den Bereichen

Werbung und PR-Aktionen wird es seinem ursprünglich angedachten Zweck allerdings oft entfremdet.

Dieser Zweck sollte dem User einen Mehrwert bieten, zum Beispiel in dem es den spielerischen Umgang mit alltäglichen Anwendungen ermöglicht und so potentiell unangenehme oder stark repetitive Aufgaben angenehmer gestaltet [Deterding u. a., 2011]. Zu den Spielelementen, welche sich in der Gamification wiederfinden, gehören nach Deterding et al., unter anderem:

- Interface-Design-Muster (Abzeichen, Ranglisten, etc.)
- Spielmechaniken
- Design-Prinzipien oder Heuristiken
- Konzeptuelle Modelle von Spieldesign-Einheiten
- Spieldesignmethoden wie Spieltest und playcentric design

Im Bereich der Mensch-Maschine-Kommunikation ist bekannt, dass eine positve Erfahrung und der Spaß bei der Verwendung einer Software zur Steigerung der Zufriedenheit führt [von Ahn und Dabbish, 2008]. Allerdings muss hierbei beachtet werden, dass trotz der spielerischen Elemente die geforderten Ziele durch den Nutzer ereicht werden. Weiterhin besteht leicht die Gefahr, Gamification-Elemente einzubinden, welche dem Nutzer keinen Mehrwert bieten. Optimalerweise erfüllen die zusätzlichen Elemente eine Aufgabe. Dies meint, dass der Nutzer von der Integration von Spiel-Elementen profitieren kann. So macht es für den Nutzer beispielsweise keinen Sinn, mit Auszeichnungen belohnt zu werden, welche für ihn keinen tieferen Sinn haben. Besser ist es, bei einem Nutzer ein positives Gefühl zu erzeugen. Dies kann beispielweise durch soziale Interaktion und der Teilhabe an einem größeren Ziel erreicht werden [McGonigal, 2011].

Innerhalb der Masterarbeit werden einige dieser Elemente Verwendung finden, um die Anwender zu einem spielerischen Umgang mit der Applikation zu ermutigen. Auch soll durch die Nutzung von motivierenden Abzeichen und Ranglisten die Partizipation am System erhöht und somit eine rege Beteiligung gefördert werden.

2.4. Mobile Computing

Die Verwendung einer mobilen Plattform zur Entwicklung einer Applikation beinhaltet einige Besonderheiten, sowohl bezogen auf die Entwicklung als auch auf den Betrieb der fertig gestellten Anwendung. Eingeschränkt wird die Entwicklung dabei durch Elemente wie geringen Speicherplatz und langsame Prozessor-Geschwindigkeit im Vergleich zur „normalen“

PC-Anwendungsentwicklung. Zur Laufzeit entstehen weitere Probleme, welche bei einer Anwendung innerhalb eines stationären Systems nicht bzw. selten auftreten können. Im Folgenden sollen einige dieser Herausforderungen näher erläutert und mögliche Lösungsansätze aufgezeigt werden.

2.4.1. Rechenkapazität und Speicherauslastung

Die Hardware mobiler Geräte wie Smartphones und Tablets hat sich in den letzten Jahren in ihrer Leistungsfähigkeit stark verbessert. Dies ist dem Anspruch der Nutzer geschuldet, die ihre Geräte für immer mehr Anwendungen nutzen möchten. Insbesondere die Darstellung von Multimedia-Elementen wie Filme und Spiele erfordern permanent technische Neuerungen, um den Erwartungen gerecht zu werden. Zwar kommt die Leistungsfähigkeit eines solchen Gerätes noch nicht an die eines handelsüblichen Computers heran, jedoch zeichnet sich eine Tendenz in diese Richtung ab, wie in Abbildung 2.1 zu sehen ist. Trotz stets verbesserter Hardware sollte der Applikationsentwickler die Ressourcen und die Prozessorlast des jeweiligen Gerätes schonen — dies ist insbesondere bei der Programmierung von Spielen wichtig. Der Grund hierfür liegt vor allem in der begrenzten Menge an vorhandener Energie, wie im nächsten Kapitel dargelegt wird.

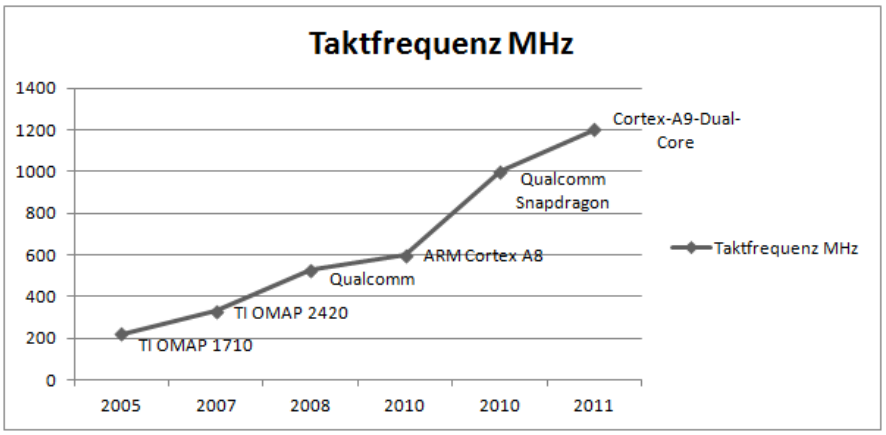

Abbildung 2.1.: Entwicklung der Rechenleistung von mobilen Prozessoren

Bei der Applikationsentwicklung ist zudem ein sorgfältiger Umgang mit den Speicherressourcen zu empfehlen. Die meisten auf dem Markt verfügbaren Geräte können zwar mittels einer SD-Karte um Speicherkapazität erweitert werden, jedoch ist bei einigen Smartphones nur ein interner Speicher von 512 Mb vorhanden [HTC, 2012]. Dieser ist allerdings durch die Installation von einigen umfangreichen Applikationen schnell erschöpft. Entwickler von Anwendungen sollten dem Nutzer daher die Installation auf einer externen SD-Karte ermöglichen. Auch zur

Laufzeit sollte die Applikation so wenig Speicherbedarf wie möglich aufzeigen: Je nach Gerät liegt die maximal verwendbare Arbeitsspeichergröße pro Anwendung zwischen 16 und 48 Mb, dieser Bereich darf durch die Applikation nicht überschritten werden [Developers, 2012b].

2.4.2. Mobilität, Laufzeit und Netzwerk

Die Verwendung eines mobilen Endgerätes, wie zum Beispiel eines Smartphones, bringt im Vergleich zu stationären Systemen einige Eigenschaften mit sich, die sich auf das Nutzerverhalten auswirken. So befinden sich 87% der Nutzer eines Smartphones „On-the-go“, während sie ihr Gerät bedienen. [thinkinsights with google, 2011]. Das heißt, ein Nutzer bewegt sich im öffentlichen Raum. Gleichzeitig ist die Nutzung zumeist durch äußere Faktoren gestört, so dass der User von der Verwendung des Gerätes abgelenkt wird [Krannich, 2010]. Der Entwickler sollte dem Applikationsnutzer deswegen die Option geben, eine Aktion jederzeit abzubrechen. Er muss ebenfalls berücksichtigen, dass der potentiell unkonzentrierte User Falscheingaben tätigen könnte. Folglich ist die Beachtung von Usability-Aspekten innerhalb der mobilen Entwicklung besonders wichtig.

Eine weitere Besonderheit der mobilen Geräte ist die begrenzte Zeit der Nutzung. Die Laufzeit eines solchen Gerätes hängt direkt von der verfügbaren Akkukapazität und dem Stromverbrauch der verbauten Komponenten ab. Die Möglichkeit zum erneuten Laden des Akkus ist in den meisten Fällen nicht gegeben, da der Nutzer diese Geräte oftmals, wie oben genannt, „On-the-go“ verwendet. Wie den Abbildungen 2.2 und 2.3 ersichtlich ist, liegt die Leistungsaufnahme bei modernen Smartphones im Stand-by-Zustand etwa bei 68 mW. Ist das Gerät im Idle-Zustand — ist also aktiv, führt aber keine Anwendungen aus — steigt die Leistungsaufnahme bereits auf 268 mW. Die Steigerung des Leistungsbedarfs liegt maßgeblich in der Aktivität des Displays und des Kommunikationsmoduls (GSM) begründet [Carroll und Heiser, 2010].

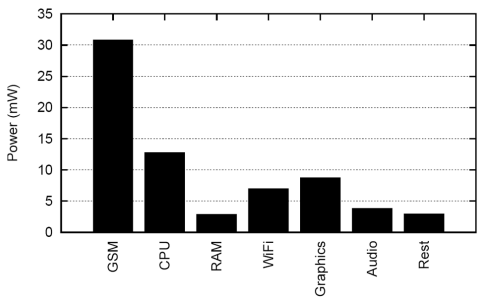

Abb. 2.2.: Stromverbrauch der Module im Stand-by-Modus

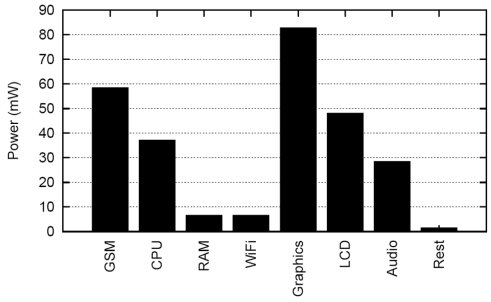

Abb. 2.3.: Stromverbrauch im Idle-Zustand ohne Hintergrundbeleuchtung

Eine intensive Nutzung des Geräts steigert die Leistungsaufnahme nochmals. Eine maximale Laufzeit von ein bis zwei Tagen ist bei modernen Geräten dementsprechend nicht ungewöhnlich. Um zumindest diese Laufzeiten gewährleisten zu können, müssen auf Entwicklerseite einige Vorgehensweisen beachtet werden. Erstens sollten benötigte Sensoren und Aktoren nur aktiv sein, wenn diese verwendet werden; so sollte etwa die Nutzung des GPS-Moduls zur Lokalisierung des Nutzers nur dann aktiv sein, wenn diese benötigt wird. Zweitens sollte auf einen schonenden Umgang mit anderen System-Ressourcen geachtet werden, da die intensive Nutzung dieser ebenfalls zu einer erhöhten Leistungsaufnahme führt.

Ein wichtiger Unterschied zu einem stationären System, wie zum Beispiel einem PC-System, ist die Verfügbarkeit einer Netzwerkverbindung. Die Qualität der aktuellen Verbindung hängt im mobilen Bereich stark von der aktuellen Netzabdeckung und etwaigen Störungsquellen ab. Weiterhin kann die verfügbare Bandbreite starken Schwankungen ausgesetzt sein, welche die Verfügbarkeit von Informationen einschränken. Sollen innerhalb einer Anwendung größere Mengen von Daten — zum Beispiel Bilder oder Videostreams — ausgetauscht werden, muss dies bereits bei der Entwicklung einer Applikation bedacht werden. Ein möglicher Lösungsansatz beim Upload von größeren Datenmengen an einen Server bzw. eine Datenbank ist das Zwischenspeichern dieser Daten, bis eine ausreichend schnelle Datenverbindung vorhanden ist (zum Beispiel WIFI - Netzwerk). Dem Nutzer erspart man so eine langwierige Wartezeit; gleichzeitig verhindert man einen möglichen Abbruch der Datenübertragung.

Für den Datenabruf — von zum Bespiel einem Webserver — sollte der Entwickler zudem ein Caching eingebaut haben. Somit wird sichergestellt, dass lediglich aktuelle Daten abgerufen werden. Damit der User die Anwendung im Offline-Modus nutzen kann, müssten bestimmte Daten einmalig herunterladbar und dann permanent abrufbar sein.

Weitere Informationen zu allgemeinen und in dieser Masterarbeit eingesetzten Techniken folgen im Kapitel zur Realisierung 6.

2.5. Android-Plattform

Android ist sowohl ein Betriebssystem für mobile Geräte wie Smartphones und Tablets, als auch eine Schnittstelle zur Entwicklung von Anwendungen für diese Geräte. Vorangetrieben durch die Open Handset Alliance — welche neben Google aus weiteren Partnern aus Telekommunikation, sowie Hard- und Software-Herstellern besteht [open handset alliance, 2011] — ist Android mittlerweile eine der führenden Plattformen. Die Nutzerzahlen sind seit der ersten Veröffentlichung im Jahr 2008 stetig gestiegen. Somit konnte Android im Jahr 2010 bereits einen Anteil von knapp 23% am Gesamtmarkt der mobilen Geräte verzeichnen [Gartner.com, 2010].

2.5.1. Architektur der Android-Plattform

Neben dem linux-basierten Betriebssystem für mobile Endgeräte, welches Aufgaben wie Speicherverwaltung, Prozess-Management, Netzwerk-Implementierungen und Treiber-Management übernimmt, bringt Android eine umfangreiche API für die Entwicklung von Anwendungen mit sich. Die Entwicklung von Applikationen erfolgt mit Java.

Die Hauptkomponenten der Android-Architektur gliedern sich in fünf Teile, welche in Abbildung 2.4 zu sehen sind. Auf der untersten Schicht befindet sich der Linux-Kernel. Hierauf aufbauend, finden sich die sogenannten Libraries. Sie enthalten Implementierungen für zum Beispiel SQLite, OpenGL, ein Media-Framework und den Surface Manager. Diese abstrahieren komplexere Systemkomponenten und bieten so eine vereinfachte Entwicklungsarbeit. Auf der Library-Ebene befindet sich auch die Android Runtime-Umgebung. Diese enthält weitere Kern-Bibliotheken, sowie die „Dalvik Virtual Machine“, welche die Laufzeitumgebung für Programme darstellt. Das System ermöglicht es, mehrere virtuelle Maschinen parallel laufen zu lassen. Dies ist notwendig, da jede Programminstanz innerhalb eines eigenen Prozesses agiert und somit eine eigene Laufzeitkomponente benötigt. Kompilierte Programme werden mittels des Tools „dx“ in das .dex Format transferiert, welches von der virtuellen Maschine gefordert wird.

Oberhalb der Libraries und der Android-Laufzeit-Umgebung befindet sich das Anwendungs-Framework. Dieses bietet Bibliotheken und Manager, welche über die API von Entwicklern angesprochen werden können. Dieses Anwendungs-Framework verfügt etwa über anwendungsrelevante Daten, Fensterverwaltung und Ressourcenzugriff und stellt die Grundlage für alle entwickelten Anwendungen dar.

Zuoberst befindet sich die Anwendungsebene — dort hinterlegt sind alle entwickelten Anwendungen — darunter befinden sich auch Applikationen, die bereits Teil des Systems im Auslieferungszustand sind. [Developers, 2011b]

Ein wichtiges Konzept von Android ist die Wiederverwendbarkeit von Komponenten in weiteren Anwendungen; eine Applikation kann dementsprechend ihre Fähigkeiten veröffentlichen: Anwendungen, die diese nutzen möchten, können auf diese Fähigkeiten zugreifen — wenn denn die system-seitigen Sicherheitseinschränkungen eingehalten werden. Bestimmte Komponenten müssen demnach nicht mehrfach implementiert werden.

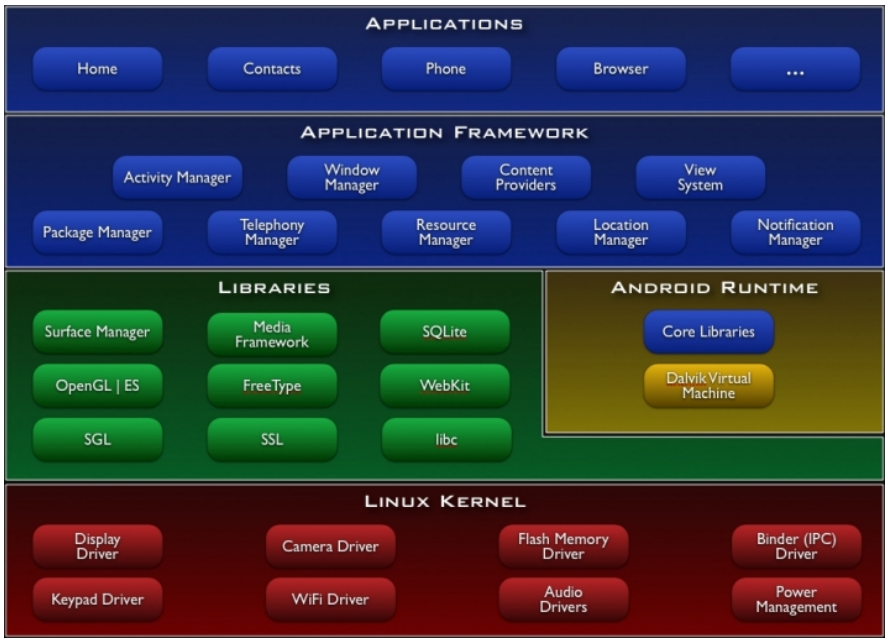

Abb. 2.4.: Architektur der Android-Plattform

2.5.2. Grundlegender Aufbau einer Anwendung

Die Anwendungsschicht von Android besteht aus vier Komponenten, die je unterschiedlichen Zwecken dienen [Developers, 2011a]:

Activities

Eine Aktivität in Android repräsentiert eine sichtbare Seite innerhalb einer Anwendung. Die Kombination von mehreren dieser Aktivitäten bildet die Grundlage einer kompletten Anwendung und ist somit für die Benutzer-Interaktion zuständig. Aktivitäten können innerhalb einer Applikation jederzeit aufgerufen werden. Auch Aktivitäten einer anderen Applikation sind aufrufbar, falls dies durch die Anwendung erlaubt wird. So kann zum Beispiel die android-interne Kameraaktivität aufgerufen werden, um ein Foto aufzunehmen.

Services

Ein Service bietet die Möglichkeit, einen Prozess oder eine lange laufende Aktion im Hintergrund durchzuführen, ohne die Nutzeraktionen zu blockieren. Ein Service bietet im Gegensatz zu einer Aktivität kein User-Interface. Eine Aktivität ist in der Lage, einen Service nur zu starten oder an sich zu binden, um mit diesem auch weiter interagieren zu können. Ein gutes Beispiel für einen Service ist das Abspielen von Musik, während der Nutzer andere Anwendungen verwendet.

Content provider

Content provider werden von Android verwendet, um Daten, die innerhalb einer Anwendung verwendet oder erstellt werden, zu speichern. Hierzu können Content provider verschiedene persistente Speicher nutzen, wie etwa die interne SQLite Datenbank, das Dateisystem oder der Speicher innerhalb eines Cloud-Dienstes. Der verwendete Content provider erlaubt es bestimmten Anwendungen, auf seine Daten zuzugreifen oder sie zu verändern. Android stellt beispielsweise einen Content provider zur Verfügung, welcher den Zugriff auf die Kontaktinformationen des aktuellen Nutzer möglich macht.

Broadcast receiver

Broadcast receiver ermöglichen die Reaktion auf systemweite Events wie beispielsweise Informationen über Akku-Zustände oder Internet-Verfügbarkeit. Ebenso wie Services verfügen Broadcast receiver über keine umfangreiche Nutzerschnittstelle. Sie können lediglich über die Statusleiste Nachrichten an den Nutzer übermitteln.

Abgesehen vom Content provider, welcher durch die Verwendung eines globalen Content Resolvers aktiviert wird, werden die o.g. genannten Komponenten mittels eines Intents aktiviert. Dieser Intent ist eine asynchrone Nachricht, die es erlaubt, verschiedene Komponenten zur Laufzeit aneinander zu binden. Um also eine Anwendung zu starten, muss mindestens ein Intent erstellt werden, der die initiale Aktivität aufruft.

Da den Komponenten einer Applikationen eine lose Kopplung zugrunde liegt, ist es wichtig festzulegen, welche Bestandteile eine Anwendung bilden. Dies geschieht über die XML-Datei „AndroidManifest.xml“. Diese Datei ist zwingender Bestandteil einer Anwendung. Innerhalb dieser Datei werden alle Applikations-Komponenten eingetragen; auch werden hier die von der Anwendung benötigten Zugriffsrechte spezifiziert. Weiterhin können noch Metainformationen angelegt werden, welche zum Beispiel die Version der Laufzeitumgebung festlegen oder Daten für die Veröffentlichung im Android-Market beschreiben.

Für die Verwaltung von Applikations-Ressourcen wie zum Beispiel Grafiken, Texten oder Sounds bietet die Android-Plattform einige gute Hilfsmittel. Durch die Vergabe eines eindeutigen Identifiers für jede angelegte Ressource können verschiedene Ressourcen innerhalb des Quellcodes jederzeit eingebunden werden. Eines der wichtigsten Features ist die Verwendung alternativer Ressourcen entsprechend der jeweiligen Gerätekonfigurationen. So können auflösungsabhängig beispielsweise verschiedene Grafiken genutzt werden; ebenso können je nach Landessprache unterschiedliche Texte platziert werden.

Für weitere Informationen zur Android-Plattform und ihrer Entwicklung wird auf weiterführende Literatur verwiesen wie beispielsweise in [Meier, 2010] und [Becker und Pant, 2010].

3. Anwendungsszenario

3.1. Standortbezogene Dienste

Die in dieser Masterarbeit konzipierte Anwendung soll der Aggregation von Informationen zu Sehenswürdigkeiten innerhalb eines Reisekontextes dienen. Um eine sinnvolle Verknüpfung von Information und dem tatsächlichen Ort zu erhalten, ist der Einsatz eines standortbezogenen Dienstes notwendig. Mit dem Aufkommen von mobilen Endgeräten wurde auch die Nutzung von Standort-Daten interessant [Steiniger u. a., 2006]. Moderne Smartphones, Tablets sowie Netbooks verfügen zumeist über einen intern verbauten GPS-Sensor oder ein Kommunikationsmodul für Mobilfunknetze. Diese sind in der Lage, die Position des Nutzers — je nach Umfeld — relativ genau zu bestimmen.

Das Open Geospatial Consortium definiert standortbezogene Dienste folgendermaßen:

> „A wireless-IP service that uses geographic information to serve a mobile user. Any application service that exploits the position of a mobile terminal." [OGC, 2008]

Der Nutzer kann also Informationen abrufen, welche sowohl die aktuelle Position, als auch den Zeitpunkt beachten. Diese an den aktuellen Kontext des Nutzers angepassten Informationen ermöglichen eine weitaus bessere system-seitige Filterung der abgefragten Daten im Vergleich zu nicht kontextbezogenen Daten. Der Nutzer erhält auf eine Anfrage dementsprechend Informationen, die er leicht weiterverarbeiten kann, ohne seinerseits eine Filterung durchführen zu müssen.

Innerhalb der standortbezogenen Dienste lässt sich zwischen zwei grundlegenden Vorgehensweisen unterscheiden [Steiniger u. a., 2006]:

Pull-Dienste

Pull-Dienste liefern Informationen erst dann, wenn diese vom Nutzer explizit abgefragt werden. Hierzu gehören beispielsweise Informationen über Restaurants in der Nähe des Nutzers oder das Bestellen eines Taxis an den aktuellen Standort. Pull-Dienste arbeiten reaktiv. Dies bietet den Vorteil, dass Sensorinformationen nur kurzweilig abgefragt werden müssen.

Push-Dienste

Push-Dienste übermitteln Informationen, auch wenn diese vom Nutzer nicht direkt abgefragt wurden. Die Übertragung der Daten wird über ein Event aktiviert. Dies kann das Eintreten in einen bestimmten Bereich sein oder die Aktivierung durch einen laufenden Timer im Gerät. Die von Push-Diensten bereitgestellten Informationen sind beispielsweise Werbeanzeigen, die an einen bestimmten Ort geknüpft werden oder Informationen über einen bestimmten Bereich innerhalb einer Stadt. Die Nutzung von Push-Diensten ist systemseitig aufwendiger, da der Nutzer keine Aktion durchführt. Folglich muss das System proaktiv handeln, damit Informationen zu einem passenden Zeitpunkt bereitgestellt werden. Um dies zu erreichen, werden beispielsweise Sensordaten benötigt, die über einen langen Zeitraum beobachtet werden. So muss etwa das GPS-Modul des Gerätes aktiv sein und dem System die aktuelle Position des Nutzers bereitstellen.

Verschiedene Basiskomponenten sind nötig, um die Infrastruktur eines standortbezogenen Dienstes aufzubauen. Diese werden im folgenden Kapitel vorgestellt und ihre Verbindung zueinander erläutert.

3.1.1. Komponenten von standortbezogenen Diensten

Die folgenden vier Basiskomponenten bilden die benötigte Infrastruktur für Standortbezogene Dienste laut Steiniger et al. [Steiniger u. a., 2006]:

Mobile Geräte

Für den Informationsabruf werden Empfangsgeräte benötigt — heutzutage also meist Smartphones, Laptops, Tablets o.a., aber auch Navigationssysteme im Automobilbereich. Mittels dieser Geräte können Nutzer Anfragen an ein System stellen und erhalten ortsbezogene und gefilterte Informationen. Die Ergebnisse der Abfrage können sowohl in Textform oder audio-visuell dargestellt werden.

Kommunikationsnetz

Ein vorhandenes Kommunikationsnetzwerk ist die zweite Basiskomponente von standortbezogenen Diensten. Mobilfunknetze wie UMTS oder GSM-Netze sind entsprechend Übertragungsrate und Bandbreite schnell und effizient. Die Geschwindigkeit der Datenübertragung variiert jedoch je nach Anzahl der verfügbaren Mobilfunk-Zellen und der aktiven Nutzer innerhalb eines Netzwerkes sowie nach Ausbau des Netzes am jeweiligen Standort.

Positionsbestimmung

Um eine Anfrage systemseitig auf den aktuellen Standtort des Nutzers abzustimmen, wird die Position des Nutzers benötigt. Für die Bestimmung der aktuellen Position stehen mehrere Möglichkeiten zur Verfügung.

Eine sehr genaue, dafür aber langsame, Bestimmung kann über das Global Positioning System (GPS) erfolgen. GPS ist auf den meisten modernen Mobilgeräten verfügbar und innerstädtisch mit einer Abweichung von wenigen Metern sehr präzise. Die Nutzung der geräteinternen GPS-Module erzeugt allerdings eine größere Last am Gerät und erhöht somit den Akkuverbrauch.

Eine weitere Möglichkeit ist die Positionierung mittels des Mobilfunknetzwerkes. Die Position des Nutzers wird hier über die Funkzelle bestimmt, in der sich dieser aktuell befindet. Die Ortung erfolgt sehr schnell und erhöht die Last am Gerät nicht. Die Methode ist jedoch relativ ungenau, wenn auch für eine initiale Ortung meist ausreichend.

Für eine Ortung innerhalb von Gebäuden, wie etwa Museen, stehen noch weitere Verfahren zur Verfügung, so beispielsweise die Triangulation von erreichbaren WIFI-Netzwerken.

Services und Anwendungen

Die vom Nutzer abgefragten Dienste werden durch die Service- und Contentprovider bereitgestellt. Als Schnittstellen für Anfragen verarbeiten sie die benötigten Informationen. Unter Berücksichtigung der aktuellen Position sowie weiteren Faktoren, wie Zeit oder Kontext des Nutzers, können so Ergebnisse erzeugt werden, die auf den jeweiligen Nutzer zugeschnitten sind. Beispielhafte Anwendungen wären das Finden einer passenden Route, die Suche nach Restaurants in der Nähe oder Informationen zu Sehenswürdigkeiten. Da ein Serviceanbieter nicht über alle Informationen verfügen kann, bedient er sich Contentprovidern, die geographische Basisdaten oder ortsabhängige Informationen anbieten.

Im Folgenden soll das Anwendungsszenario der in der Masterarbeit entstandenen Anwendung erläutert werden. Es wird erklärt, welche Komponenten benötigt werden und welche Dienste die Applikation dem Nutzer bietet. Weiterhin wird die Verwendung der Software im städtischen Kontext erläutert.

3.2. Anwendungsszenario GeoBee

Im Folgenden soll ein Anwendungsszenario für die in dieser Masterarbeit entstandene Android-Applikation *GeoBee* vorgestellt werden:

Wie im Abschnitt Motivation 1.1 bereits beschrieben, soll der Nutzer mithilfe der Anwendung die Sehenswürdigkeiten und interessanten Plätze unbekannter Orte entdecken können. *GeoBee* ist also eine Alternative zu herkömmlichen Reiseführern in analoger oder digitaler Form, welche sich zwangsläufig oftmals nicht den Interessen der Nutzer anpassen können. Der Vorteil von *GeoBee* liegt im Einsatz von Crowdsourcing-Mechanismen. Die Empfehlung von Stätten oder auch unbekannten Orten erfolgt nicht durch wenige Reisespezialisten sondern durch viele Personen, die in ihren Vorlieben divers sind wie die Nutzer selber.

Damit ein Reisender das Potential von *GeoBee* ausschöpfen kann, sollte er die Anwendung über ein mobiles Endgerät nutzen. So kann der Nutzer die Stadt „auf eigene Faust“ entdecken oder die Empfehlungen und Touren nutzen, welche die Applikation bereitstellt. Jeder Anwender der Software ist gleichzeitig sowohl Nutzer als auch Autor und kann dementsprechend auch seine selber entdeckten Favoriten hinzufügen. Die Anzahl der in der Software vorhandenen Empfehlungen ist hierbei unbegrenzt. Da sich eine Stadt oder Region in vielerlei Hinsicht stetig wandelt, sind die vorhandenen Informationen durch die Nutzer editierbar. Es wird dadurch sichergestellt, dass die Informationen zu verschiedenen Orten erweitert und aktualisiert werden können.

Da der Reisende zumeist „On-the-go“ ist (vgl. [Krannich, 2010]), soll die Nutzung der Anwendung möglichst intuitiv und fehlerresistent sein und den Reisenden bestmöglich unterstützen. Der Fokus des Nutzers soll allerdings weiterhin auf der Umgebung liegen und nicht auf die Applikation gerichtet sein. Es wurden deswegen Designkonzepte entworfen, die es der Software ermöglichen, proaktiv zu agieren. Beispielsweise wurde die Navigation zu einem unbekannten Ort nicht in Form einer Kartennavigation gestaltet, sondern in Form eines intuitiven Kompasses, welcher den Nutzer zum Ziel leitet, ohne zu viel Aufmerksamkeit zu verlangen. Ein weiteres Beispiel ist die geplante Einbindung von Audiofeatures wie Audiotouren oder Audiokommentaren zu Sehenswürdigkeiten. Der Nutzer erhält dadurch Informationen, ohne dass er sich auf sein mobiles Gerät fokussieren muss.

Folgende Komponenten sind geplant:

Orte in der Umgebung

Dem Nutzer sollen die Sehenswürdigkeiten in seiner Nähe angezeigt werden. Diese werden anhand der aktuellen Position des Nutzers ermittelt und an das mobile Endgerät übermittelt.

Der Nutzer kann die Ergebnisse innerhalb einer Listenansicht einsehen und für weitere Informationen auswählen.

Hinzufügen eines Ortes

Jeder Nutzer kann eigene Wort- und Bildeinträge hinzufügen. Dies stellt eine der Kernfunktionalitäten dar. Weiterhin können noch Schlagworte und eine Bezeichnung für den Ort oder das Event eingegeben werden. Die aktuelle Position des Nutzers sowie der Benutzername und ein Zeitstempel werden durch die Anwendung automatisch hinzugefügt. Hier bieten sich noch einige Erweiterungsmöglichkeiten an. Beispielsweise könnte der Nutzer noch weitere Bilder, Audioaufnahmen oder Videoaufnahmen hinzufügen. Auch könnte man Links zu anderen Plattformen herstellen, beispielsweise ein Link zu Wikipedia mit weiterführender Beschreibung für ein historisches Gebäude.

Nutzerprofil

Jeder Nutzer erhält ein Profil. Er kann dort seine bisher getätigten Einträge, erhaltenen Abzeichen und erreichten Punktestände abrufen, sowie seine Nutzerinformationen anpassen. Andere Nutzerprofile sind über die Detailansicht eines Eintrages zu finden. Falls die eigenen Vorlieben dem eines anderen Nutzers oft entsprechen, könnte es die Anwendung ermöglichen, andere Nutzer als Favoriten zu speichern.

Suchfunktion

Der Nutzer kann nach Orten suchen. Dies kann sowohl per Freitextsuche erfolgen als auch mittels der hinterlegten Vorlieben des Nutzers. Diese Vorlieben kann der Nutzer innerhalb der Anwendung anhand von Schlagwörtern einstellen. Diese können jederzeit an die Bedürfnisse angepasst und verändert werden. Auch kann man nach Orten suchen, die sich nicht unbedingt in der Nähe des Nutzers befinden. So wird es möglich, bereits vor Beginn der Reise Sehenswürdigkeiten am Reiseziel zu entdecken und diese vorzumerken.

Kartenansicht

Mittels einer Kartenansicht soll der Nutzer die Position von Sehenswürdigkeiten betrachten und weitere Ziele in der Umgebung ausmachen können. Für eine erste Orientierung in einer fremden Gegend ist eine Kartenübersicht zudem ebenso hilfreich wie für die Zusammenstellung einer Reiseroute.

Kompass

Ein intuitiver Kompass soll den Nutzer innerhalb einer Umgebung zu seinem Ziel führen. Der Nutzer kann so auch ohne stetige Konsultierung der Karte seinen Weg einfach finden und dabei eigene Eindrücke von seiner näheren Umgebung sammeln.

Spielelemente

Teil der Anwendung sollen spielerische Elemente aus dem Bereich der Gamification sein: Ranglisten für Nutzer, die besonderes schöne oder besonders viele Orte hinzugefügt haben. Abzeichen, die ein Nutzer bei dem Erreichen eines systemseitigen Ziels erhält, sind ebenfalls Bestandteile der Anwendung. Entsprechende Ziele wären beispielsweise das erste Hinzufügen eines Ortes, die erste Bewertung oder ein erster Kommentar. Die Erweiterung dieser Ziele soll jederzeit möglich sein, um sehr aktive Nutzer auch weiterhin darin zu bestärken, sich zu beteiligen.

Anlegen von Routen

Der Nutzer soll die Option haben, sich mehrere Sehenswürdigkeiten zu einer Route zusammenzustellen — vor Reiseantritt oder vor Ort. Sobald die Route gestartet wird, leitet das System ihn von Ort zu Ort. Auch sollen Autoren Einträge zu einer Route zusammenfassen können. Sie können eigene oder auch andere Einträge zu einer Route hinzufügen und diese im Anschluss für andere Benutzer freigeben. Routen sollen, genauso wie normale Orte, bewertet und kommentiert werden können. Kommt ein Besucher einer Route an einen Ort, an dem beispielsweise Ton-, Bild- oder weitere Textinformationen hinterlegt sind, so werden diese vom System automatisch abgespielt oder angezeigt.

Heatmaps

Die Implementierung von Heatmaps soll es einem Nutzer ermöglichen, bestimmte Informationen übersichtlich auf einer Karte angezeigt zu bekommen. So kann dieser beispielsweise die Bereiche einer Stadt ausmachen, die am meisten Sehenswürdigkeiten aufweisen oder Stadtteile markieren lassen, welche seinen Vorlieben am nächsten kommen. Die Darstellung als Heatmap ist also ein Abstraktions-Feature, durch welches bestimmte Bereiche schnell und einfach ausgemacht werden können.

Diese Komponenten sind für die Entwicklung der Applikation geplant. Die konkreten Anforderungen an die Anwendung sollen im Kapitel zur Analyse 4 vorgestellt werden.

4. Analyse

Der erste Teil dieses Kapitels befasst sich mit bereits bestehenden Arbeiten zu Standortbezogenen Diensten. Exemplarisch vorgestellt werden dazu im Abschnitt „Related Work“ 4.1 drei Anwendungen mit ihren jeweiligen Besonderheiten und ihrem jeweiligen Potential. Zusätzlich erfolgt ein Vergleich der Funktionalitäten zwischen *GeoBee* und den betrachteten Arbeiten. Im Anschluss daran werden die allgemeinen Anforderungen an eine standortbezogene Crowdsourcing-Applikation erarbeitet — dadurch wird der Kern der Anwendung erkennbar, welcher den hauptsächlichen Unterschied zu anderen Diensten ausmacht. Der Abschnitt „Anforderungen an GeoBee“ 4.3, wird genutzt, um die wichtigsten Applikationsmerkmale und Anforderungen an die Anwendung aufzuzeigen. Es werden sowohl konkrete Anwendungsfälle als auch erste prototypische Konzepte diskutiert. Die Konzepte dienen als Grundlage für die Entwicklung der Anwendung und die Konzipierung der Nutzerschnittstelle.

4.1. Related Work

In diesem Abschnitt werden drei Anwendungen vorgestellt — wie bei GeoBee handelt es sich bei den Applikationen um standortbezogene Dienste. Die aufgezeigten Systeme werden in ihrer Funktionalität, ihrem Umfang und ihren besonderen Merkmalen beschrieben.

4.1.1. Google Places

Google — ursprünglich lediglich eine Suchmaschine im World Wide Web — bietet mittlerweile auch viele weitere Dienste an, welche es dem Nutzer ermöglichen, verschiedenste Aufgaben zu erledigen. Mehr als 20% der Suchanfragen auf Google.com beziehen sich mittlerweile auf standortbezogene Daten [Noack, 2011]. Somit ist es sinnvoll, eine Software anzubieten, welche diese Daten für den Nutzer möglichst angenehm darstellt. Google Places ist etwa ein Dienst, den User nutzen können, um auf der Basis des Dienstes Google Maps potentiell sehenswerte Orte und Geschäfte in die Kartenlandschaft einzutragen und so mit anderen zu teilen. Die User können diese Orte beschreiben, bewerten und Fotos hinzufügen. Anhand von Schlagwörtern

und Suchbegriffen werden Nutzern eine Vielzahl von Orten in ihrer jeweiligen Umgebung vorgeschlagen, welche sie besuchen könnten.

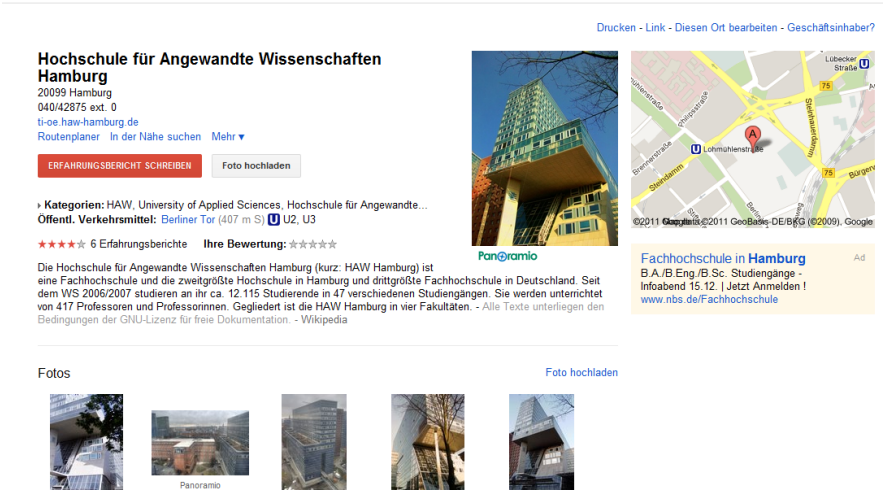

Abb. 4.1.: Google Places innerhalb eines Web-Browsers

Zu der Zielgruppe dieses Dienstes gehören dementsprechend auch Geschäftsleute, die beispielsweise ihren Gastronomiebetrieb auf der Webseite von Google Places eintragen und um beliebige Informationen ergänzen können; so etwa Kontaktadressen, Öffnungszeiten, verfügbare Bezahlsysteme oder relevante Videos. Auch Statistiken können abgefragt werden — zu der jeweiligen Anzahl der Suchanfragen, verwendeten Suchwörtern, Herkunft der Suchanfragen u. ä.. Mit diesen Informationen können Unternehmer auf Kundenwünsche und Anforderungen gezielt reagieren und beispielsweise an einem Standort mit vielen Anfragen eine Zweigstelle eröffnen.

Neben der Web-Anwendung existieren auf verschiedenen Plattformen mobile Versionen von Google Places, diese können Nutzer lokalisieren und führen für sie eine kontextbezogene Suche durch. Auch die gezielte und selektierende Suche mit Hilfe von Typologisierungen wie etwa „Restaurant“ oder Schlagwörtern wie „italienische Küche“ wird bei diesen mobilen Version angeboten. Der Nutzer eines mobilen Endgerätes kann so in seiner unmittelbaren Umgebung schnell und einfach nach Orten, Diensten oder Events suchen. Mittlerweile werden so ca. 50 % der Anfragen auf Google Maps [Noack, 2011] — und somit auch innerhalb des integrierten Google Places — von einem mobilen Gerät aus gestellt.

Der Dienst Google Places erfreut sich großer Beliebtheit und zeigt, das Geoinformationen den Nutzern einen Mehrwert bieten. So existieren 50 Millionen Seiten innerhalb von Places, wovon acht Millionen Einträge von Geschäftsleuten stammen, welche aktuelle Informationen zu ihrem Unternehmen veröffentlichen. Weiterhin wurden durch die Nutzer bereits fünf Millionen

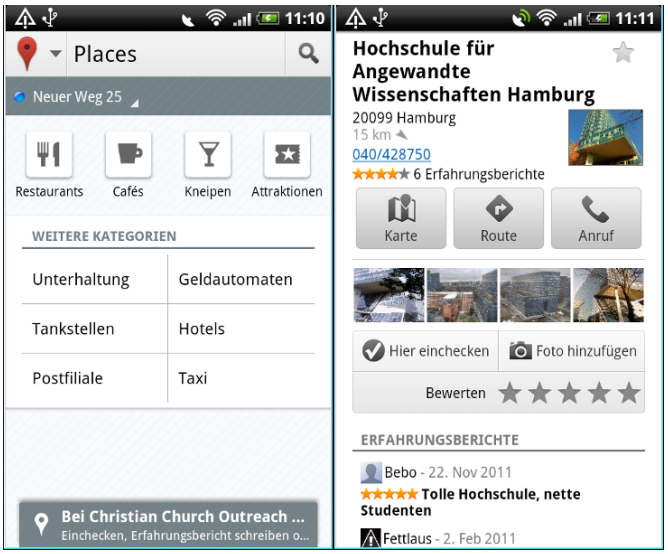

Abb. 4.2.: Places-Ansicht auf einem Android-Gerät

Bewertungen und Kommentare für entsprechende Orte und Geschäfte abgegeben [Noack, 2011]. Mit Diensten wie Google Latitude können Nutzer an Orten „einzchecken“ und somit ihren Standort und ihre Vorlieben mit Freunden und Bekannten teilen.

Die Anwendung *GeoBee* bietet ähnliche Features wie Google Places. Allerdings ist die innerhalb dieser Masterarbeit konzipierte Applikation mehr auf die Nutzung durch Touristen und Besucher einer fremden Stadt ausgelegt. Durch geplante Module wie die Möglichkeit des Erstellens von Touren oder die Einbindung von Audiokommentaren und Audiotouren liegt der Fokus in dieser Anwendung auf der Vermittlung von Informationen und Hintergründen zu Sehenswürdigkeiten. Weiterhin machen Heatmaps eine Kategorisierung von Stadtteilen und bestimmten Gebieten über verschieden Schlagwörter möglich. So kann sich ein Stadtfremder durch einen Blick auf eine Karte über die verschiedenen Gegebenheiten einer Stadt informieren.

4.1.2. Mobile Phone Tour Guide System

Die Arbeit von Bao et al. „Integration of Multimedia and location-based Services on mobile phone tour guide system“ [Bao u. a., 2009] beschreibt die Implementierung eines mobilen Dienstes zur Darstellung von multimedialen Inhalten innerhalb einer Kartenanwendung für Touristen. Der entstandene Dienst — der im Jahr 2009 publizierten Arbeit — beinhaltet alle grundlegenden Komponenten einer standortbezogenen Anwendung. Der Nutzer wird anhand eines GPS-Moduls lokalisiert und die entsprechend verknüpften Inhalte anschließend durch einen Remote-Service zur Verfügung gestellt. Neben der Anzeige von Kartenmaterial für die aktuelle Position des

Nutzers kann auch ein einfaches Routing zwischen zwei markierten Orten vollzogen werden. Ebenso können während der Verwendung der Applikation multimediale Inhalte zur Verfügung gestellt werden — abhängig vom aktuellen Standort. Neben der Möglichkeit, die teilweise sehr umfangreichen Daten vorab an einem Terminal auf das mobile Gerät zu transferieren, werden weitere Daten zur Laufzeit der Anwendung komprimiert verschickt. Dies spart Bandbreite und erhöht die Kommunikationsgeschwindigkeit zwischen Client- und Server-Anwendung.

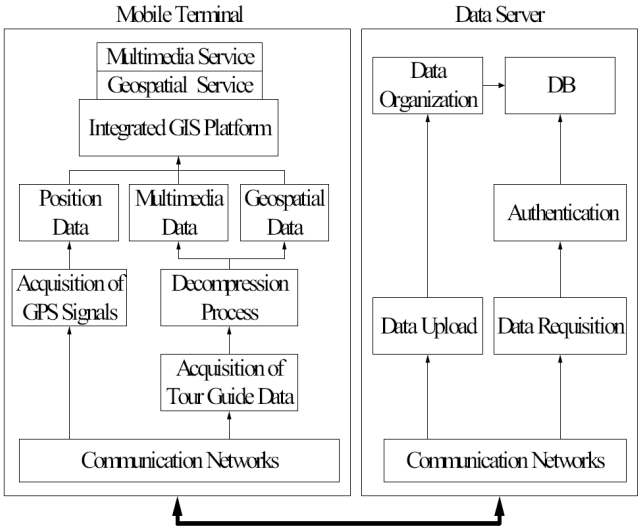

Abb. 4.3.: Funktionales Diagramm [Bao u. a., 2009]

Die Darstellung der Karten erfolgt hier nicht über eine vorhandene Anwendung, sondern wird durch die Applikation selbst bereitgestellt. Hierzu liegen vorbereitete Geoinformationen über bestimmte Regionen auf einer Datenbank vor, welche durch die Client-Anwendung abgefragt werden können. Diese zeichnet anhand der erhaltenen Daten eine Karte mittels einfacher geometrischer Formen und fügt weitere interaktive Elemente auf der Karte hinzu. Die Multimediainhalte wie zum Beispiel Audiotouren, Bilder oder ergänzendes Videomaterial liegen auf dem mobilen Gerät bereits vor. Um sie zum richtigen Zeitpunkt anzuzeigen, werden sie, mittels einer ortsabhängigen ID, an den jeweiligen Standort gebunden. Befindet sich der Nutzer an einer Position für die Inhalte vorliegen, werden diese automatisch angezeigt.

Innerhalb der Arbeit von Bao et al. werden die Ergebnisse eines Systemtests vorgestellt, welche eine schnelle Systemreaktion gezeigt haben. Das generierte Kartenmaterial wurde von den Testpersonen als angenehm und übersichtlich empfunden. Im Vergleich zu traditionellen Audio-Guide-Systemen bietet die vorgestellte Anwendung weitaus mehr Informationen für Reisende

Abb. 4.4.: Resultierende Anwendung [Bao u. a., 2009]

und ermöglicht eine autonomere Interaktion mit dem System. Weiterhin ist das System flexibler und kann leicht um weitere Module ergänzt werden.

Besondere Merkmale des vorgestellten Systems sind zum einen die Generierung von Kartenmaterial anhand der aktuellen Position des Nutzers, zum anderen die intensive Nutzung von multimedialen Inhalten. Multimediale Inhalte nehmen auch bei *GeoBee* einen zentralen Raum ein: Nutzer sollen die Möglichkeit haben, zu vorhandenen Orten ihre Bilder und Audioaufnahmen hinzuzufügen. Das Erzeugen von standortbezogenen Videoaufnahmen ist ebenfalls eine denkbare Erweiterung von *GeoBee*, welche zunächst aber nicht umgesetzt wird.

4.1.3. Place-Tags, Discovering and Promoting Places

Das letzte hier angeführte Projekt stammt aus dem Jahr 2010. Delev et al. stellen in ihrer Arbeit „Place-Tags, Discovering and Promoting Places Through Mobile Phones and Collaborative Filtering“ [Delev u. a., 2010] eine mobile Anwendung vor, welche das Verschlagworten von Plätzen in der Umgebung ermöglicht. Die entstandene Anwendung lässt Nutzer Informationen zu Standorten anlegen und diese über einen Webdienst speichern und teilen. Die so generierten Daten werden serverseitig aggregiert und können anhand von Schlagwörtern und Nutzerpositionen anderen Anwendern zur Verfügung gestellt werden.

Der Nutzer kann eigens erstelle Orte einsehen, bearbeiten und mit der Serverapplikation synchronisieren. Weiterhin können Orte mittels einer Suchfunktion gesucht oder anhand der aktuellen Position mögliche Ziele vorgeschlagen werden.

Die technische Realisierung der Applikation von Delev et al. erfolgt mittels J2ME auf einem GPS-fähigen Nokia-Gerät. Der Webdienst wurde mittels Phyton erstellt und läuft innerhalb eines von Google bereitgestellten Clouddienstes namens Google App Engine. Die vom Nutzer erstellten Daten werden mittels *JSON* an die Serverapplikation übertragen und gespeichert.

Nicht nur auf einem mobilen Endgerät, auch auf Webbrowsern können die angelegten Orte mittels Google Map oder Microsoft Live Map eingesehen werden. Angelegte Orte können durch andere Nutzer bewertet werden. Diese Bewertungen, sowie die Informationen über Häufigkeiten von Besuchen, werden server-seitig genutzt, um durch kollaborative Filter-Algorithmen Empfehlungen für andere Nutzer zu erstellen. Die Filterung wird hier mittels des Korrelationskoeffizienten nach Pearson erstellt [Delev u. a., 2010]. Dieser Koeffizient ermöglicht es, die Ähnlichkeit zwischen zwei gegebenen Datensätzen zu bestimmen. Der verwendete Algorithmus ist im Vergleich zu einer Euklidischen Distanz zwischen zwei Merkmalen etwas komplizierter, bringt allerdings ein besseres Ergebnis hervor, wenn Datensätze nicht gut normalisiert sind.

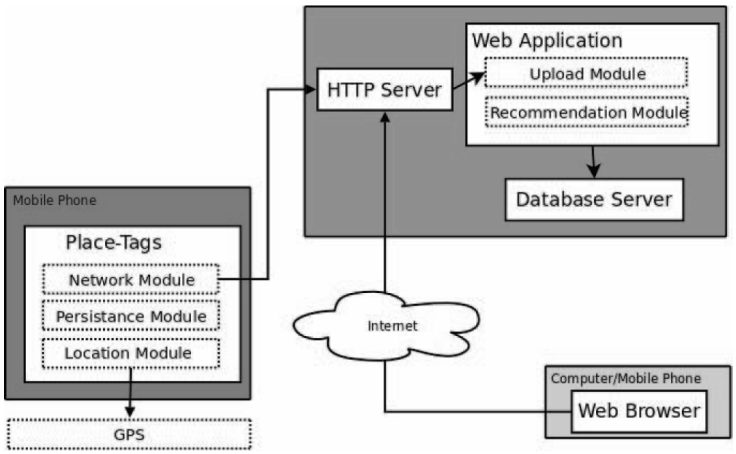

Abb. 4.5.: Software-Architektur als Block-Diagramm [Delev u. a., 2010]

Auch bei *GeoBee* können die generierten Informationen zwischen Nutzern ausgetauscht werden. Die Applikation von Delev et al. kann jedoch keine multimedialen Inhalte aufnehmen. Dieses Feature ist bei *GeoBee* gegeben und könnte so diese Anwendung für Benutzer noch attraktiver machen und die Partizipation am System erhöhen. Ebenso stellt die Anwendung von Delev et al. keine Gamification-Elemente bereit. Dies ist eine weitere Möglichkeit, dem Nutzer ein besseres Erlebnis bei der Verwendung der Applikation zu bieten und die Partizipation so weiter zu steigern.

Alle obig vorgestellten standortbezogenen Dienste bieten Informationen über sehenswerte Orte, sind jedoch teils sehr unterschiedlich konzeptioniert. Bei Google Places sowie bei Delev et al. können die Nutzer Orte verschlagworten und weitere Inhalte hinzufügen. Die so generierten Informationen werden genutzt, um anderen Personen Vorschläge zu etwaigen Zielen zu geben. Der in dieser Masterarbeit entstandene Dienst vereint die Fähigkeiten dieser Anwendungen und erlaubt so ein umfangreiches Nutzererlebnis. Im Abschnitt zur Realisierung 6 werden

zudem weitere Verbesserungen vorgeschlagen, welche zum Beispiel die Nutzbarkeit der mobilen Applikation optimieren sollen.

Die drei vorgestellten Anwendungen wurden gewählt, um einzelne Aspekte der hier entwickelten Applikation herauszustellen. Google Places ist für die Öffentlichkeit erreichbar und wird von dieser sehr gut angenommen. Wie obenstehend im Text genannt, existieren mittlerweile ca. 50 Millionen Seiten auf Google Places, welche sowohl Sehenswürdigkeiten als auch Geschäfte abdecken [Noack, 2011]. Dies zeigt, dass ein solcher Dienst einen Mehrwert für die Nutzer bietet und ein starkes Interesse an geographischen Diensten besteht.

Die zweite vorgestellte Arbeit, „Mobile Phone Tour Guide System“, beinhaltet die Option, multimediale Inhalte zu nutzen und so die erzeugten Daten für den Nutzer gut zu visualisieren. Auch *GeoBee* nutzt mediale Inhalte, um die Einträge für den Nutzer interessanter zu machen und eine bessere Darstellung zu erreichen.

Zuletzt wurde die Arbeit von Delev et al. vorgestellt. Nutzer dieser Anwendung können Orte bewerten und somit Empfehlungen für andere Nutzer generieren. Die so erstellten Vorschläge können anhand der aktuelle Position und mittels der Eingabe von Schlagwörtern gesucht werden. Nutzern der Anwendung *GeoBee* ist es ebenfalls möglich, Bewertungen und Kommentare abzugeben. Weiterhin kann ein Nutzer einen Ort als „besucht“ markieren. Auf Grundlage dieser Empfehlungen sowie den bereits besuchten Orten kann der Nutzer Vorschläge für weitere interessante Plätze erhalten.

Die besonderen Merkmale werden in Tabelle 4.1 noch einmal übersichtlich dargestellt.

Dienst	**Öffentlich**	**Medien**	**Sozial**	**Karten**	**Navigation**	**Touren**
Google Places	X	X	X	X	X	-
Mobile Phone Tour	-	X	-	X	X	-
Place-Tags	-	-	X	X	-	-
GeoBee	-	X	X	X	X	(X)

Tabelle 4.1.: Merkmale der verschiedenen Anwendungen

4.2. Anforderungen an standortbezogenes Crowdsourcing

Dieses Kapitel wird genutzt, um mögliche Anforderungen an standortbezogene Dienste festzulegen. Insbesondere wird hier auf die Anforderungen an mobile Crowdsorcing-Applikationen eingegangen. Zunächst ist es wichtig, dass ein solcher Dienst einen mobilen Charakter aufweist. Das bedeutet, dass die Crowdsourcing-Anwendung im mobilen Umfeld sinnvoll eingesetzt wird. So würde die Lösung eines Problems — wie beispielsweise die Identifikation von Elementen in

Bilder — durch mobiles Crowdsourcing wenig sinnvoll sein, da der mobile Charakter hier keinen Mehrwert verspricht. Problemfelder wie das Verschlagworten von Orten oder die Aufnahme von Gebäuden innerhalb einer Stadt würden hingegen auf einem stationären System wenig Sinn ergeben.

Eine weitere Bedingung, die für einen sinnvollen Einsatz eines solchen Dienstes gegeben sein muss, ist die Nutzung der Positionsdaten des Nutzers. Dies kann, wie oben beschrieben, durch ein GPS-Modul oder durch Funkzellenortung geschehen. Werden diese Daten nicht innerhalb der Anwendung genutzt oder sind sie für die Lösung der Aufgabe nicht zwingend notwendig, können sie ebenso auf einem stationären Systen bearbeitet werden. Ausgenommen hiervon wären Applikationen, die darauf ausgelegt sind, in speziellen Nutzerkontexten verwendet zu werden. So beispielsweise während einer Fahrt in einem öffentlichen Verkehrsmittel oder in Wartesituationen. Diese Anwendungen sind allerdings nicht für einen standortbezogenes Szenario entworfen worden und sollen deswegen nicht Teil dieser Arbeit sein.

Betrachtet man die Notwendigkeit einer verfügbaren Lokalisierung, ergibt sich hieraus eine weitere Anforderung. Die Anforderungen an die Genauigkeit der aktuellen Nutzerposition dürfen bei der Konzipierung einer mobilen Crowdsourcing-Applikation nicht zu hoch angelegt werden. Auch wenn moderne Smartphones und GPS-Geräte eine relativ hohe Auflösung der Position bieten, ist — je nach Erreichbarkeit von vorhandenen Satelliten — mit einer Abweichung von einigen Metern zu rechnen. Auch die Lokalisierung mittels Funkzelle bringt eine Abweichung mit sich, welche je nach Gebiet, Größe der Funkzellen und genutztem Verfahren zwischen fünf Metern und mehreren Kilometern beträgt. Der Einsatzweck von standortbezogenen Daten sollte demnach vorher mit den Anforderungen an das System abgestimmt werden. Beispielsweise ist die genaue Lokalisierung von kleinen Objekten in der Umgebung — ohne Einsatz von ergänzenden Techniken — nicht ohne weiteres möglich und eignet sich von daher nicht für eine mobile Crowdsourcing-Applikation.

Bedingt durch die mobile Nutzung der Anwendung und der somit oft schlechten Verbindung zu einem Netzwerk mit ausreichend großer Bandbreite, ergibt sich eine weitere Anforderung an des System. Die Crowdsourcing-Applikation sollte so gestaltet werden, dass der benötigte Datentransfer zwischen Client- und Server-Applikation möglichst klein gehalten wird. Wie der Abbildung 4.6 zu entnehmen, schwanken die verfügbaren Bandbreiten je nach verwendetem Standard. Diese stellen allerdings die maximalen Bandbreiten dar und können je nach Umfeld deutlich schlechter ausfallen.

Der Datenverkehr muss im Verhältnis zur Laufzeit also begrenzt werden, um die etwaig eingeschränkt zur Verfügung stehende Datenmenge nicht allzu sehr zu belasten und die Wartezeit bei der Verwendung der Applikation möglichst gering zu halten. Eine Möglichkeit, welcher

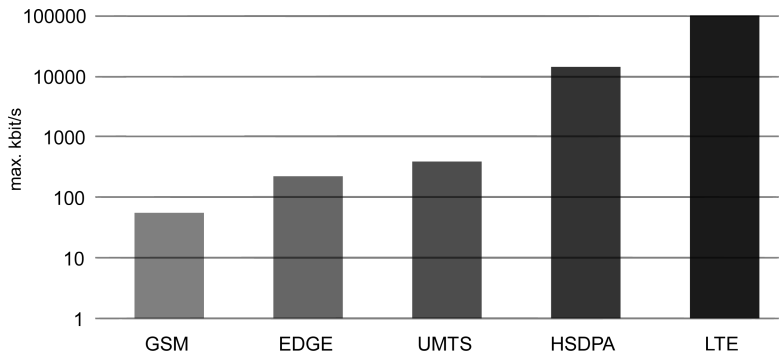

Abb. 4.6.: Bitraten der unterschiedlichen Mobilfunksysteme [McZusatz, 2011]

sich *GeoBee* bedient, ist es, größere Datenmengen in einem Cache vorzuhalten. Werden durch den Nutzer beispielsweise Bilder erzeugt, die mit einem Server synchronisiert werden sollen, werden diese temporär gespeichert und erst dann zum Server übertragen, wenn ein ausreichend gutes Netz zur Verfügung steht (WIFI). Zur Übertragung von generierten Datensätzen oder Serverantworten sollte ein möglichst kleines Format verwendet werden. Ebenfalls sollten die zu übertragenden Daten auch hier nur die Informationen enthalten, die aktuell erforderlich sind.

Es besteht eine weitere wichtige Anforderung an mobile Dienste: Da sich der Nutzer in einem mobilen Kontext befindet, ist er leicht externen Störungen und Unterbrechungen ausgesetzt [Krannich, 2010]. So kann ein externes Ereignis — beispielsweise wird dem Nutzer eine Frage gestellt — zu einer Fehleingabe oder einer nötigen Unterbrechung führen. Bei der Entwicklung einer mobilen Anwendung — insbesondere im Bereich standortbezogener Dienste — sollte eine ungewollte Störung des Nutzers in Betracht gezogen werden. So muss es dem Anwender möglich sein, Eingaben zu korrigieren oder Schritte rückgängig zu machen. Ein weiterer Aspekt, der hier beachtet werden muss, ist die Nutzungsdauer. Ein Anwender, der eine Applikation im mobilen Kontext nutzt, kann dies nur für eine gewisse Zeit konzentriert tun. Erfordert eine Anwendung die Konzentration des Nutzers über einen längeren Zeitraum, so sollte sie die Möglichkeit bieten, pausiert zu werden und den aktuellen Stand abzuspeichern. Ist der Nutzer dann wieder in der Lage, sich auf die Anwendung zu konzentrieren, kann er seine Eingaben fortführen, ohne schon getätigte Eingaben zu wiederholen. Ist die Crowdsourcing-Applikation in ein Spiel eingebettet, so kann der Nutzer eine pausierte Anwendung an der Stelle weiternutzen, an der er unterbrochen wurde.

4.3. Anforderungen an GeoBee

Dieser Abschnitt soll die konkreten Anforderungen an die entwickelte Applikation in Form von Anwendungsfällen aufzeigen. Neben den im vorherigen Abschnitt erörterten Anforderungen, werden hier die Funktionalitäten vorgestellt, welche die Anwendung bereitstellen muss um das Anwendungsszenario aus 3.2 zu erfüllen. Weiterhin sollen die im Kapitel „Related Work“ 4.1 vorgestellten Anwendungen genutzt werden, um Vergleiche zu den hier aufgestellten Anforderungen zu ziehen und Funktionalitäten abzugleichen.

4.3.1. Anwendungsfälle

Im Folgenden werden die an das System gestellten Anwendungsfälle identifiziert und der Systemkontext festgelegt. Die Anwendungsfälle werden genauer spezifiziert und die funktionalen sowie die technischen Anforderungen an das System abgeleitet. Die so entstandenen Anforderungen werden im Kapitel „Konzeption“ 5 aufgegriffen, um die benötigten Module für die Umsetzung zu bestimmen.

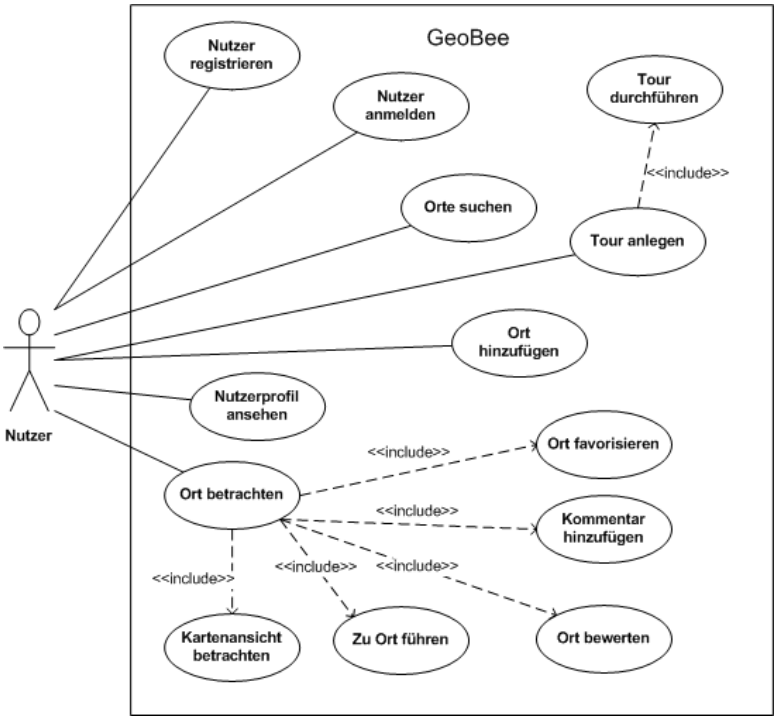

Abb. 4.7.: Anwendungsfall Diagramm GeoBee

Der Systemkontext (siehe Abbildung 4.7) wird mittels eines UML-Anwendungsfalldiagramms dargestellt [Kecher, 2011]. Die identifizierten Anwendungsfälle sowie der beteiligte Akteur werden innerhalb dieses Diagramms dargestellt und ihre Verbindungen miteinander visualisiert.

Die Visualisierung der Anwendungsfälle mittels eines UML-Anwendungsfalldiagramms ermöglicht zum einen die eindeutige Identifizierung der nötigen Bestandteile des Systems, zum anderen die der involvierten Akteure. Weiterhin werden die Systemgrenzen festgelegt und etwaige Schnittstellen zu anderen Systemen aufgezeigt, was eine Abgrenzung zu anderen Systemen erlaubt.

Anwendungsfallbeschreibungen

Innerhalb dieses Abschnittes wird beispielhaft ein Anwendungsfall aus dem in Abbildung 4.7 dargestellten Diagramm genauer spezifiziert. Dies soll der eindeutigen Festlegung der Bestandteile sowie der nötigen Bedingungen dienen. Die Spezifikation erfolgt tabellarisch und stellt so sämtliche Bestandteile und Bedingungen übersichtlich dar (siehe Tabelle 4.2). Weitere Tabellen zu den Anwendungsfällen sind im Anhang zu finden.

Name	**Ort hinzufügen**
Kurzbeschreibung	Der Nutzer fügt einen neuen Ort in das System ein.
Anwendungsfälle	Ort favorisieren, Kommentar hinzufügen, Ort bewerten, Zu Ort führen, Kartenansicht betrachten
Akteure	Nutzer
Vorbedingung	Der Nutzer ist im System registriert und mit seinem Konto angemeldet.
Ergebnis	Ein neuer Ort wurde im System angelegt und an den Server übertragen oder temporär zwischengespeichert.
Nachbedingung	Der Nutzer gelangt in die Hauptansicht und kann mit der Verwendung der Software fortfahren
Standard-Ablaufschritte	1. Der Nutzer meldet sich mit Benutzerkennung und Passwort an. 2. Der Nutzer wählt im Dashboard den Menüpunkt zum Hinzufügen eines neuen Ortes. 3. Der Nutzer wählt ein vorhandenes Foto aus seinem Album oder nutzt die Kamera, um ein Bild aufzunehmen. 4. Der Nutzer vergibt einen Namen für den Ort. 5. Der Nutzer fügt eine Beschreibung des Ortes ein. 6. Der Nutzer gibt Schlagwörter für den getätigten Eintrag an. 7. Der Nutzer tätigt den Button „Ort erstellen“.
Alternative Ablaufschritte	—

Tabelle 4.2.: Anwendungsfallbeschreibung „Ort hinzufügen“

4.3.2. Anforderungen

Im Folgenden werden für die in Abschnitt 4.3.1 identifizierten Anwendungsfälle Systemanforderungen diskutiert. Diese werden eingeteilt in funktionale, technische und nicht-funktionale Anforderungen. Die Identifizierung von Anforderungen hilft bei der Spezifikation des Systems und der benötigten Module.

Funktionale Anforderungen

Beispielhaft werden nachfolgend die funktionalen Anforderungen an den Anwendungsfall „Nutzer registrieren“ beschrieben. Diese zeigen, welche Ziele der Nutzer bei der Verwendung der Funktion erreichen soll. Die weiteren funktionalen Anforderungen finden sich im Anhang B.

Nutzer registrieren

FA-01 [Nachdem der Anwender die Applikation gestartet hat], soll die Applikation die Loginseite anzeigen.

FA-02 [Nachdem der Nutzer auf den Button „Register“ gedrückt hat], soll die Anwendung auf die Registrierungsseite wechseln und diese anzeigen.

FA-03 [Hat der Nutzer seine EMail-Adresse, einen Nutzernamen und ein Passwort eingegeben und den „Register“-Button gedrückt], soll die mobile Anwendung diese Daten an den Server übertragen.

FA-04 [Hat der Nutzer auf die systemseitige „Zurück“-Taste gedrückt], soll das System die Loginseite anzeigen.

FA-05 [Falls der Nutzer keine Email-Adresse, keinen Benutzernamen und/oder Passwort eingegeben hat], soll eine Fehlermeldung am Gerät angezeigt werden.

FA-06 [Nachdem die Serveranwendung die übertragenen Daten empfangen hat], soll die Verfügbarkeit des Benutzernamens überprüft werden und wenn möglich, der neue Nutzer angelegt werden.

FA-07 [Wenn der neue Nutzer angelegt wurde oder schon im System vorhanden ist], soll die Serveranwendung eine Fehler- oder Erfolgsmeldung verschicken.

FA-08 [Hat die Client-Applikation die Antwort vom Server erhalten, soll entweder eine Fehlermeldung angezeigt werden oder der Nutzer — nach einer erfolgreichen Registrierung — auf das Dashboard gelangen.]

Technische Anforderungen

Technische Anforderungen an die technischen Komponenten des Systems beschreiben die Eigenschaften, über welche die Software verfügen soll. Sie beeinträchtigen die Konzipierung der einzelnen System-Komponenten. Nachfolgend sollen beispielhaft einiger dieser technischen Anforderungen identifiziert werden:

TA-01 Die eingesetzten mobilen Endgeräte benötigen ein Modul zur Positionierung des Nutzers.

TA-02 Das eingesetzte Gerät muss mit dem Betriebssystem Android betrieben werden und mindestens in der Version 2.2 verfügbar sein.

TA-03 Eine Internetverbindung ist bei der Verwendung der Software notwendig, um mit der serverseitigen Applikation zu kommunizieren.

TA-04 Die eingesetzten Geräte müssen über eine integrierte Kamera verfügen.

TA-05 Für die Serveranwendung muss eine MySQL-Datenbank vorhanden sein.

TA-06 Die Client-Anwendung soll es ermöglichen, Daten bei nicht ausreichender Datenverbindung zwischenzuspeichern.

Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen an ein System dienen der Sicherstellung der Softwarequalität. Die im Folgenden aufgeführten Qualitätsmerkmale sind an die Merkmale der ISO/IEC 9126 angelehnt [for Standardization, 2011]:

NFA-01 Die beschriebenen Anwendungsfälle für die Software wurden umgesetzt, und die funktionalen Anforderungen aus 4.3.2 und Anhang B wurden erfüllt.

NFA-02 Die Anwendung darf bei normalem Umgang nicht abstürzen oder Fehler erzeugen. Sollte unerwartet ein Fehler auftreten, muss der Zustand der letzten Verwendung wiederhergestellt werden.

NFA-03 Die Verwendung der Software und und ihrer Bestandteile muss in 80% der Anfragen eine Reaktionszeit von < 5 Sekunden bieten können, um den Nutzer eine angenehme Benutzung zu erlauben.

NFA-04 Die Position des Nutzers muss in 90% der Fälle innerhalb von < 10 Sekunden geschehen. Dabei ist aufgrund der Funkzellenortung eine höhere Abweichung möglich. Die Optimierung der Position durch GPS erfolgt im Hintergrund und bedarf keiner Aktion des Nutzers.

NFA-05 Die Entwicklung der Software erfolgt nach den Prinzipen und Methoden der objektorientierten Programmierung. Die Architektur erlaubt eine lose Kopplung der Module und somit eine leichte Erweiterbarkeit der Software.

NFA-06 Die Software ist geeignet für die Nutzung auf der Android-Plattform. Die Software soll auf allen bisherigen Versionen dieser Plattform lauffähig sein (bis Version 2.2).

5. Konzeption

Dieses Kapitel befasst sich mit der Konzeption der Applikation. Die verschiedenen Architekturen der Client- und Serveranwendung werden vorgestellt und detailliert in ihrem geplanten Aufbau beschrieben. Erläutert wird auch das Kommunikationsprotokoll der Applikation, welches auf dem JSON-Format basiert. Die festgelegten Anforderungen aus Kapitel 4, welche sich aus den Anwendungsfällen ergeben, sollen hier nun zur Konzeption für ein geeignetes Nutzer-Interface verwendet werden. Um dieses Interface zu visualisieren, werden UI-Sketches erstellt und die Funktionsweisen der einzelnen Interface-Elemente charakterisiert. Abgeschlossen wird Kapitel 5 mit einer Konzeption des nötigen Caching von Daten, indem einige mögliche Vorgehensweisen aufgezeigt und Vor- sowie Nachteile diskutiert werden.

Zur Darstellung des Architekturmodells wird sich an den *4+1 Sichten* nach Philippe Kruchten orientiert [Kruchten, 2004]. Aus diesem Modell werden die „Scenarios“ anhand des UML-Anwendungsfalldiagramms, der „Logical view“ durch Klassen- und Sequenzdiagramm, der „Development view“ mittels UML-Komponentendiagramm und der „Process view“ durch UML-Sequenzdigramme visualisert. Der „Physical view“, also die physikalische Verteilung der Softwarekomponenten, wird in dieser Masterarbeit nicht explizit beschrieben. Weiterhin erfolgt die Modellierung des Systems hier nur beispielhaft und deckt nicht jedes Modul des Systems ab.

5.1. Fachliche Architektur der Anwendung

Im Folgenden werden die fachlichen Module vorgestellt, welche in die Applikation eingehen sollen. Diese fachliche Architektur der Anwendung berücksichtigt keine technischen Bestandteile und spiegelt die in Kapitel 4 festgelegten Anforderungen wider.

Anhand der Anforderungen an das System, welche in Abschnitt 4.3.2 und B definiert werden, lassen sich vier Hauptkomponenten identifizieren, welche in die fachliche Architektur der Anwendung eingehen (siehe Abbildung 5.1). Diese sind eine Crowdsourcing-Komponente, eine Komponente, welche die soziale Interaktion zwischen den Nutzern aufgreift, die demographische Komponente und die Spiel-Elemente, welche ebenfalls in einer fachlichen Komponente zusammengefasst sind.

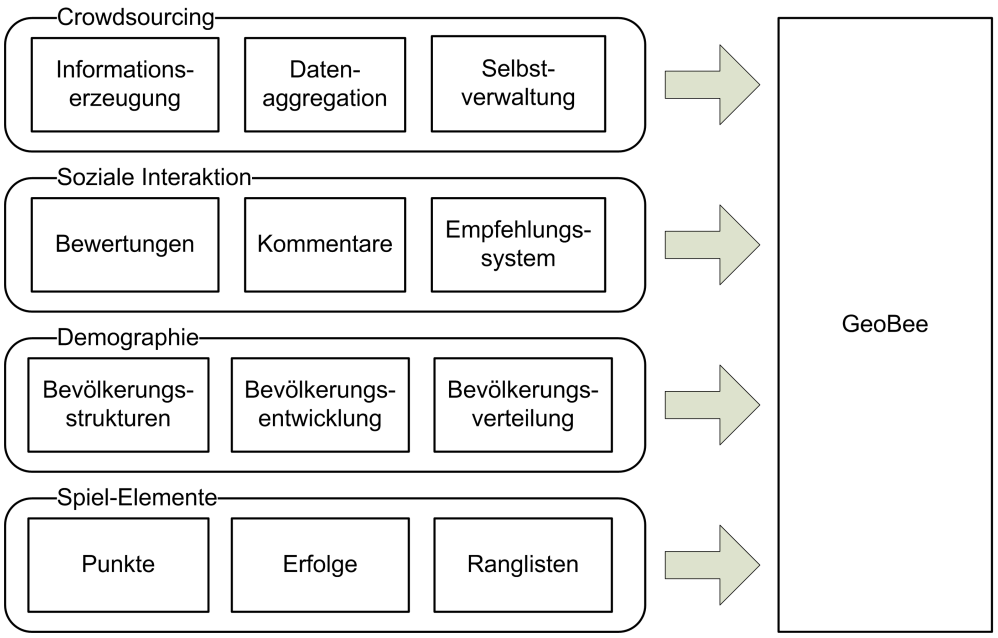

Abb. 5.1.: Fachliche Architektur der Anwendung

Die Module, welche in den Komponenten enthalten sind, werden im Folgenden beschrieben und beispielhaft erläutert:

Crowdsourcing

Die hauptsächliche Aufgabe der Applikation liegt in der Nutzung von Crowdsourcing-Konzepten zur Aggregation und Verbreitung von Informationen. Die fachlichen Anforderungen an diese Komponente sind Informationserzeugung, Datenaggregation und Selbstverwaltung. Das Modul Selbstverwaltung dient der Organisation der Daten sowie der Sicherstellung eines Qualitätsanspruchs. Die Nutzer haben so die Möglichkeit, die Inhalte der Plattform zu kontrollieren und unerwünschte Beiträge zu entfernen.

Soziale Interaktion

Um die Interaktion der Nutzer zu fördern und ein „Miteinander-Gefühl“ zu erzeugen, spielen Funktionen, die soziale Interaktion ermöglichen, eine wichtige Rolle. Durch gegenseitige Bewertungen und Kommentare der Einträge wird die Interaktion gefördert und zudem die Motivation gesteigert, qualitative Inhalte zu erzeugen. Ein Anreiz für Nutzer ist auch das Empfehlungssystem: Die Kontakte der Nutzer zueinander und ihre individuellen Interessen können dort vermerkt werden; die Nutzer können einander zudem Informationen zu interessanten Stätten übermitteln.

Demographie

Durch die Erzeugung von verschlagworteten Standort-Informationen und Sehenswürdigkeiten ergibt sich die Möglichkeit, Demographien über die Bewohner einer Stadt zu generieren. Die Aggregation der vorliegenden Daten erlaubt es zum einen, dem Nutzer eine einfache Übersicht über bestimmte Städte oder Stadtteile zu bieten, zum anderen können die gewonnenen Informationen auch Aufschluss über die Bevölkerungsentwicklung, die Bevölkerungsstruktur und -verteilung einer Stadt geben. Die Daten müssen selbstverständlich anonymisiert werden, um geltene Datenschutzbestimmungen einzuhalten.

Spiel-Elemente

Zuletzt wird die fachliche Komponente der Spiel-Elemente erläutert. Die Nutzung von spielerischen Elementen innerhalb einer Applikation — also der Einsatz von Gamification-Methoden, ermöglicht es, die Partizipation des Nutzers am System positiv zu beeinflussen. Weiterhin wird die Verwendung einer Applikation durch motivierende Elemente wie beispielsweise Ranglisten für den Nutzer ansprechender. Die Module dieser Komponente sollen zunächst grundlegende Funktionalitäten wie ein Punktesystem, Erfolge und Ranglisten bieten. Die Erweiterung um andere Spiel-Elemente ist für die Zukunft sinnvoll.

5.2. Systemarchitektur

Die Auswahl der passenden Architektur für das gegebene Anwendungsszenario muss den Anforderungen an das vorliegende Problemfeld genügen und die Entwicklung einer Lösung unterstützen. Weiterhin sind Faktoren wie Leistungsfähigkeit, Skalierbarkeit, Reaktionszeit, Ressourcenbedarf und Robustheitsanforderungen [Dunkel u. a., 2008] bei der Auswahl einer geeigneten Architektur zu beachten.

Im Folgenden werden zwei beispielhafte Architekturen vorgestellt und ihre Vor- und Nachteile bezüglich der Anforderungen der hier entwickelten Applikaktion evaluiert [Dunkel u. a., 2008].

Peer-To-Peer

Diese Architektur kommt ohne zentrale Instanz aus und verbindet die Teilnehmer gleichbereichtigt miteinander, wie in Abbildung 5.2 zu sehen ist. Der Aufbau einer verteilten Architektur durch ein Peer-To-Peer-Netz eigent sich vornehmlich für Anwendungsfälle wie File-Sharing oder Instant-Messaging. Aufgrund der fehlenden zentralen Instanz einer solchen Architektur werden die Ressourcen der Teilnehmer verwendet, um beispielsweise große Datenmengen auszutauschen oder gleichzeitig Daten an viele Empfänger zu verteilen. Vorteilhaft ist zum einen die Fehlertoleranz, welche den Ausfall einzelner Teilnehmer

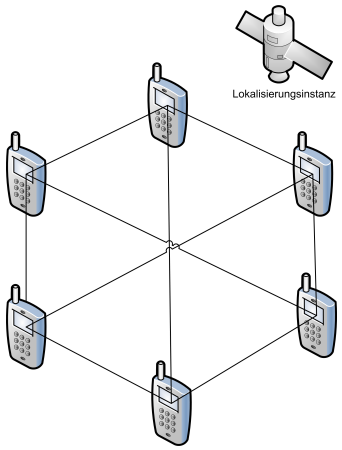

Abb. 5.2.: Aufbau einer Peer-to-Peer-Architektur

erlaubt, zum anderen die lose Kopplung der einzelnen beteiligten Geräte. Weiterhin bietet eine Peer-To-Peer-Architektur eine gute Skalierbarkeit, da bei steigender Teilnehmerzahl gleichzeitig auch weitere Ressourcen verfügbar werden. Der Einsatz einer solchen Architektur ist für die hier konzipierte Applikation dennoch wenig sinnvoll. Da die eingesetzten mobilen Endgeräte über eine begrenzte Rechenleistung sowie eine geringe Speicherkapazität verfügen, sind diese nicht geeigent, um große Mengen an Daten zu verwalten und untereinander auszutauschen. Desweiteren müssten die Endgeräte in der Lage sein, Daten für andere Empfänger weiterzuleiten oder zwischenzuspeichern. Dies würde bedingen, dass die Geräte eine dauerhafte und ausreichend gute Datenverbindung aufweisen können, was aufgrund des mobilen Charakters der Teilnehmer nicht gegeben ist.

Client-Server-Architektur

Das folgende Konzept für die Architektur der Anwendung gehört zu den meist verwendeten. Die Client-Server-Architektur ist von ihren Konzepten gut verstanden und hat sich bereits seit vielen Jahren bewährt. Sie ist ebenfalls Grundlage vieler Weiterentwicklungen und bietet vor allem durch ihren einfachen und übersichtlichen Aufbau eine gute Alternative (siehe Abbildung 5.7). Lediglich bei der Konzeption großer und sehr umfangreicher Systeme, kann die Modellierung einer solchen Architektur unübersichtlich und komplex werden [Dunkel u. a., 2008]. Es bietet sich vor allem für Systeme an, welche nur wenige, eher große Komponenten beinhalten. Ein wichtiger Vorteil bezüglich des hier entworfenden Systems ist die Beteiligung einer zentralen Instanz. Diese ist zum einen durch eine Vielzahl von mobilen Endgeräten leicht erreichbar, zum anderen bietet sie den verhältnismäßig leistungsarmen Clients die Möglichkeit, Daten zu speichern und vorzuverarbeiten. Der

einfache Aufbau eines solchen Systems macht es weiterhin möglich, die Modellierung der Komponenten übersichtlich zu gestalten. Neben der Option die Hardware für die Server-Applikation eigenhändig bereitzustellen, ist es auch denkbar, diese über einen Cloud-Dienst anzumieten. Der Vorteil einer solchen Lösung wäre zum einen der reduzierte Administrationsaufwand, da die Hardware durch den Dienstanbieter gewartet wird, zum anderen die bessere Skalierbarkeit. Sollten für eine Optimierung der Leistungsfähigkeit des Systems weitere Hardware-Ressourcen benötigt werden, können diese je nach Bedarf beim Cloud-Dienst-Anbieter nachgebucht werden. Dies erlaubt eine einfache Skalierung ohne Beschaffung von weiteren Ressourcen. Nachteilhaft wäre hier, dass die server-seitige Programmiersprache für die Umsetzung teilweise durch die Anbieter eines Cloud-Dienstes vorgegeben werden. Somit kann es notwendig sein, den Web-Dienst mittels einer anderen Programmiersprach neu zu implementieren.

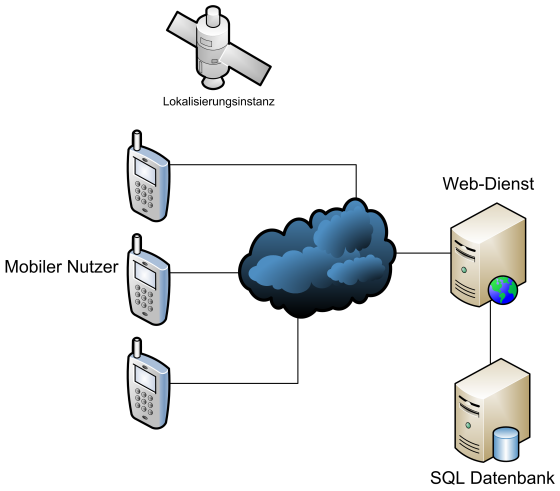

Abb. 5.3.: Client-Server-Architektur der Applikation

Da die Client-Server-Architektur für die gegebene Problemstellung die beste Unterstützung liefert und das Anmieten einer Cloud-Infrastruktur den Rahmen einer prototypischen Umsetzung übersteigt, wurde sich bei der Wahl der Architektur für eine klassische Client-Server-Architektur entschieden. Die Realisierung mittels einer Peer-To-Peer-Architektur ist bei den Gegebenheiten und dem mobilen Charakter der Clients wenig sinnvoll und soll ebenfalls nicht erfolgen.

5.3. Technische Architektur der Client-Anwendung

Der folgende Abschnitt beschreibt die Architektur der Client-Anwendung. Es wird zum einen die von Android vorgegebene Struktur erläutert, zum anderen werden Erweiterungen aufgezeigt, die während der Entwicklung hinzugefügt wurden. Die wichtigsten Komponenten werden anhand von UML-Komponentendiagrammen dargestellt, sowie Teile der Architektur mittels UML-Klassendiagrammen verdeutlicht. Zuletzt wird die verwendete Architektur für die Server-Kommunikation aufgezeigt. Die Umsetzung der Benachrichtigung bei empfangenen Daten und die Aktualisierung der vorhandenen Modelle werden erklärt.

5.3.1. Android-Architektur (Model-View-Controller)

Die Anwendungsentwicklung auf der Android-Plattform erfolgt nach einem vorgegebenen Schema. Dieses Schema ist keine zwingende Bedingung bei der Entwicklung, bietet sich allerdings aufgrund der gegebenen Struktur an. Die wichtigsten Bestandteile bei der Entwicklung einer Anwendung sind die folgenden:

XML-Layout-Dokument
: Die in Android enthaltenen XML-Layout-Dokumente dienen der Umsetzung der sichtbaren Elemente einer Anwendung. Es können alle verfügbaren visuellen Komponenten eingefügt, mögliche Aktionen für Buttons oder andere Elemente festgelegt und ebenso Erweiterungen und eigene UI-Komponenten erstellt und eingebunden werden. Weiterhin bietet das Layout-System von Android die Funktionalität, alle Elemente auf einem Bildschirm mittels verschiedener Layout-Container anzuordnen und gegebenenfalls pixelgenau die Positionen anzupassen.

Aktivität
: Eine Aktivität stellt die Klasse für je einen sichtbaren Bildschirm dar. Die oben genannten Layout-Dateien werden einer Aktivität zugeordnet und an sie gebunden. Innerhalb einer Aktivität können weitere Interface-Elemente hinzugefügt oder vorhandene angepasst oder entfernt werden. Auch findet innerhalb einer Aktivität zu großen Teilen die Geschäftslogik statt. Diese kann weitere Aktivitäten starten oder Daten anderer Aktivitäten anfordern.

Das von Android verwendete Architektur-Prinzip kann aufgrund der vorhandenen Elemente als Model-View-Controller-Architektur verstanden werden. So existiert eine Entkopplung von Geschäftslogik und Repräsentation der vorhandenen Daten. Die eingesetzten Layout-Dateien stellen hierbei den View-Teil dar und dienen lediglich der Anzeige von Daten und der Nutzerinteraktion. Die zugehörige Aktivität kann in diesem Fall als Controller verstanden werden,

auch wenn teils noch eine starke Kopplung zwischen dem Controller und dem vorhandenen View besteht. Diese Kopplung von View und Controller sowie die mögliche Einflussnahme des Controllers auf anzeigespezifische Elemente kann dazu führen, dass die vorhandene Struktur vom Entwickler vernachlässigt wird und somit keine Trennung zwischen Geschäftslogik und Anzeige erfolgt. Das Model — also die Repräsentation der vorhandenen Daten — ist vom Entwickler entsprechend anzulegen. Hierbei ist zu beachten, dass möglichst keine Programmlogik innerhalb dieser Modelle geführt wird. Die Daten sollten strikt von der Visualisierung sowie der Verarbeitung getrennt werden. Wird dies eingehalten, so ist es denkbar, die vorhandenen Datenstrukturen auch in anderen Anwendungen zu übernehmen und lediglich die Verarbeitung und visuelle Repräsentation an das jeweilige System anzupassen.

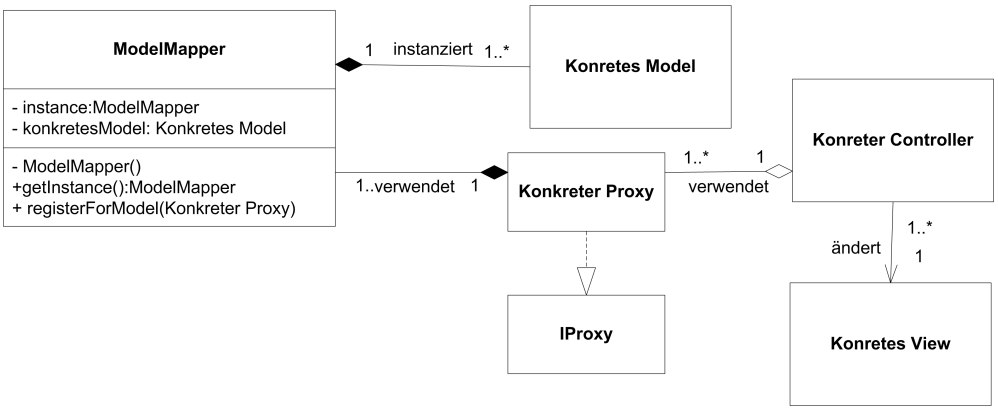

Abb. 5.4.: MVC-Architektur der Anwendung GeoBee

Innerhalb dieser Arbeit wird die von Android gegebene Anwendungs-Architektur, wie in Abbildung 5.4 ersichtlich, als eine MVC-Architektur aufgegriffen. Das Modell wurde hier mittels des Einsatzes von Model-Klassen und zugehöriger „Value Objects“ umgesetzt. So ist es dem Modell möglich, ein oder mehrere Objekte vorzuhalten, welche die vorhandenen Daten enthalten. Das Model kann auf diese Daten zugreifen, sie verändern oder entfernen. Die Entkopplung der Daten von der Geschäftslogik und der verwendeten Visualisierung fördert zum einen die Wiederverwendbarkeit und zum anderen eine gute Strukturierung der Applikation. Damit Modelle von Controllern nutzbar sind, aber nicht mehrfach durch die Anwendung instanziiert werden, sollen innerhalb dieser Arbeit Proxys verwendet werden. Die Implementierung von Proxys ermöglicht eine einfache und sichere Zugriffsmöglichkeit auf vorhandene Model-Objekte. Proxys können aus jeder Klasse erzeugt werden und stellen somit eine Art leichtgewichtige „Helferklassen“ dar. So kann ein Controller mehrere dieser Proxys instanziieren, je nachdem auf welche Daten der Controller Zugriff benötigt. Um auf die Daten eines Modells zugreifen zu können, muss sich ein

Proxy bei einem ModelMapper registrieren. Erfüllt der Proxy das erwartete Interface, so wird durch den ModelMapper die Instanz des Models, an den Proxy übergeben. Proxys haben den Vorteil, dass Operationen auf Daten der Modelle gekapselt werden. Somit können Operationen je nach Anwendungsfall spezifiziert und erzeugt werden. Beispielsweise kann das Einfügen eines neuen Datensatzes durch weitere Operationen ergänzt werden. Diese Operationen werden nicht im Model implementiert, sondern innerhalb des jeweiligen Proxys. So wird verhindert, dass die Modelle spezialisiertes Verhalten aufweisen, und die verwendeten Controller bleiben übersichtlicher. Weiterhin können Proxys auf unterschiedliche Modelle zugreifen. Dies ermöglicht es ihnen, umfangreiche Operationen auf unterschiedlichen Daten auszuführen, ohne dabei unübersichtlich zu werden.

5.3.2. Konzeption der Netzwerkkommunikation

Dieser Abschnitt soll die Konzeption der client-seitigen Kommunikation mit der Server-Anwendung vorstellen. Da es sich bei dem verwendeten Protokoll um ein zustandsloses HTTP handelt und die Kommunikation mit dem Server weiterhin asynchron erfolgt, müssen einige Besonderheiten bei der Verarbeitung der Daten beachtet werden. Neben den gerade erwähnten Merkmalen sollte die Architektur der Netzwerkkomponente eine möglichst generische Schnittstelle bieten und leicht von anderen Komponenten verwendet werden können. Um dies zu erlauben und auf die vorhandene Asynchronität des Systems einzugehen, wurde sich hier für eine Architektur nach dem Beobachter-Muster entschieden [Freeman u. a., 2004]. Dies erlaubt es, Anfragen jederzeit asynchron zu versenden und den Aufrufer zu benachrichtigen, wenn eine Antwort vom Server erfolgt ist. Die konzipierte Architektur ist in der Abbildung 5.5 zu sehen.

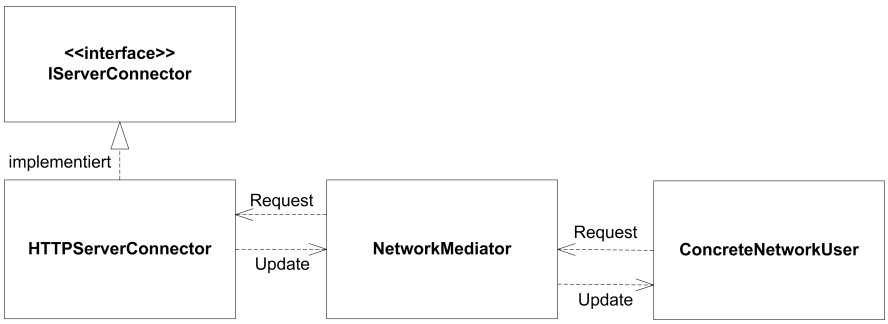

Abb. 5.5.: Architektur der Netzwerkkomponente

Um allen Klassen eine generische Schnittstelle zum Versand von Netzwerknachrichten zu bieten und dabei eine gewisse Unabhängigkeit von der tatsächlichen Implementierung des Kommunikationsprotokolls zu erreichen, wurde bei der Konzeption auf eine Entkopplung von der

Netzwerk-Implementierung geachtet. Durch die Verwendung eines Netzwerk-Mediators werden diese Anforderungen erfüllt. Der Mediator stellt zeitgleich das zu beobachtende Subjekt dar und erlaubt es den Beobachtern, sich zu registrieren und wieder abzumelden. Erreicht die konkrete Netzwerkschnittstelle eine Antwort vom Server, so wird dies zunächst an den Mediator gemeldet. Dieser informiert anschließend alle registrierten Beobachter über die Nachricht. Durch die beschriebene Entkopplung der tatsächlichen Implementierung des verwendeten Netzwerkprotokolls ist es leicht, andere Kommunikationsmodule zu realisieren und einzubinden. So könnten für unterschiedliche Anfragen verschiedene Kommunikations-Protokolle genutzt werden.

5.3.3. Komponenten der Client-Anwendung

Die Client-Anwendung deckt die in Abschnitt 4.3 festgelegten Anwendungsfälle ab. Die Abhängigkeiten sowie die geplante Struktur der Applikation sollen im Folgenden anhand von UML-Komponentendiagrammen aufgezeigt werden. Komponentendiagramme dienen der Gliederung einer Softwareanwendung in modulare Teile. Es werden vornehmlich die nach außen sichtbaren Schnittstellen definiert. Die innere Sicht einer Komponente wird nicht im Detail spezifiziert. Somit ergibt sich für die Komponenten ein „Black-Box“-Sicht, welche mit Hilfe von Klassendiagrammen weiter verfeinert werden kann.

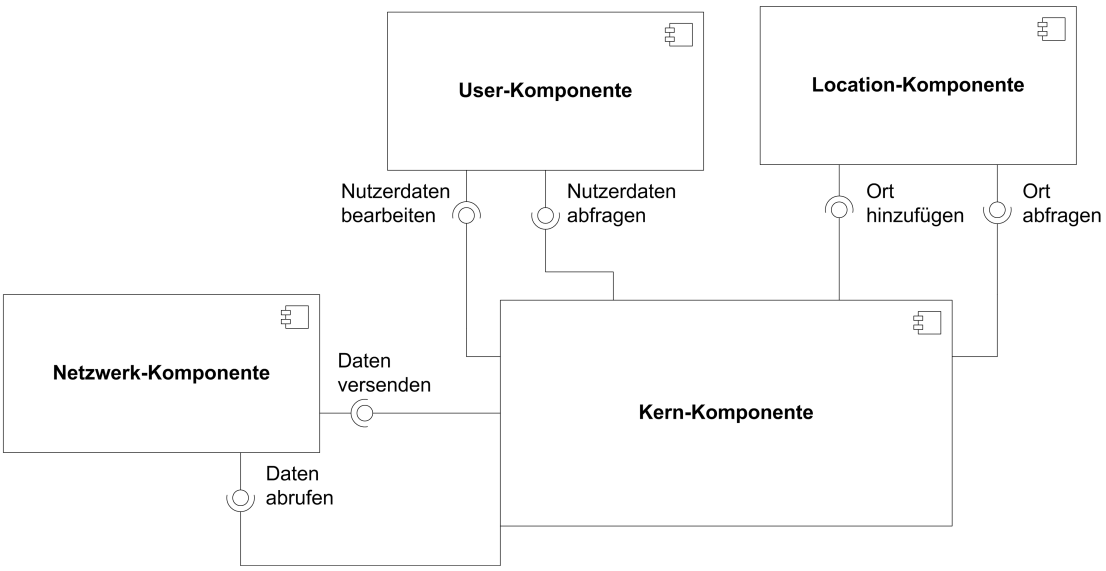

Abb. 5.6.: Komponentendiagramm der Anwendung

In Abbildung 5.6 ist das Komponentendiagramm von *GeoBee* zu sehen. Dieses gliedert sich in vier Haupt-Komponenten. Die User-Komponente beinhaltet Informationen zum aktuellen

Nutzer und bietet Schnittstellen zur Manipulation der vorhandenen Daten. Hierüber kann die Anwendung Daten wie Erfolge, grundlegende Nutzerinformationen, sowie die vom Nutzer generierten Einträge abrufen. Die zweite Komponente beinhaltet Informationen über verfügbare Orte. Weiterhin bietet auch sie Schnittstellen zur Manipulation der Daten. Es können neue Orte hinzugefügt oder bearbeitet werden. Die Location-Komponente bietet die Möglichkeit, Übersichten von Orten abzufragen oder detailliertere Informationen über einen bestimmten Ort zu liefern. Die Kapselung der Nutzer- und Ortsverwaltung in eigenständigen Modulen ermöglicht es, diese innerhalb weiterer Anwendungen — welche auf den vorhandenen Daten der Applikation aufbauen — wiederverwenden zu können. Die Netzwerk-Komponente ist ebenfalls in sich gekapselt. Die Art bzw. die technische Umsetzung der Kommunikation des Clients mit der Server-Anwendung kann so einfach geändert werden. Diese Komponente bietet zum einen eine Schnittstelle zum Versand von Anfragen an die Server-Applikation und erhält zum anderen bei eintreffenden Antworten eine Benachrichtigung. Da die Kommunikation auf Basis des HTTP-Protokolls geplant ist, erfolgt die Kommunikation asynchron. Diese Art der Nachrichtenübermittlung erfordert, dass das System auch entsprechend konzipiert wird und verspätet eintreffende Nachrichten trotzdem korrekt verarbeitet werden. Die vierte Komponente stellt den Kern der Applikation dar. Innerhalb dieser Komponente sind alle Anforderungen aus den Anwendungsfällen implementiert. Da sich die konkreten Komponenten der Kern-Funktionalitäten nicht maßgeblich unterscheiden, wurden diese — der Übersicht halber — innerhalb dieser Komponente zusammengefasst. Die Implementierungen innerhalb der Kern-Komponente nutzen die angebotenen Schnittstellen der anderen Einheiten, um dem Nutzer die geforderten Funktionalitäten zu bieten. Auf die detaillierte Umsetzung bzw. Konzeption einiger dieser innenliegenden Funktionalitäten wird im Kapitel 6 zur Realisierung noch genauer eingegangen.

5.4. Dynamische Modellierung der Anwendung

Um die Dynamik des Systems zu beschreiben, werden im Folgenden zwei beispielhafte UML-Aktivitätsdiagramme aufgezeigt. Diese zeigen einzelne Komponenten des Systems auf, welche anhand der Use-Cases identifiziert wurden. UML-Aktivitätsdiagramme stellen den Daten- und Kontrollfluss einer Applikation dar und ermöglichen es somit, Prozesse und Kommunikationsschnittstellen zu visualisieren.

Abbildung 5.7 zeigt den Anwendungsfall „Nutzer anmelden“ in Form eines solchen Aktivitätsdiagramms. Die Teilung der Bereiche erfolgt hier nach Nutzer und der server-seitigen Benutzerverwaltung. Neben dem Erfolgsfall bei der Anmeldung ist auch die fehlerhafte Eingabe

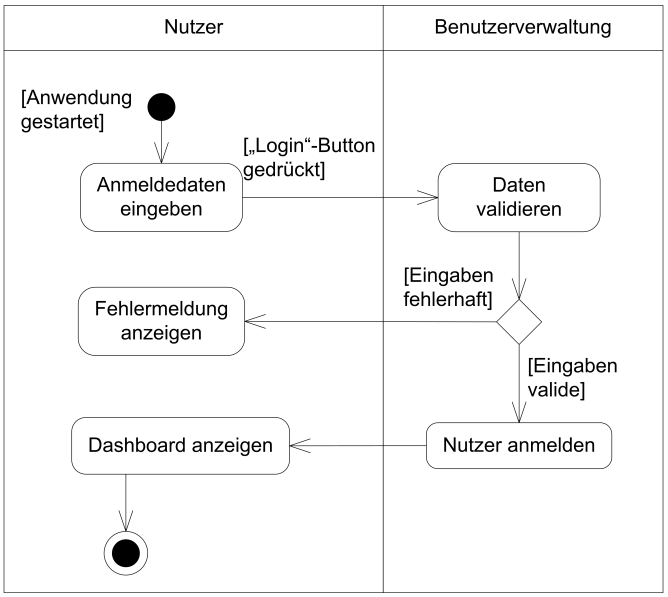

Abb. 5.7.: Aktivitätsdiagramm des Anwendungsfalls „Nutzer anmelden“

der Nutzerdaten modelliert. Diese wird dem Nutzer mittels einer Fehlermeldung am Endgerät angezeigt.

In Abbildung 5.8 ist das Aktivitätsdiagramm für den Anwendungsfall „Ort hinzufügen“ zu sehen. Dies teilt sich in die Bereiche Nutzer, Ortungs-Dienst und Location-Komponente. Hat der Nutzer ein Bild des anzulegenden Ortes aus der Gallery gewählt oder ein Bild mittels der geräte-internen Kamera aufgenommen, vergibt er einen Ortsnamen und eine Beschreibung. Anschließend wählt er Schlagworte innerhalb der zugehörigen Aktivität. Klickt er den entsprechenden Button zur Erstellung eines neuen Ortes, ermittelt der Ortungs-Dienst zunächst die Geo-Koordinaten der aktuellen Position des Nutzers. Anhand dieser Koordinaten werden die entsprechenden Adressdaten abgerufen. Schließlich werden die übermittelten Daten durch die Location-Komponente validiert. Sind sie korrekt, werden sie abgespeichert, und der Nutzer erhält eine Erfolgsmeldung. Sind die Daten nicht valide, erhält der Nutzer eine entsprechende Fehlermeldung.

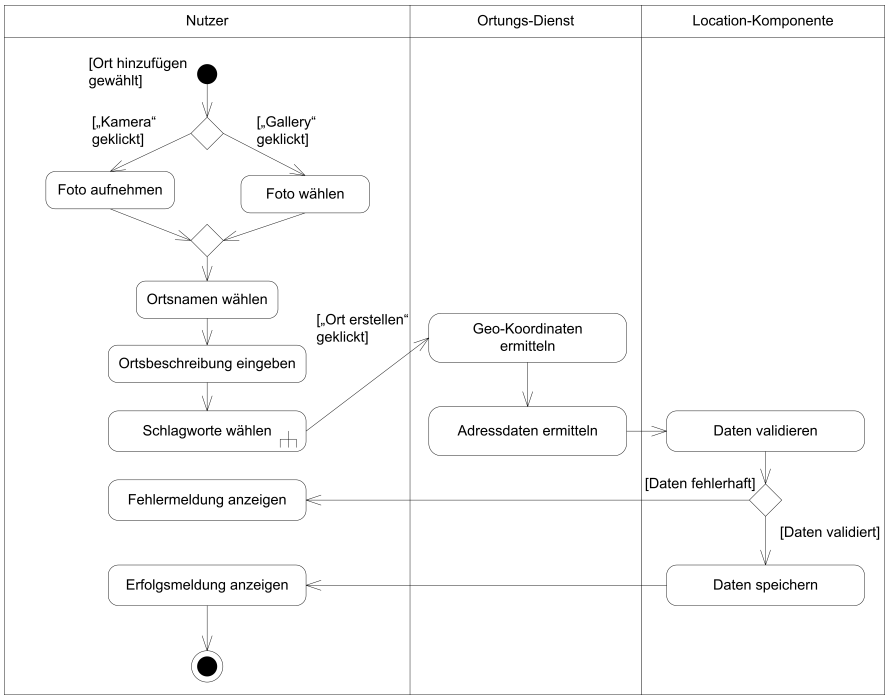

Abb. 5.8.: Aktivitätsdiagramm des Anwendungsfalls „Ort hinzufügen“

5.5. Lokalisierung des Nutzers - mathematisches Konzept

Für die Kompass-Anwendung, welche den Nutzer zum gewählten Zielort führen soll, wird die Messung einer Entfernung sowie die Peilung des Nutzer anhand der aktuellen Position benötigt. Nachfolgend werden die mathematischen Prinzipien erläutert, welche die Berechnung der nötigen Werte ermöglichen. Die Realisierung der Komponenten werden im Kapitel 6 beschrieben.

Es gibt mehrere mathematische Verfahren, mit denen die Distanz von einem aktuellen Standort zu einem bestimmten Zielort berechnet werden kann. Zu den einfachsten gehört die Berechnung einer Distanz zwischen zwei Punkten auf einer planaren Fläche mittels Anwendung des Kosinussatzes, welcher in Gleichung 5.1 zu sehen ist. Gleichung 5.2 zeigt den Ausdruck für den Fall im rechtwinkligem Dreieck (Satz des Pythagoras).

$$c^2 = a^2 + b^2 - 2ab * cos\gamma \tag{5.1}$$

$$c^2 = a^2 + b^2 \tag{5.2}$$

Für die Berechnung einer Distanz zwischen zwei Punkten auf einer Sphäre und somit auf einer kugelförmigen Oberfläche reicht, bei genügend großem Abstand der Punkte, diese Gleichung nicht aus. Für diesen Zustand existieren zwei Verfahren: die Haversine-Formel und der Seiten-Kosinussatz. Beide sind konzeptuell in Abbildung 5.9 dargestellt.

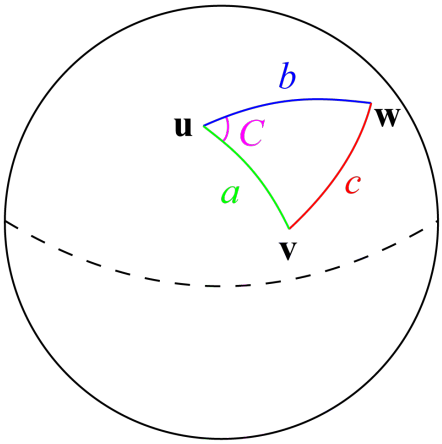

Abb. 5.9.: Ermittlung der Distanz auf einer Kugeloberfläche [Wikipedia, 2012]

Beide Verfahren ermöglichen eine gute Approximation der Entfernung zweier Punkte auf einer Kugeloberfläche. Der Einsatz der komplexen Haversine-Formel 5.3 liefert insbesondere bei kleinen Entfernungen genaue Ergebnisse.

$$\begin{aligned} a &= sin^2(\Delta lat/2) + cos(lat_1) * cos(lat_2) * sin(\Delta long/2) \\ c &= 2 * atan2(\sqrt{a}, \sqrt{(1-a)}) \\ d &= R_{earth} * c \end{aligned} \tag{5.3}$$

Der Seiten-Kosinussatz in Gleichung 5.4 erlaubt hingegen aufgrund seiner geringeren Komplexität eine Implementierung in modernen Programmiersprachen. Durch den heutigen Einsatz von präzisen Fließkomma-Berechnungen sind Rundungsfehler, welche das Ergebnis stark beeinträchtigen können, vernachlässigbar. In dieser Arbeit wird der Seiten-Kosinussatz genutzt. Die verwendeten Parameter sind die Geo-Koordinaten der involvierten Orte — jeweils mit Latitude und Longitude. Da eine Implementierung innerhalb der verwendeten *osmdroid*-Bibliothek bereits existiert, wurde von einer erneuten Umsetzung abgesehen.

$$
\begin{aligned}
d = acos(\quad & sin(lat_1) * sin(lat_2) \quad + \\
& cos(lat_1) * cos(lat_2) \quad * \\
& cos(lon_2 - lon_1)) \quad * \quad R_{earth}
\end{aligned} \tag{5.4}
$$

Für die Durchführung einer Peilung zwischen aktuellem Standort und einem entfernten Zielort kann Formel 5.5 verwendet werden. Diese errechnet das sogenannte „Bearing“, also die Peilung eines Ortes anhand einer Referenz. Um diese Peilung zu ermitteln, wird der Winkel errechnet, welcher sich zwischen den beiden Orten aufspannt.

$$
\begin{aligned}
\theta = atan2(\quad & sin(\Delta long) \quad * cos(lat_2), \\
& cos(lat_1) \quad * sin(lat_2) - \\
& sin(lat_1) \quad * cos(lat_2) * cos(\Delta long))
\end{aligned} \tag{5.5}
$$

5.6. Interface-Konzeption der Client-Anwendung

Dieser Abschnitt soll genutzt werden, um die Anwendung aus Sicht des Nutzer-Interfaces zu konzipieren. Dabei werden Interface-Konzepte erstellt und visualisiert. Die Konzeption des Nutzerinterfaces erfolgt im Abgleich mit den herausgearbeiteten Anwendungsfällen. Neben der reinen Erfüllung der Anwendungsfälle soll das Interface auch eine angenehme Benutzbarkeit und eine gute „User Experience“ mit sich bringen. Der Nutzer soll bei der Verwendung der Applikation also neben der Option, bestimmte Aktionen auszuführen, auch Spaß bei der Nutzung haben. Die grafische Gestaltung der Anwendung, ein einfacher Aufbau und interessante Interface-Elemente sollen hierzu beitragen.

5.6.1. Nutzerinterface-Skizzen

Innerhalb dieses Abschnittes sollen Interface-Skizzen für sechs Nutzerinterfaces gezeigt und beschrieben werden. Die verbleibenden Interfaces — wie zum Beispiel die Suche — sind sehr einfach gehalten und werden hier nicht im Detail vorgestellt.

Dashboard

Das sogenannte Dashboard ist das Hauptmenü der Anwendung. Von diesem Interface aus kann der Nutzer auf alle weiteren Funktionalitäten zugreifen. Die Verwendung eines Dashboards ist in vielen mobilen Applikationen verbreitet, da es eine übersichtliche und leicht verständliche Nutzerinteraktion ermöglicht.

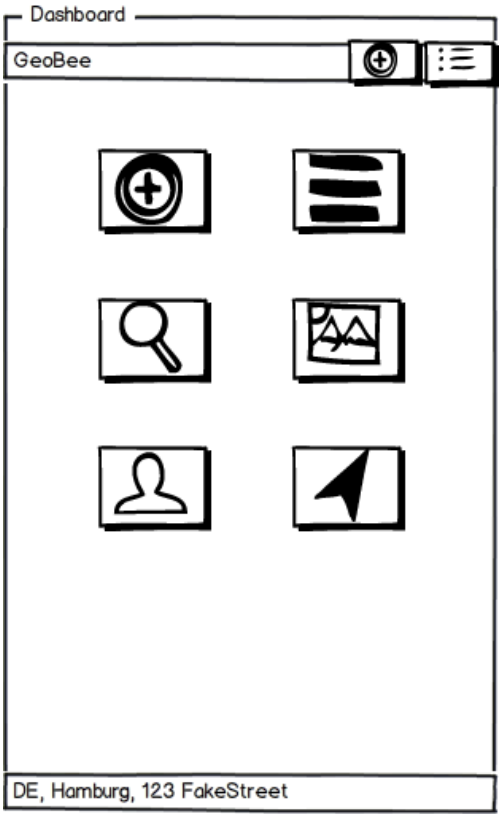

Abb. 5.10.: UI-Konzept für das Dashboard

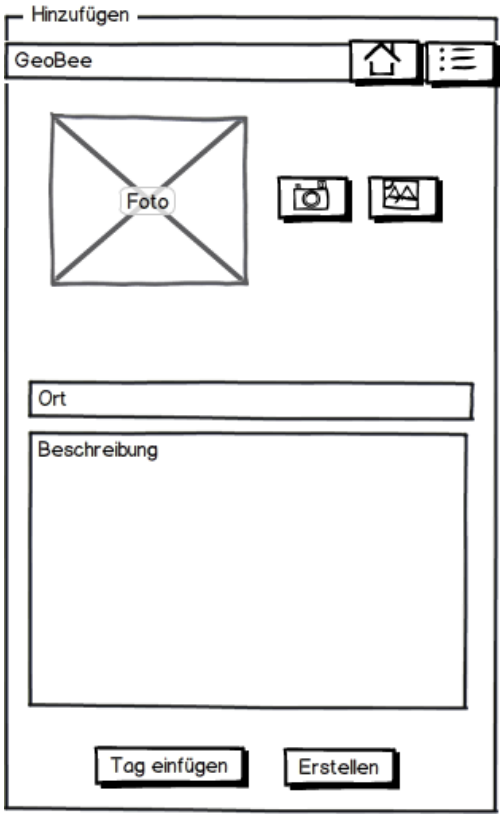

Abb. 5.11.: Hinzufügen eines Ortes - Skizze

Abbildung 5.10 zeigt das UI-Mockup für das Dashboard der Anwendung. Der Aufbau des Dashboards ist einfach gehalten und dient so der Übersichtlichkeit. Den Hauptteil des Inhalts bilden die Buttons, mit welchen der Nutzer die verschiedenen Funktionalitäten erreicht. Diese sind zunächst: das Hinzufügen eines Ortes, eine Listenansicht der Orte in der Umgebung, eine Suche, eine Kartenansicht, das Nutzerprofil und die Routenansicht. Im unteren Bereich des Dashboards wird dem Nutzer seine aktuelle Position in Form von Land, Adresse und Stadtnamen angezeigt. Dies dient der Orientierung innerhalb einer fremden Stadt.

Der obere Bereich der Anwendung wird durch eine sogenannte „ActionBar“ gefüllt. Diese ist innerhalb der Anwendung jederzeit verfügbar. Die ActionBar ermöglicht es dem Nutzer, schnell und einfach Funktionen aufzurufen, die dem aktuellen Kontext der Anwendung angepasst werden. Das Konzept der Actionbar wird seit der Version 3 von Android direkt unterstützt und ist zu einem Standard geworden, welchen heutzutage viele Anwendungen erfüllen. Da die Applikation unter der Android-Version 2.3 entwickelt wurde, steht die Standard-Aktionsleiste nicht zur Verfügung, deswegen wird eine einfachere Variante dieses Interface-Elements selbst implementiert. Hier sollen jeweils zwei Shortcuts vorhanden sein, welche dem Nutzer einen schnellen Zugriff auf andere Funktionen anbieten.

Hinzufügen eines Ortes

Der Bildschirm, über den Nutzer einen neuen Ort anlegen können, verfügt über drei Gruppen von Elementen (Abbildung 5.11). Zunächst kann der Nutzer ein Foto für die neue Sehenswürdigkeit auswählen. Hierzu kann er entweder ein vorhandenes Bild aus seiner Galerie wählen oder mittels der integrierten Kamera ein Foto machen. Das gewählte Bild wird anschließend in einer verkleinerten Version neben den Buttons angezeigt. Die zweite Gruppe der Interface-Elemente dient der Beschreibung des Ortes. Hier kann der Nutzer einen Namen für den Ort vergeben und eine ergänzende Beschreibung einfügen. Die Anzahl der verfügbaren Zeichen ist hierbei nicht begrenzt. Weitere Informationen wie Adresse, Stadt und Länderkennung werden von der Anwendung automatisch erzeugt. Die aktuelle GPS-Position des Nutzers — und somit der beschriebenen Sehenswürdigkeit — wird ebenfalls durch die Applikation erkannt und zusammen mit allen anderen Informationen an den Server übertragen. Der letzte Punkt bei der Erstellung eines neuen Ortes ist die Vergabe von Schlagwörtern. Hierzu kann der Nutzer über einen Button zu einem weiteren Bildschirm gelangen. Dort kann er aus einer vordefinierten Auswahl von Schlagwörtern die passenden auswählen. Die Schlagwörter werden danach unter der Beschreibung angezeigt.

Detailansicht

In der Detailansicht — welche in Abbildung 5.12 zu sehen ist — hat der Nutzer die Option, einen im System vorhandenen Ort genauer zu betrachten. Die angezeigten Informationen zeigen den Ort mit Namen und Beschreibung. Zudem ist die Entfernung zu diesem Ort vom aktuellen Standort aus ersichtlich. Das für den Ort hinterlegte Bild nimmt einen großen Teil des Bildschirms ein, damit auch Details erkennbar sind. Unterhalb des Bildes kann der Nutzer die verfügbaren Aktionen wählen. Hierzu gehören das Bewerten eines Ortes, das Markieren von bereits besuchten Orten, eine Kartenansicht, die den aktuellen Ort anzeigt, sowie die Option, sich mithilfe des integrierten Kompasses zu diesem Ort führen zu lassen.

Unterhalb der vorhandenen Informationen befindet sich ein Eingabefeld, in welches Kommentare zum aktuellen Ort eingefügt werden können. Die neusten Kommentare — auch von anderen Nutzern — werden am Ende des Bildschirms angezeigt.

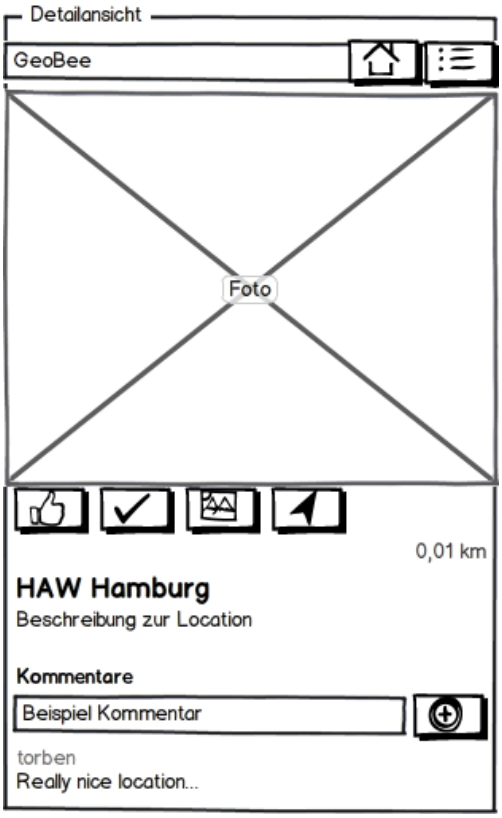

Abb. 5.12.: Detailansicht eines Ortes

Abb. 5.13.: Kartenübersicht zu angelegten Orten

Kartenansicht

Die Kartenansicht aus Abbildung 5.13 ist über verschiedene Bildschirme der Applikation erreichbar. Das Hauptelement dieser Ansicht ist die Karte, auf der der Nutzer vorhandene Orte und den eigenen Standort sehen kann. Weitere Informationen zur Kartenansicht finden sich im Kapitel 6 zur Realisierung.

Kompass

Um die Handhabung zu vereinfachen, ist das Interface, welches dem Nutzer einen Kompass zur Verfügung stellt, sehr simpel gehalten. Wie in Abbildung 5.14 zu sehen ist, besteht dieses

Interface lediglich aus drei Elementen. An oberster Stelle wird die Entfernung des Nutzers vom Zielort angezeigt. Diese wird regelmäßig aktualisiert. Am unteren Ende des Bildschirms wird die Adresse des aktuellen Zielortes angezeigt. Der Nutzer kann so, zusätzlich zum Kompass, weitere Informationen heranziehen. Den Hauptteil der Funktion bildet der grafische Kompass. Dieser zeigt mittels der Nadel in die Richtung, in welche der Nutzer gehen muss, um die vorher ausgewählte Sehenswürdigkeit zu erreichen. Es wurde bewusst auf eine Kartendarstellung verzichtet, um den Nutzer nicht zu sehr auf das Gerät zu fokussieren. Mittels dieses Interface-Mockups kann der Nutzer schnell die wichtigsten Informationen einsehen, ohne dabei langfristig auf das Gerät zu blicken.

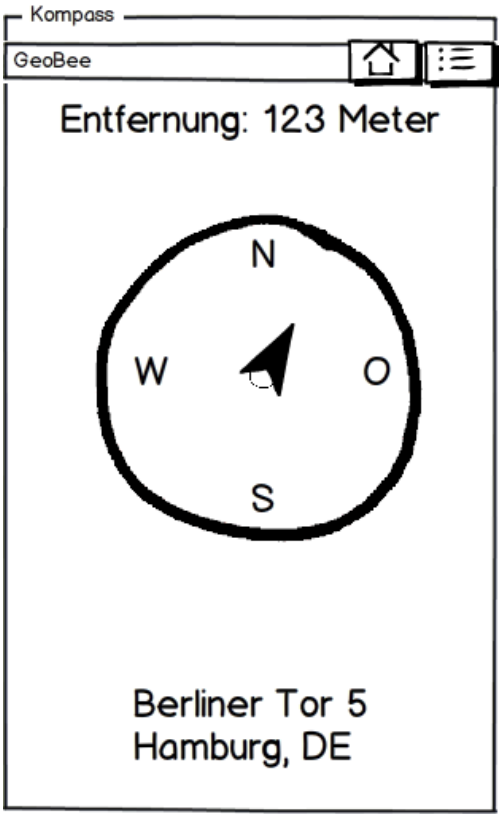

Abb. 5.14.: Kompass zur Navigation zu Orten

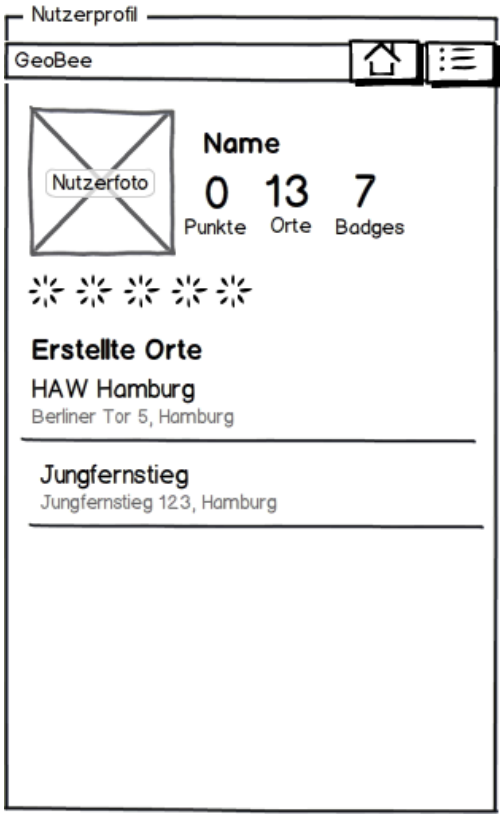

Abb. 5.15.: UI-Mockup für das Nutzerprofil

Nutzerprofil

Zuletzt soll das Nutzerprofil-Konzept erklärt werden (Abbildung 5.15). Dieses ermöglicht es dem Nutzer, seine aktuellen Informationen und Statistiken einzusehen. Weiterhin werden auch

die bisher von diesem Nutzer erstellten Orte aufgelistet. Im oberen Bereich ist das jeweilige Nutzerbild zu sehen. Auf der rechten Seite neben dem Bild werden Benutzername und die aktuellen Statistiken des Nutzers angezeigt. Diese beinhalten Informationen über die Anzahl der erstellten Orte, erreichte Punktezahlen und die Anzahl der Erfolge. Zusätzlich werden unterhalb dieser Angaben die Erfolge noch als Piktogramm dargestellt. Die vom individuellen Nutzer bisher erstellten Orte bilden den Abschluss dieses Interfaces: Angezeigt werden die Namen und Adressen dieser Orte. Der Nutzer kann die einzelnen Einträge anklicken und gelangt so zur Detailansicht des Ortes. Das Benutzerprofil soll nicht nur über das Dashboard für den aktuellen Nutzer selber erreichbar sein, sondern auch über die erstellten Orte als Link auf das Erstellerprofil.

5.7. Technische Architektur der Server-Anwendung

Dieser Abschnitt stellt die Struktur und die Architektur der Server-Applikation vor. Neben der in Abschnitt 5.3 erläuterten Client-Anwendung ist die Server-Applikation die zweite wichtige Komponente des Gesamtsystems. Um eine universelle Schnittstelle zu bieten, welche von heterogenen Clients genutzt werden kann, wurde die Server-Anwendung als Web-Service konzipiert. So ist es für eine Client-Applikation leicht möglich, auf die vom Server-Dienst angebotenen Funktionen zuzugreifen. Da innerhalb dieser Masterarbeit ein mobiler Client entwickelt werden soll, ist es weiterhin von Vorteil, dass die Server-Kommunikation über eine zustandslose Verbindung erfolgt. Wäre eine dauerhafte Verbindung zum Server erforderlich, wäre dieser Dienst für die mobile Nutzung nicht primär geeignet, da mobile Endgeräte oftmals nur über eine langsamere Verbindung verfügen oder vertragliche Einschränkungen greifen.

Um zu gewährleisten, dass die Server-Applikation eine gute Performance erreicht und eine gute Skalierbarkeit aufweist, wurde sich bei der Konzipierung der Applikation für eine relativ einfache und „leichtgewichtige“ Software-Architektur entschieden. Die Architektur als Web-Dienst ist sinnvoll, da es sich bei der Art der Anfragen an den Dienst um in sich abgeschlossene Aktionen handelt, welche eine Funktion des Dienstes nutzen und anschließend keine Verbindung mehr benötigen. Weiterhin ermöglicht diese Art der Architektur eine einfache und schnelle Erweiterbarkeit, da neue Funktionen leicht im System integriert werden können.

Abbildung 5.16 zeigt die Struktur der Server-Applikation als UML-Komponentendiagramm. Hier ist erkennbar, dass die Server-Anwendung über einen zentralen Punkt zu erreichen ist. Diese nach außen hin sichtbare Schnittstelle kann über eine URI erreicht werden und liefert, je nach Typ der Anfrage, die entsprechenden Ergebnisse an den Aufrufer. Die gewünschte Funktion, welche vom Client genutzt werden soll, wird anhand der übermittelten Daten erkannt und folglich von der Schnittstelle aufgerufen. Die empfangenen Daten werden hierbei an die jeweilige

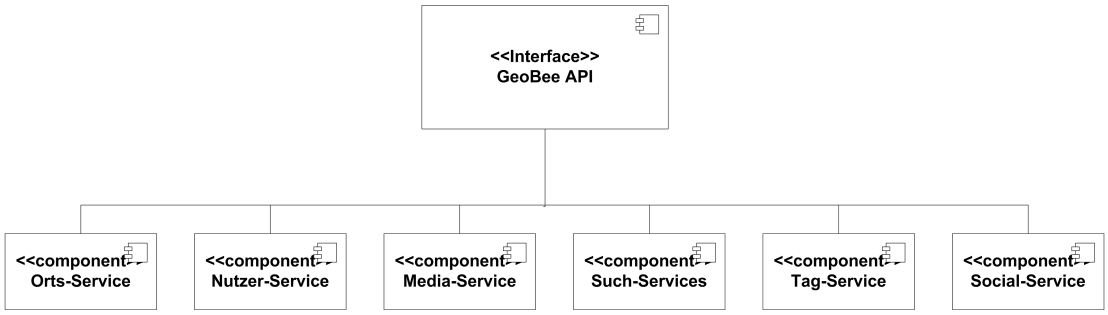

Abb. 5.16.: Komponentendiagramm des realisierten Web-Dienstes

Klasse zur Weiterverarbeitung übergeben. Die unterschiedlichen Arten der verfügbaren Dienste wurden hierbei in Komponenten gekapselt und können so leicht ausgetauscht oder erweitert werden. Die innerhalb dieser Arbeit entstandenen Dienste sind folgende: Ein Orts-Service, eine Benutzer-Verwaltung, ein Media-Service, ein Service zur Suche von Orten, ein Tag-Service und ein Service, welcher soziale Funktionen, wie beispielsweise Kommentare, bereitstellt.

5.7.1. Kommunikationsprotokoll

Die Kommunikation zwischen Server und mobilen Clients erfolgt über eine HTTP-Verbindung, welche per Definition zustandslos ist [Kurose und Ross, 2008]. Abbildung 5.17 zeigt den Aufbau des Systems und die Kommunikationswege zwischen Client- und Server-Anwendung.

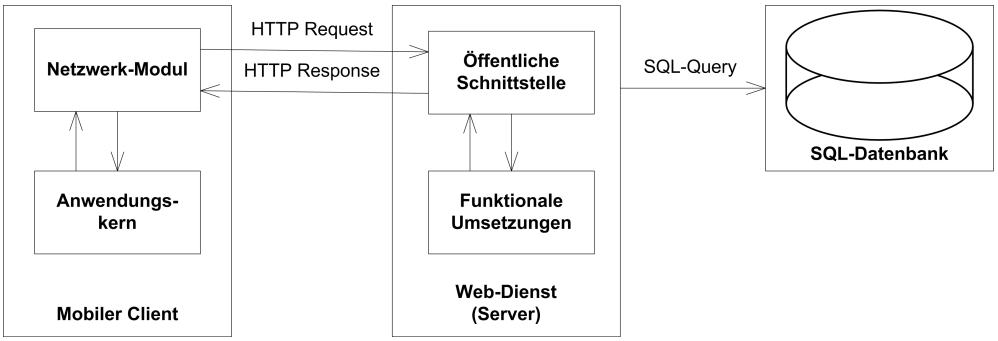

Abb. 5.17.: Kommunikationsablauf zwischen Server und Client

Um eine einheitliche Kommunikationsschnittstelle zu ermöglichen, wurde die Konzeption des Protokolls an das im Jahr 2000 von Roy Thomas Fielding [Fielding, 2000] vorgestellte Representational State Transfer (REST)-Prinzip angelehnt. Dieses beschreibt eine zustandslose Kommunikation zwischen Server und Client. Die Bedingungen für ein derartiges System beinhalten unter anderem:

Adressierbarkeit

Ein REST-konformer Dienst muss über eine eindeutige Adresse erreichbar sein, was über einen Uniform Ressource Identifier (URI) sichergestellt wird. Dieser URI ist die allgemeine Bezeichnung einer im Internet verfügbaren Ressource.

Unterschiedliche Repräsentationen

Je nach angeforderter Art der Repräsentation kann ein solcher Dienst die Antwort auf eine Anfrage in verschiedenen Darstellungsformen ausliefern (z.B. HTML, XML oder JSON).

Zustandslosigkeit

Die Zustandslosigkeit eines solchen Dienstes wird durch die Abgeschlossenheit der Anfragen an das System gegeben. Jede Anfrage an den Dienst beinhaltet alle nötigen Informationen, um auf diese Anfrage reagieren zu können. Somit kann weiterhin eine einfache Lastverteilung der Anfrage erfolgen, und das System ist leicht skalierbar.

Operationen

Die Operationen, die für die Nutzung des Dienstes vorgesehen sind, müssen konform sein und bei gleicher Art der Anfrage auch das gleiche Ergebnis liefern.

Die oben genannten Bedingungen werden durch den hier vorgestellten Dienst erfüllt. So ist eine allgemeine Schnittstelle über den eindeutigen URI aufrufbar, welche es erlaubt, alle Anfragen über eine einzige Adresse abzusetzen. Dies bietet zum einen den Vorteil, dass alle vom mobilen Endgerät getätigten Anfragen an den server-seitigen Dienst über eine Instanz innerhalb des Clients erfolgen können. Zum anderen ist es so möglich, über verschiedene Endgeräte die gleichen Informationen zu erhalten, ohne dabei auf alternative Adressen zugreifen zu müssen. Beides unterstützt die zweite Bedingung an ein solches System. Der entworfene Dienst kann bei minimaler Anpassung der server-seitigen Applikationen unterschiedliche Repräsentationen der Antwort-Daten liefern. Die Informationen können dadurch zum Beispiel nicht nur von mobilen Anwendungen, sondern auch von anderen Web-Services abgerufen werden. Andere Betreiber ähnlicher Dienste können die Informationen dementsprechend abrufen, erweitern oder anpassen.

Weiterhin wird auch die Bedingung an die Zustandslosigkeit erfüllt. Das verwendete Kommunikationsprotokoll basiert auf dem JavaScript Object Notation (JSON). Mithilfe dieses — im Vergleich zu XML sehr kompakte Datenformat — können alle nötigen Daten für eine Anfrage an den Dienst eingebunden werden. Auch bringt JSON den Vorteil mit sich, dass es neben der maschinenlesbaren Form auch von Menschen leicht gelesen werden kann. Da sowohl in der Anfrage als auch der Antwort innerhalb einer Kommunikation alle nötigen Bestandteile und Informationen enthalten sind, ist es für beide Anwendungen — Client- und Server-Applikation —

nicht notwendig, Zustände zu speichern. Eine Sessionverwaltung kann somit entfallen, da die jeweiligen Nutzerinformationen bei Anfragen an den Dienst mitgeliefert werden können. Die Operationen, welche durch den Client angefordert werden können, sind innerhalb des Protokolls spezifiziert und einheitlich. Sie können auch von anderen Web-Diensten genutzt werden, um Informationen abzufragen, anzupassen oder hinzuzufügen. Die gewünschte Operation ist Teil des JSON-Objektes und kann von Client- und Server-Anwendung extrahiert werden. Die verfügbaren Operationen sind in Tabelle 5.1 aufgeführt. Die Erweiterung dieser Operationen kann leicht erfolgen und ermöglicht so eine einfache Weiterentwicklung des Dienstes. Ergänzende Informationen zur client- und server-seitigen Umsetzung der Kommunikation finden sich in den nachfolgenden Abschnitten.

Operation	**Beschreibung**
loc.add	Fügt einen neuen Standort im System hinzu
loc.get	Gibt vorhandene Informationen über einen Standort zurück
loc.comment	Fügt einen neuen Kommentar im System hinzu
loc.like	Erhöht das Rating für einen bestimmten Standort
loc.visited	Erhöht die Besucherzahl eines Standortes
loc.fav	Fügt einen Standort bei einem Nutzer als Favorit ein
loc.stats	Zeigt alle Statistiken über einen Standort, wie z. B. Besucher, Ratings, etc.
usr.add	Fügt einen neuen Benutzer im System ein
usr.login	Autorisiert einen Nutzer für die Verwendung des Dienstes
usr.get	Gibt grundlegende Nutzerinformationen über einen bestimmten Nutzer an
usr.stats	Zeigt Statistiken über einen bestimmten Nutzer, wie z. B. Anzahl angelegter Orte, Punktestand, Achievements, etc.
usr.pic	Ermöglicht den Zugriff auf ein hinterlegtes Nutzerbild
med.gP	Ermöglicht den Zugriff auf ein größeres Bild eines Standortes
srch.loc	Ermöglicht die Suche nach einem bestimmten Standort, welcher im System existiert
tag.get	Zeigt alle im System vorhandenen Tags an
tag.add	Fügt ein neues Tag in das System ein
com.get	Zeigt alle im System vorhandenen Kommentare zu einen Standort an
com.add	Fügt einen neuen Kommentar für einen Standort in das System ein

Tabelle 5.1.: Mögliche Operationen des Dienstes

Abbildung 5.18 zeigt den exemplarischen Ablauf einer Client-Server-Kommunikation anhand des Anwendungsfalles „Ort hinzufügen“ in einem UML-Sequenzdiagram [Kecher, 2011]. Der Ablauf der gesamten Kommunikation über die beteiligten Module ist erkennbar. Beginnend mit

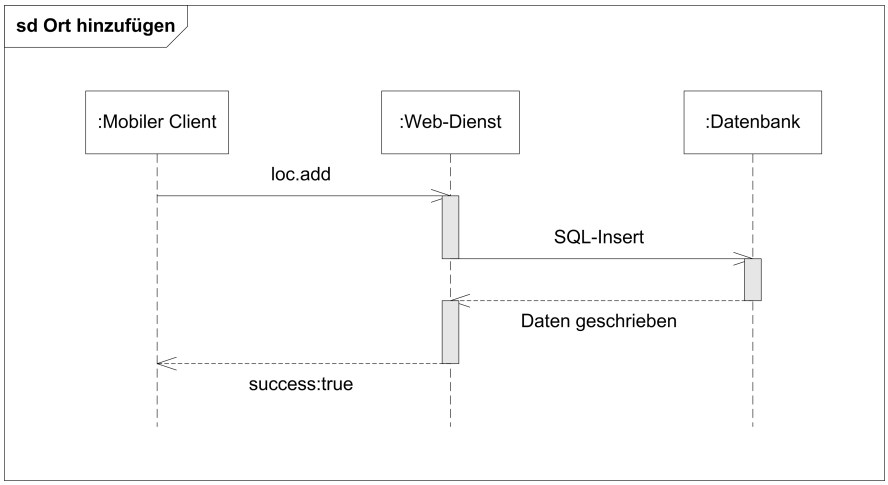

Abb. 5.18.: UML-Sequenzdiagram für den Anwendungsfall „Ort hinzufügen“

dem Objekt, welches die geforderten Daten vom Server abfragt, über den *NetzwerkMediator* bis hin zur konkreten Netzwerkschnittstelle. Wird die Nachricht durch die Server-Applikation empfangen, leitet die öffentliche Schnittstelle diese an das zuständige Modul zur Verarbeitung weiter. Die Antwort auf die Anfrage wird anschließend über die beteiligten Elemente an den Aufrufer zurück geleitet.

5.8. Caching von Daten

Der folgende Abschnitt soll die Notwendigkeit eines Cachings von Daten erläutern und thematisiert verschiedene mögliche Konzeptionen hierzu.

Da der mobile Charakter der Applikation ein Zwischenspeichern von Daten erforderlich macht, um eine zu intensive Internet-Nutzung zu verhindern, ist es sinnvoll, bei der Konzeption der Anwendung bereits die Möglichkeit des Zwischenspeicherns von Daten vorzusehen. Bei *GeoBee* ist beispielsweise das Zwischenspeichern von Daten wie Orten, Beschreibungen und Bildern angebracht. Auch andere Daten wie Nutzerinformationen, Nutzerbilder, Suchergebnisse können für eine Umgebungssuche vorgehalten werden. Diese werden bei Aufruf der entsprechenden Informationen aus dem Zwischenspeicher und nicht aus dem Netzwerk geladen.

Zur Realisierung eines solchen Cachings gibt es verschiedene Ansätze. Beispielsweise können die vom Server geladenen Daten innerhalb der Modelle vorgehalten werden. So steigt zwar der Speicherverbrauch der Applikation zur Laufzeit, es müssen aber bei wiederholtem Aufruf einer Information keine Daten vom Server geladen werden. Um zu verhindern, dass aktualisierte Daten

lediglich in einem veralteten Zustand angezeigt werden, könnte die Berechnung eines „Hashwertes“ eingesetzt werden, um eine Veränderung zu identifizieren. Mittels dieses „Hashwertes“ kann der Stand der Daten mit denen auf dem Server verglichen — und wenn nötig neu geladen — werden.

Um beim Hinzufügen von neuen Sehenswürdigkeiten auf eine schlechte oder nicht vorhandene Datenverbindung zu reagieren, sollen die erzeugten Daten ebenfalls zwischengespeichert werden. Dies erfolgt nach festgelegten Richtlinien. Zum Beispiel ist es möglich, größere Datenmengen — wie ein neues Foto eines Ortes — nur dann an die Server-Applikation zu übermitteln, wenn beispielsweise eine WIFI-Datenverbindung besteht.

Das Zwischenspeichern von Daten innerhalb des Arbeitsspeichers des Geräts bringt allerdings ein Problem mit sich. So werden die Daten beim Beenden der Applikation durch den Nutzer oder das Betriebssystem verworfen und stehen anschließend nicht mehr zur Verfügung. Um dieses Problem zu lösen, kann sich der integrierten Datenbank des Android-Systems bedient werden. Diese SQL-Lite-Datenbank gestattet es, die zwischengespeicherten Daten vorzuhalten, auch wenn die Anwendung beendet wird. Das Speichern innerhalb der Datenbank könnte beispielsweise immer dann geschehen, wenn die Anwendung beendet oder in den Hintergrund verschoben wird. So wird sichergestellt, dass zwischengespeicherte Daten weiterhin noch zur Verfügung stehen.

6. Realisierung

Kapitel 5 befasst sich mit der Applikationskonzeption und Applikationstruktur im Systemkontext; dieses Kapitel 6 beschreibt dementsprechend die Realisierung. Einzelne Module der Client-Anwendung werden im Detail erörtert; exemplarisch werden zudem die Realisierung von Server-Schnittstelle und einigen Server-Komponenten thematisiert. Abschließend wird die Umsetzung einer „Heatmap“ theoretisch erörtert und die integrierten Komponenten vorgestellt.

6.1. Realisierungsdetails

In diesem Abschnitt wird die Realisierung und der Umfang der im vorherigen Kapitel vorgestellten Konzeption beschrieben. Danach wird auf einige essentielle Details der Server- und Client-Applikationen eingegangen. Aufgrund des Umfangs der Implementierungen kann hier nicht jedes Detail der Realisierung vorgestellt werden, dementsprechend wurden hierfür nur Komponenten von besonderem Interesse ausgewählt.

6.1.1. Realisierungsumfang

Innerhalb des praktischen Teils dieser Masterarbeit wurde die konzipierte Anwendung implementiert. Dies erfolgte client-seitig auf der bereits vorgestellten Android-Plattform. Für die Umsetzung wurde ein „HTC Desire“-Smartphone als Test-Gerät eingesetzt. Dieses verfügt über die Android-Version 2.1 sowie die Standardausführung des von HTC angepassten Systems. Für die server-seitige Umsetzung wurde die Programmiersprache PHP (Version 5.2.12-nmm2) gewählt. Dies liegt darin begründet, dass diese auf dem zur Verfügung stehenden Webserver leicht implementiert werden konnte und keine weitere Software nötig war. Um die Funktionalität zu zeigen und einen ersten Prototypen der Software zu erstellen, ist die Leistungsfähigkeit von PHP ausreichend. Für eine Umsetzung innerhalb eines „Live“-Kontextes müssten Skalierbarkeit und Leistungsfähigkeit der Server-Implementierung weiter evaluiert werden.

Bei der Implementierung werden nicht alle Bestandteile der Konzeption berücksichtigt. So dient die prototypische Implementierung des Dienstes vornehmlich dem Testen der geplanten Funktionalität sowie Anschauungszwecken. Die Kernfunktionalität der Applikationen wurde in

der praktischen Arbeit umgesetzt. So kann der Nutzer neue Orte in das System einfügen, vorhandene Orte betrachten, eine Kartendarstellung wählen, einen Kompass nutzen und sein Profil betrachten. Auch können Orte in der Umgebung abgerufen werden. Es wurde ein rudimentäres *Queueing* realisiert und weitere system-seitig notwendige Funktionen.

Eine ausführliche Auswertung der realisierten Funktionen unter Beachtung der Anwendungsfälle erfolgt im Kapitel 8 „Evaluation“.

6.1.2. Realisierungen innerhalb der Client-Anwendung

Im Folgenden sollen einige ausgewählte Beispiele aus der Realisierung der Client-Applikation vorgestellt werden. Dies erfolgt nicht in einer detaillierten Auflistung aller Module, stattdessen werden die Implementierungen der konzipierten Architektur sowie der funktionalen Anforderungen aufgezeigt. Die Herangehensweisen bei auftretenden Problemen sowie lösungsorientierte Ansätze sollen innerhalb dieses Abschnittes dargelegt werden.

Netzwerk-Kommunikation

Die client-seitige Realisierung für die Kommunikation mit dem Server orientiert sich an der in Abschnitt 5.3.2 vorgestellten Konzeption. Den Kern der Umsetzung bilden vor allem die Klassen „NetworkMediator“ und „HTTPServerConnector“. Die Realisierung der tatsächlichen Server-Kommunikation ist in der Klasse „HTTPServerConnector“ zu finden. Die Kapselung der Realisierung ermöglicht einen leichten Austausch des tatsächlichen Kommunikations-Moduls. Die Klasse „NetworkMediator“ dient als zu beobachtendes Subjekt, bei dem sich andere Klassen registrieren können, um eine Kommunikation durchzuführen. Um eine einzige Instanziierung der Klasse „NetworkMediator“ sicherzustellen, wurde diese mittels des Singelton-Musters [Freeman u. a., 2004] realisiert.

```
1 public synchronized static NetworkMediator getInstance() {
2  if (instance == null) {
3   instance = new NetworkMediator();
4  }
5  return instance;
6 }
```

Listing 6.1: “NetworkMediator als Singleton“

Der statische Zugriff auf die Methode getInstance() sowie ein privater Konstruktor innerhalb der Klasse stellen sicher, dass nur eine einzige Instance der Klasse erzeugt wird. Das nachfolgende Listing 6.2 zeigt die Methoden, welche durch den „NetworkMediator“ bereitgestellt werden

müssen, um als beobachtbares Subjekt zu fungieren. Es handelt sich dabei einerseits um Methoden zum Hinzufügen und zum Entfernen von Beobachtern, andererseits um die Methode „notifyObservers()“, welche alle Beobachter über Änderungen informiert.

```
1  public void update(JSONObject res) {
2   super.setChanged();
3   super.notifyObservers(res);
4  }
5
6  @Override
7  public void addObserver(Observer observer) {
8   super.addObserver(observer);
9  }
10
11 @Override
12 public synchronized void deleteObserver(Observer observer) {
13  super.deleteObserver(observer);
14 }
15
16 @Override
17 public void notifyObservers(Object data) {
18  super.notifyObservers(data);
19 }
```

Listing 6.2: “Methoden des NetworkMediators“

Zum Versenden von Daten an den Server-Dienst können Objekte die Methode *sendData(final JSONObject data, final byte[] imgData)* aufrufen und das gewünschte JSON-Kommando übergeben. Zusätzlich kann ein Byte-Array übergeben werden, welches bei Bedarf an den Server übertragen wird. Innerhalb der Klasse *NetworkMediator* wurde diese Methode implementiert — wie in Listing 6.3 zu sehen ist.

```
1  public boolean sendData(final JSONObject data, final byte[]
       imgData) {
2   NetworkInfo network = ConnectionProxy.getInstance().
       getNetworkInfo();
3   TelephonyManager mobil = ConnectionProxy.getInstance().
       getTelephony();
4   Boolean success = false;
5
6   if((null != network) && network.isConnected()) {
```

```
7    try {
8     if(data.getString("action").equals("loc.add") &&
9      !network.getTypeName().equals("WIFI") ){
10
11     requestQueue.addLast(new RequestVo(data, imgData));
12    }else {
13     new HttpServerConnector().execute(data, imgData, this);
14     success = true;
15    }
16   } catch (JSONException e) {
17    e.printStackTrace();
18   }
19  }else {
20   Toast t = Toast.makeText(ConnectionProxy.getInstance().
        getContext(), "Please enable a internet connection.", Toast.
        LENGTH_SHORT);
21   t.show();
22  }
23  return success;
24 }
```

Listing 6.3: “sendData()-Funktion des NetworkMediators“

Die hauptsächlichen Funktionen dieser Methoden sind das Überprüfen einer verfügbaren Internet-Verbindung und das Sicherstellen einer ausreichend guten Bandbreite beim Erstellen eines neuen Ortes. Diese ist notwendig, da beim Anlegen eines Ortes große Mengen von Bilddaten übertragen werden; die Übertragung kann dementsprechend nur erfolgen, wenn WIFI-Netzwerke verfügbar sind. Ist dies nicht der Fall, wird die Anfrage innerhalb der *NetworkMediator*-Klasse vorgehalten und erst dann durchgeführt, wenn ein WIFI-Netzwerk ereichbar ist.

Ist die vorhandene Bandbreite ausreichend, wird eine Instanz der Klasse *HttpServerConnector* erzeugt und anschließend die Methode Execute aufgerufen. Um die Übertragung von Daten weiter von der sichtbaren Anwendung zu trennen, wurde die tatsächliche Implementierung der Kommunikation in einen *asynchronen Task* ausgelagert. Dies ermöglicht es, die Kommunikation in einen Thread durchzuführen, damit der Rest der Applikation nicht blockiert wird.

Innerhalb der Klasse *HttpServerConnector*, welcher die Schnittstellen des Interfaces *IServer-Connector* erfüllen muss, wird für die Durchführung eines Kommunikationsvorgangs die Methode *doInBackground* ausgeführt, welche in Listing 6.4 zu sehen ist.

```
@Override
protected JSONObject doInBackground(Object... params) {
 this.mediator = (NetworkMediator)params[2];
 return this.sendData((JSONObject)params[0], (byte[])params[1]);
}
```

Listing 6.4: “Threading der Netzwerk-Kommunikation“

Die hier aufgerufene Methode *sendData* führt einen HTTP-Request durch, welcher neben Header-Informationen auch das geforderte JSON-Kommando und weitere Daten enthält. Sollte das Kommando Bilddaten enthalten, werden diese per *ByteArrayBody* an den Request angehängt und mitversendet.

Die Instanz des erzeugten Server-Connectors wartet im Hintergrund auf eine Antwort des Server. Wird diese empfangen, wird sie mittels der Connector-Klasse vorverarbeitet und zu einem *JSONObject* gewandelt. Abschließend erfolgt die Rückmeldung an den Mediator, welcher dann alle registrierten Klassen über eine Aktualisierung benachrichtigt.

Beispielhaft soll im Folgenden der Aufbau eines JSON-Kommandos gezeigt werden. Dieses Kommando wird gesendet, wenn der Nutzer einen neuen Ort erstellen will. Es sind alle notwendigen Informationen innerhalb des Kommandos enthalten, um eine abgeschlossene Transaktion durchzuführen, für die kein Status vorgehalten werden muss. Im Folgenden ist der Aufbau des Kommandos *action:loc.add* zu sehen, welcher an den Server übertragen wird.

> ’action’:’loc.add’,’locationid’:’Haw+Hamburg’,’la’:53.5571997222, ’lo’:10.0227186111,
> ’cc’:’DE’,’cn’:’Hamburg’,’al’:’Berliner Tor5’,’desc’:’Informatikum+der+Haw+Hamburg.’,
> ’img’:’binärdaten’, ’uid’:31,’tags’:’Wissen%2CHochschule%2C’

Angehängt an diese Operation sind Informationen, welche durch den Nutzer oder die Client-Applikation selber generiert wurden. Dazu gehören z.B. Latitude und Longitude, der Name des Ortes, die Beschreibung und Adressdaten wie Straße und Stadt.

Queueing mittels FIFO

Dieser Abschnitt beschreibt die Realisierung der Queueing-Funktionalität. Hauptsächlich wird das Zwischenspeichern von noch nicht gesendeten Daten an die Server-Anwendung thematisiert, welches auftritt, wenn keine ausreichende Bandbreite vorhanden ist. Ist bei der Erstellung eines neuen Ortes keine ausreichende Bandbreite verfügbar, so wird dieser Request an den Server im Speicher der Anwendung zwischengespeichert. Wie bereits beschrieben, dient dies dazu, Anfragen an den Server in annehmbarer Zeit abzuarbeiten und fehlerfrei zu übertragen.

Das Zwischenspeichern von Anfragen wurde innerhalb des *NetworkMediators* umgesetzt. Dieser überprüft die Verfügbarkeit von Netzwerk und Netzwerkbandbreite. Reichen sie nicht aus, werden die Daten als neues *RequestVO* in einer Queue gespeichert 6.5. Diese Queue funktioniert nach dem FIFO-Prinzip; somit werden neue Einträge am Ende eingefügt, und bei ausreichender Bandbreite werden die Elemente — beginnend mit dem ersten — versendet.

```
requestQueue.addLast(new RequestVo(data, imgData));
```

Listing 6.5: "Queueing der Server-Anfragen"

Ruft der Nutzer die Anwendung erneut auf, nachdem diese in den Hintergrund verschoben wurde — z.B. durch Nutzung einer anderen Applikation — wird durch die Anwendung der aktuelle Netzwerkstatus abgerufen. Ist dieser ausreichend gut, wird die Methode *flushRequests()* im *NetworkMediator* aufgerufen. Diese ist im Listing 6.6 zu sehen.

```
public void flushRequests() {
 while(this.requestQueue.size() != 0) {
  RequestVo req = this.requestQueue.poll();
  new HttpServerConnector().execute(req.getData(), req.getImgData
      (), this);
 }
}
```

Listing 6.6: "flushRequests()-Methode"

Diese Methode führt für jede noch ausstehende Anfrage einen Server-Request durch. Dabei wird — wie bei einer normalen Anfrage — eine Instanz des Server-Connectors erzeugt und dieser als Thread ausgeführt. Um dem Nutzer die Möglichkeit zu bieten, die Anwendung nicht nur in den Hintergrund zu verschieben, sondern auch zu beenden, müssten die Daten innerhalb der in Android enthaltenen SQL-Lite-Datenbank abgelegt werden. Dies wurde bei dieser prototypischen Realisierung nicht umgesetzt.

Open Street Maps - Einbindung

Nachfolgend soll die Einbindung von Kartenmaterial und Kartenansichten innerhalb der Anwendung beschrieben werden. Um den Nutzer zu befähigen, eine Kartenansicht der eingestellten Orte in seiner Umgebung zu betrachten, wurde auf ein bereits vorhandenes Modul zurückgegriffen. Anstatt die von Google bereitgestellte Anwendung Google Maps zu verwenden, wurde eine Open-Source-Lösung gewählt. Diese bietet zum einen den Vorteil, dass sie ohne Anmeldung beim Dienst und ohne Einschränkungen verwendet werden kann. Zum anderen ist es aus der Sicht

des Autors sinnvoll, bei einer Forschungsarbeit Open-Source-Komponenten einzusetzen, um den freien Austausch von Wissen zu unterstützen.

Bei dem gewählten Kartenmaterial handelt es sich um die Geo-Daten des Open-Street-Map-Projekts. Es wurde im Jahr 2004 gegründet und organisiert eine Sammlung von Geo-Daten aller Art [Deutschland, 2012]. Die Daten können von jeder Person erweitert und korrigiert werden. Somit passt die Ausrichtung des Projektes zum Thema dieser Arbeit, da es sich hierbei auch um ein Crowdsourcing-Vorhaben handelt. Um die Einbindung in die Client-Anwendung umzusetzen, wurde auf ein weiteres Programm zurückgegriffen: das OSMDroid-Projekt [osmdroid, 2012]. Dies ist ein für Android erstelltes Paket zur Einbindung von Open-Street-Map-Kartendiensten und ersetzt die *MapView*-Klasse, welche zur Einbindung von Google Maps verwendet wird. Um es in einer Anwendung zu nutzen muss die vom Projekt bereitgestellte *.jar*-Datei in die eigene Software eingebunden und instanziiert werden. Dies erfolgt zunächst über die erstellte Layout-Datei, wie in Listing 6.7 zu sehen ist.

```
<org.osmdroid.views.MapView
 android:id="@+id/mapview"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:clickable="true"
 android:layout_below="@id/actionbarbg_map"
/>
```

Listing 6.7: “Layout-Integration der Open-Street-Maps“

Anschließend kann über die zugehörige Klasse auf das Objekt zugegriffen werden. So wird es möglich, verschiedene Einstellungen vorzunehmen, etwa die Zoomstufe oder die Anzeige von Navigations-Elementen. Ersichtlich sind diese Einstellungen in Listing 6.8.

```
map = (org.osmdroid.views.MapView)findViewById(R.id.mapview);
map.setBuiltInZoomControls(true);
map.setMultiTouchControls(true);
map.getController().setZoom(13);
```

Listing 6.8: “Initialisierung des OSM-Layers“

Um den aktuellen Standort des Nutzers auf der Karte anzuzeigen, bedient man sich eines der vielen verfügbaren *Overlays* für die Karten-Applikation. Einige *Overlays* sind bereits im Framework enthalten, weitere — auf die eigenen Ansprüche zugeschnittene — *Overlays* können leicht hinzugefügt werden. Die Anpassung wird im Abschnitt zur Generierung der Heatmaps 6.2 weiter erläutert. Um den Standort des Nutzers auf der Karte mittels eines Piktogramms anzuzeigen, wird ein neues Objekt der Klasse *SimpleLocationOverlay* erzeugt — dadurch kann

eine neue Anzeige eines Ortes auf der Karte eingefügt werden. Um die Markierungen anzuzeigen, muss das neu erzeugte *Overlay* der Karte hinzugefügt werden 6.9.

```
locationOverlay = new SimpleLocationOverlay(this);
map.getOverlays().add(locationOverlay);
```

Listing 6.9: “Initialisierung des SimpleLocationOverlay“

Um nun einen neuen Ort auf dem *Overlay* anzuzeigen, wird die Methode *setLocation()* aufgerufen und ein Parameter des Typs *GeoPoint* übergeben. Das *Overlay* erzeugt auf der Karte so eine Markierung in Form eines kleinen Icons. Wie im Listing 6.10 zu sehen ist, sollte nach dem Setzen der aktuellen Position die Karte noch auf diesen Punkt zentriert werden, um dem Nutzer den aktuellen Standort anzuzeigen.

```
locationOverlay.setLocation(new GeoPoint(this.latitude, this.
    longitude));
map.getController().setCenter(new GeoPoint(this.latitude, this.
    longitude));
```

Listing 6.10: “Setzen der aktuellen Position des Nutzers“

Die Orte, welche in der Umgebung des Nutzers liegen, werden zusätzlich zum aktuellen Standort auf der Kartenansicht angezeigt. Zur Realisierung dieser Anzeige wird zunächst eine *ArrayList* in Java mit den vorhandenen Orten erzeugt. Die Elemente der Liste stellen *OverlayItems* dar. Diese beinhalten neben den Informationen über Längen- und Breitengrade des Ortes auch den Ortsnamen sowie die Ortsbeschreibung. Der Nutzer erhält bei der Anwahl eines Ortes auf der Karte dementsprechend auf Wunsch relevante Zusatzinformationen. Ist die Liste erzeugt, wird für die Anzeige auf der Karte ein neues *Overlay* generiert. Hierfür wird das in *osmdroid* vorhandene *ItemizedIconOverlay* genutzt. Mittels dieser Klasse können Objekte vom Typ *OverlayItem* auf einer Karte dargestellt werden. Weiterhin wird hier die Methode für das einfache Antippen eines Ortes, sowie das „Gedrückthalten“ auf einen Ort realisiert (Listing 6.11). Zuletzt wird das neue *Overlay* — welches nun die aktuellen Orte in der Umgebung enthält — der Kartenansicht hinzugefügt.

```
this.itemOverlay = new ItemizedIconOverlay<OverlayItem>(items,
 new ItemizedIconOverlay.OnItemGestureListener<OverlayItem>() {
   @Override
   public boolean onItemSingleTapUp(final int index, final
       OverlayItem item) {
   }
   @Override
```

```
7     public boolean onItemLongPress(final int index, final
          OverlayItem item) {
8     }, mResourceProxy);
9 this.map.getOverlays().add(this.itemOverlay);
```

Listing 6.11: “Erzeugung des Orte-Overlays“

Die bereitgestellte Ansicht der verfügbaren Orte in der Umgebung des Nutzers wird in Abbildung 6.1 gezeigt. Die angezeigten Markierungen sind durch den Nutzer „klickbar“, und eine Navigation innerhalb der Karte kann per Gestensteuerung erfolgen (zoomen, verschieben, etc.).

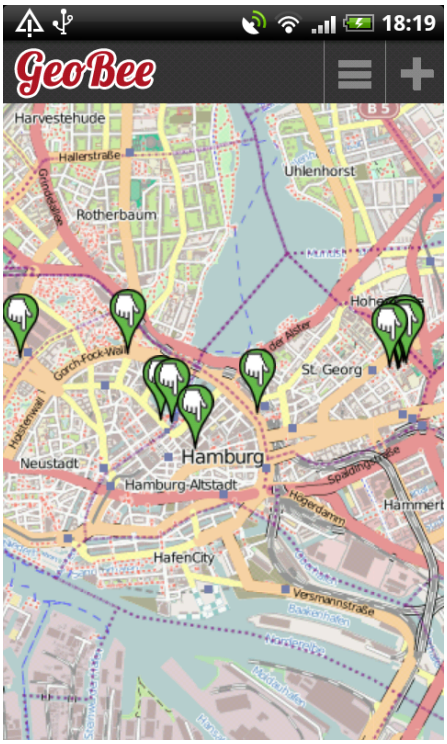

Abb. 6.1.: Kartenansicht für Orte in der Umgebung

Kompass

Der folgende Abschnitt beschreibt die Einbettung eines Kompasses in *GeoBee*. Hierzu soll zunächst eine kleine Einleitung zu Funktionsweise und besonderen Eigenschaften des Global Positioning System (GPS), sowie der „Best Performance“ unter Android gegeben werden. Anschließend werden einige Begrifflichkeiten erklärt, welche dem Verständnis der Materie dienen. Zuletzt wird die Implementierung des Kompasses in *GeoBee* vorgestellt.

Das Global Positioning System, etwa im Jahr 1990 voll ausgebaut, entstand im Rahmen eines Projektes des US-Verteidigungsministeriums und war ursprünglich für den Einsatz bei militärischen Anwendungen gedacht [Roth, 2005]. Heutzutage ist die Genauigkeit von GPS mit etwa 25m horizontal und 43m vertikal auch für Zivilpersonen brauchbar exakt, denn im Jahr 2000 wurde die Signalverschlechterung für zivile Anwendungen abgeschaltet [Roth, 2005]. Im zivilen Bereich wird das System beispielsweise in den Bereichen öffentlicher Nahverkehr, Navigationsgeräte für die Fahrzeugnavigation, sowie bei See- und Luftfahrtanwendungen eingesetzt. Mit der Integration von GPS-Empfängern in moderne Smartphones erweiterten sich die möglichen Anwendungsgebiete erneut. Heutzutage ist die Positionsbestimmung von mobilen Endgeräten in den meisten Betriebssystemen verfügbar und wird für viele verschiedene Aufgaben genutzt. Neben der relativ genauen Lokalisierung mittels GPS gibt es weitere Methoden, um eine Positionierung durchzuführen. So existieren Ansätze, welche mittels der erreichbaren WIFI-Netzwerke die Position des Nutzers ermitteln. Auch können die Funkzellen und deren Verteilung innerhalb einer Region genutzt werden, um eine Lokalisierung eines Anwenders durchzuführen.
Das zuletzt genannte Verfahren ermöglicht eine sehr schnelle Positionsbestimmung, welche allerdings — je nach Größe der aktuellen Funkzelle — eine Ungenauigkeit bis zu mehreren Kilometern aufweisen kann. Für eine schnelle, initiale Ortung reicht diese Lokalisierung allerdings meistens aus. Die genaue Positionierung des Nutzers erfolgt anschließend meist per GPS und kann dadurch weiter eingeschränkt werden.

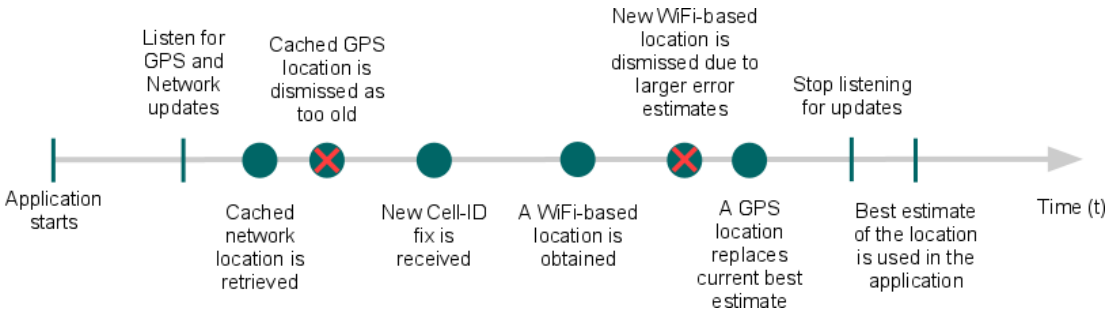

Abb. 6.2.: Ablauf der Lokalisierung mittels Android [Developers, 2012a]

Innerhalb von Android existieren mehrere Möglichkeiten zur Lokalisierung eines Anwenders. Auch wird dem Entwickler eine Art „best practices“-Anleitung bereitgestellt, welche den Ablauf einer Ortung in verschiedenen Zuständen beschreibt (Abbildung 6.2). Orientiert sich der Entwickler an diesen Vorgaben, erlaubt er dem Nutzer einen angenehmen Umgang mit der Ortungs-Funktionalität sowie eine effiziente Nutzung der Ressourcen unter Beachtung der verfügbaren Akkuleistung.

Die Realisierung des Kompasses besteht aus zwei Hauptelementen: zum einen aus der Anzeige für die Distanz vom aktuellen Standort zum gewünschten Zielort, zum anderen aus der eigentlichen Richtungsanzeige für den Zielort in Bezug auf die aktuelle Laufrichtung des Nutzers. Beide Elemente werden im Folgenden in ihrer Realisierung und ihren Bestandteilen erläutert.

Die errechnete Entfernung zum Zielort in Metern — welche konzeptionell in Abschnitt 5.5 diskutiert wurden — wird unterhalb der Richtungskomponente dargestellt und aktualisiert sich jeweils bei Änderung der Nutzerposition. Dies ermöglicht eine ungefähre Abschätzung der verbleibenden Entfernung zum Ziel und hilft dem Nutzer bei der Navigation.

Die Richtungsanzeige des Kompass-Moduls befähigt den Nutzer, seine Abweichung von der korrekten Laufrichtung zu korrigieren. Für die Umsetzung dieses Moduls wird ein im Gerät verbauter Orientierungssensor benötigt. Ist ein solcher Sensor im Gerät vorhanden, bietet Android über die API eine Schnittstelle, um Werte dieses Sensors zu erhalten. Die Registrierung einer Klasse für Werte eines Sensors erfolgt über den *SensorManager*. Bei der Anmeldung bei einem solchen Dienst muss der Typ des gewünschten Dienstes sowie die Verzögerungszeit für Aktualisierungen angegeben werden. Im Listing 6.12 ist die Registrierung für den Orientierungs-Sensor mit dem schnellsten Aktualisierungs-Intervall (SENSOR_DELAY_GAME) zu sehen.

```
1 sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
2 sensorManager.registerListener(this,
3  sensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION),
4  SensorManager.SENSOR_DELAY_GAME);
```

Listing 6.12: “Registrierung für den Orientierungs-Sensor“

Um Änderungen des Sensors in der Klasse weiterverarbeiten zu können, muss diese die Schnittstelle *SensorEventListener* implementieren. Die Methode *onSensorChanged(SensorEvent event)* ermöglicht den Empfang der aktualisierten Daten. Hier können die neuen Daten verarbeitet werden, und auf Änderungen kann entsprechend reagiert werden.

Um die aktuelle Richtungsänderung zu errechnen, wird zunächst aus den erhaltenen Sensor-Daten der sogenannte Azimuth extrahiert. Dieser beschreibt den Winkel zwischen dem magnetischen Nordpol und der Y-Achse, bezogen auf die Z-Achse des Systems (siehe Abbildung 6.3). Danach wird das Bearing berechnet. Das Bearing erlaubt die Bestimmung eines Winkels zwischen der Richtung eines gepeilten Objekts und einer Bezugsrichtung. In diesem Fall ist der aktuelle Standort der Bezugspunkt. Das angepeilte Objekt ist hier der gewünschte Zielort (Listing 6.13).

```
1 float bearing = (float) current.bearingTo(new GeoPoint(dest.
    getLat(), dest.getLng()));
```

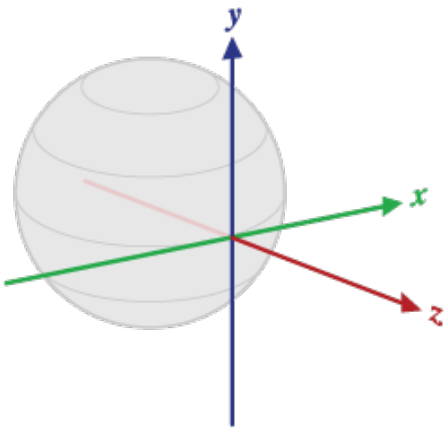

Abb. 6.3.: Referenz-Koordinatensystem für Android-Geräte [Developers, 2012c]

Listing 6.13: “Peilung zwischen Standort und Zielort“

Es wurde die bereits existierende Implementierung aus der *osmdroid*-Bibliothek verwendet. Diese setzt für die Berechnung der Peilung den Algorithmus aus Gleichung 5.5 ein.

Nachfolgend werden die Werte für den Azimuth sowie die errechneten Werte für die Peilung miteinander verrechnet. So ergibt sich die gewünschte Rotation der Kompass-Nadel in Richtung des Zielortes. Eine zweite Berechnung findet statt, um die Richtung des *True North* anzuzeigen — also die der nicht-magnetischen Nordrichtung. Beide Rotationen werden danach mittels einer Animation visuell durchgeführt (Listing 6.14).

```
1 float rotation = (360 - azimuth) + bearing;
2 Animation ani = new RotateAnimation(rotation, rotation, Animation
      .RELATIVE_TO_SELF, 0.5f,  Animation.RELATIVE_TO_SELF, 0.5f);
3 ani.setFillAfter(true);
4 compassGfx.startAnimation(ani);
5
6 float trueNorth = (360 - azimuth);
7 Animation northAni = new RotateAnimation(trueNorth, trueNorth,
      Animation.RELATIVE_TO_SELF, 0.5f, Animation.RELATIVE_TO_SELF,
      0.5f);
8 ani.setFillAfter(true);
9 compassNorth.startAnimation(northAni);
```

Listing 6.14: “Berechnung der nötigen Rotation für den Kompass“

6.1.3. Realisierungen innerhalb der Server-Anwendung

Die Realisierungen für den konzipierten Server-Dienst werden im Folgenden aufgezeigt. Hierbei werden neben der Implementierung der Dienst-Schnittstelle beispielhafte Umsetzungen erklärt, sowie die Interaktion mit der Persistenz-Schicht erläutert. Die Implementierungen erfolgen mit Bezug auf die innerhalb der Konzeption erarbeiteten Anwendungsfälle und stellen so den Dienst zur Verfügung, welche die mobilen Clients benötigen, um dem Nutzer die gewünschten Funktionen bieten zu können.

Öffentliche Schnittstelle des Dienstes

Zunächst wird innerhalb der Dienst-Schnittstelle überprüft, ob bei der Anfrage durch die client-seitige Anwendung eine Bilddatei enthalten ist. Diese wird, sofern vorhanden, durch die Schnittstelle extrahiert und in verschiedenen Größen auf dem Server abgespeichert. Die Speicherung erfolgt in unterschiedlichen Größen, um je nach Anwendung kleinere oder größere Bilder ausliefern zu können. Für die prototypische Realisierung liegen auf dem Server folgende Dateien vor: die Original-Datei mit den originalen Abmessungen, eine verkleinerte Bilddatei mit einer Größe von 500 x 500 Pixeln und eine „Thumbnail“-Variante mit 60 x 60 Pixeln. Die Original-Daten werden für die verkleinerten Ansichten mittels der Funktion *cropThumbnailImage()* auf eine rechteckige Größe zugeschnitten. Hierzu werden die Ränder der Bilder beschnitten, sodass der Inhalt auf den Bildern größtenteils vorhanden bleibt. Die zugeschnittenen Bilder werden in verschiedene Unterverzeichnisse auf dem Server abgelegt und können anschließend zur Anzeige auf dem Client ausgeliefert werden.

Im nächsten Schritt wird der vom mobilen Gerät gesendete JSON-Befehl aus der Anfrage extrahiert und durch den Server vorverarbeitet. Es werden die — bei der Kodierung eingesetzten — Leerzeichen entfernt und der resultierende String mittels *json_decode()* dekodiert. Die *action*-Eigenschaft des JSON-Objektes wird zerlegt und so die zu erstellende Klasse sowie die Methode identifiziert, welche im Anschluss aufgerufen wird. Listing 6.15 zeigt diese Vorverarbeitung aus der server-seitigen Umsetzung.

```
1 $_POST['json'] = stripslashes($_POST['json']);
2 $payload = $_POST['json'];
3 $payloadObj = json_decode($payload);
4 $identifier = explode(".", $payloadObj->action);
5 $classname = $identifier[0];
6 $methodname = $identifier[1];
```

Listing 6.15: “Vorverarbeitung der Client-Anfrage“

Die so gewonnenen Informationen über die gewünschte Aktion werden danach verwendet, um eine entsprechende Instanz der zuständigen Klasse zu erstellen. Ist diese Klasse der Schnittstelle nicht bekannt, wird versucht, sie zu importieren. Sollte dies ebenfalls fehlschlagen, wird eine Fehlermeldung an den Client gesendet. Ist die Instanziierung der Klasse erfolgt, wird die entsprechende Methode für die gewünschte Operation aufgerufen. Sollte die Methode nicht existieren, erhält der Client ebenfalls eine Fehlermeldung. Das Instanziieren der Klasse sowie der Aufruf der entsprechenden Methode sind im Listing 6.16 beispielhaft aufgezeigt. Nach dem Aufrufen der entsprechenden Methode erfolgt innerhalb der *API* keine weitere Aktion. Die Verantwortlichkeiten für die Ausführung der gewünschten Operationen werden somit an die entsprechende Instanz weitergegeben. Das Versenden der „Response“ an den Client erfolgt ebenfalls innerhalb der entsprechenden Methode. So wird gewährleistet, dass die Bearbeitung der Anfrage von der öffentlichen Schnittstelle des Dienstes entkoppelt ist und keine Abhängigkeiten entstehen. Sollte ein vorhandener Dienst andere Objekte benötigen — was zur Zeit nicht der Fall ist — kann dieser die entsprechende Klasse erzeugen, um die benötigte Funktionalität zu erhalten.

```
1  if(class_exists($classname)) {
2   $classObj = new $classname;
3   if(method_exists($classObj, $methodname)) {
4    $classObj->$methodname($payloadObj);
5   }else {
6   $outputArray =
7    Array("success" => false);
8    echo json_encode($outputArray);
9   }
10 }else {
11 ...
12 }
```

Listing 6.16: “Dynamische Instanziierung der Klasse und Aufruf der benötigten Methode“

Datenbank-Kommunikationsmodul

Die nötige Kommunikation mit der Datenbank für die server-seitige Anwendung zur Speicherung der Daten erfolgt über eine entkoppelte Schnittstelle. Diese ist als unabhängige Klasse implementiert und kann von anderen Objekten erzeugt werden. Die Klasse stellt eine Verbindung zur Datenbank her und verwaltet die Daten für die Anmeldung beim Datenbank-Management-System (DBMS). Es stehen drei Methoden zur Verfügung. Diese ermöglichen den Aufbau einer Verbindung inklusive eines Fehlermanagements sowie das Trennen einer Verbindung, wenn

diese nicht mehr benötigt wird. Mithilfe der Methode *query($query)* wird eine Anfrage an die Datenbank gesendet. Hierzu wird der Funktion lediglich die gewünschte SQL-Aktion übergeben, und das erhaltene Resultat der Datenbank-Abfrage wird an den ursprünglichen Aufrufer zurückgegeben.

Realisierung der Anwendungsfälle

Im Folgenden werden beispielhaft einige der realisierten Anwendungsfälle aufgezeigt und in ihrer Umsetzung erläutert, da eine detaillierte Beschreibung aller Funktionalitäten den Rahmen dieser Arbeit überschreiten würde.

Ort hinzufügen

Der Anwendungsfall „Ort hinzufügen“ wurde in der Klasse *loc* durch die Methode *add()* realisiert. Diese Operation wird mittels der Aktion „loc.add“ durch den Client aufgerufen — gibt der Nutzer die nötigen Parameter an, kann er eine neue Sehenswürdigkeit in das System einpflegen. Die nötigen Parameter für die Aktion sind in Tabelle 6.1 aufgelistet.

Parameterkürzel	**Beschreibung**
locationid	Name des Ortes
la	Latitude
lo	Longitude
cc	Ländercode der Nutzerposition
cn	Stadtname des Ortes
al	Adresszeile des neuen Ortes
desc	Beschreibung des Ortes
uid	Nutzer-ID
tags	Schlagworte für den Ort
img	Bilddaten für den neuen Ort (falls vorhanden)

Tabelle 6.1.: Parameter für das Einfügen eines Ortes

Ist eine Instanz der Klasse durch die Schnittstelle der Applikation erstellt und die Methode *add()* aufgerufen worden, wird zunächst der Benutzername des Erstellers von der Datenbank abgefragt. Dies ist notwendig, da der Client innerhalb der Anfrage nur die Nutzer-ID mitsendet. Anschließend werden die erhaltenen Parameter mittels eines SQL-Statements an die Datenbank übertragen. Ergänzend zu den Parametern des Clients wird der Nutzername und ein Zeitstempel eingesetzt, welcher das Erstellungsdatum kennzeichnet. Für einen neuen Ort wird innerhalb der Tabelle *GeoBeeLocations* eine neuer Eintrag generiert.

Zuletzt sendet die Funktion noch eine Antwort an den Client. Diese enthält alle geschriebenen Daten, welche zum neuen Ort gehören. Die vom Server versendete Antwort enthält weiterhin die Bezeichnung für die durchgeführte Aktion sowie ein Markierung, welche den Erfolg oder Misserfolg einer Aktion kennzeichnet. Die Wiederholung des Aktionstyps innerhalb der Server-Antwort wird vom Client benötigt, um die Antwort dem korrekten Empfänger zuordnen zu können (Listing 6.17). Die an den Client übertragenen Daten werden zuletzt noch in ein JSON-Format kodiert.

```
$this->database->connect();
$res = $this->database->query("INSERT INTO GeoBeeLocations
    VALUES('', '".$data->locationid."',
  '".$data->la."',
  '".$data->lo."',
  '".$data->cc."',
  '".$data->cn."',
  '".$data->al."',
  '".$data->desc."',
  '".$data->img."',
  '".$data->rating."',
  '".$data->views."',
  '".$username."',
  '".time()."',
  '".$data->tags."')");
$this->database->disconnect();

$outputArray =
 Array("action" => $data->action,
 "success" => $res,
 ...
 );
echo json_encode($outputArray);
```

Listing 6.17: “Server-seitiges Hinzufügen eines Ortes“

Bilddaten abrufen

Die zweite Funktionalität, die erläutert werden soll, ist das Abfragen eines Detailbildes vom Server-Dienst. Dies wird beispielsweise vom Client durchgeführt, wenn der Nutzer eine Detailansicht zu einem bestimmen Ort betrachten möchte. Die vom Server gespeicherten Bilder befinden sich in verschiedenen Verzeichnissen. Innerhalb dieser Methode wird dem Client ein Bild in der Größe 500 x 500 Pixel ausgeliefert. Die Operation wird durch

das Kommando „med.gP“ aufgerufen. Der einzige Übergabe-Parameter ist hierbei der Dateiname der angeforderten Bilddatei, welcher innerhalb der Tabelle *GeoBeeLocations* hinterlegt ist (siehe Tabelle 6.2).

Parameterkürzel	**Beschreibung**
img	Dateiname des angeforderten Bildes

Tabelle 6.2.: Parameter für das Abrufen eines Bildes

Die Server-Applikation überprüft zunächst, ob das angeforderte Bild auf dem Datenträger vorhanden ist. Ist dies der Fall, werden die Bilddaten mittels der Funktion *file_get_contents()* vom Speicher des Servers geladen. Anschließend erfolgt eine Kodierung der Daten durch den Aufruf der Methode *base64_encode()*. Die Kodierung ist notwendig, um die binären Bilddaten für die Übertragung an den Client aufzubereiten. Erfolgt dieser Schritt nicht, kommt es zu Fehlinterpretationen auf Seiten des Clients. Zuletzt werden die aufbereiteten Daten an den Client gesendet (Listing 6.18).

```
1  if(file_exists("detail/".$data->img)) {
2   $imgData = file_get_contents("detail/".$data->img);
3   $value = base64_encode($imgData);
4  }else {
5   $value = "";
6  }
7  $outputArray =
8   Array("action" => $data->action,
9   "success" => file_exists("detail/".$data->img),
10  "imgData" => $value);
```

Listing 6.18: “Versenden von Bilddaten durch den Server“

Nutzer registrieren

Verwendet ein neuer Nutzer die Anwendung zum ersten Mal, muss er sich ein Benutzerkonto erstellen, mit welchem er sich bei der Software anmelden kann. Dies erfolgt über die mobile Software auf einem Android-Gerät. Nach dem Start der Anwendung bekommt der Nutzer über den „Registrieren“-Button die entsprechenden Felder angezeigt. In diese trägt er E-Mail-Adresse, Nutzernamen und ein Passwort ein. Der Client überträgt die Daten im Anschluss an den Web-Dienst (siehe Tabelle 6.3).

Zur Erstellung eines neuen Kontos muss das Kommando *usr.add* an den Server gesendet werden. Die Methode *add* in der Klasse *usr* führt dann die nötigen Schritte aus, welche

Parameterkürzel	Beschreibung
username	Gewünschter Benutzername
email	E-Mail-Adresse des Nutzers
password	Gewähltes Passwort des Nutzers

Tabelle 6.3.: Parameter zum Erzeugen eines Benutzerkontos

für eine Registrierung nötig sind. Das Listing 6.19 zeigt, dass zunächst durch den Dienst überprüft wird, ob bereits ein Nutzer mit dem gleichen Benutzernamen existiert. Ist dies der Fall, erhält der Client die Fehlermeldung *error.doubleuser*, und der Nutzer muss einen anderen Benutzernamen wählen, um sich registrieren zu können. Ist der gewählte Benutzername noch verfügbar, wird mit der Erstellung des Nutzerkontos fortgefahren.

```
1  $this->database->connect();
2  $testUsername = $this->database->query("SELECT uid FROM
       GeoBeeUser WHERE username = '".$data->username."'");
3  $this->database->disconnect();
4
5  if(@mysql_num_rows($testUsername) == 1) {
6   $outputArray =
7    Array("action" => $data->action,
8     "success" => "error.doubleuser",
9   );
10 echo json_encode($outputArray);
```

Listing 6.19: "Prüfung des Benutzernamens"

Um das Benutzerkonto im System zu erzeugen, wird innerhalb der Datenbank ein neuer Eintrag in der Tabelle *GeoBeeUser* generiert. Hierzu werden Benutzername, E-Mail-Adresse und das mittels *crypt()* verschlüsselte Passwort hinterlegt. Weiterhin wird für den Nutzer ein Zeitstempel generiert, um ihn nach der Registrierung am System anzumelden. Die eingesetzte Verschlüsselung ist von der zugrundeliegenden PHP-Version und der Konfiguration abhängig. So wird für das Erzeugen eines Hash-Wertes entweder der DES-Algorithmus oder MD5 verwendet. Der benötigte Salt für das Hashing wird durch die Funktion *crypt()* selbständig erzeugt. Listing 6.20 zeigt die Implementierung des beschriebenen Vorgangs.

```
1  $this->database->connect();
2  $res = $this->database->query("INSERT INTO GeoBeeUser VALUES
       ('', '".$data->username."',
3    '".$data->email."',
```

```
4     '".crypt($data->password)."',
5     '".time()."','')");
6 $this->database->disconnect();
7 ...
8 $outputArray =
9  Array("action" => $data->action,
10   "success" => $res,
11   "uid" => $uid
12  );
13 echo json_encode($outputArray);
```

Listing 6.20: "Erstellen eines neuen Benutzerkontos"

6.2. Generierung der Heatmaps

Zu den geplanten Funktionen der Anwendung gehörte unter anderem die Darstellung von kategorisierten Orten in Form einer Heatmap. Sie soll eine einfache und für den Nutzer gut verständliche Darstellung von Informationen zu Sehenswürdigkeiten und interessanten Plätzen einer Stadt bieten. Weiterhin bietet die Verarbeitung und Aggregation von Orten einer Stadt nach bestimmten Eigenschaften wie beispielsweise Interessenlagen oder Durchschnittsalter eine gute Möglichkeit, Demographien über eine Stadt oder Region anzulegen. So könnten die Besucher einer Stadt anhand der Heatmap-Visualisierungen auf einer Stadtkarte die Bereiche einer Stadt ausmachen, welche ihren Vorlieben oder ihrem Alter am ehesten entsprechen.

Da die komplette Umsetzung eines solchen Sub-Systems den Rahmen der praktischen Realisierbarkeit überschreiten würde und es weiterhin nur Ziel war, einen prototypische Umsetzung durchzuführen, soll im folgenden Abschnitt eine theoretische Realisierung thematisiert werden. Aufbauend auf einigen kleineren Umsetzungen, die für die Darstellung von Heatmaps bereits erstellt wurden, sollen mögliche Verbesserungen und Erweiterungen vorgestellt werden. Um die Sinnhaftigkeit dieser Darstellungart aufzuzeigen, werden zudem zwei Systeme vorgestellt, welche eine solche Visualisierung bereits umgesetzt haben.

6.2.1. Prototypische Einbindung in GeoBee

Um eine erste prototypische Umsetzung der Heatmaps zu ermöglichen und die Einbindung in das konzipierte System zu überprüfen, wurde eine einfache Darstellung von Heatmaps auf Basis des *osmDroid*-Frameworks vorgenommen. Das Framework erlaubt es, eigene Visualisierungsebenen (Overlays) in die Kartenanwendung einzubinden. Um solch ein angepasstes *Overlay* zu erzeu-

gen, wird die von *osmDroid* bereitgestellte Klasse *Overlay* erweitert. Bei der Instanziierung der abgeleiteten Klasse — in diesem Fall die Klasse *HeatmapOverlay* — wird zum einen der aktuelle Kontext der Applikation übergeben, zum anderen müssen noch ein MapView sowie eine Liste von Orten an die Klasse weitergegeben werden. Um ein derartiges *Overlay* nutzen zu können, wird eine weitere Klasse benötigt, welche eine Android-Activity erweitert. Diese muss ebenfalls das genannte MapView erzeugen und die Instanziierung des *Overlays* durchführen. Um das generierte *Overlay* einzubinden, wird der Befehl aus Listing 6.21 in die Karten eingefügt.

```
1 heatmapOverlay = new HeatmapOverlay(this, map, list);
2 map.getOverlays().add(heatmapOverlay);
```

Listing 6.21: “Einbinden des HeatmapOverlays in die Karte“

Zur Darstellung der Punkte auf der Kartenanwendung wurde aufgrund der prototypischen Umsetzung eine einfache Methode gewählt. Diese clustert die vorliegenden Orte in drei Stufen und zeichnet danach die Orte nach einem vorgegebenen Farbschema. Die Clusterung der Daten erfolgte, wie in Tabelle 6.4 ersichtlich, anhand der Entfernung zu benachbarten Orten:

Entfernung	**Farbschema**
< 50 Meter	Rot-Färbung des Ortes
> 50 Meter < 100 Meter	Orange-Färbung des Ortes
> 100 Meter	Gelb-Färbung des Ortes

Tabelle 6.4.: Klassifizierung der Datensätze anhand der Entfernung zueinander

Entsprechend dieser Einordnung werden die Orte auf der Karte hinzugefügt und mittels des Farbschemas mit einem farbigen Kreis markiert. Die farbigen Kreise, welche einen Ort markieren, werden in der Client-Software mit Hilfe des Paint-Objektes erzeugt (siehe Listing 6.22). Dies ermöglicht es, sowohl Farbwerte mit Alpha-Informationen einzustellen, als auch Eigenschaften für Linien-Stärke, Größe und Linien-Art zu definieren. Anschließend wird der Ort mithilfe der vorliegenden Geo-Koordinaten in die Karte eingefügt. Im letzten Schritt wird für den entsprechenden Ort ein Kreis gezeichnet, welcher den entsprechenden Farbwert besitzt.

```
1
2 Paint paint = new Paint();
3 paint.setARGB(150, 255, 0, 0);
4 paint.setDither(true);
5 paint.setStyle(Paint.Style.FILL_AND_STROKE);
6 paint.setStrokeJoin(Paint.Join.ROUND);
7 paint.setStrokeCap(Paint.Cap.ROUND);
```

```
8  paint.setStrokeWidth(40);
9
10 Point point = new Point();
11 projection = map.getProjection();
12 GeoPoint geoPoint = new GeoPoint(loc.getLat(), loc.getLng());
13 projection.toPixels(geoPoint, point);
14
15 arg0.drawPoint(point.x, point.y, paint);
```

Listing 6.22: “Zeichnen der Markierungen von Orten als Heatmap“

Damit es bei der Ansammlung von vielen Orten innerhalb eines kleinen Radius zu Überlagerungseffekten kommen kann, ist der Alpha-Wert der Füllung eines Kreises auf einen Wert von etwa 50% gesetzt. Somit können die Farbflächen durch andere Flächen leicht durchscheinen und bieten eine gute Darstellung von vielen Orten auf kleinem Raum.

Das Ergebnis ist in Abbildung 6.4 zu sehen. Es zeigt erste Ansätze für eine Darstellung in Form einer Heatmap, ist allerdings nur als prototypische Umsetzung zu verstehen und somit noch verbesserungswürdig. Nachfolgend sollen zwei Realisierungen vorgestellt werden, in denen eine solche Visualisierung umgesetzt wurde.

6.2.2. Einsatz von Heatmaps in anderen Systemen

Um den sinnvollen Einsatz von Visualisierungen von ortsbezogenden Daten anhand von Heatmaps zu belegen, werden im Folgenden beispielhaft zwei Systeme vorgestellt, welche diese Art der Abbildung einsetzen.

Das erste System ist die Kartenanwendung der Firma Nokia. Diese ist sowohl als klassische Karten-Applikation in Form einer Webseite zu erreichen, als auch auf von Nokia produzierten mobilen Endgeräten. Zumindest der Web-Dienst ermöglicht es seit 2011 [Blog, 2012], bestimmte Eigenschaften einer Gegend in Form von Heatmaps zu zeigen. Dafür muss der Nutzer unter einem Menüpunkt eine Kategorie wählen — beispielsweise Sehenswürdigkeiten, Restaurants, Bars oder Geschäfte. Aus der Sammlung von Orten werden dem Nutzer dementsprechend besonders lohnenswerte Orte rötlich dargestellt. Die Abstufungen der Farben reichen von rot bis gelb und zeigen so besonders geeignete Gebiete leichtverständlich auf. Eine exemplarische Ansicht einer solchen Visualisierung ist in Abbildung 6.5 zu sehen. Sie zeigt die Stadt Hamburg und markiert die besten Gegenden für Sehenswürdigkeiten.

Eine zweite nennenswerte Anwendung ist durch Patrick Wied entstanden [Wied, 2012]. Diese *JavaScript*-Umsetzung eines Frameworks zur Darstellung von Heatmaps ermöglicht eine einfache Einbindung in verschiedene Applikationen. So entstanden angepasste Frameworks —

Abb. 6.4.: Ergebnis der prototypischen Heatmap-Visualisierung

unter anderem auch eine Implementierung für den Einsatz auf Karten. Es existieren hier sowohl Umsetzungen für Google Maps als auch für das Open-Street-Maps-Projekt. Die einzubindende JavaScript Klasse erlaubt es, angelegte Datensätze anhand der Koordinaten sowie einer Häufigkeit von beispielsweise Besuchen durch Nutzer, eine Heatmap zu generieren. Der direkte Einsatz des Frameworks innerhalb von GeoBee ist zurzeit nicht möglich, da es zunächst in *Java* — die native Programmiersprache für Android — portiert werden müsste.

6.2.3. Zusammenfassung und Ausblick

Die prototypische Umsetzung von Heatmaps in dieser Arbeit ist sehr einfach gehalten und muss an vielen Stellen verbessert und erweitert werden. Anhand der vorgestellten Projekte, welche diese Art der Visualisierung verwenden, ist ersichtlich, dass eine solche Art der Darstellung von Informationen sinnvoll und angebracht ist. Eine Möglichkeit der Erweiterung und Verbesserung wäre die Portierung der Applikation von Wied [Wied, 2012] für die Einbindung in native Android-Anwendungen. Denkbar wäre auch die Überarbeitung des hier entwickelten *Overlays*. Hierzu müsste zunächst ein verbessertes Clustering erfolgen und die Option, vorhandene Orte nach einer

Kategorie auszuwählen und zu sortieren. Zudem müsste die Darstellungsform einem typischen Heatmap-Stil angepasst werden.

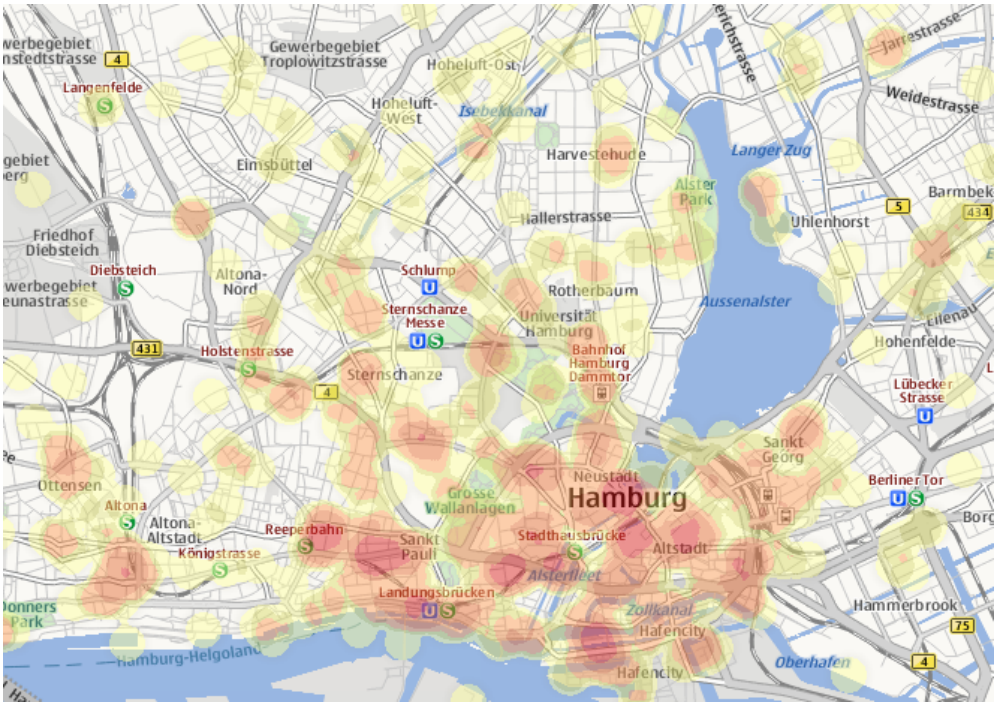

Abb. 6.5.: Heatmap-Visualisierung innerhalb von Nokia Maps [Maps, 2012]

7. Test

Entwickelt man eine Software-Anwendung, muss man diese unbedingt auch testen, um zu ermitteln, ob die vorab festgelegten Anforderungen erfüllt worden und hinreichend qualitativ sind. Die Hauptmotivation des Testens einer Applikation liegt im Auffinden von Fehlern und in der Vorbeugung von potentiellen Fehlerquellen. Da die Beseitigung von Fehlern mit dem Voranschreiten der Software-Entwicklung immer schwieriger und komplexer wird, ist es wichtig, frühzeitig mit der Überprüfung einzelner Software-Bestandteile zu beginnen. Unter anderem bestehen bezüglich des Testens von Software zwei wichtige Prinzipien:

Prinzip 1

> „Testing shows the presence of defects, not their absence."[Spillner u. a., 2011]

Prinzip 2

> „Exhaustive testing is not possible."[Spillner u. a., 2011]

Diese Prinzipien zeigen, dass auch eine sehr umfangreiche Durchführung von Tests nicht gewährleisten kann, dass keine Fehler mehr auftreten. Allerdings ermöglicht die Erhebung von Testfällen eine gewisse Art der Vorsorge, sodass Fehler beim Regelbetrieb der Software kaum auftreten sollten. Lediglich in extremen Situation, z.B. unter großer Auslastung, könnte ein Software-System Fehler erzeugen.

7.1. Testvorgehen

Verschiedene Modelle bieten sich an, um die Entwicklungsschritte im frühen Stadium der Software-Programmierung zu testen. Eines der Bekanntesten ist das V-Modell aus Abbildung 7.1. Es reflektiert die Verbindungen zwischen Entwicklung und Software-Test. Die linke Seite des V-Modells zeigt die Schritte während des Entwicklungsprozesses, welche nach dem Durchlaufen der Definition, dem Entwurf und der Spezifikation mit der Implementierung enden. Die rechte Seite des Modells beschreibt die Tests und Integrationsprozesse, welche die korrespondierenden Schritte in der Entwicklung verifizieren. Die Verwendung des V-Modells während der Entwicklung einer

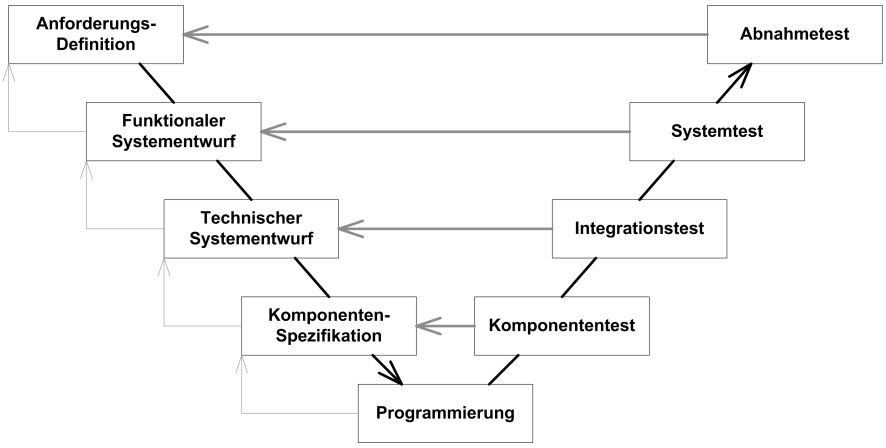

Abb. 7.1.: Aufbau des generellen V-Modells

Anwendung ermöglicht so eine gute Testabdeckung und eine Verifikation der Funktionalitäten der Software mit den gestellten Anforderungen an das System.

Die integrierten Teststufen des V-Modells sollen im Folgenden kurz erläutert werden:

Komponententest

Die Durchführung von Komponententests erfolgt durch die Entwickler der Software. Gestestet werden Module, wie beispielsweise Klassen. Dieser Vorgang ermöglicht es, die Bewertung und Sicherstellung für eine technische Lauffähigkeit und korrekte fachliche Ergebnisse durch diese Module zu gewährleisten.

Integrationstest

Diese Art von Tests prüfen die Interaktion zwischen verschiedenen Modulen auf Korrektheit. Die durch das System-Design vorgegebene Zusammenarbeit von Komponenten wird geprüft, und der Test von definierten Schnittstellen wird durchgeführt.

Systemtest

Um das gesamte System zu untersuchen und die gegebenen fachlichen und nicht-fachlichen Anforderungen an die Software mit der Realisierung abzugleichen, erfolgt ein Systemtest. Meistens werden zur Prüfung der Funktionalitäten Testdaten verwendet, welche den späteren Einsatz der Software simulieren.

Abnahmetest

Die letzte Stufe des Testablaufs erfolgt in den meisten Fällen durch den Auftraggeber. Dieser testet die erhaltene Software auf die vorgegebenen Anforderungen hin. Da lediglich

das Verhalten des Systems getestet wird, dient hierzu ein „BlackBox“-Test. Das bedeutet, dass nicht die Implementierung der Software überprüft wird, sondern die spezifizierten Handlungen der Software beim Gebrauch.

7.2. Testdurchführung für GeoBee

Im Folgenden wird die Testdurchführung für die hier entwickelte Anwendung *GeoBee* vorgestellt. Es werden Testziele erläutert und die Teststrategie aufgezeigt. Anschließend wird der Testaufbau sowohl für die Client- als auch für die Server-Applikation beispielhaft vorgestellt. Anhand der Teststufen des o.g. V-Modells wird der Software-Test erläutert und abschließend ein Ausblick auf mögliche Verbesserungen und Automatisierungen gegeben.

7.2.1. Testziele

Die festgelegten Ziele eines Softwaretests sollen die Qualitätsansprüche an die Software widerspiegeln. Sie dienen deshalb hauptsächlich dem Auffinden von Fehlern innerhalb der Applikation. Die Testziele variieren je nach Teststufe und Softwareprodukt. Somit müssen diese individuell an die Art des Tests und die vorliegende Software angepasst werden. Innerhalb dieser Masterarbeit wird das Verfahren an das V-Modell angelehnt. So ergeben sich folgende Testziele:

- Validierung der einzelnen Module innerhalb von Client- und Server-Anwendung durch Unit-Tests.
- Überprüfung der Integration von Modulen durch manuelle Schnittstellentests.
- Systemtests durch Verwendung der Software und Überprüfung der funktionalen Anforderungen.
- Abnahmetests von möglichen Nutzern. Feedback bezüglich der Funktionalitäten sowie der Nutzerschnittstelle.

Die Umsetzung der Tests wird im Abschnitt zum Testaufbau 7.2.3 anhand von Beispielen erläutert.

7.2.2. Teststrategie

Da ein umfassender Software-Test aufgrund der Vielfalt an Implementierungen nicht möglich ist, ist es wichtig, eine Strategie festzulegen, anhand welcher die Tests durchgeführt werden. Diese Strategie soll die Testanforderungen festlegen und den Rahmen der zu erstellenden Testfälle

abstecken. Die Festlegung einer Strategie zeigt weiterhin die Abdeckung an, welche mittels der durchgeführten Tests erreicht werden kann.

Nachfolgend wird der Umfang der Tests beschrieben, welche für die hier entstandene Software durchgeführt wurden:

Unit-Tests

Die Module der Client- und der Server-Anwendung werden mittels Unit-Tests auf Korrektheit überprüft. Da die entstandenen Module sehr umfangreich sind, und die Durchführung eines Tests nicht den Hauptteil der vorliegenden Arbeit ausmacht, soll dies nur beispielhaft erfolgen. Die Beispiele zeigen allerdings das grundsätzliche Vorgehen und den Einsatz von Unit-Test-Frameworks.

Integrationstests

Die Integration neuer Module soll anhand von Schnittstellentests erfolgen. Hierzu werden Teile der Software manuell auf Fehlerfreiheit getestet; ebenfalls überprüft werden die Schnittstellen zwischen den Modulen — dabei insbesondere die Kommunikation zwischen Client und Server.

Systemtests

Ein vollständiger Systemtest erfolgt anhand der in Abschnitt 4.3.2 und im Anhang B aufgestellten funktionalen Anforderungen. Diese beschreiben die nötigen Funktionen, die die Software erfüllen muss. Die Systemtests erfolgen manuell und validieren den nötigen Funktionsumfang.

Abnahmetests

Da die entwickelte Software nicht für einen einzelnen Auftraggeber erfolgte, wird der Abnahmetest als Feldversuch durchgeführt. Hierzu ergaben sich während der Entwicklung der Anwendung zwei Gelegenheiten. Zum einen die jährlich stattfindende „Nacht des Wissens“ in Hamburg, zum anderen das jährliche Alumni-Treffen des Departments Informatik an der Hochschule für angewandte Wissenschaften in Hamburg. Diese beiden Veranstaltungen wurden genutzt, um Abnahmetests bzw. Nutzertests durchzuführen. Geprüft wurden dabei die verfügbaren Funktionalitäten von GeoBee, auch wurde das Nutzerinterface von den Veranstaltungsbesuchern verwendet und bewertet.

7.2.3. Testaufbau

Dieser Abschnitt soll den Testaufbau und die Umsetzung der verschiedenen Teststufen erläutern. Beispielhaft werden einige Realisierungen aufgezeigt und die Rahmenbedingungen für die jeweiligen Stufen erläutert.

Unit-Tests für Client und Server

Sowohl für die Client- als auch die Server-Applikation wurden exemplarisch Unit-Tests durchgeführt, um die entwickelten Module auf Korrektheit zu überprüfen. Im Folgenden werden die Frameworks zur Umsetzung solcher Unit-Tests vorgestellt und ihr Einsatz anhand von Beispielen erklärt:

Bei der Entwicklung der Unit-Tests für den Client wurde das von Android bereitgestellte Test-Framework verwendet. Es basiert auf dem JUnit-Framework und weiteren Tools, welche eine Automatisierung ermöglichen oder das Testen von Applikationen auf Mobilgeräten erleichtern [Developers, 2012d]. Zusätzlich wurde das für Android erhältliche Robotium-Framework eingesetzt [Robotium, 2012]. Dieses bietet eine einfache Schnittstelle, um Nutzerinteraktion zu simulieren und so automatisierte Oberflächen-Tests durchzuführen.

Abbildung 7.2 zeigt die Testumgebung, welche als Plugin für Eclipse verfügbar ist. Die erstellten Testfälle werden nacheinander ausgeführt und Ausgaben erstellt, falls ein Test fehlschlägt.

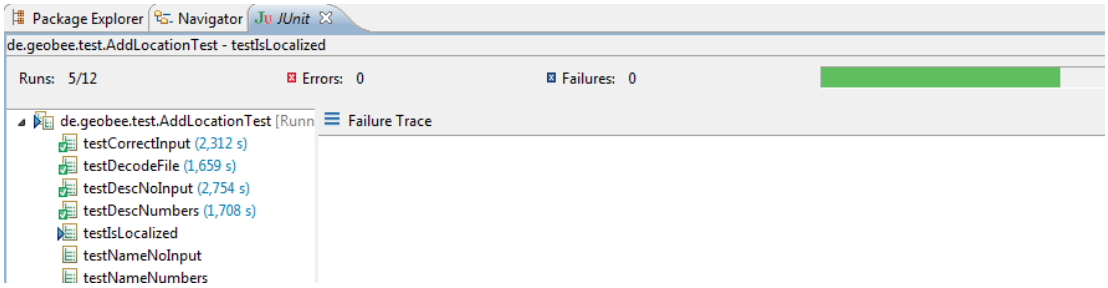

Abb. 7.2.: Testumgebung von JUnit für Android

Um Unit-Tests für Android-Applikationen zu realisieren, wird ein neues Test-Projekt angelegt und mit der zu überprüfenden Anwendung verknüpft. Die „Test-Cases“ für die verschiedenen Klassen der Anwendung werden hier ebenfalls in Klassen aufgeteilt. Das folgende Listing zeigt den grundsätzlichen Aufbau einer solchen Test-Klasse. Diese muss zum Testen einer Activity die Klasse *ActivityInstrumentationTestCase2<T>* erweitern. Neben dem Aufruf des Kontruktors erfolgt der Aufruf der Methode *setUp()*. Diese erlaubt es, nötige Vorbedingungen für den Test

umzusetzen. Sind alle Testfälle durchgeführt worden, wird die überprüfte Aktivität innerhalb der Methode *tearDown()* wieder beendet.

```
1  public class AddLocationTest extends
       ActivityInstrumentationTestCase2<AddView> {
2   private Solo solo;
3
4   public AddLocationTest() {
5    super("de.geobee.view.AddView", AddView.class);
6   }
7   @Override
8   protected void setUp() throws Exception {
9    super.setUp();
10   solo = new Solo(getInstrumentation(), getActivity());
11  }
12  @Override
13  protected void tearDown() throws Exception{
14   solo.finishOpenedActivities();
15  }
16 }
```

Listing 7.1: “Notwendige Bestandteile einer Activity-Test-Klasse“

Die Realisierung von zwei Testfällen wird im Folgenden beispielhaft gezeigt. Es handelt sich dabei erstens um die Überprüfung der Verfügbarkeit einer Lokalisierung während des Hinzufügens eines neuen Ortes und zweitens um die Überprüfung der Validierung von Eingabedaten in zwei Textfelder des Interfaces. In beiden Fällen wird mittels der *assertTrue()*-Methode geprüft, ob die erwünschte Bedingung erfüllt wird. Das Ergebnis bezüglich der Verfügbarkeit von Lokalisierungen kann dementsprechend nur positiv ausfallen, wenn GPS-Koordinaten vorhanden sind. Da für die Lokalisierung des Nutzers Zeit benötigt wird, wartet die Testanwendung zehn Sekunden, bevor die Bedingung geprüft wird.

```
1  public void testIsLocalized() {
2   synchronized (solo) {
3    try {
4     solo.wait(10000);
5    } catch (InterruptedException e) {
6     e.printStackTrace();
7    }
8   }
9
```

```
10  Boolean result = getActivity().isLocalized();
11  assertTrue(result != false);
12 }
```

Listing 7.2: “Testen der Lokalisierung des Nutzers“

```
1 public void testCorrectInput() {
2  EditText name = (EditText) solo.getView(R.id.uid);
3  EditText desc = (EditText) solo.getView(R.id.desc);
4
5  solo.enterText(name, "NameOfLocation");
6  solo.enterText(desc, "Description of the Location");
7
8  assertTrue(getActivity().verifyInput());
9 }
```

Listing 7.3: “Testfall für eine korrekte Nutzereingabe“

Im zweiten Testfall werden Eingaben innerhalb der Anwendung durchgeführt. Dies geschieht automatisch durch das Robotium-Framework. Werden die Testdaten durch die Applikation als korrekt ausgewertet, ist auch dieser Testfall erfolgreich. Neben der Validierung von richtigen Eingaben wurden auch Testfälle zur Überprüfung von Falscheingaben umgesetzt. Diese sind „erfolgreich“, wenn das System mit einer Fehlermeldung reagiert und somit die Eingabe nicht zulässt.

Die Module der Server-Seite wurden ebenfalls durch den Einsatz von Unit-Tests validiert. Hierzu wurde das SimpleTest-Framework in der Version 1.1.0 eingesetzt [SimpleTest, 2012]. Dieses ermöglicht eine einfache Integration von Testfällen für die entsprechenden Server-Module. Für die Verwendung des Frameworks im eigenen Projekt wird lediglich der Ordner, welcher die nötigen Dateien des Frameworks enthält, in die Server-Struktur mit aufgenommen. Die Testfälle werden in Test-Klassen gekapselt und gelten je für eine Realisierungs-Klasse. Diese wird innerhalb der Test-Klasse instanziiert, anschließend werden die definierten Testfälle abgearbeitet. Das Ergbenis des jeweiligen Tests ist nachfolgend als Ausgabe zu sehen (siehe Abbildung 7.3).

Damit eine Test-Klasse unter SimpleTest erstellt werden kann, muss diese die Klasse *UnitTestCase* erweitern. Weiterhin ist es erforderlich, dass die Klasse *autorun.php* und die zu testende Klasse eingebunden werden. Der Aufbau einer Test-Klasse ist in Listing 7.4 zu sehen.

```
1 <?php
2 require_once('../simpletest/autorun.php');
3 require_once('../usr.php');
4
```

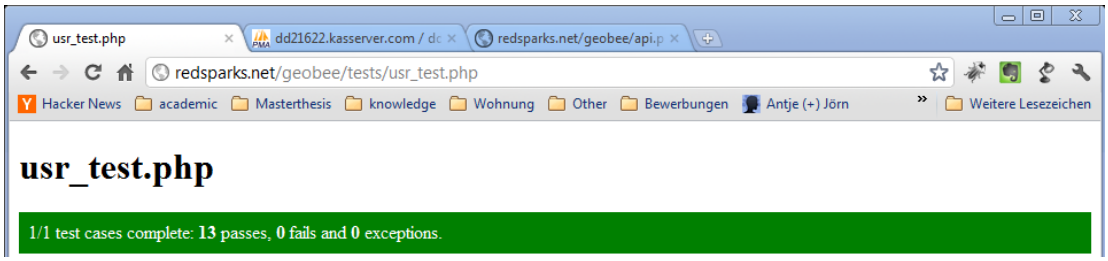

Abb. 7.3.: Alle Testfälle für die Klasse *Usr* der Serveranwendung erfolgreich

```
5 class TestofUsr extends UnitTestCase {
6 ...
7 }
```

Listing 7.4: "Aufbau der Test-Klasse für einen Unit-Test"

Beispielhaft für die Realisierung eines solchen Unit-Tests zeigt Listing 7.5 die Validierung des Login-Vorgangs mit Testdaten. Hierzu wird durch die Test-Klasse eine Anfrage an den Server simuliert und anschließend die entsprechende Methode des Servers aufgerufen. Die Überprüfung von Datenbank-Abfragen und die Überprüfung der Daten im Server erfolgt danach. Zuletzt werden die Funktionen der Passwort-Verschlüsselung geprüft und sichergestellt, dass ein Benutzerbild auf dem Server hinterlegt ist. Werden alle Testfälle erfüllt, ist das Ergebnis für diesen Test positiv.

```
1  function testUsrLoginUserSuccess() {
2   $usr = new usr();
3   $mockRequest = Array("action" => "usr.login","password" => "test
        ","username" => "Test");
4   $usr->login($mockRequest);
5
6   $this->assertTrue(@mysql_num_rows($usr->res) == 1);
7   $this->assertNotNull($usr->passServer);
8   $this->assertNotNull($usr->uid);
9   $this->assertNotNull($usr->username);
10
11  $this->assertTrue(crypt($mockRequest['password'], $usr->
        passServer) == $usr->passServer);
12  $this->assertTrue(file_exists("../userimg/".$usr->uid.".jpg"));
13 }
```

Listing 7.5: "Unit-Test für eine erfolgreiche Authentifizierung"

Integrationstests

Die Integrationstests für die entwickelte Anwendung erfolgten parallel zur Realisierung dieser. Neue Module wurden in das System eingebettet, direkt danach wurde die Funktionalität geprüft. Auch wurden die verfügbaren Schnittstellen getestet. Client-seitig erfolgte die Überprüfung der Schnittstellen durch die Verwendung der Software und die Verifizierung der korrekten Ausgaben einer Schnittstelle mittels der Log-Funktionalität der Entwicklungsumgebung.

Um die server-seitige Integration von Modulen und die Interaktion zwischen Client- und Server-Modulen zu überprüfen, wurde eine eigens entwickelte Web-Anwendung genutzt. Diese ermöglicht es, mittels eines einfachen Interfaces ohne Zeitverlust Anfragen an den Dienst zu stellen. Mit der Web-Applikation kann der Entwickler ein JSON-Kommando an den Server senden und die Antwort des Dienstes als Ausgabe erhalten. Abbildungen 7.4 und 7.5 zeigen die Test-Applikation und das erhaltene Ergebnis einer Abfrage als JSON-formatierte Antwort. Während der Entwicklungsarbeit ist dies ein hilfreiches Werkzeug, um Module des Dienstes zu testen, ohne dass bereits eine konkrete Client-Anwendung existiert. Es erlaubt zudem eine gewisse Unabhängigkeit von Client- und Server-Entwicklung zueinander.

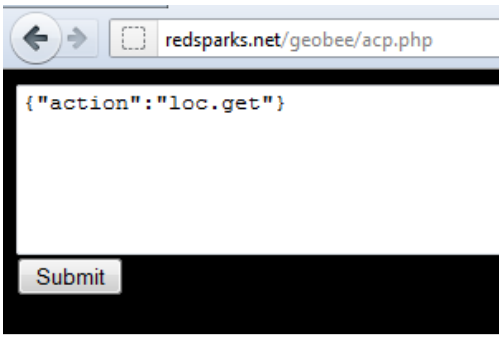

Abb. 7.4.: Eingabemaske der Test-Applikation

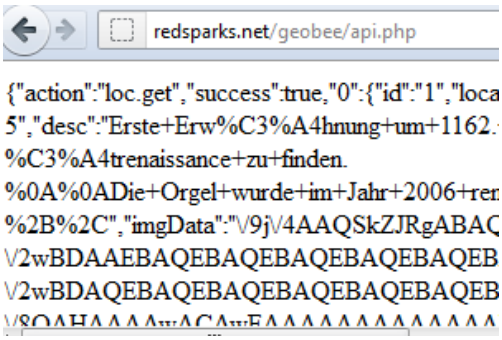

Abb. 7.5.: Ergebnis der Server-Abfrage

Systemtests für GeoBee

Die Systemtests erfolgten während der Entwicklung der Applikation iterativ. So wurden bei Fertigstellung eines Use-Cases die funktionalen Anforderungen an die Anwendung manuell geprüft. Die iterative Vorgehensweise beim Testen des bisher realisierten Systems zeigt frühzeitig etwaige Fehler in der Umsetzung auf. Zudem konnten durch die Verwendung der Software fehlende Anforderungen ergänzt und bestehende Anforderungen überarbeitet werden. Auch wenn der Entwickler der Anwendung das Nutzerinterface kennt, können auf diesem Weg trotzdem erste Probleme bei der Verwendung der Software aufgezeigt werden. Die Anpassung des Interfaces

und die Optimierung der Usability resultierte ebenfalls unmittelbar aus den durchgeführten Systemtests.

Abnahmetest / Nutzertest

Für den Abnahmetest bzw. den Nutzertest wurden, wie oben angegeben, die „Nacht des Wissens“ und das jährliche Alumni-Treffen an der HAW Hamburg genutzt (siehe Abbildung 7.6). GeoBee wurde dem Publikum vorgestellt und im persönlichen Gespräch detailliert erklärt. Die Besucher hatten die Möglichkeit, die Anwendung und ihre Funktionalitäten „live“ zu testen.

Abb. 7.6.: Feldtest der Anwendung auf der Nacht des Wissens 2011

Der Nutzertest erlaubte die Durchführung eines „Black-Box“-Tests, da die Nutzer die Implementierung der Software nicht kannten. Weiterhin konnten durch Beobachtung der Anwender bei der Nutzung des Systems weitere Probleme im Bereich der Usability und der User-Experience identifiziert werden. Das Feedback der Nutzer zu den Funktionalitäten und möglichen Erweiterungen des Systems konnte wiederum in die fachlichen Anforderungen an die Software eingehen. Abschnitt 8.3 stellt einige der möglichen Erweiterungen vor, welche durch Nutzer der Software genannt wurden. Leider konnten diese aus zeitlichen Gründen nicht mehr in die Entwicklung einfließen und dienen nun als Ausblick auf einen Ausbau der Software.

7.2.4. Testergebnisse und Testauswertung

Die Test-Ergebnisse der verschiedenen Testphasen wurden bei der Entwicklung und Anpassung der Applikation berücksichtigt. Die Durchführung der Unit-Tests für Client und Server-Anwendung validieren die Korrektheit der einzelnen Module für sich. Leider konnten aus zeitlichen Gründen nur exemplarisch Unit-Test implementiert werden. Für einen umfassenderen Test der Module müsste diese für jede realisierte Klasse erzeugt werden. Das Testen der Integration der einzelnen Module in das Gesamtsystem sowie die iterativen Systemtests ermöglichen eine Abstimmung der gestellten Anforderungen mit den erfolgten Realisierungen. Weiterhin helfen sie dabei eine Optimierung von Nutzerinterface und User-Experience bei der Verwendung der Software durchzuführen. Der Nutzertest bzw. Abnahmetest, welcher als Feldversuch erfolgte, ermöglichte es, das Nutzerinterface sowie die Nutzerakzeptanz der Applikation zu überprüfen. Hilfreich war auch das Feedback bei der Anpassung der Anforderungen an das System; einige Vorschläge konnten bereits umgesetzt werden, andere dienen als Ausblick — beispielsweise die Anbindung an Soziale Netzwerke oder der vermehrte Einsatz von multimedialen Inhalten.

Der Test der Anwendung erfolgt zur Zeit größtenteils manuell. Lediglich die implementierten Unit-Tests für Client- und Server-Applikation können automatisiert ausgeführt werden. Um eine gute Testabdeckung für den gesamten Entwicklungszeitraum zu gewährleisten, ist eine Automatisierng der Tests anzustreben. Eine gute Erweiterung der bisherigen Testdurchführung wäre die kontinuierliche Integration, welche den Entwickler schnell auf Integrationsprobleme und fehlgeschlagene Unit-Tests aufmerksam macht. Für die Umsetzung eines solchen Systems stehen Software-Lösungen wie beispielsweise Jenkins [Jenkins, 2012] oder Hudson [Hudson, 2012] zur Verfügung.

8. Evaluation

In diesem Kapitel werden das Konzept aus Kapitel 5 und die prototypische Realisierung des Dienstes aus Kapitel 6 mit den aufgestellten funktionalen, technischen und den nicht-funktionalen Anforderungen aus dem Kapitel 4 verglichen und evaluiert. Hierzu wird ein kritischer Vergleich der Anforderungen an den Dienst mit der Konzeption sowie der Umsetzung der einzelnen Module durchgeführt. Weiterhin werden im Abschnitt 8.3 mögliche Erweiterungen und Ergänzungen für das System vorgestellt. Abschließend werden in Abschnitt 8.4 Vorschläge und Ansätze für eine verbesserte Aggregation der Daten, sowie eine automatische Qualitätssicherung der erzeugten Daten durch die Nutzer aufgezeigt.

8.1. Abgleich der Anforderungen mit Konzeption und Realisierung

Dieser Abschnitt dient dem Abgleich der gestellten funktionalen, technischen und nicht-funktionalen Anforderungen aus Kapitel 4 mit den Konzeptionen und den erfolgten Realisierungen. Fehlende oder nicht vollständig umgesetzte Anforderungen werden so identifiziert und nachfolgend diskutiert.

8.1.1. Funktionale Anforderungen

In Tabelle 8.1 werden die funktionalen Anforderungen an das System aufgezeigt. Ihr Bedeutung während der Konzeptions- sowie der Realisierungsphase wird betrachtet.

Die in der Analyse 4 erarbeiteten Anwendungsfälle und die dazugehörigen funktionalen Anforderungen an das System wurden — bis auf wenige Ausnahmen — erfüllt. Sowohl in der Konzeption als auch in der prototypischen Realisierung des Dienstes wurden die geforderten Funktionalitäten beachtet und umgesetzt. Der Anwendungsfall „Nutzerprofil ansehen“ wurde in der Realisierung zwar grundlegend implementiert, allerdings fehlen hier noch die geplanten Funktionalitäten für die Berechnung der erreichten Punkte, sowie das automatische, systemseitige Anlegen von Erfolgen für den Nutzer. Weiterhin ist es innerhalb des Prototypen nicht möglich,

Anforderungen	Anwendungsfall	Konzept	Realisierung
FA-01 — FA-08	Nutzer registrieren	√	√
FA-09 — FA-11	Nutzer anmelden	√	√
FA-12 — FA-18	Orte suchen	√	√
FA-19 — FA-32	Ort hinzufügen	√	√
FA-33 — FA-38	Nutzerprofil ansehen	√	(√)
FA-39 — FA-43	Ort betrachten	√	√
FA-44 — FA-46	Ort favorisieren	√	√
FA-47 — FA-49	Ort bewerten	√	√
FA-50 — FA-52	Ort besuchen	√	√
FA-53 — FA-55	Kommentar hinzufügen	√	√
FA-56 — FA-58	Zu Ort führen	√	√
FA-59 — FA-61	Kartenansicht betrachten	√	√
FA-62 — FA-65	Tour durchführen	(√)	-
FA-66 — FA-74	Tour anlegen	(√)	-

Tabelle 8.1.: Konzept und Realisierung: funktionale Anforderungen

Nutzerprofile anderer Nutzer einzusehen. Um soziale Interaktion zu fördern, sollte eine entsprechende Implementierung noch erfolgen. Um den Einsatz von Gamification-Elementen einfacher zu gestalten, wird für das System eine sinnvolle Struktur zur Erweiterung der möglichen Erfolge benötigt. Der Bereich der Touren wurde innerhalb der Konzeption nur theoretisch betrachtet. Eine praktische Umsetzung war für den Prototypen zeitlich nicht möglich. Aufgrund des modularen Aufbaus der Applikation ist das Einbinden von Touren allerdings leicht durchführbar.

8.1.2. Technische Anforderungen

Nachfolgend werden die in Abschnitt 4.3.2 aufgestellten technischen Anforderungen mit der Konzeption und Realisierung des Systems in Tabelle 8.2 abgeglichen.

Anforderungen	Beschreibung	Konzept	Realisierung
TA-01	Positionierungsmodul vorhanden	√	√
TA-02	Android mindestens Version 2.2	√	√
TA-03	Datenverbindung vorhanden	√	√
TA-04	Kameramodul integriert	√	√
TA-05	Persistenzschicht vorhanden	√	√
TA-06	Zwischenspeicherung der Daten	√	(√)

Tabelle 8.2.: Abgleich der technischen Anforderungen mit Konzeption und Realisierung

Wie Tabelle 8.2 zu entnehmen ist, wurden alle technischen Anforderungen bei der Konzeption und Umsetzung des Dienstes beachtet. Lediglich die Anforderung TA-06 (Zwischenspeicherung der Daten), konnte nicht vollständig implementiert werden. So wurde — unter Beachtung dieser Anforderung — ein Queueing realisiert, welches bei fehlender oder schlechter Datenverbindung die zu versendenen Daten zwischenspeichert. Um eine Reduktion des Datenverkehrs zu erreichen, wäre zudem ein Caching der bereits geladenen Daten nötig. Zur Identifizierung von nicht aktuellen Daten wäre der Einsatz einer Prüfsumme erforderlich. Erst wenn eine Inkonsistenz der Daten zwischen Server und Client festgestellt wird, wäre ein erneutes Laden von Nöten.

8.1.3. Nicht-funktionale Anforderungen

Abschließend werden die nicht-funktionalen Anforderungen der Konzeption mit der Realisierung innerhalb der Applikation abgeglichen:

Anforderungen	Beschreibung	Konzept	Realisierung
NFA-01	Anwendungsfälle und fkt. Anforderungen	√	(√)
NFA-02	Fehlertoleranz und Wiederherstellbarkeit	√	(√)
NFA-03	Anforderung an Reaktionszeiten	√	√
NFA-04	Anforderung an die Positionierung	√	(√)
NFA-05	Wartbarkeit und Objekt-Orientierung	√	√
NFA-06	Einsatz auf Android-Geräten bis Version 2.2	√	√

Tabelle 8.3.: Abgleich der nicht-funktionalen Anforderungen mit Konzeption und Realisierung

Die nicht-funktionalen Anforderungen, welche mit Bezug auf die ISO/IEC 9126 [for Standardization, 2011] im Abschnitt 4.3.2 festgelegt wurden, sind bei Konzeption und Realisierung der Anwendung zum tragen gekommen. Lediglich die Punkte NFA-01 (Anwendungsfälle und fkt. Anforderungen), NFA-02 (Fehlertoleranz und Wiederherstellbarkeit) und NFA-04 (Anforderung an die Positionierung) wurden nur bedingt umgesetzt. Das heißt nicht, dass ihnen innerhalb der Realisierung keine Bedeutung zukam. Sie können lediglich nicht als vollständig erfüllt angesehen werden. Der erste Punkt ist nur bedingt erfüllt, da nicht alle aufgestellten Anforderungen komplett in Konzeption und Realisierung beachtet werden konnten. Dies ist auf den zeitlich vorgegebenen Rahmen zurückzuführen, welcher die Möglichkeit zur Umsetzung von Modulen eingeschränkt hat.

Die Anforderung an Fehlertoleranz und Wiederherstellbarkeit wurden im Allgemeinen beachtet. Allerdings ist aufgrund des prototypischen Charakters der Anwendung nicht in vollem Umfang gewährleistet, dass diese Anforderungen für alle Module gelten. So wäre es bei einer Erweiterung der Funktionalität notwendig zu prüfen, ob Seiteneffekte auftreten oder ob der aktu-

elle Zustand eines Moduls nach einer unerwarteten Beendigung der Software wiederhergestellt werden kann. Da obig genannte Punkte zu berücksichtigen sind, wurde die Anforderung an Fehlertoleranz und Wiederherstellbarkeit innerhalb der Realisierung nur bedingt umgesetzt.

Zuletzt ist die nicht-funktionale Anforderung NFA-04 zu nennen. Diese wurde in der Realisierng nur bedingt umgesetzt. Die Positionierung des Nutzer erfolgt im ersten Schritt per Funkzellenortung. Dies erfüllt in 90% der Fälle eine Ortung in unter 10 Sekunden. Allerdings ist die Aktualisierung der Positionsdaten durch eine GPS-Ortung im Hintergrund nicht vollständig umgesetzt worden. Bei der Erzeugung eines neuen Ortes werden Positionsdaten aus einer GPS-Ortung verwendet, welche durchgeführt wird, sobald der Nutzer die „Ort hinzufügen"-Seite öffnet. Auch wird mittels GPS lokalisiert, wenn die Karten-Anwendung aufgerufen wird. Eine allgemeine Verbesserung der Lokalisierung im Hintergund der Anwendung ist allerdings noch nicht umgesetzt.

8.1.4. Anforderungen an Skalierbarkeit und Performance

Die Anforderungen an Skalierbarkeit und Performance der Applikation werden, abgesehen der Anforderung NFA-03, welche konkrete Anforderungen an die Reaktionszeiten bei der Verwendung stellt, nicht weiter beachtet. Dies liegt im prototypischen Charakter der Anwendung begründet. Die Evaluierung der Ziele bezüglich Skalierbarkeit und Performance, im Falle eines „Live"-Betriebs der Software, müsste durch Erprobungen und Messungen der Anwendung erfolgen. Besonders der Bereich Skalierbarkeit ist bei der hier entwickelten Applikation nicht wichtig, da die Anwendung lediglich durch einige wenige Personen verwendet wurde. Bei der Durchführung eines umfangreichen Beta-Tests, bei dem eine Vielzahl von Nutzer die Anwendung erproben können, müssten die Server-Systeme entsprechend ausgelegt werden und bezüglich ihrerer Erweiterbarkeit und Skalierbarkeit bewertet werden.

8.2. Evaluierung des Crowdsourcing-Konzepts

Die Evaluierung des Meta-Konzepts dieser Arbeit — also das Crowdsourcing von Reiseinformationen — kann im Vergleich zu den aufgestellten Anforderungen nicht anhand eines Abgleichs erfolgen. Um die Tragfähigkeit des Konzepts zu überprüfen, müsste die Applikation von einer Vielzahl von realen Nutzern verwendet werden. Einerseits könnte anhand eines Nutzertests die Akzeptanz von und das Interesse an einem solchen System evaluiert werden, andererseits könnten die bis dato definierten Anforderungen optimiert und ergänzt werden.

Um Faktoren wie Nutzbarkeit und Akzeptanz optimal auswerten zu können, ist die Menge an erzeugten Inhalten interessant. Wären innerhalb eines festgelegten Probezeitraums viele Datensät-

ze generiert worden, und wäre eine rege Beteiligung der Nutzer am System erkennbar, so wären dies gute Indikatoren für eine hohe Nutzerakzeptanz und die Funktionsfähigkeit des Entwurfs. Zum anderen wäre es von Vorteil, die Nutzerinteraktionen mit der Applikation auszuwerten und diese Daten für den Entwurf von Optimierungen und Erweiterungen heranzuziehen. Um dies zu erreichen, könnten Fragebögen oder Software-Module genutzt werden, welche zum einen die Anliegen der Nutzer schriftlich aufzeigen und zum anderen die Verwendung der Software analysieren.

Neben der Konzeptevaluation wären weitere Nutzertests sinnvoll. Dazu gehört beispielsweise die Durchführung eines Usability-Tests. Aufbau, Konzept und Nutzerinterface könnten dadurch gestestet und Fehler oder schwer verständliche Bestandteile entfernt werden.

8.3. Erweiterungen

Die in dieser Arbeit durchgeführte prototypische Umsetzung des Systems ermöglicht eine Verwendung der Kernfunktionalitäten, welche anhand der „Use-Cases“ in Abschnitt 4.3.1 herausgearbeitet wurden. Nutzer können die Software in einem realen Kontext verwenden. Die bis dato entwickelte Architektur und die Umsetzung der Anforderungen können sich jedoch erst durch konkrete Nutzung bewähren. So muss bei einer Fortführung der Entwicklung eine Anpassung der Software an die Bedürfnisse der Nutzer erfolgen. Weiterhin gibt es eine Vielzahl von möglichen Erweiterungen, welche im Folgenden beispielhaft aufgezählt und erläutert werden:

Selbstverwaltung / Qualitätssicherung

Die durch die Nutzer generierten Inhalte zu interessanten Orten und Sehenswürdigkeiten sind in der aktuellen Version des Systems frei wählbar und können nicht überarbeitet werden. Dies bringt die Gefahr mitsich, dass Nutzer die Plattform für die ungewollte Verbreitung von beispielsweise Werbung oder anstößigen Inhalten missbrauchen. Die Sicherstellung der Einhaltung von Richtlinien bezüglich der zugelassenen Inhalte würde einen massiven Verwaltungsaufwand bedeuten. Damit ein solcher Aufwand verhindert wird, existieren verschiedene Methoden zur Regulierung. Um Inhalte zu entfernen, welche nicht den Richtlinien entsprechen, ist es sinnvoll, eine Möglichkeit zu bieten, solche Inhalte melden zu können. Wird ein Eintrag mehrfach als ungeeignet markiert, kann dieser automatisch durch das System entfernt oder zumindest ausgeblendet werden.

Ein weiterer wichtiger Punkt ist die Aktualität der Einträge. Es kann nicht davon ausgegangen werden, dass ursprüngliche Erzeuger eines Eintrags diesen auch weiterhin pflegen. Es würde deswegen Sinn machen, die Editierung von Einträgen zuzulassen. Nutzer könnten

so bestehende Einträge anpassen oder neuere Bilder hinzufügen. Wird ein Eintrag editiert, sollte eine Historie erstellt werden, in welcher ältere Versionen, sowie die Namen der Nutzer, welche Einträge verändert haben, gespeichert werden.

Erweiterung um Multimediaihalte

Neben der Einbindung von Bildern könnte das System um weitere multimediale Aspekte erweitert werden. Hierzu gehört beispielsweise die bereits erwähnte Integration von Audiotouren, welche vornehmlich bei der Nutzung der Touren-Funktionalität sinnvoll wären. Auch Video-Ansichten für Orte würden Text und Bilder gut ergänzen. Zu beachten ist bei der Erweiterung um solche Aspekte vor allem die Netzwerkbandbreite, die dem Nutzer zur Verfügung steht.

Da die hier entwickelte Anwendung vornehmlich im mobilen Kontext verwendet werden soll, muss das Streaming von Video- und Audio-Inhalten unter Berücksichtigung des aktuellen Kontexts des Nutzer geschehen. So sollte bei geringer Bandbreite nicht automatisch umfangreiches Videomaterial vom Server geladen werden. Vorstellbar wäre auch, dass Nutzer Video-Inhalte schon im Voraus während der Reiseplanung herunterladen. Diese Inhalte wären im Anschluss auf dem Gerät hinterlegt und könnten dann abgerufen werden, wenn sie benötigt werden. Insbesondere im Multimedia-Bereich besteht noch sehr viel Spielraum für Erweiterungen, mit denen Nutzern ein umfangreiches und angenehmes Erlebnis geboten werden kann.

Anbindung an Enzyklopädien

Um die durch die Nutzer erstellten Inhalte zu ergänzen, wäre eine Anbindung an bestehende Online-Dienste wie etwa Wikipedia denkbar. Zum einen könnten Inhalte von *GeoBee* durch Inhalte auf Wikipedia ergänzt werden, zum anderen könnte dies auch andersherum gelten. Denkbar wäre auch eine Verknüpfung mit Artikeln aus Online-Diensten, um weiterführende Informationen zu bieten — etwa durch die Verknüpfung passender Beiträge von Online-Magazinen oder Nachrichten-Plattformen mit Orten einer Stadt.

Integration von sozialen Netzwerken

Integriert man soziale Netzwerke in GeoBee, können Nutzer ihre Lieblingsplätze oder Interessen mit Freunden auf anderen Online-Plattformen teilen. Zum einen würde so der soziale Aspekt der Applikation weiter gestärkt, zum anderen erhöht eine Verknüpfung zwischen Interessen und Empfehlungen von Freunden die Partizipation am System. Auch der spielerische Aspekt der Software könnte so um soziale Elemente erweitert werden. Die Veröffentlichung von erreichten Zielen oder Medaillen im Kontext von sozialen Netzwerken

könnte die Verbreitung der Software erhöhen und die Optionen für ergänzende spielerische Elemente erweitern.

Die verfügbaren Erweiterungen sind sehr umfangreich, und nicht alle Möglichkeiten können hier aufgezählt werden. Die Umsetzung der entwickelten Software bietet allerdings die vorerst angedachten Funktionalitäten.

8.4. Datenaggregation und -analyse

Die server-seitige Verarbeitung der vorligenden Daten beschränkt sich zum aktuellen Zeitpunkt auf die Erzeugung von Inhalten. So können neue Orte hinzugefügt, Bewertungen abgegeben und Kommentare erstellt werden, um nur einige Beispiele zu nennen. Durch die prototypische Implementierung der Heatmaps können Nutzer diese Daten in einer abstrahierten Übersicht einfach einsehen und bestimmte Gebiete als interessant einordnen. Besonders dieser Bereich bietet allerdings noch großes Potenzial bezüglich möglicher Aggregation und der Analyse vorliegender Daten. Eine optimierte Aggregation — etwa durch die Bildung von verbesserten Clustern — würde zu einer noch besseren Kategorisierung der einzelnen Gebiete führen. Reisende könnten dadurch ihre Interessen und Vorlieben einfacher mit dem System abgleichen. Die Analyse der Daten könnte bei der Umsetzung eines verbesserten Empfehlungssystems hilfreich sein. Der Einsatz von kollaborativen Filter-Algorithmen wie bei [Delev u. a., 2010] ist sinnvoll, um qualitative Empfehlungen für andere Nutzer zu erstellen. Die Umsetzung solcher Funktionalitäten müsste allerdings immer unter Berücksichtigung des Datenschutzes erfolgen. Die Verarbeitung der Daten müsste also anonymisiert erfolgen.

9. Zusammenfassung

Im Rahmen dieser Arbeit wurde ein mobiler Online-Dienst auf der Basis des Crowdsourcings konzipiert und realisiert, welcher Nutzern den Austausch über interessante Stätten verschiedenster Städte bietet. Auch wurden Gamification-Elemente eingesetzt, um die Verwendung der Software für Nutzer attraktiver zu gestalten.

Ein Überblick über die Bestandteile des Systems wurde zu Beginn in Kapitel 2 gegeben. Erklärt wurden dort etwa Themenfelder wie Kollektive Intelligenz, Crowdsourcing und Gamification. Nachfolgend wurde ein Einblick in das Gebiet des mobilen Computing gegeben. Hierzu gehörten erstens Ausführungen zu Rechenkapazität und Speicherauslastung sowie zu Besonderheiten bezüglich Mobilität, Akkulaufzeiten und Netzwerk-Verfügbarkeit. Zweitens wurde eine kurze Einführung in die Android-Plattform gegeben und die Architektur und der Aufbau von Android-Anwendungen erklärt.

In Kapitel 3 wurde ein konkretes Szenario für den Einsatz der Software aufgezeigt. Zusätzlich wurden die Komponenten diskutiert, welche für die Umsetzung eines standortbezogenen Dienstes notwendig sind. Kapitel 4 wurde genutzt, um artverwandte Anwendungen vorzustellen und mit der hier entwickelten Applikation zu vergleichen. Bezogen auf das Anwendungsszenario wurden in diesem Kapitel auch allgemeine Anforderungen an einen standortbezogenen Dienst abgeleitet; anschließend wurden Anwendungsfälle extrahiert und beschrieben. Diese dienten dann zur Spezifizierung von funktionalen, technischen und nicht-funktionalen Anforderungen. Diese Anforderungen dienen im Folgenden zur Konzeption und der Realisierung des Systems.

Die Konzeption wurde in Kapitel 5 vorgestellt. Hierzu gehörten die fachliche Architektur, das Zusammenspiel der einzelnen fachlichen Komponenten und die technische Architektur der Anwendung. Auch wurde das mathematische Konzept für die Lokalisierung des Nutzers vorgestellt. Die Gestaltung der Nutzerschnittstelle wurde anhand von Interface-Skizzen diskutiert und die beinhalteten Komponenten im Detail vorgestellt.

Das Kapitel 6 zur Realisierung gab einen Überblick über die Entwicklung der Anwendung sowie über die verwendeten Module wie *Open Street Map*. Parallel zur Realisierung der Applikation wurden die in Kapitel 7 erläuterten Tests durchgeführt, welche zu einer iterativen Optimierung der Software genutzt wurden. Auch konnten in Feldversuchen mit realen Nutzern

wichtige Erkenntnisse und Feedback gesammelt werden, das direkt in die Weiterentwicklung der Anforderungen eingegangen ist. Vorschläge für nicht umgesetzte Erweiterungen werden im Kapitel 8 zur Evaluation aufgegriffen und dienen als Ausblick für die Weiterentwicklung. Auch wurden die definierten Anforderungen innerhalb der Evaluation mit der tatsächlichen Realisierung abgeglichen.

GeoBee bietet den Nutzern die Kern-Funktionalitäten, welche für den Applikations-Einsatz als mobile Citymap notwendig sind. Die identifizierten Anwendungsfälle wurden, bis auf wenige Ausnahmen, umgesetzt und von einigen Nutzern verwendet. Die Tragfähigkeit des Crowdsourcing-Konzepts sowie die Akzeptanz durch die Nutzer muss allerdings durch einen umfangreicheren Nutzertest validiert werden.

A. Anwendungsfälle (Tabellen)

Name	**Nutzer registrieren**
Kurzbeschreibung	Der Nutzer legt ein neues Konto im System an.
Verwendete Anwendungsfälle	—
Akteure	Nutzer
Vorbedingung	Die Anwendung wurde vom Nutzer gestartet.
Ergebnis	Ein neues Benutzerkonto wurde erstellt.
Nachbedingung	Der Nutzer ist am System angemeldet und gelangt auf das Dashboard der Applikation.
Standard-Ablaufschritte	1. Der Nutzer klickt auf dem Login-Bildschirm auf „Register“. 2. Der Nutzer gibt seine E-Mail Adresse, einen Benutzernamen und ein Passwort ein. 3. Der Nutzer wählt den Button „Register“.
Alternative Ablaufschritte	1. Der Nutzer erhält die Meldung, dass sein Benutzername schon vergeben ist. 2. Der Nutzer wählt auf der Registrierungsseite einen neuen Benutzernamen. 3. Der Nutzer wählt den Button „Register“.

Tabelle A.1.: Anwendungsfallbeschreibung „Nutzer registrieren“

Name	**Nutzer anmelden**
Kurzbeschreibung	Der Nutzer meldet sich beim System an.
Verwendete Anwendungsfälle	—
Akteure	Nutzer
Vorbedingung	Die Anwendung wurde vom Nutzer gestartet.
Ergebnis	Der Nutzer ist im System angemeldet.
Nachbedingung	Der Nutzer gelangt auf das Dashboard der Applikation.
Standard-Ablaufschritte	1. Der Nutzer gibt auf der Login-Seite seinen Benutzernamen ein. 2. Der Nutzer gibt sein Passwort ein. 3. Der Nutzer wählt den Button „Login“.
Alternative Ablaufschritte	1. Der Nutzer kann nicht am System angemeldet werden. 2. Der Nutzer korrigiert seinen Benutzernamen oder sein Passwort. 3. Der Nutzer wählt den Button „Login“.

Tabelle A.2.: Anwendungsfallbeschreibung „Nutzer anmelden“

Name	**Ort suchen**
Kurzbeschreibung	Der Nutzer kann vorhandene Orte im System anhand von Schlagwörtern suchen.
Verwendete Anwendungsfälle	—
Akteure	Nutzer
Vorbedingung	Der Nutzer ist im System eingeloggt.
Ergebnis	Die gefundenen Orte werden dem Nutzer in einer Liste angezeigt.
Nachbedingung	Der Benutzer kann die angezeigten Orte auswählen, um weitere Informationen zu erhalten.
Standard-Ablaufschritte	1. Der Nutzer wählt auf dem Dashboard den Menüpunkt Suchen (Lupen-Symbol). 2. Der Nutzer gibt im Suchfeld den gewünschten Suchbegriff ein. 3. Der Nutzer wählt den Button zum Suchen der Einträge.
Alternative Ablaufschritte	—

Tabelle A.3.: Anwendungsfallbeschreibung „Ort suchen“

Name	**Nutzerprofil ansehen**
Kurzbeschreibung	Der Nutzer kann sein eigenes Profil einsehen.
Verwendete Anwendungsfälle	—
Akteure	Nutzer
Vorbedingung	Der Nutzer ist im System eingeloggt.
Ergebnis	Das Profil inkl. der Statistiken und der erstellten Orte werden angezeigt.
Nachbedingung	Der Benutzer kann die angezeigten Orte auswählen, um weitere Informationen zu erhalten.
Standard-Ablaufschritte	1. Der Nutzer wählt auf dem Dashboard den Menüpunkt Profil (Personen-Symbol).
Alternative Ablaufschritte	—

Tabelle A.4.: Anwendungsfallbeschreibung „Nutzerprofil ansehen“

Name	**Ort betrachten**
Kurzbeschreibung	Der Nutzer kann die Detailansicht eines Ortes einsehen.
Verwendete Anwendungsfälle	—
Akteure	Nutzer
Vorbedingung	Der Nutzer ist im System eingeloggt, und eine Liste von Orten ist vorhanden.
Ergebnis	Die Detailansicht des ausgewählten Ortes wird angezeigt.
Nachbedingung	—
Standard-Ablaufschritte	1. Der Nutzer wählt auf dem Dashboard den Menüpunkt Listenansicht (Listen-Symbol). 2. Der Nutzer wählt aus den aufgelisteten Orten einen Ort aus, um diesen im Detail zu betrachten.
Alternative Ablaufschritte	1. Der Nutzer wählt einen Ort aus, der das Ergebnis einer Suche war. 2. Der Nutzer wählt einen Ort aus, welcher in seinem Nutzerprofil angezeigt wird.

Tabelle A.5.: Anwendungsfallbeschreibung „Ort betrachten“

Name	**Ort favorisieren**
Kurzbeschreibung	Der Nutzer kann einen Ort zu seinen Favoriten hinzufügen.
Verwendete Anwendungsfälle	Ort betrachten
Akteure	Nutzer
Vorbedingung	Der Nutzer hat einen Ort für die Detailansicht ausgewählt.
Ergebnis	Der Ort ist als Favorit des Nutzers gespeichert.
Nachbedingung	Der Nutzer kann mit der Betrachtung der Detailansicht fortfahren. Das Lesezeichen-Symbol ist rot eingefärbt.
Standard-Ablaufschritte	1. Der Nutzer wählt den Button „Favorisieren“ in der Detailansicht (Lesezeichen-Symbol).
Alternative Ablaufschritte	—

Tabelle A.6.: Anwendungsfallbeschreibung „Ort favorisieren“

Name	**Ort bewerten**
Kurzbeschreibung	Der Nutzer kann einen Ort als „sehenswert“ markieren.
Verwendete Anwendungsfälle	Ort betrachten
Akteure	Nutzer
Vorbedingung	Der Nutzer hat einen Ort für die Detailansicht ausgewählt.
Ergebnis	Der Ort wurde durch den Nutzer als „sehenswert“ markiert.
Nachbedingung	Der Nutzer kann mit der Betrachtung der Detailansicht fortfahren. Das Stern-Symbol ist gelb eingefärbt.
Standard-Ablaufschritte	1. Der Nutzer wählt den Button „Bewerten“ in der Detailansicht (Stern-Symbol).
Alternative Ablaufschritte	—

Tabelle A.7.: Anwendungsfallbeschreibung „Ort bewerten“

Name	**Kommentar hinzufügen**
Kurzbeschreibung	Der Nutzer kann für einen Ort einen Kommentar hinzufügen.
Verwendete Anwendungsfälle	Ort betrachten
Akteure	Nutzer
Vorbedingung	Der Nutzer hat einen Ort für die Detailansicht ausgewählt.
Ergebnis	Der Kommentar wurde an den entsprechenden Ort angefügt.
Nachbedingung	Der Nutzer kann mit der Betrachtung der Detailansicht fortfahren.
Standard-Ablaufschritte	1. Der Nutzer gibt einen Kommentar in das entsprechende Feld in der Detailansicht ein. 2. Der Nutzer wählt den Button „Kommentar einfügen“ in der Detailansicht (+-Symbol).
Alternative Ablaufschritte	—

Tabelle A.8.: Anwendungsfallbeschreibung „Kommentar hinzufügen“

Name	**Zu Ort führen**
Kurzbeschreibung	Der Nutzer wird durch die Anwendung zu einem Zielort navigiert.
Verwendete Anwendungsfälle	Ort betrachten
Akteure	Nutzer
Vorbedingung	Der Nutzer hat einen Ort für die Detailansicht ausgewählt.
Ergebnis	Der Nutzer ist am entsprechenden Zielort angekommen.
Nachbedingung	Der Nutzer kann Details zu diesem Ort in der Anwendung einsehen.
Standard-Ablaufschritte	1. Der Nutzer wählt das „Kompass“-Symbol aus. 2. Der Nutzer erhält eine Ansicht, in welcher ein Kompass die Laufrichtung anzeigt, die restliche Entfernung zum Ziel wird dem Nutzer dargestellt.
Alternative Ablaufschritte	—

Tabelle A.9.: Anwendungsfallbeschreibung „Zu Ort führen“

Name	**Kartenansicht betrachten**
Kurzbeschreibung	Der Nutzer kann seine Position und die Orte in der Umgebung auf einer Karte einsehen.
Verwendete Anwendungsfälle	Ort betrachten
Akteure	Nutzer
Vorbedingung	Der Nutzer hat einen Ort für die Detailansicht ausgewählt.
Ergebnis	Die Position des Nutzers und der ausgewählte Ort werden auf einer Karte angezeigt.
Nachbedingung	Der Nutzer kann innerhalb der Karte navigieren und Zoomen.
Standard-Ablaufschritte	1. Der Nutzer wählt das „Karten“-Symbol aus. 2. Der Nutzer erhält eine Ansicht, in welcher eigener Standort und der Zielort durch Markierungen aufgezeigt werden.
Alternative Ablaufschritte	—

Tabelle A.10.: Anwendungsfallbeschreibung „Kartenansicht betrachten“

Name	**Touren anlegen**
Kurzbeschreibung	Der Nutzer kann eine neue Tour im System einfügen.
Verwendete Anwendungsfälle	—
Akteure	Nutzer
Vorbedingung	Der Nutzer ist am System angemeldet
Ergebnis	Eine neue Tour wurde im System hinzugefügt.
Nachbedingung	Die neu angelegte Tour kann von allen Nutzern eingesehen und verwendet werden.
Standard-Ablaufschritte	1. Der Nutzer wählt auf dem Dashboard das Touren-Symbol aus. 2. Der Nutzer wählt im Kontext-Menü den Menüpunkt „Tour anlegen“. 3. Der Nutzer vergibt einen Namen für die Tour. 4. Der Nutzer sucht über die Eingabe von Städtenamen in einem Suchfeld nach Orten, die er hinzufügen möchte. 5. Der Nutzer wählt aus den Ergebnissen den passenden Ort aus. 6. Der Nutzer wählt zum Abschluss den Button „Tour erstellen“.
Alternative Ablaufschritte	1. Der Nutzer wählt weitere Orte mittels der Suche nach Städten aus.

Tabelle A.11.: Anwendungsfallbeschreibung „Touren anlegen“

Name	**Touren durchführen**
Kurzbeschreibung	Der Nutzer läuft eine Tour von Orten ab.
Verwendete Anwendungsfälle	—
Akteure	Nutzer
Vorbedingung	Der Nutzer ist am System angemeldet
Ergebnis	Dem Nutzer wird die aktuelle Tour angezeigt.
Nachbedingung	Der Nutzer kann die einzelnen Stationen der Tour ablaufen.
Standard-Ablaufschritte	1. Der Nutzer wählt auf dem Dashboard das Touren-Symbol aus. 2. Der Nutzer wählt aus einer Liste von Touren in der Umgebung eine Tour aus. 3. Der Nutzer läuft die Stationen der Tour nacheinander ab.
Alternative Ablaufschritte	—

Tabelle A.12.: Anwendungsfallbeschreibung „Touren durchführen“

B. Funktionale Anforderungen (GeoBee)

Nutzer anmelden

FA-09 [Nachdem der Anwender die Applikation gestartet hat], soll die Applikation die Loginseite anzeigen.

FA-10 [Hat der Nutzer seinen Benutzernamen und sein Passwort eingegeben und den „Anmelden"-Button geklickt], werden die Daten an den Web-Dienst gesendet.

FA-11 [Wurden die Daten vom Server empfangen], werden diese auf Korrektheit überprüft.

FA-12 [Ist die Authentifizierung erfolgreich oder schlägt fehl], versendet die Server-Anwendung eine Erfolgs- oder Fehlermeldung an den Client.

FA-13 [Hat der Client die Antwort vom Server erhalten], gelangt der Nutzer auf das Dashboard der Applikation oder ihm wird eine Fehlermeldung angezeigt, und er muss den Login-Prozess erneut durchführen.

Orte suchen

FA-14 [Hat der Nutzer, ausgehend vom Dashboard, das Lupen-Symbol geklickt], soll die Applikation die Suchseite anzeigen.

FA-15 [Hat der Nutzer einen Suchbegriff eingegeben und den „Suchen"-Button geklickt], wird der Suchbegriff an den Web-Dienst übertragen.

FA-16 [Wurden die Daten vom Server empfangen], wird eine Suche in den vorhandenen Orten in der Datenbank durchgeführt.

FA-17 [Ist die Suche beendet], versendet die Server-Anwendung alle gefundenen Einträge an den Client.

FA-18 [Hat der Client die Antwort vom Server erhalten], werden alle gefundenen Orte in einer Liste angezeigt.

FA-19 [Wählt der Nutzer einen der Orte durch anklicken aus], gelangt der Nutzer zur Detailansicht für diesen Ort.

FA-20 [Verwendet der Nutzer die systemseitige „Zurück"-Taste], gelangt er wieder zur Liste der Suchergebnisse.

Ort hinzufügen

FA-21 [Hat der Nutzer, ausgehend vom Dashboard, das „Plus"-Symbol geklickt], soll die Applikation die Seite zum Hinzufügen eines Ortes anzeigen.

FA-22 [Hat der Nutzer den Button für die Kamera-Applikation geklickt], öffnet das System die Kamera-Anwendung.

FA-23 [Hat der Nutzer mit der Kamera-Applikation ein Foto aufgenommen und „Fertig" geklickt], gelangt er zurück zur Seite „Ort hinzufügen" und das aufgenommene Bild wird klein angezeigt.

FA-24 [Hat der Nutzer den Button für die Gallery-Anwendung geklickt], öffnet das System die Gallery-Anwendung.

FA-25 [Hat der Nutzer in der Gallery ein Foto gewählt], gelangt er zurück zur Seite „Ort hinzufügen" und das gewählte Bild wird klein angezeigt.

FA-26 [Ist das Foto ausgewählt], kann der Nutzer einen Namen und eine Beschreibung für den zu erstellenden Ort angeben.

FA-27 [Hat der Nutzer auf die Taste „Tag this location" geklickt], werden alle verfügbaren Kategorien und zugehörige Schlagwörter vom Server an den Client übertragen.

FA-28 [Sind passende Schlagwörter durch den Nutzer aus der Liste gewählt worden und hat er die Taste „Return" geklickt], gelangt der Nutzer zur „Ort hinzufügen"-Seite, und die gewählten Schlagwörter werden in einer Liste dargestellt. Weiterhin ist der „Tag this location"-Button ausgeblendet.

FA-29 [Hat der Nutzer auf die Taste „add location" geklickt], wird geprüft, ob eine Datenverbindung existiert und ob diese vom Typ WIFI ist.

FA-30 [Ist keine Datenverbindung vorhanden oder die Bandbreite zu gering], werden die Daten zwischengespeichert, und dem Nutzer wird eine entsprechende Meldung angezeigt. Die

Daten werden bei der nächsten Verfügbarkeit einer WIFI-Verbindung an den Server gesendet.

FA-31 [Ist eine ausreichende Verbindung verfügbar], werden die Daten an den Web-Dienst übertragen. Zusätzlich zu den eingegeben Daten kommen Geo-Koordinaten und Adress-Informationen hinzu. Dem Nutzer wird eine Warte-Animation angezeigt.

FA-32 [Hat die Server-Anwendung die Daten empfangen], wird ein neuer Ort in der Datenbank angelegt und das zugehörige Bild auf dem Server hinterlegt.

FA-33 [Ist das Bild abgespeichert und der Ort im System angelegt], versendet die Server-Applikationen eine Erfolgsmeldung an den Client.

FA-34 [Hat der Client die Antwort des Servers empfangen], wird die Warte-Animation beendet. Der Nutzer erhält stattdessen eine Meldung vom System, dass der neue Ort angelegt wurde und gelangt automatisch zurück zum Dashboard.

Nutzerprofil ansehen

FA-35 [Hat der Nutzer, ausgehend vom Dashboard, das Personen-Symbol geklickt], werden die Statistiken des Nutzers von der Server-Applikation abgefragt. Der Nutzer sieht eine Warte-Animation.

FA-36 [Erhält der Web-Dienst die Anfrage], werden die aktuellen Statistiken des Nutzers aggregiert. Hierzu gehören Anzahl der Punkte, Anzahl der erstellen Orte und erreichte Medaillen.

FA-37 [Liegen alle Informationen zum Nutzer vor], versendet der Server diese an die Client-Anwendung.

FA-38 [Wurde die Antwort des Servers vom Client empfangen], öffnet die Anwendung die Seite für das Nutzerprofil.

FA-39 [Wurde die Profilseite geöffnet], kann der Nutzer seine Statistiken einsehen. Hierzu gehören Anzahl der Punkte, Anzahl der angelegten Orte, Anzahl der Medaillen. Es werden dem Nutzer seine erreichten Medaillen grafisch als Icons angezeigt. Weiterhin werden alle von ihm erzeugten Orte in einer Liste angezeigt.

FA-40 [Hat der Nutzer einen seiner angelegten Orte angeklickt], bekommt er die Detailseite dieses Ortes angezeigt.

Ort betrachten

FA-41 [Hat der Nutzer die Detailseite eines Ortes ausgewählt], wird die Detailansicht-Seite von der Anwendung geöffnet. Dem Nutzer wird eine Warte-Animation angezeigt.

FA-42 [Sind die erforderlichen Daten für die Anzeige nicht vorhanden], werden diese vom Server-Dienst abgefragt.

FA-43 [Hat der Server die erforderlichen Daten von der Datenbank geladen], werden diese an die Client-Applikation gesendet.

FA-44 [Hat der Client die angefragten Daten erhalten], wird die Entfernung vom angezeigten zum aktuellen Standort des Nutzer berechnet.

FA-45 [Ist die Vorverarbeitung der Daten abgeschlossen], werden diese auf der Detailseite dem Nutzer angezeigt.

Ort favorisieren

FA-46 [Hat der Nutzer auf der Detailansicht das „Lesezeichen"-Symbol geklickt], wird eine Anfrage an den Server-Dienst gestellt.

FA-47 [Hat der Server die Anfrage erhalten], wird versucht, einen neuen Lesezeichen-Eintrag für den Ort und den aktuellen Nutzer anzulegen und eine Erfolgs- oder Fehlermeldung an den Client gesendet.

FA-48 [Hat der Client die Antwort des Servers erhalten], wird im Erfolgsfall das „Lesezeichen"-Symbol rot eingefärbt. Im Fehlerfall erhält der Nutzer eine Fehlermeldung.

Ort bewerten

FA-49 [Hat der Nutzer auf der Detailansicht das „Stern"-Symbol geklickt], wird eine Anfrage an den Server-Dienst gestellt.

FA-50 [Hat der Server die Anfrage erhalten], wird versucht, eine neue Bewertung für den Ort anzulegen und eine Erfolgs- oder Fehlermeldung an den Client gesendet.

FA-51 [Hat der Client die Antwort des Servers erhalten], wird im Erfolgsfall das „Stern"-Symbol gelb eingefärbt, und der Nutzer erhält eine Erfolgsmeldung. Im Fehlerfall erhält der Nutzer eine Fehlermeldung.

Ort besuchen

FA-52 [Hat der Nutzer auf der Detailansicht das „Markierungs“-Symbol geklickt], wird eine Anfrage an den Server-Dienst gestellt.

FA-53 [Hat der Server die Anfrage erhalten], wird versucht, einen neuen Besucher-Eintrag für den Ort und den aktuellen Benutzer anzulegen und eine Erfolgs- oder Fehlermeldung an den Client gesendet.

FA-54 [Hat der Client die Antwort des Servers erhalten], wird im Erfolgsfall das „Markierungs“-Symbol rot eingefärbt, und der Nutzer erhält eine Erfolgsmeldung. Im Fehlerfall erhält der Nutzer eine Fehlermeldung.

Kommentar hinzufügen

FA-55 [Hat der Nutzer auf der Detailansicht einen Eintrag in das Kommentar-Feld eingegeben und auf den „+“-Button geklickt], wird eine Anfrage an den Server-Dienst gestellt. Dem Nutzer wird eine Warte-Animation angezeigt.

FA-56 [Hat der Server die Anfrage erhalten], wird ein neuer Kommentar für den aktuellen Ort und den aktuellen Benutzer hinzugefügt. Anschließend wird eine Erfolgs- oder Fehlermeldung an den Client gesendet.

FA-57 [Hat der Client die Antwort des Servers erhalten], wird im Erfolgsfall der Kommentar im Bereich „Comments“ hinzugefügt. Im Fehlerfall erhält der Nutzer eine Fehlermeldung.

Zu Ort führen

FA-58 [Hat der Nutzer auf der Detailansicht auf das „Kompass“-Symbol geklickt], wird ihm danach die Kompass-Seite angezeigt.

FA-59 [Ist der Nutzer auf dem Weg zum Zielort], aktualisiert sich sowohl die aktuelle Zielrichtung des Kompasses als auch die verbleibende Entfernung zum aktuellen Zielort.

FA-60 [Ist der Nutzer am Zielort angekommen], beträgt die Entfernung 0 Meter. Der Nutzer kann über den systemseitigen „Zurück“-Button zum Dashboard der Applikation gelangen.

Kartenansicht betrachten

FA-61 [Hat der Nutzer auf der Detailansicht auf das „Karten“-Symbol geklickt], wird ihm danach die Kartenansicht der Applikation angezeigt.

FA-62 [Ist die Kartenansicht geladen], wird dem Nutzer auf der Karte eine Markierung für den aktuellen Ort, den er betrachtet hat, angezeigt. Zusätzlich ist eine Markierung (Personen-Symbol) vorhanden, welche die aktuelle Position des Nutzers anzeigt.

FA-63 [Hat der Nutzer auf eine Orts-Markierung geklickt], wird eine Hinweismeldung angezeigt, welche den Namen des geklickten Ortes anzeigt.

Tour durchführen

FA-64 [Hat der Nutzer auf der Detailansicht auf das „Touren"-Symbol geklickt], wird ihm danach die Tourenansicht der Applikation angezeigt. Diese zeigt in Listenansicht alle verfügbaren Touren, welche in einem Umkreis von 10 km erreichbar sind.

FA-65 [Hat der Nutzer eine der angezeigten Touren per Klick ausgewählt], wird ihm eine Ansicht angezeigt, in welcher ein Kompass mit der Richtung zum nächsten Ziel zu sehen ist. Zusätzlich werden Entfernung und Name des nächsten Zielortes angezeigt.

FA-66 [Ist der Nutzer in der Nähe eines Zielortes angekommen], wird eine akustische Meldung vom System ausgegeben. Die Seite zur Tour zeigt nun die Detailseite des aktuellen Ortes an.

FA-67 [Hat der Nutzer auf den Button „Tour fortführen" geklickt], wird wieder die Seite aus **FA-63** angezeigt, diesmal mit Informationen zum nächsten Zielort.

Tour anlegen

FA-68 [Hat der Nutzer auf der Detailansicht auf das „Touren"-Symbol geklickt], wird ihm danach die Tourenansicht der Applikation angezeigt. Diese zeigt in Listenansicht alle verfügbaren Touren, welche in einem Umkreis von 10 km erreichbar sind.

FA-69 [Hat der Nutzer im Android-Kontextmenü den Punkt Tour anlegen gewählt], wird ihm ein Suchfeld angezeigt. Zusätzlich werden ein Feld zur Benennung der Tour und die aktuelle Anzahl der enthaltenen Orte angezeigt.

FA-70 [Hat der Nutzer im Suchfeld eine Stadt eingegeben und auf Suchen geklickt], wird eine Anfrage an die Server-Applikation gestellt. Dem Nutzer wird eine Warte-Animation angezeigt.

FA-71 [Hat der Server die Anfrage erhalten], werden alle in der gesuchten Stadt vorhandenen Orte aus der Datenbank abgefragt und an die Client-Anwendung gesendet.

FA-72 [Hat der Client die Antwort erhalten], werden dem Nutzer die Orte dieser Stadt in einer Liste angezeigt.

FA-73 [Hat der Nutzer einen dieser Orte in der Liste ausgewählt], wird dieser der Tour hinzugefügt. Die Anzahl der Orte der Tour erhöht sich um eins.

FA-74 [Hat der Nutzer den Button „Tour erstellen“ geklickt], wir eine Anfrage mit der Liste der Orte und dem Namen der Tour an den Server gesendet.

FA-75 [Hat der Server die Anfrage erhalten], wird im System eine neue Tour angelegt, welche mit den enthaltenen Orten verknüpft wird. Der Server versendet eine Erfolgs- oder Fehlermeldung an den Client.

FA-76 [Hat der Client die Antwort des Servers erhalten], wird dem Nutzer eine Erfolgs- oder Fehlermeldung angezeigt.

C. Mailverkehr zu Google Places

Hallo Torben,
hier alle Infos, die wir zu Google Places herausgeben dürfen.

- **Local information is extremely important to both users and businesses.**
  - **More than 20% of searches on Google are related to location**
    - Meanwhile, we've significantly improved our ability to give users relevant local information and to help business owners connect with potential customers
    - We offer detailed information, directions, reviews and more for millions of local queries like shops, restaurants, parks and landmarks right on Google.com and on Google Maps
    - Place Search provides a richer local experience, and offers 30-40 diverse sources of information on a single results page. It saves people an average of two seconds
  - **Google helps local businesses expand their reach and build a better web in the process**
- **Geo and local are showing real momentum**
  - **User momentum**:
    - Google Maps was born on the web, but today more than 50% of global Google Maps usage is mobile
    - More than 10M people use Google Maps Navigation, driving more than 12B miles a year with this tool. With the **route around traffic feature**, users save two years per day
    - Users have contributed 5M ratings and reviews to **Google Places,** and the rating widget available on Androids and iPhones enables users to share their opinions on the go
  - **Business momentum:**
    - There are 50M **Place pages** worldwide. More than 8M business owners have claimed their free listings in Google Places to ensure that users have access to up-to-date information about their businesses, and increase the quality and accuracy of our local search results
    - **Google AdWords Express** is a fast and easy way to participate in the paid ad auction. We create the ad campaign using basic information the business has already inputted in Google Places
    - Through collaborations with a limited number of partners, we're experimenting with showing sponsored **hotel prices** alongside Google Maps search results and on hotel Place pages.
- **Geo+Local+Mobile is a winning combination**
  - **Check-ins via Google Latitude** enable users to share the places they go with friends and add context to the locations they visit
- **We're opening up the world to millions and millions of people**
  - Google is building a digital mirror of the world that is open, dynamic, easy to search and designed for sharing--whether you're at your computer, on your mobile or in the car
  - Maps are no longer static images, but living and breathing reflections of the world around us
  - Maps are personal and interactive: Users can create custom guides with **My Maps**, share photos with **Panoramio**, contribute to the map with **Map Maker**, make 3D models with **SketchUp** and **Building Maker**, update listings and data with **Google Maps**, and share ratings and recommendations with friends to get personalized suggestions with **Google Places**
  - **Art Project** and **Street View special collections** organize photographs by location and enables users to pivot around places
    - Over 40M My Maps have been created by users
    - Over 20M photos have been uploaded to Panoramio
    - Google Map Maker is available in more than 180 countries/territories
    - Our users make tens of thousands of corrections to local data and map data per day
    - 5M user contributions (ratings and reviews) have been made via Google Places
    - The integration of **Panoramio** and **Latitude** enables users to automatically geotag their photos to their time and location history, and to view those photographs in **Google Earth**

D. Beigelegte CD

Textform
Die vorliegende Masterarbeit als PDF-Dokument.

Server-Anwendung
Die Quelltexte der Server-Applikation inkl. der umgesetzten Unit-Tests.

Client-Anwendung
Das gesamte Android-Projekt für die Client-Applikation. Beinhaltet alle Quelltexte, Layout-Dateien und Grafiken.

Datenbankdump
SQL-Dumb der verwendeten Datenbank.

Client-Test-Anwendung
Das Test-Projekt für die Client-Applikation.

Plakat
Das Plakat für die Präsentation der Arbeit bei der Nacht des Wissens und dem Alumni-Treffen.

Literaturverzeichnis

[von Ahn und Dabbish 2008] Ahn, L. von ; Dabbish, L: Designing Games With a Purpose. In: *Comm. ACM*, 2008, S. 58–67

[von Ahn 2009] Ahn, Luis von: Human computation. In: *Proceedings of the 46th Annual Design Automation Conference*. New York, NY, USA : ACM, 2009 (DAC '09), S. 418–419. – URL http://doi.acm.org/10.1145/1629911.1630023. – ISBN 978-1-60558-497-3

[open handset alliance 2011] alliance open handset: *Members*. Website. 2011. – http://www.openhandsetalliance.com/oha_members.html; aufgerufen am 21. November 2011

[Aristoteles 2003] Aristoteles: *Politik*. o. J. N.-A.. Nachdr. 2003. DTV Deutscher Taschenbuch, 2003. – ISBN 3423301376

[Bao u. a. 2009] Bao, Xiehao ; Bie, Hongxia ; Wang, Mingxiao: Integration of multimedia and location-based services on mobile phone tour guide system. In: *Network Infrastructure and Digital Content, 2009. IC-NIDC 2009. IEEE International Conference on*, nov. 2009, S. 642 –646

[Becker und Pant 2010] Becker, Arno ; Pant, Marcus: *Android 2 - Grundlagen und Programmierung*. 2. aktualisierte und erweiterte Auflage. Heidelberg : Dpunkt.Verlag GmbH, 2010. – ISBN 3898646777

[Beni und Wang 1989] Beni, Gerardo ; Wang, Jing: Swarm Intelligence in Cellular Robotic Systems. In: *Proceedings of NATO Advanced Workshop on Robots and Biological Systems* Bd. 102, 1989

[Blog 2012] Blog, Nokia M.: *Heat Maps: Explore the city and see where the action is!* Website. 2012. – http://blog.maps.nokia.com/archive/heat-maps-explore-the-city-and-see-where-the-action-is; aufgerufen am 21. Januar 2012

[Carroll und Heiser 2010] Carroll, Aaron ; Heiser, Gernot: An analysis of power consumption in a smartphone. In: *Proceedings of the 2010 USENIX conference on USENIX annual technical*

conference. Berkeley, CA, USA : USENIX Association, 2010 (USENIXATC'10), S. 21–21. – URL http://dl.acm.org/citation.cfm?id=1855840.1855861

[Cooper u. a. 2010] Cooper, Seth ; Treuille, Adrien ; Barbero, Janos ; Leaver-Fay, Andrew ; Tuite, Kathleen ; Khatib, Firas ; Snyder, Alex C. ; Beenen, Michael ; Salesin, David ; Baker, David ; Popović, Zoran: The challenge of designing scientific discovery games. In: *Proceedings of the Fifth International Conference on the Foundations of Digital Games*. New York, NY, USA : ACM, 2010 (FDG '10), S. 40–47. – URL http://doi.acm.org/10.1145/1822348.1822354. – ISBN 978-1-60558-937-4

[Delev u. a. 2010] Delev, T. ; Gjorgjevik, D. ; Madzarov, G.: Place-Tags, discovering and promoting places through mobile phones and collaborative filtering. In: *Information Technology Interfaces (ITI), 2010 32nd International Conference on*, june 2010, S. 225 –230. – ISSN 1330-1012

[Deterding u. a. 2011] Deterding, Sebastian ; Khaled, Rilla ; Nacke, Lennart E. ; Dan, Dixon: Gamification: Toward a Definition, ACM, 2011 (CHI 2011)

[Deutschland 2012] Deutschland, OpenStreetMap: *Fragen und Antworten*. Website. 2012. – http://openstreetmap.de/faq.html#was_ist_osm; aufgerufen am 12. Januar 2012

[Developers 2011a] Developers, Android: *Application Fundamentals*. Website. 2011. – http://developer.android.com/guide/topics/fundamentals.html; aufgerufen am 14. November 2011

[Developers 2011b] Developers, Android: *What is Android?* Website. 2011. – http://developer.android.com/guide/basics/what-is-android.html; aufgerufen am 07. November 2011

[Developers 2012a] Developers, Android: *Obtaining User Location*. Website. 2012. – http://developer.android.com/guide/topics/location/obtaining-user-location.html; aufgerufen am 14. Januar 2012

[Developers 2012b] Developers, Android: *Reference - ActivityManager*. Website. 2012. – http://developer.android.com/reference/android/app/ActivityManager.html#getMemoryClass(); aufgerufen am 07. Januar 2012

[Developers 2012c] Developers, Android: *Sensor Event*. Website. 2012. – http://developer.android.com/reference/android/hardware/SensorEvent.html; aufgerufen am 17. Januar 2012

[Developers 2012d] Developers, Android: *Testing Fundamentals*. Website. 2012. – http://developer.android.com/guide/topics/testing/testing_android.html; aufgerufen am 05. Februar 2012

[Dunkel u. a. 2008] Dunkel, J. ; Eberhart, A. ; Fischer, S. ; Kleiner, C. ; Koschel, A.: *Systemarchitekturen für Verteilte Anwendungen: Client-Server, Multi-Tier, SOA, Event Driven Architectures, P2P, Grid, Web 2.0*. Hanser Fachbuchverlag, 2008. – URL http://books.google.de/books?id=601izhXEesIC. – ISBN 9783446413214

[Eberhart und Kennedy 2001] Eberhart, Russell C. ; Kennedy, James: *Swarm intelligence*. 340 Pine Street, San Francisco, CA : Morgan Kaufmann, 2001. – ISBN 1558605959

[Falk 2012] Falk: *Falk Guide*. Website. 2012. – http://www.falk-navigation.de/de/mobile-apps/falk-guide/applikation.html; aufgerufen am 25. Januar 2012

[Fielding 2000] Fielding, Roy T.: *Architectural Styles and the Design of Network-based Software Architectures*, Information and Computer Science - UNIVERSITY OF CALIFORNIA, IRVINE, Doctor of Philosophy, 2000

[Freeman u. a. 2004] Freeman, Eric ; Freeman, Elisabeth ; Bates, Bert ; Sierra, Kathy: *Head First design patterns*. 1. Aufl. Sebastopol, CA : O'Reilly Media, Inc., 2004. – ISBN 0596007124

[Gartner.com 2010] Gartner.com: *Gartner Says Worldwide Mobile Device Sales to End Users Reached 1.6 Billion Units in 2010; Smartphone Sales Grew 72 Percent in 2010*. Website. 2010. – http://www.gartner.com/it/page.jsp?id=1543014; aufgerufen am 14. November 2011

[thinkinsights with google 2011] google thinkinsights with: *The Mobile Movement - Understanding Smartphone Users*. Presentation. 2011. – http://www.thinkwithgoogle.com/insights/library/studies/the-mobile-movement/; aufgerufen am 15. November 2011

[Howe 2006] Howe, Jeff: *The Rise of Crowdsourcing*. Website. 2006. – http://www.wired.com/wired/archive/14.06/crowds.html; aufgerufen am 08. November 2011

[HTC 2012] HTC: *Specs - HTC Wildfire S*. Website. 2012. – http://www.htc.com/www/smartphones/htc-wildfire-s/#specs; aufgerufen am 07. Januar 2012

[Hudson 2012] Hudson: *Extensible continuous integration server*. Website. 2012. – http://hudson-ci.org/; aufgerufen am 14. Februar 2012

[Jenkins 2012] Jenkins: *An extendable open source continuous integration server*. Website. 2012. – http://jenkins-ci.org/; aufgerufen am 14. Februar 2012

[Kecher 2011] Kecher, Christoph: *UML 2.3 - Das umfassende Handbuch*. 4. Aufl. Bonn : Galileo Press, 2011. – ISBN 383621752X

[Krannich 2010] Krannich, Dennis: *Mobile Usability-Testing - Ein toolbasiertes Vorgehensmodell zum Rapid-Prototyping und Usability-Testing von Mobilen Systemen im originären Benutzungskontext.*, Fachbereich 3 (Mathematik und Informatik) - Universität Bremen, Doktorgrades der Ingenieurwissenschaften, 2010

[Kruchten 2004] Kruchten, Philippe: *The rational unified process - an introduction*. 3. A. Boston : Addison-Wesley Professional, 2004. – ISBN 0321197704

[Kurose und Ross 2008] Kurose, J.F. ; Ross, K.W.: *Computernetzwerke: Der Top-Down-Ansatz*. München : Pearson Studium, 2008 (Pearson Studium). – ISBN 9783827373304

[Maps 2012] Maps, Nokia: *Kartenanwendung mit Heatmapvisualisierung*. Website. 2012. – http://maps.nokia.com; aufgerufen am 21. Januar 2012

[McGonigal 2011] McGonigal, Jane: *Reality is Broken - Why games make us better and how they can change the World*. Random House, 20 Vauxhall Bridge Road, London SW1V 2SA : Jonathan Cape, 2011. – ISBN 0224089250

[McZusatz 2011] McZusatz: *MobileBitRate*. Website. 2011. – http://upload.wikimedia.org/wikipedia/commons/4/42/MobileBitRate-logScale.svg; aufgerufen am 05. Dezember 2011

[Mechanical-Turk 2011] Mechanical-Turk, Amazon: *Startpage*. Website. 2011. – https://www.mturk.com/mturk/welcome; aufgerufen am 15. November 2011

[Meier 2010] Meier, Reto: *Professional Android 2 Application Development*. 2. Auflage. Indianapolis, Indiana : John Wiley and Sons, 2010. – ISBN 0470565527

[Noack 2011] Noack, David: *Google Places für die Masterarbeit*. Emailverkehr. 2011. – Emailverkehr mit Google Account Manager - Retail David Noack

[OGC 2008] OGC, Open Geospatial C.: *OpenGIS® Location Services (OpenLS): Core Services - Version 1.2*. OpenGIS® Project Document. 2008. – http://www.opengeospatial.org/standards/ols; aufgerufen am 26. November 2011

[osmdroid 2012] OSMDROID: *Summary - OpenStreetMap-Tool for Android*. Website. 2012. – http://code.google.com/p/osmdroid/; aufgerufen am 12. Januar 2012

[Planet 2012a] PLANET, Lonely: *Compass City Guide Apps for Android*. Website. 2012. – http://www.lonelyplanet.com/apps-and-ebooks/android/index.php; aufgerufen am 25. Januar 2012

[Planet 2012b] PLANET, Lonely: *iTunes In-App Sales*. Website. 2012. – http://itunes.apple.com/de/app/lonely-planet-travel-guides/id317165182?mt=8; aufgerufen am 25. Januar 2012

[Polo 2012] POLO, Marco: *MARCO POLO CityGuide meets iPhone*. Website. 2012. – http://www.marcopolo.de/magazin/artikel/cityguide-meets-iphone; aufgerufen am 25. Januar 2012

[Rheingold 2003] RHEINGOLD, Howard: *Smart mobs - the next social revolution*. Reprint. 387 Park Avenue South New York, NY 10016 : Basic Books, 2003. – ISBN 0738208612

[Robotium 2012] ROBOTIUM: *User scenario testing for Android*. Website. 2012. – http://code.google.com/p/robotium/; aufgerufen am 05. Februar 2012

[Ross u. a. 2010] ROSS, Joel ; IRANI, Lilly ; SILBERMAN, M. S. ; ZALDIVAR, Andrew ; TOMLINSON, Bill: Who are the crowdworkers?: shifting demographics in mechanical turk. In: *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*. New York, NY, USA : ACM, 2010 (CHI EA '10), S. 2863–2872. – URL http://doi.acm.org/10.1145/1753846.1753873. – ISBN 978-1-60558-930-5

[Roth 2005] ROTH, Jörg: *Mobile Computing - Grundlagen, Technik, Konzepte*. 2. aktualis. A. Heidelberg : Dpunkt-Verl., 2005. – ISBN 3898643662

[SimpleTest 2012] SIMPLETEST: *Unit Testing for PHP*. Website. 2012. – http://simpletest.org/index.html; aufgerufen am 03. Februar 2012

[Spillner u. a. 2011] SPILLNER, Andreas ; LINZ, Tilo ; SCHAEFER, Hans: *Software Testing Foundations - A Study Guide for the Certified Tester Exam*. 3. Aufl. 26 West Mission Street Ste 3; Santa Barbara, CA 93101 : Rocky Nook, 2011. – ISBN 1933952784

[for Standardization 2011] STANDARDIZATION, (International O. for: *ISO/IEC 9126*. Norm. 2011

[Steiniger u. a. 2006] STEINIGER, Stefan ; NEUN, Moritz ; EDWARDES, Alistair: *Foundations of Location Based Services*. Lecture Notes on LBS. 2006. – http://www.spatial.cs.umn.

edu/Courses/Fall07/8715/papers/IM7_steiniger.pdf; aufgerufen am 24. November 2011

[Surowiecki 2005] Surowiecki, James: *Die Weisheit der Vielen - warum Gruppen klüger sind als Einzelne und wie wir das kollektive Wissen für unser wirtschaftliches, soziales und politisches Handeln nützen können.* Sonderausgabe. München : C. Bertelsmann, 2005. – ISBN 3570006875

[Weiser 1991] Weiser, Mark: *The Computer for the 21st Century.* Website. 1991. – http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html; aufgerufen am 26. Januar 2012

[Wied 2012] Wied, Patrick: *OpenLayers Heatmap Overlay.* Website. 2012. – http://www.patrick-wied.at/static/heatmapjs/demo/maps_heatmap_layer/openlayers.php; aufgerufen am 21. Januar 2012

[Wikipedia 2012] Wikipedia: *Law-of-haversines.* Website. 2012. – http://en.wikipedia.org/wiki/File:Law-of-haversines.svg; aufgerufen am 16. Januar 2012

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 19. Februar 2012 Torben Wallbaum, B.Sc.