ulm university universität
uulm

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Datenbanken und Informationssysteme

Analyse von ortsabhängigen, mobilen Anwendungsszenarien im Bereich der Gastronomie am Beispiel einer Android-Anwendung

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Alpay Artun
alpay.artun@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Marc Schickler

2015

Fassung 28. April 2015

Satz: PDF-LaTeX 2ε

Kurzfassung

Der mit dem Bluetooth 4.0 Standard eingeführte Low-Energy Standard ermöglicht nicht nur einen energiearmen Gebrauch von Bluetooth-Funktionalitäten, sondern ebnete auch den Weg für sogenannte Beacons, die letztendlich dank Apple zu einem großen Bekanntheitsgrad gelangen konnten. Das sind kleine Sender, die als Signalgeber platziert werden und auf dem Sender-Empfänger-Prinzip basieren. Sie senden in festen Zeitintervallen Signale, die beispielsweise von Smartphones empfangen werden können. Nichtsdestrotz finden sich in zahlreichen Branchen kaum bekannte Anwendungen, die sinnvollen Gebrauch der neuen Möglichkeiten machen. In dieser Arbeit werden die kleinen Beacons genutzt, um Anwendungsszenarien im Bereich der Gastronomie zu erörtern und schlussendlich anhand einer mobilen Anwendung auf Basis des Android-Betriebssystems in die Praxis umzusetzen. Die genutzte Technologie ermöglicht eine Ortung von Smartphones in geschlossenen Räumen, wo sonst das bisher genutzte GPS-Signal wirkungslos war. Zur Erweiterung der Möglichkeiten finden Geofences in dieser Arbeit Verwendung, um im Zusammenspiel mit einem Server nicht nur interessante Einsatzmöglichkeiten aktueller Technologien für den Gebrauch einer mobilen Anwendung in der Gastronomie zu demonstrieren, sondern vor allem, um einen positiven Mehrzweck für Gäste und Branchen-Mitarbeiter zu schaffen.

Danksagung

Ich bedanke mich bei meinen Freunden, die während meines Studiums und insbesondere während meiner Bachelorarbeit nicht nur geduldig waren und meine Abwesenheit tapfer hingenommen haben, sondern mich auch stets ermutigt und motiviert haben.

Ich bedanke mich bei Marc Schickler vom Institut für Datenbanken- und Informationssysteme für die Betreuung während meiner Arbeit an diesem Schriftstück.

Ich danke Philipp Spira für die Genehmigung der Verwendung der Markenbilder des „Choclet“.

Und natürlich danke ich meiner Mutter, die mir alles erst ermöglicht hat.

Inhaltsverzeichnis

1 Einleitung

Das Interesse war groß, als Apple auf der Worldwide Developers Conference (kurz: WWDC) in ihrer Präsentation über Bluetooth Low-Energy (kurz: BLE) und ihre iBeacons gesprochen hat. Obwohl die technischen Gegebenheiten schon im Jahr zuvor vorhanden waren, erhoffte sich die Branche einen frischen Impuls für mögliche Anwendungsszenarien mit Smartphones und dem „Internet der Dinge“ [9]. Und obwohl diese Präsentation im Juni des Jahres 2014 stattfand, kann von einer Omnipräsenz der genannten, neuen technologischen Möglichkeiten heute nicht die Rede sein.

Offensichtlich ist die Idee der neuen Möglichkeiten, wie der nun möglichen Ortung eines Smartphones innerhalb eines geschlossenen Raumes („Indoor-Ortung“), noch nicht in alle Bereiche vorgedrungen. Eine Recherche zeigt beispielsweise, dass es für die Gastronomie zwar einige wenige mobile Anwendungen gibt, allerdings nur eine, die besagte Indoor-Ortung umsetzt, und auch nur für iOS-basierte Smartphones. Eine solche Anwendung für Android-Geräte gibt es nicht.

Dabei würde gerade die Gastronomie von frischen Ideen profitieren, weil sich dort zwar in der Art der Dienstleistungen zwar kaum etwas ändert, der Trend aber zu mehr Restaurant-Ketten mit immer größeren Gasträumen und dadurch ein potenziell höheres Gästeaufkommen geht, was letzten Endes eine größere Arbeitsbelastung für Mitarbeiter bedeutet. Anschauliche Beispiele hierfür seien Starbucks, Coffee Fellows, L'Osteria, Vapiano, Cafe Extrablatt oder Henry's Coffee World.

Wie sollen diese und natürlich auch kleinere Betriebe in einem hart umkämpften Markt sowohl ausreichende Umsätze erzielen, als auch die Gast-Erfahrung verbessern und interne Arbeitsabläufe optimieren?

1.1 Ziel der Arbeit

Angesichts besagter Problemstellung soll es Ziel dieser Arbeit sein, Anwendungsszenarien zu finden und zu erörtern, die im Kontext einer mobilen, ortsabhängigen Android-Anwendung [16] zur Verbesserung und Optimierung des Gast-Erlebnisses und der internen Arbeitsabläufe in der Gastronomie beiträgt. Im Idealfall trägt dies positiv zur Verbesserung der Arbeitsbelastung von Mitarbeitern in der Gastronomie bei.

In diesem Sinne sollen technische Möglichkeiten ausgeschöpft werden, um sinnvolle Anwendungsmöglichkeiten anzubieten, die innerhalb und außerhalb von geschlossenen Räumen ge-

nutzt werden können. Plattform für eine solche mobile Anwendung sollen Android Smartphones sein, da es gerade hierfür ein klares Defizit für interessante Anwendungen gibt, obwohl Android hierzulande seit Jahren stabil den größten Marktanteil hält, wie folgende Abbildung 1.1 zeigt.

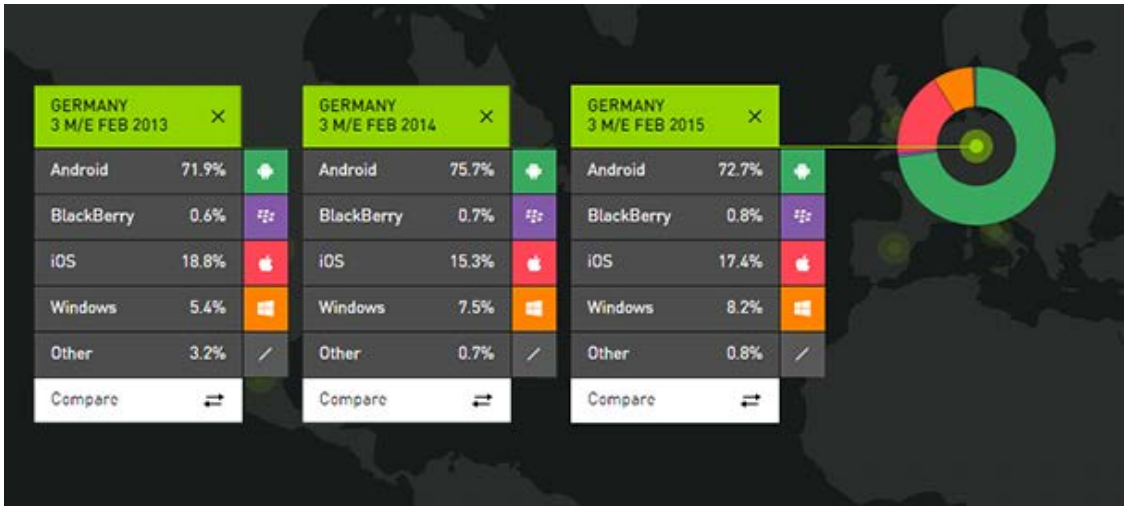

Abbildung 1.1: Marktanteile von 2013 bis 2015 [5].

Diese mobile Anwendung soll nach Fertigstellung sowohl Gästen als auch Mitarbeitern in der Gastronomie nützlich sein und als eine willkommene Unterstützung und Erweiterungen bisheriger Gastronomie-Konzepte dienen.

1.2 Aufbau der Arbeit

Entsprechend dieser Gegebenheiten wird diese Arbeit im folgenden Kapitel 2, *Anwendungsszenarien*, zunächst potenzielle Anwendungsszenarien besprechen und bewerten. Daraus resultiert der Funktionsumfang der mobilen Anwendung in Kapitel 3, *Anforderungsanalyse*, welcher an dortiger Stelle zusammenfassend erläutert wird.

Kapitel 4, *Aufbau*, soll einen groben Überblick über die Funktionsweise der Server-Komponente und der mobilen Anwendung verschaffen, bevor alle konkreten Interaktionsmöglichkeiten im Kapitel 5, *Anwendungsfälle*, erläutert werden.

Der technische Aspekt wird in Kapitel 6, *Architektur und Implementierung*, ausführlich besprochen, wobei Server und mobile Anwendung separat voneinander behandelt werden.

Weiterhin findet sich in Kapitel 7, *Anforderungsabgleich*, ein Abgleich über theoretische und in die Praxis umgesetzte Anforderungen, bevor im Kapitel 8, *Diskussion*, einige kontroverse Themen diskutiert werden.

Zuletzt führt das Kapitel 9, *Schlussgedanken*, mit einem Überblick zu den Herausforderungen während der Arbeit, einem Ausblick und einem persönlichen Fazit diese Arbeit zu einem Abschluss.

2 Anwendungsszenarien

Als technische Voraussetzungen oder Mittel sind gegeben:

- Bluetooth Beacons (kurz: *Beacons*), die auf Basis von Bluetooth Low Energy aus dem Blueooth 4.0 Standard ein Signal senden, um prinzipiell Auskunft über sich selbst zu erteilen.
- Android Smartphones, die mindestens kompatibel zu *Bluetooth Low Energy* (kurz: BLE) sind.
- *Geofences*, die digital einen – normalerweise kreisförmigen – Bereich einer realen Umgebung abstecken und damit ein Gebiet definieren [11]. In dieses kann jemand mittels Smartphone eintreten („enter“), darin verweilen („dwell“) oder es verlassen („exit“) (siehe Abbildung 2.1).

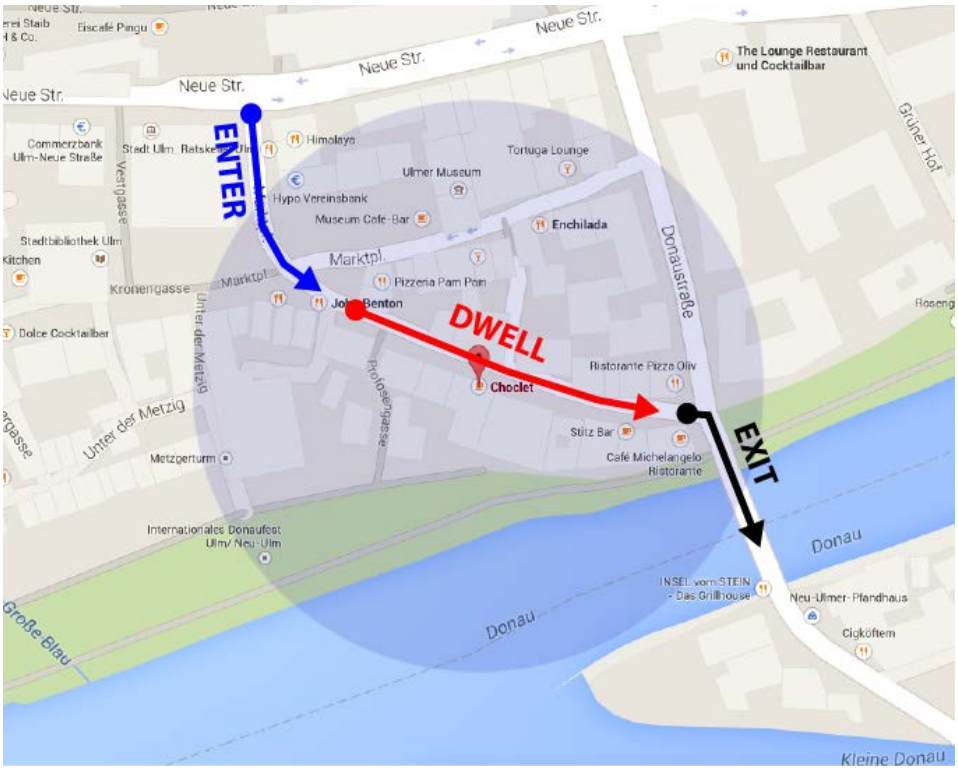

Abbildung 2.1: Ein Geofence und mögliche, eigene Zustände.

Im Folgenden werden zunächst mögliche Anwendungsszenarien im Kontext der gegebenen Prämisse aus dem vorherigen Kapitel untersucht und deren Vor- und Nachteile gegenübergestellt,

bevor ein Fazit gezogen wird, welches über eine technische Umsetzung entscheidet. Diese theoretische Auseinandersetzung soll die Grundlage für eine darauffolgende Programmierung als Teil einer mobilen Anwendung werden [15]. In Hinblick auf sinnvolle Anwendungsszenarien soll kein Gebrauch der NFC-Technologie gemacht werden, weil diese eine aktive Interaktion des Benutzers mit einem NFC-Tag, nämlich die Berührung mit dem Smartphone, erforderlich machen würde. Im Sinne einer positiven Gast-Erfahrung kommen daher nur anfangs genannte Technologien in Frage, da Anwendungen mit diesen im Hintergrund ablaufen können, also ohne die zusätzliche Interaktion des Benutzers.

2.1 Digitale Menükarte

Ein häufig zu sehendes Problem von Gastronomiebetrieben sind Menükarten, die nicht in digitaler Version vorliegen und daher auf Internetseiten lediglich als PDF-Datei heruntergeladen werden können, mangels befriedigender, ganzheitlicher Lösungen für verschiedene Plattformen, die darüber hinaus bezahlbar sind. Eine sowohl für Gäste als auch Gastronomen vorteilhafte Anwendung könnte an diesem Problem ansetzen und ein vollständig modulares System zur Digitalisierung von Menükarten anbieten, welche anhand ordentlich konzipierter Schnittstellen für verschiedene Plattformen, wie Smartphones oder Websites, verfügbar gemacht wird. Dabei soll es möglich sein, nahezu jede Art von Getränke- und Speisekarten abzubilden.

Idee

Zunächst kann ein solches System auf einem Server eingerichtet werden, wobei mit Hilfe einer Website die Bearbeitung der Menükarte ermöglicht werden soll. Mit einer entsprechenden Schnittstelle kann eine plattformunabhängige Verwendung sichergestellt werden, sodass sowohl Internetseiten als auch mobile Anwendungen Zugriff auf die online zur Verfügung gestellte Menükarte haben. Ein Administrator, beispielsweise ein Geschäftsführer oder Filialleiter, soll in der Lage sein, Einträge zu erstellen, zu bearbeiten oder zu löschen. Ein System für die Versionierung soll zudem Sorge tragen, dass auf mobilen Anwendungen stets die neueste Version angezeigt wird. Die Menükarte soll demnach lokal auf den Smartphones gespeichert bleiben und nur im Fall einer notwendigen Aktualisierung verändert werden, um die Netzwerkverbindung des Smartphones möglichst wenig zu belasten.

Vor- und Nachteile

Der Vorteil für Gastronomen liegt darin, dass sie sofortige Änderungen an der Menükarte zumindest auf digitalen Plattformen vornehmen können, wohingegen bei gedruckten Varianten starke, durch Kosten verursachte Einschränkungen vorliegen. Eine Server-Schnittstelle für verschiedene Plattformen gewährleistet darüber hinaus, dass plattformübergreifend stets die gleichen Inhalte angezeigt werden. Gäste haben mit Hilfe eines solchen Systems den einfachen Zugang

zu dieser Menükarte und sind dadurch stets auf dem aktuellsten Stand der Getränke- und Speisenauswahl, anders als man es bei gedruckten Menükarten gewohnt ist.

Fazit

Eine solche Anwendung erscheint äußerst hilfreich und nützlich, daher sollte ein derartiges System etabliert werden. Auch um möglicherweise weitere Anwendungen auf einer solchen Menükarte basierend umsetzen zu können.

2.2 Echtzeit-Anzeige der Tischbelegung

Eine aus Sicht des Gastes offensichtlich sehr nützliche Funktion wäre, wenn die mobile Anwendung anzeigt, ob es noch freie Tische gibt. So lässt sich von vornherein Zeit sparen, da im Falle eines voll belegten Lokals der Gast nicht erst den Gastraum zum Nachsehen betreten muss.

Idee

Eine praktische Umsetzung dieser Idee kann mit zweierlei Ansätzen erfolgen. Die erste, automatisierte Möglichkeite wäre es, anhand der Indoor-Ortung jeden einzelnen Gast zu erfassen und dadurch auf freie Tische zu schließen. Ein Server erhält diese Informationen und leitet sie an die mobile Anwendung weiter, wodurch potenzielle Gäste über die Tischbelegenung informiert werden können, ohne das Lokal betreten zu müssen.

Die andere, manuelle Möglichkeit wäre es, an einer Station im Arbeitsbereich der Mitarbeiter einzelne Tische als frei oder belegt markieren zu lassen. Es liegt also am Mitarbeiter, den Status eines Tisches an einer Art Terminal einzugeben. Von diesem wiederum geht die Information an einen Server, der diese an die mobile Anwendung senden kann.

Vor- und Nachteile

Die Schwierigkeiten beider Konzepte sind offensichtlich. Im ersten Falle ist vorausgesetzt, dass jeder Gast ein Smartphone besitzt, welches erstens ein aktiviertes Bluetooth-Modul besitzt, das zweitens kompatibel zu BLE sein muss und drittens muss der Gast darüber hinaus die mobile Anwendung installiert haben. Erst dann ergibt sich ein zu fast 100% zuverlässiges, lückenloses Ergebnis.

Bei der zweiten Möglichkeit muss zunächst die passende Infrastruktur geschaffen sein, damit Mitarbeiter an einem Terminal den Status ihrer zugeteilten Tische eingeben können. Da bisherige Kassensysteme der Gastronomie als geschlossene System verkauft werden und daher nicht erweiterbar sind, muss eine solche Infrastruktur erst erarbeitet werden. Darüber hinaus müssen mehrere solcher Terminals zugänglich sein, da insbesondere bei hohem Gästeaufkommen

und -durchlauf sonst eine Warteschlange vor nur einem Terminal entstehen könnte. Auch der zusätzliche Aufwand im Allgemeinen führt im Falle einer bereits stressigen Arbeitssituation zu einer kaum zumutbaren Mehrbelastung.

Fazit

Es liegt folglich auf der Hand, dass dieses Anwendungsszenario, trotz interessanter Motivation, entweder, wie im ersten Falle, aufgrund seiner erforderlichen Prämissen realitätsfern oder, wie im zweiten Fall, aufgrund des Kosten- und Nutzenfaktors unbrauchbar erscheint. Außerdem lohnt sich hier durchaus auch der Blick auf den wirtschaftlichen Aspekt, denn mit einer derartigen mobilen Anwendung würden dem Gastronomen auch die Gäste fehlen, die bereit wären, im Eingangsbereich auf einen Tisch zu warten. Dadurch würde der Umsatz leiden, was die mobile Anwendung für Gastronomen vollkommen uninteressant macht.

2.3 Exakte Tisch-Reservierung

Reserviert ein Gast einen Tisch, ist es üblich, dass der Tisch vom Mitarbeiter ausgewählt wird. Eine Verbesserung dieses Vorgangs wäre es, wenn der Gast beispielsweise den Tisch, an dem er sich momentan befindet, für einen anderen Zeitpunkt reservieren könnte.

Idee

Anhand von Beacons soll die mobile Anwendung erfahren können, an welchem Tisch sich der Gast gegenwärtig befindet. So soll bei einem Reservierungsvorgang über die mobile Anwendung also die Auswahlmöglichkeit bestehen, ob der Gast sich den selben Tisch wünscht.

Vor- und Nachteile

In der Tat ist es so, dass sich der Nutzen in Grenzen hält, da mutmaßlich eher in wenigen Fällen der Gast sich den Tisch reservieren möchte, an dem er sich momentan befindet. Der Aufwand kann eher als gering geschätzt werden, da die mobile Anwendung nur eine E-Mail versenden muss, die wie andere Reservierungsanfragen auch von einem Mitarbeiter manuell bearbeitet wird.

Fazit

Aufgrund des geringen Aufwandes kann es lohnenswert sein, dieses Szenario in der Praxis anzuwenden, da der Nutzen vermutlich zwar kaum massentauglich ist, jedoch einige Gäste von dieser Verbesserung profitieren können. Eine praktische Umsetzung ist also erstrebenswert.

2.4 Nachverfolgung bei Sitzplatz-Wechsel

Ein großes Problem, insbesondere bei einem bis auf den letzten Tisch belegten Lokal, ist es oft, dass Gäste ohne Ankündigung den Tisch wechseln. Da der Gastraum des Lokals intern in verschiedene Bereiche unterteilt ist, denen verschiedene Mitarbeiter zugeordnet sind, können Gäste nach einem Wechsel oft nicht schnell genug wiedergefunden werden, um deren Bestellungen auf den neuen Tisch umzubuchen. Bei Hochbetrieb stellt dies für die Mitarbeiter eine zusätzliche Belastung dar und kann leicht den Betriebsablauf stören.

Idee

Wegen der an den Tischen befestigten Beacons kann dieses Problem verringert werden. Dazu soll die mobile Anwendung in der Lage sein, zu erkennen, wenn sich die Tischnummer im Lauf des Aufenthalts ändert. Da jedes Beacon eine Tischnummer repräsentiert und die mobile Anwendung die Tischnummer des Beacons, dessen Signal am stärksten ist, als die seine versteht, kann ein Wechsel technisch nachvollziehbar gemacht werden. In diesem Falle soll die Anwendung eine Mitteilung, beispielsweise an eine Website, senden, die für die Mitarbeiter jederzeit sichtbar ist. So können Mitarbeiter sehen, wenn ein Gast von einem Tisch zu einem anderen wechselt, wodurch sich Zeit und Arbeit sparen lässt.

Vor- und Nachteile

Der Vorteil liegt hierbei auf der Hand. Zeitersparnis und Reduzierung der Belastung sind für Mitarbeiter der Gastronomie von großem Nutzen, wohingegen der Gast nur indirekt profitiert. Der Nachteil allerdings zeigt sich auf der praktischen Seite. Der erzielte, positive Effekt dieser Anwendung kann nur so gut sein wie die Anzahl der Gäste hoch ist, die die mobile Anwendung verwenden.

Fazit

Ist die Infrastruktur für diese Anwendung erstmal vorhanden, kann davon ausgegangen werden, dass der Nutzen in Zukunft größer wird, wenn mehr Gästen der Nutzen der Anwendung näher gebracht wird. Die Einrichtung dieser Infrastruktur erscheint zumindest thereotisch nicht zu groß und sollte im Rahmen dieser Arbeit umgesetzt werden.

2.5 Digitale Bestellung

Steht eine digitale Menükarte zur Verfügung, kann in eine mobile Anwendung auch ein Bestellsystem implementiert werden. Demnach kann der Gast aus den Getränken und Speisen verbindlich wählen und seine Bestellung absenden, ohne dazu einen Mitarbeiter zu benötigen.

Idee

Es müsste zunächst geprüft werden, ob ein solches Vorgehen mit einem bestehenden Kassensystem verbunden werden kann. Andernfalls muss ein zentrales System konzipiert werden, das eingehende Bestellungen entgegennimmt, speichert und abrufbar macht. Da aufgrund von Beacons die Tischnummer für die mobile Anwendung bekannt ist, können Bestellungen Gästen zugeordnet werden. Dies könnte man mit einem mobilen Bezahlsystem, wie beispielsweise Paypal, verbinden, um auch den Bezahlvorgang mobil abwickeln zu können.

Vor- und Nachteile

Der Vorteil liegt natürlich auf der Hand, da Gäste nicht mehr erst auf einen Mitarbeiter warten müssen. Ein Gastronom kann auf diese Weise außerdem Personalkosten sparen. Andererseits bietet ein solches Konzept kaum Möglichkeiten, Empfehlungen einzuholen oder Nachfragen zu tätigen, wenn kein direkter Ansprechpartner gegeben ist. Aus technischer Sicht spricht dagegen, dass eine Zuordnung zu einem Tisch zu 100% korrekt sein muss, was zunächst einmal nicht garantiert werden kann, aufgrund verschiedener Störfaktoren für das Bluetooth-Signal.

Fazit

Das Angebot eines mobilen Bestellvorgangs liegt zwar nahe, dennoch muss bedacht werden, dass eine Umsetzung nicht möglich ist, solange keine Garantie gegeben werden kann, dass Gäste mit Hilfe von Beacons korrekt Tischen zugeordnet werden.

2.6 Bestellhilfe mit Bereitschaftsruf

Insbesondere umfangreiche Menüs können im Fall einer vollständigen Menüzusammenstellung für eine Bestellung zu einer Belastungsprobe des Kurzzeitgedächtnisses des Gastes werden. Die Medienpsychologie lehrt, dass das menschliche Gehirn zu einer kurzzeitigen Informationsspeicherung von etwa sieben Einheiten in der Lage ist [7]. Eine Stütze im Sinne eines digitalen Notizzettels würde dem Benutzer eine erleichternde Hilfe bieten. Zusätzlich kann als weitere Optimierung des gesamten Betriebsablaufs ein Bereitschaftsruf im Hintergrund für den Service-Mitarbeiter die Bestellbereitschaft signalisieren. Dies ist gerade dann praktisch, wenn ein Gast noch in der Menükarte stöbert, obwohl er sich bereits im Klaren ist, was er bestellen möchte, was dem Service-Mitarbeiter dann nicht ersichtlich ist.

Idee

Beim Durchstöbern der digitalen Menükarte in der mobilen Anwendung können mittels Berührung einzelne Produkte markiert werden, die im Hintergrund zwischengespeichert werden. Be-

endet der Gast seine Auswahl, so wird er zu einer Übersicht seiner gewählten Produkte weitergeleitet, die er dann bei der Bestellung nur noch ablesen muss. Gleichzeitig wird eine Benachrichtigung an das Terminal der Mitarbeiter gesendet, wonach der Service-Mitarbeiter über die Bestellbereitschaft des Gastes informiert wird.

Vor- und Nachteile

Der Nutzen eines solchen Anwendungsszenarios ist schnell ersichtlich: Eine Entlastung für den Gast und eine kleine Untersützung für den Service-Mitarbeiter. Ein Nachteil ergibt sich hieraus nicht.

Fazit

Die Umsetzung einer solchen Idee ist sinnvoll und sollte daher in der Praxis getestet werden.

2.7 Benachrichtigungen mit Geofences

Der Benutzer dieser mobilen Anwendung erhält im Fall der räumlichen Nähe zum Lokal kontextsensitive Benachrichtigungen. In einem konkreten Beispiel könnte der Benutzer in der Mittagszeit eine Benachrichtigung über das aktuelle Tagesgericht-Angebot erhalten, wenn er sich innerhalb von 100m im Umkreis des Lokals befindet. Oder abends über die in Kürze anstehende Happy-Hour.

Idee

Zunächst soll es auf einer Administratoren-Website möglich sein, Geofence-Punkte auf einer Karte zu markieren und den Radius zu bestimmen. Dazu soll eine gewünschte Aktion, die auf der mobilen Anwendung ausgelöst werden soll, ausgewählt werden. Theoretisch kann es beliebig viele Geofences geben, allerdings existiert eine technische Beschränkung von maximal 100 Geofences pro Gerät und Benutzer [3], was bei der Implementierung natürlich berücksichtigt werden muss. Benachrichtigungen soll der Benutzer auch dann erhalten können, wenn er die Anwendung noch nicht gestartet hat, sodass er jederzeit eine Benachrichtigung mit kontextsensitivem Inhalt erhalten kann.

Vor- und Nachteile

Der Vorteil für Benutzer liegt darin, auf für sie möglicherweise interessante Angebote hingewiesen zu werden, während die Mitarbeiter der Gastronomie schlussfolgernd von zusätzlichen Gästen profitieren. Andererseits kann es aber auch dazu kommen, dass sich der Benutzer belästigt fühlt, wenn er zu viele nicht relevante Benachrichtigungen erhält. Außerdem muss hier auf

eine ressourcen-schonende Implementierung geachtet werden, da sowohl die Ortungs- als auch die Bluetooth-Funktionen schnell zu einer Belastung für den Energieverbrauch des Geräts werden können. Ein weiterer nicht zu vernachlässigender Nachteil ist der zusätzliche Aufwand bei der Umsetzung, um eine Website für den Administrator zu erstellen, auf der Geofences intuitiv verwaltet werden können.

Fazit

Dieses Anwendungsszenario birgt theoretisch ein interessantes Potenzial, dessen Effektivität aber auch davon abhängt, mit welchen Benachrichtigungen der Administrator Benutzer informieren möchte. Kann hier ein guter Mittelweg gefunden werden, können alle davon profitieren. Die Umsetzung dieses Geofence-Szenarios rundet das Gesamtbild der mobilen Anwendung daher sinnvoll ab.

2.8 Konklusion

Aus dieser Analyse ist also gut zu erkennen, dass nicht jedes, zunächst interessantes Anwendungsszenario aus verschiedenen Gründen tatsächlich umgesetzt werden kann. Die Schlussfolgerung ist nun, dass folgende Anwendungsszenarien aufgegriffen werden sollen, um sie in der Praxis unter realen Bedingungen testen zu können. Die digitale Menükarte soll die Voraussetzung sein für eine Bestellhilfe mit Bereitschaftsruf am Smartphone und der Benachrichtigung über Angebote, wie Tagesgerichte, durch Geofences. Dank der Ortung im geschlossenen Raum durch Beacons soll es außerdem möglich sein, Gastronomie-Mitarbeiter automatisch zu benachrichtigen, wenn ein Gast Platz genommen oder seinen Sitzplatz gewechselt hat. Mit der gleichen Technik soll der Gast außerdem exakt seinen Tisch für einen späteren Besuch reservieren können.

3 Anforderungsanalyse

Nachdem nun verschiedene Anwendungsszenarien besprochen sind und nun klar ist, welche davon als Teil der mobilen Anwendung umgesetzt werden, können in der folgenden Übersicht alle Funktionen der Anwendung hergeleitet werden. Diese Anforderungsanalyse ist in zwei Kategorien unterteilt, die *funktionalen Anforderungen* und die *nicht-funktionalen Anforderungen*. Ob und in welchem Maße diese erfüllt werden, wird am Ende in Kapitel 7, *Anforderungsabgleich*, analysiert.

3.1 Funktionale Anforderungen

Die funktionalen Anforderungen (kurz: FA) beschreiben, was die mobile Anwendung oder der Server tun soll.

3.1.1 Mobile Anwendung

#1: Geofence Die Anwendung soll erkennen, wenn das Gerät ein Geofence betritt oder verlässt.

#2: Kontext-sensitive Benachrichtigung Innerhalb eines Geofences soll die Anwendung über aktuelle Angebote benachrichtigen.

#3: Begrüßung durch Anwendung Begrüßung bei Betreten des Lokals durch die Anwendung mit Hilfe eines Beacons im Eingangsbereich.

#4: Indoor-Ortung Erfassung der Position im Lokal mit Hilfe von Beacons und Zuordnung zu nächstgelegenem Beacon.

#5: Mitteilungen an Server via Anwendung Mitteilung an den Server bei erstmaligem Platznehmen oder Platzwechsel im Lokal mit Hilfe der Indoor-Ortung.

#6: Mitteilung über Zahlungswunsch Benachrichtigung an ein Terminal für Service-Mitarbeiter über Zahlungswunsch mit Angabe der Zahlungsweise durch die Anwendung.

#7: Menükarte Digitale Menükarte mit verschiedenen Kategorien.

#8: Digitaler Notizzettel mit Bereitschafts-Ruf Markierung von Speisen oder Getränken mit anschließender Auflistung und Benachrichtigung des Service-Mitarbeiters an einem Terminal über Bestellbereitschaft.

#9: Reservierung mit Tischnummer Bei der Reservierung über die Anwendung soll, falls möglich, die aktuelle Tischnummer auf Wunsch mit angegeben werden.

#10: Einstellungen Möglichkeiten zur Einstellung im Kontext der Benachrichtigungsfunktionalitäten.

3.1.2 Server

#1: Konten-System Konten-System zur Registrierung von authorisierten Benutzern und Administratoren.

#2: Kategorien Kategorien der Menükarte verwalten und Zuordnung von Ober- und Untergruppen.

#3: Produkte Produkte aus der Menükarte verwalten und Zuordnung zu Kategorien.

#4: Beacons Beacons und deren nächstgelegene Beacons sollen verwaltet werden können.

#5: Benachrichtigungen Alle Benachrichtigungen, die von den mobilen Anwendungen gesendet werden, sollen verarbeitet und kategorisch geordnet angezeigt werden.

#6: Geofence Verwaltung von Geofences.

#7: REST-konforme Schnittstelle und JSON Die vom Administrator verwalteten Kategorien, Produkte und Beacons sollen via REST-konforme Schnittstelle abrufbar sein und im JSON Format zur Verfügung gestellt werden.

3.2 Nicht-Funktionale Anforderungen

Nicht-funktionale Anforderungen (kurz: NFA) bleiben dem Benutzer zunächst verborgen, da sie keine inhaltlichen Aspekte beinhalten, sondern beschreiben, welche Eigenschaften die mobile Anwendung oder der Server haben soll. Sie stellen einen qualitativen Anspruch aus handwerklicher Sicht an die Anwendung dar. Werden entsprechende Anforderungen eingehalten, sorgen sie für einen positiven Gesamteindruck der Anwendung beim Benutzer und erhöhen die Motivation. Gleichzeitig sollen dadurch Sicherheitslücken vermieden und die allgemeine Wartbarkeit erhöht werden.

#1: Hohe Kompatibilität Anwendung soll auf möglichst vielen Geräten zum Einsatz kommen können und daher mindestens mit Android Version 4.3, API-Level 18, kompatibel sein.

#2: Datensicherheit Daten, die insbesondere am Server verwaltet werden, sollen vor Verlust und unbefugtem Zugriff geschützt sein.

#3: Einhaltung des MVC-Prinzips Aus Gründen der Übersichtlichkeit, Sicherheit und leichten Wartbarkeit soll das Model-View-Controller-Prinzip (kurz: MVC-Prinzip) eingehalten werden.

#4: Plattform-Kompatibilitäten Alle Schnittstellen sollen prinzipiell die Anwendung auf verschiedenen Plattformen ermöglichen (iOS, Websites etc.).

#5: Performanz und Effizient Aufgrund technischer Gegebenheiten muss die Anwendung insbesondere Ressourcen schonen.

#6: Stabilität und Robustheit Zur Verbesserung des Benutzererlebnisses und der Erhaltung dieser soll die Anwendung stabil laufen und Abstürze (und möglicherweise daraus resultierende Daten-Inkonsistenzen) vermeiden.

#7: Intuitive und funktionale Benutzeroberfläche Eine funktionale Benutzeroberfläche mit intuitiver Steuerung soll die Benutzung erleichtern und Frustration verhindern.

#8: Ansprechendes Design Optisch ansprechende Aufbereitung und möglichst berührungsbasierte Interaktion sollen Benutzer einen positiven Gesamteindruck vermitteln.

4 Aufbau

Um sich einen Überblick über das gesamte System verschaffen zu können, sowie um den Aufbau und die Struktur zu verstehen, sollen im folgenden Kapitel der Server und die mobile Anwendung in Hinblick auf ihre Bedienung separat voneinander erläutert werden, wie in Abbildung 4.1 dargestellt. Einen detaillierten Überblick der gesamten Architektur gibt es in Abbildung 6.1 auf Seite 23.

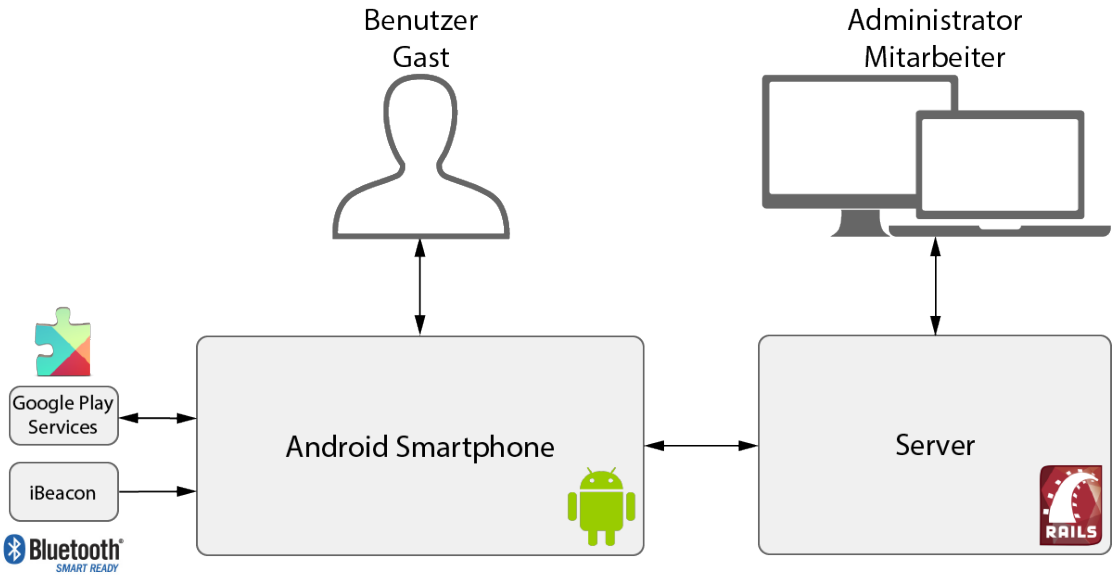

Abbildung 4.1: Grober Überblick der Gesamtarchitektur.

4.1 Server

Ruft ein Administrator das Online-Portal des Servers auf, so kann er zunächst wählen, ob er sich die Kategorien, das Sortiment oder die Beacons ansehen und gegebenenfalls verwalten möchte. Außerdem haben der Administrator oder andere Mitarbeiter hier die Möglichkeit, zur Seite mit Benachrichtigungen zu gelangen.

Beim Sortiment besteht zum einen die Möglichkeit, Kategorie-Gruppen zu erstellen, löschen oder bearbeiten, wobei es auch möglich ist, Gruppen hierarchisch als Ober- oder Untergruppen untereinander zu definieren, sodass beispielsweise „Getränke“ die Untergruppe „Kaffee“ besitzt. Entsprechende Beziehungen, werden in der Detail-Ansicht einzelner Gruppen angezeigt. Au-

ßerdem muss der Administrator wählen, ob eine Gruppe eine Untergruppe *oder* zugeordnete Produkte haben kann.

Zum anderen können einzelne Produkte ebenfalls erstellt, gelöscht oder bearbeitet werden und einer Gruppe zugeordnet werden. Außerdem besteht die Möglichkeit, ein Produktbild hochzuladen, das dann in den digitalen Menükarten der mobilen Anwendungen zusätzlich angezeigt wird.

Auf der Benachrichtigungsseite können theoretisch von Jedermann Informationen abgerufen werden, die von den mobilen Anwendungen generiert und an den Server gesendet werden, wie beispielsweise einen Zahlungswunsch. Die eindeutig identifizierende Android ID, die bei der Erstbenutzung eines Smartphones generiert wird, unterscheidet hierbei einzelne Gäste im Lokal voneinander. Zusammenfassend finden sich hier Informationen darüber, ob ein Gast sich soeben neu an einen Tisch gesetzt hat, sich umgesetzt hat, bezahlen möchte oder nun bereit für die Bestellung ist.

4.2 Mobile Anwendung

Beim Start der mobilen Anwendung soll ein Splash-Screen genutzt werden, um im Hintergrund alle Ressourcen zu laden. Der Benutzer findet sich dann auf der Haupt-Anzeige wieder. Von dort und überall sonst kann er auf das Menü über eine obere linke Schaltfläche zugreifen oder jederzeit durch einen Fingertipp auf das Logo der Anwendung oben in die Hauptanzeige zurückkehren. In der ständig präsenten oberen Bildschirmleiste („Actionbar“) findet sich außerdem eine Anzeige, die über den Status der Bluetooth-Funktionen und der Netzwerkverbindung informiert.

In der Hauptanzeige soll ein Countdown die Zeit bis zur nächsten Happy Hour anzeigen und im Anschluss die verbleibende Zeit der Happy Hour herunterzählen. Außerdem soll ein Bild des Lokals die Räumlichkeiten präsentieren und zur weiteren Entdeckung der mobilen Anwendung einladen.

Im Menü findet sich an erster Stelle ein Punkt, der zur dieswöchigen Tageskarte führt. Diese sollte im Idealfall immer über den Server am Wochenende aktualisiert werden und sich die darauf folgende Woche nicht mehr ändern. Weiter im Menü finden sich separat Speise- und Getränkekarte. Dort kann der Benutzer das gesamte Sortiment durchstöbern und bei Bedarf mit einer Berührung einzelne Produkte markieren und für eine spätere Bestellung aus einer Übersicht heraus ablesen. In den Einstellungen soll dem Benutzer die Freiheit gelassen werden, zu entscheiden, ob er durch Geofence ausgelöste Benachrichtigungen erhalten möchte oder nicht.

5 Anwendungsfälle

Einen detaillierten Einblick in den Aufbau sollen Anwendungsfälle bieten, die alle mögliche Funktionen und Interaktionsmöglichkeiten von Server und mobiler Anwendung schematisieren und beschreiben.

5.1 Server

Abbildung 5.1 zeigt den Anwendungsfall der Registrierung (rote Linie) und des Logins (blaue Linie). Das Prozedere ist in beiden Fällen sehr ähnlich und kann abschließend mit Klick auf die Bestätigungsschaltfläche beendet werden. Im Anschluss an die Registrierung kann das Login-Verfahren gestartet werden.

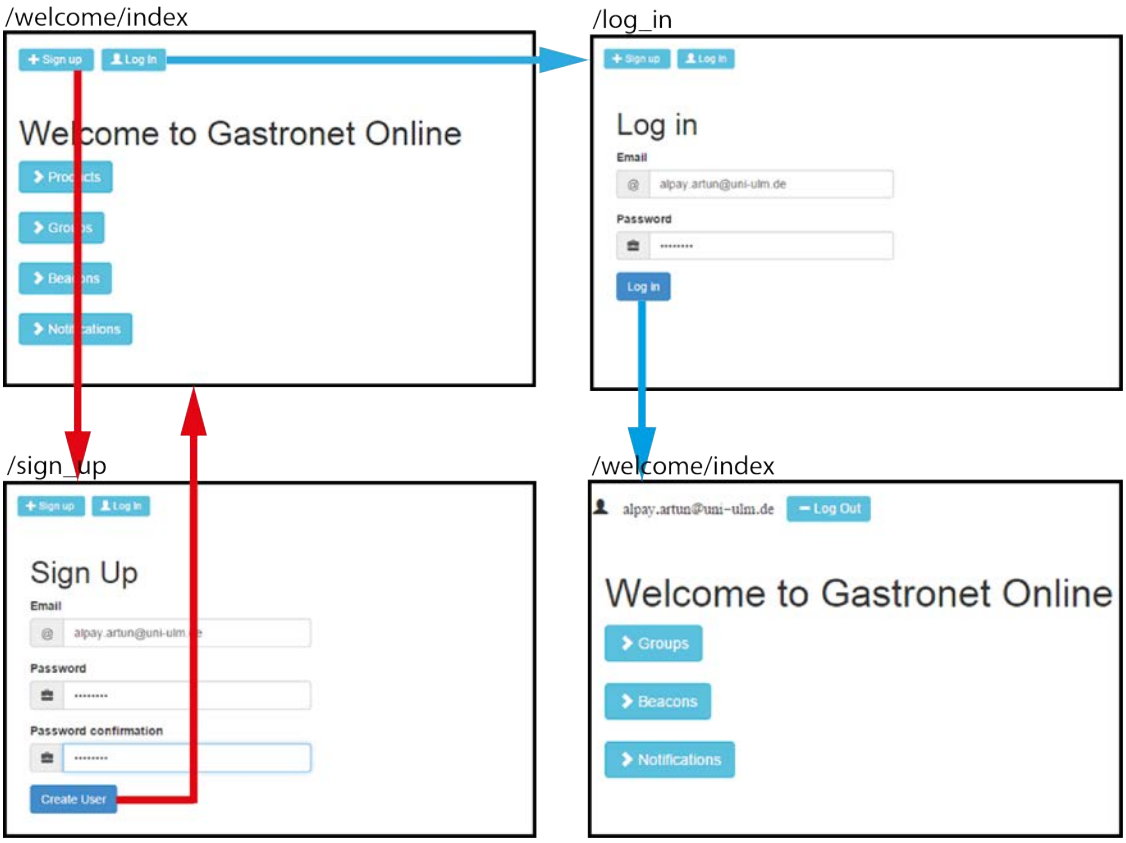

Abbildung 5.1: Anwendungsfall – Registrierung und Login.

Abbildung 5.2 zeigt den Anwendungsfall des Erstellens (rote Linie), des Bearbeitens (gelbe Linie) und des Löschens (blaue Linie) von Kategorien. Die Vorgänge bei Erstellung und Bearbeitung ähneln sich sehr, während beim Löschen lediglich ein Pop-Up erscheint, welches eine Bestätigung für den Abschluss benötigt. Es muss beachtet werden, dass alle Vorgänge nur abgeschlossen werden können, wenn der Benutzer sich zuvor angemeldet hat, da andernfalls eine Fehlermeldung erscheint

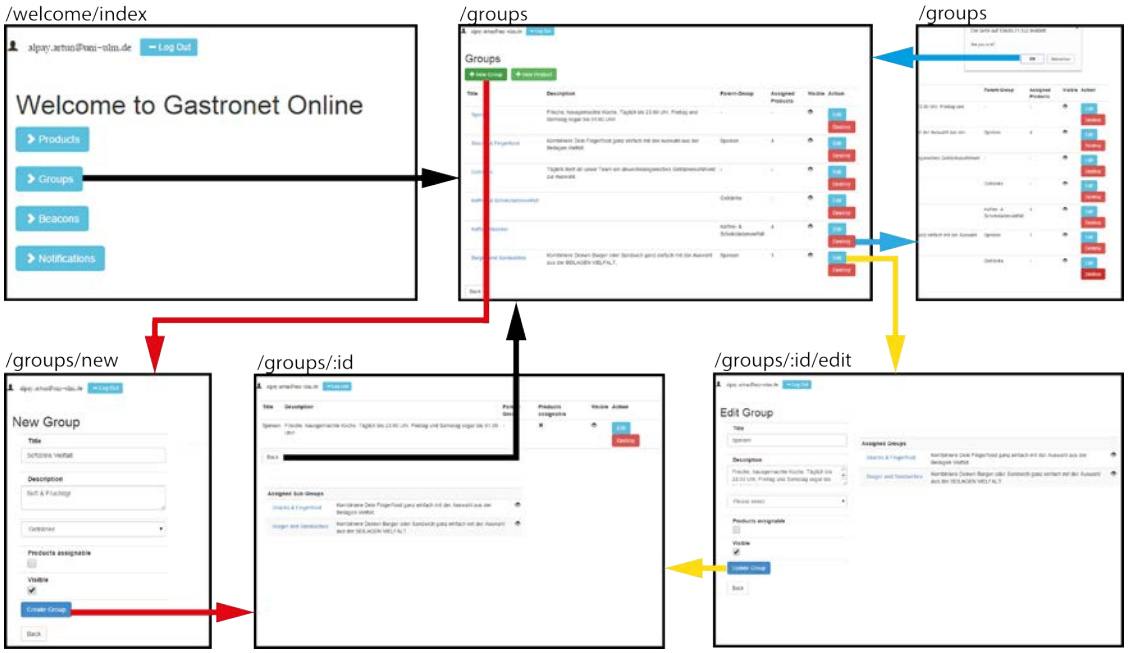

Abbildung 5.2: Anwendungsfall – Erstellen, Bearbeiten und Löschen einer Kategoriegruppe.

Für die Verwaltung der Produkte gilt die Vorgehensweise wie bei Kategorien analog, da sowohl Layout als auch struktureller Aufbau identisch sind. Hierfür kann das Navigationsschema der Abbildung 5.2 für Kategorien verwendet werden.

Abbildung 5.3 zeigt den Anwendungsfall des Erstellens (rote Linie), des Bearbeitens (gelbe Linie) und des Löschens (blaue Linie) von Beacons. Die Vorgänge bei Erstellung und Bearbeitung ähneln sich sehr, allerdings erscheint beim Löschvorgang zunächst ein Pop-Up, welches eine Bestätigung für den Abschluss benötigt. Dieser Anwendungsfall kongruiert strukturell in den meisten Fällen mit dem zuvor veranschaulichten Anwendungsfall für Kategorien und Produkte. Durch dieses Muster soll ein Wiedererkennungswert gefördert und so die intuitive Bedienung optimiert werden (siehe NFA #7 auf Seite 13).

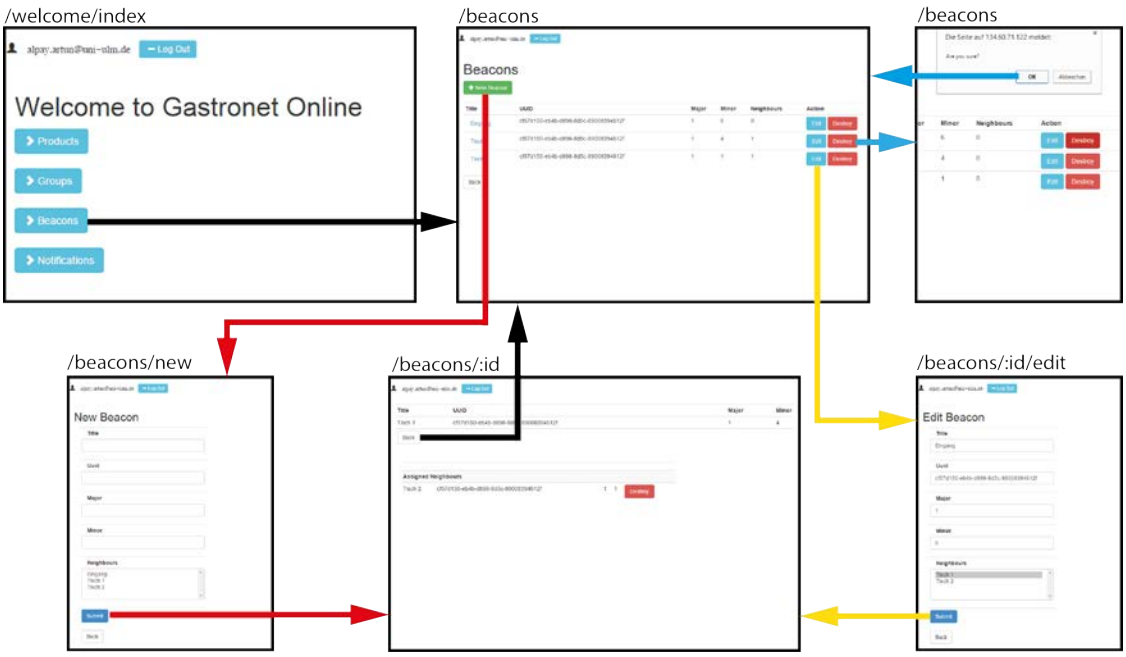

Abbildung 5.3: Anwendungsfall – Erstellen, Bearbeiten und Löschen eines Beacons.

Abbildung 5.4 zeigt die Seite der Benachrichtigungen, die der Server von den mobilen Anwendungen erhält. Dies kann einfach von der Startseite aus aufgerufen werden. Auf dieser Seite befinden sich Informationen über neue Gäste und Gäste, die ihren Sitzplatz wechseln. Außerdem ist hier einsehbar, wer gerne bestellen oder bezahlen möchte.

Sign up Log In

Notifications

NEW

Android Device UID	Beacon Title	Time Created
ea622a4c1c7f230c	Tisch 1	21:08:41

CHANGE

Android Device UID	Beacon Change	Time Created
ea622a4c1c7f230c	Tisch 2 → **Tisch 1**	21:11:19

PAYMENT REQUEST

Android Device UID	Beacon Title	Payment Method	Time Created
ea622a4c1c7f230c	Tisch 2	Bar	21:10:42
df344j343j3hj433	Tisch 3	Bar	21:14:44

READY TO ORDER

Android Device UID	Beacon Title	Time Created
df344j343j3hj433	Tisch 3	20:10:44

Abbildung 5.4: Anzeige der Notifications.

5.2 Mobile Anwendung

Abbildung 5.5 zeigt den Anwendungsfall des Aufrufs und des Durchstöberns der Getränkekarte (rote Linie), was analog für Speisekarte gilt. Nach einer Berührung der Menü-Schaltfläche oben links erscheint das gesamte Menü und mit einer weiteren Berührung kann die Getränkekarte geladen werden. Ein Fingertipp auf ein Produkt markiert dieses und präsentiert es dann als Teil des Notizzettels, nachdem man oben rechts die „Fertig“-Schaltfläche berührt. Mit seitlichen Wischgesten können die einzelnen Kategorien der Getränkekarte gewechselt werden (schwarze Linie). Innerhalb der Kategorien kann mit vertikalen Wischgesten die Produktliste erkundet werden. Ein Fingertipp auf ein Produktbild zeigt eine größere Ansicht und lässt sich mit einem Fingertipp auf die „schließen“-Schaltfläche wieder ausblenden (blaue Linie).

Abbildung 5.5: Anwendungsfall – Aufruf und Durchstöbern der Menükarte mit Notizzettel-Funktion.

Abbildung 5.6 zeigt den Anwendungsfall einer Benachrichtigung über ein kontext-senstives Angebot, das durch das Betreten eines Geofences ausgelöst wird (rote Linie). Das Gerät zeigt eine Benachrichtigung in der oberen Display-Leiste an, die gelesen und angetippt werden kann. Letzteres öffnet die mobile Anwendung und weist auf detaillliertere Informationen des Ereignisauslösers hin. Im illustrierten Fall wird man direkt zum korrekten Tagesgericht geführt, wo man beispielsweise direkt dazu übergehen kann, das Tagesgericht und andere Produkte für eine spätere Bestellung zu markieren, wie im Anwendungsfall zuvor gezeigt.

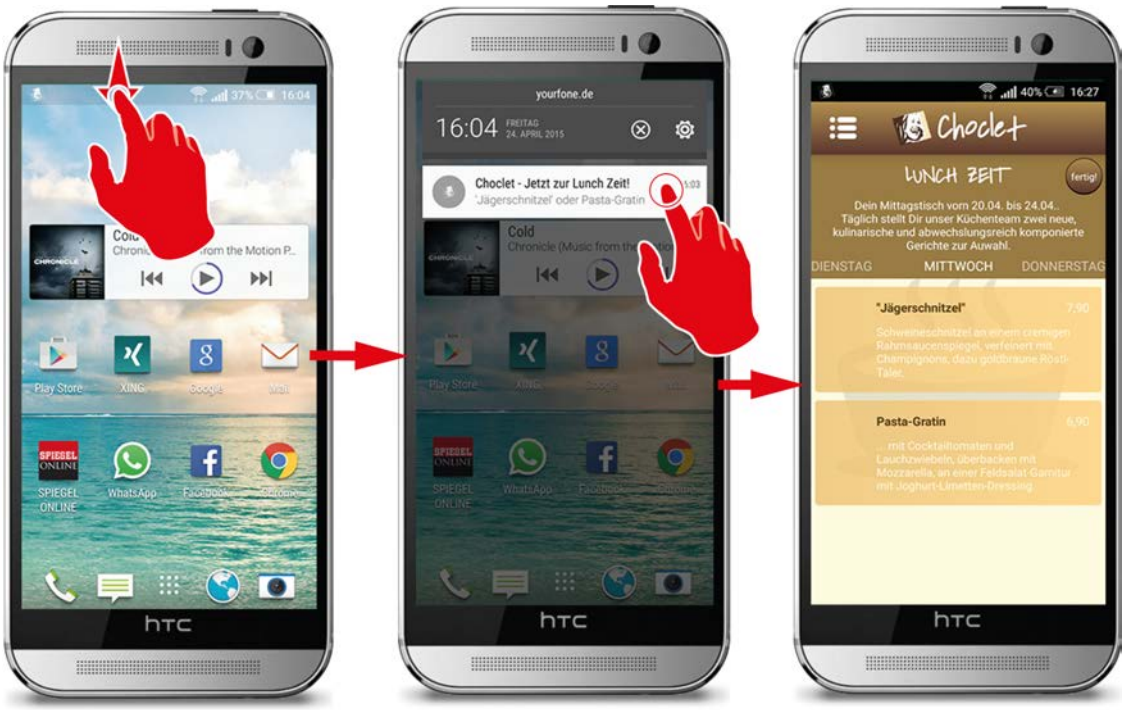

Abbildung 5.6: Anwendungsfall – Geofence-Benachrichtigung für ein Tagesgericht.

Abbildung 5.7 zeigt den Anwendungsfall der verschiedenen Dienstleistungen, die der Benutzer anfordern kann (rote Linie). Dort finden sich die Möglichkeiten, einen Zahlungswunsch oder einen Bestellwunsch zu äußern. Ist jedoch noch kein Beacon als aktueller Sitzplatz registriert, sind die Schaltflächen ausgegraut. Über das Menü können außerdem die Einstellungen aufgerufen werden (blaue Linie). Dort können Einstellungen bezüglich der möglichen Benachrichtigungen gesteuert werden. Dies können allgemein alle möglichen Benachrichtigungen oder speziell nur die Benachrichtigungen über eine anstehende „Happy Hour“, sofern man sich in der Nähe des Lokals befindet, sein.

Abbildung 5.7: Anwendungsfall – Anzeige der Services und der Einstellungen.

Abbildung 5.8 zeigt wie der aktuelle Status der Bluetooth- und Netzwerkverbindungsfunktionen als blinkende Symbole oben rechts angezeigt wird, sollten diese nicht aktiviert sein. In diesem Fall können davon betroffene Funktionen der Anwendung nicht ausgeführt werden. Werden besagte Funktionen während der Verwendung der Anwendung verfügbar, werden die Symbole entsprechend wieder ausgeblendet.

Abbildung 5.8: Anzeige fehlender Systemfunktionen.

6 Architektur und Implementierung

Prinzipiell muss die Gesamtarchitektur in zwei Komponenten unterschieden werden, nämlich dem *Server* zum einen und der lokalen, *mobilen Anwendung* zum anderen (siehe Abbildung 6.1). Beide sollen im Folgenden in Hinsicht auf *Architektur*, *Layout* und *technische Implementierung* hin erläutert werden.

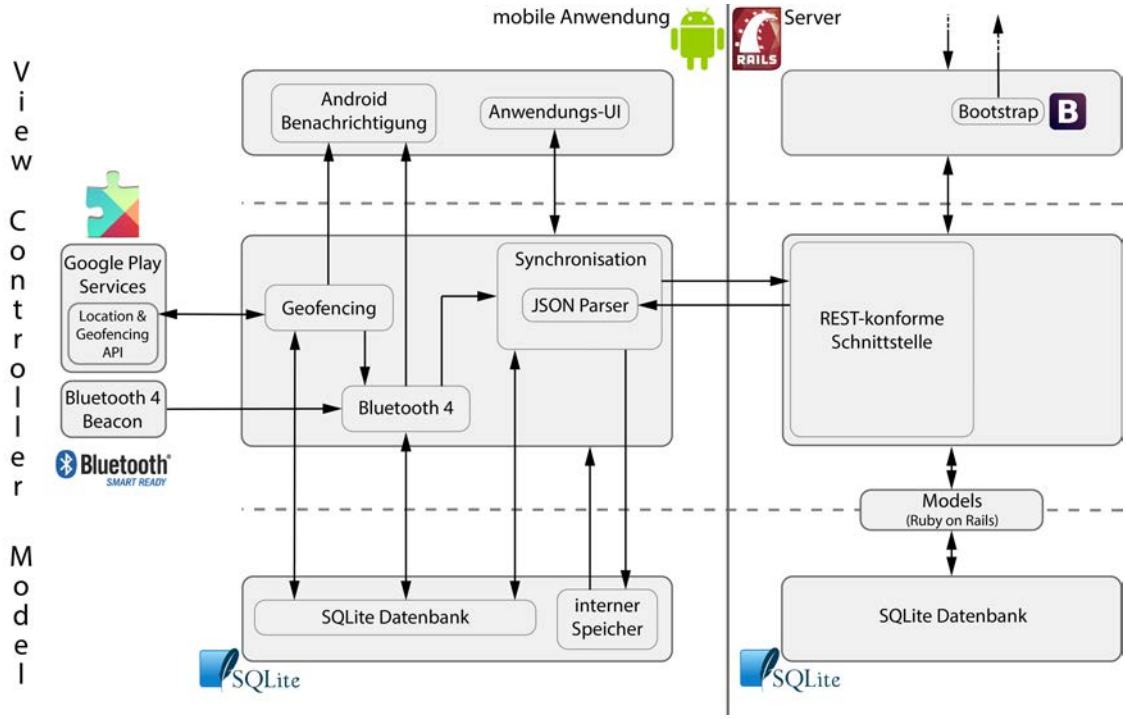

Abbildung 6.1: Detaillierter Überblick der Gesamtarchitektur.

6.1 Server

Der Server, das Back-End, speichert einen Großteil der Inhalte, die auf der mobilen Anwendung dargestellt werden. Ein Administrator mit entsprechendem Konto kann sich auf einem Online-Portal anmelden und diese Inhalte bearbeiten. Mitarbeiter ohne Konto können die selben Inhalte sehen, jedoch keine Änderungen daran vornehmen.

6.1.1 Architektur

Die Wahl der Programmiersprache für das Back-End fiel auf Ruby, das sich genauso wie das Web-Framework Ruby on Rails wachsender Beliebtheit erfreut. Das für diese Abschlussarbeit verwendete Ruby on Rails bietet bereits von vornherein eine strenge Unterteilung im Sinne des MVC-Prinzips (siehe NFA #3, Seite 12). Die für Ruby typisch gute Lesbarkeit erweist sich für den Programmierprozess als äußerst hilfreich. Auch kann eine REST-konforme Schnittstelle ohne besonders großen Aufwand umgesetzt werden (mehr dazu in Kapitel 6.1.3 auf Seite 25). Auch liefert Ruby on Rails entsprechende Methoden, um Objekte automatisiert in JSON-Formate umzuwandeln, um sie auf Anfragen entsprechend an die mobilen Anwendungen zu senden. Als nennenswerte Erweiterungen („Gems“) von Ruby on Rails kommen Paperclip für den Upload von Bilddateien und Bootstrap mit SASS für das UI-Design zum Einsatz (siehe NFA #8, Seite 13). Abbildung 6.2 zeigt einen Überblick über die Architektur des Servers.

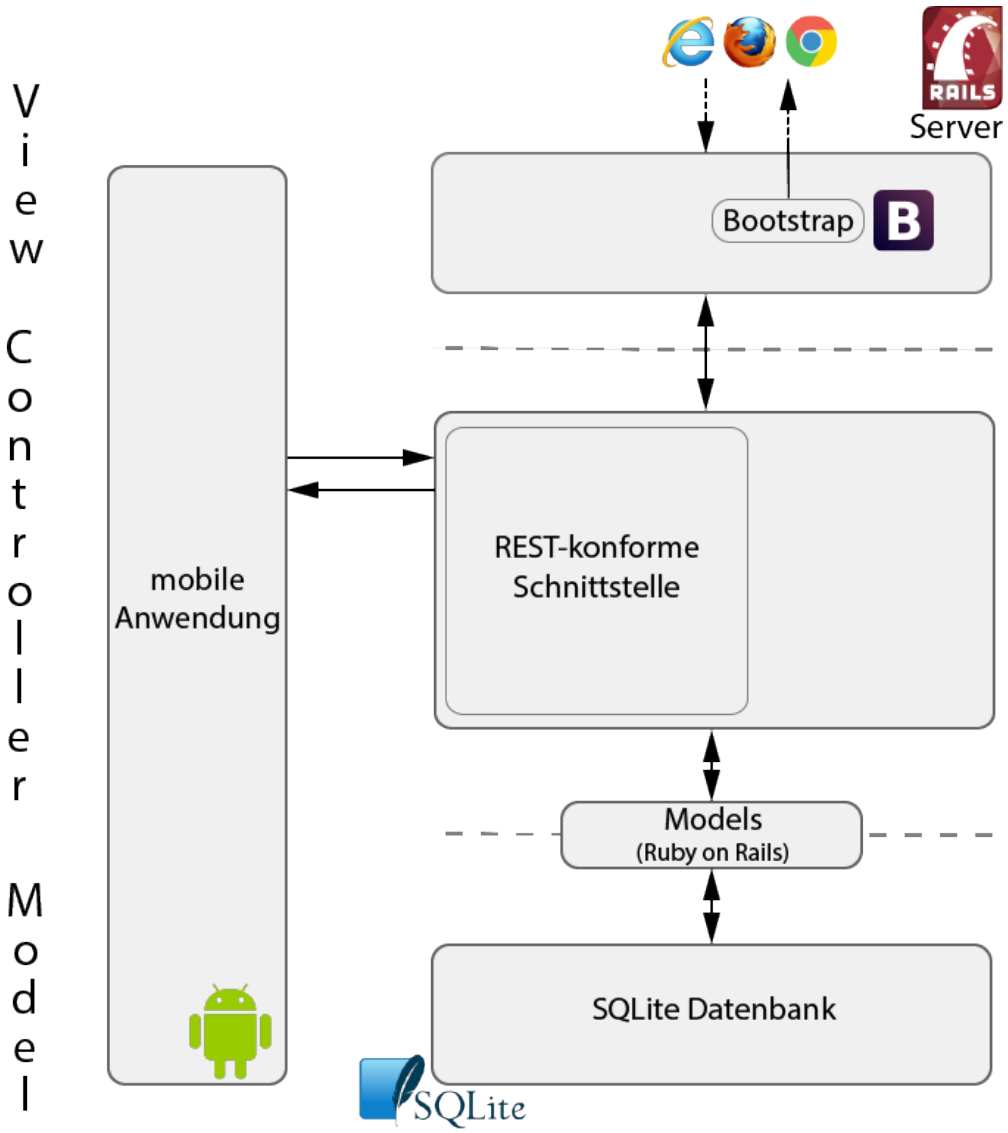

Abbildung 6.2: Architektur des Servers gemäß MVC-Prinzip.

Um Missbrauch zu vermeiden und die Datensicherheit zu erhöhen, können Änderungen an den Daten nur mit Authentifizierungs-Tokens getätigt werden, die man über die Anmeldung erhält. In diesem Sinne sind zahlreiche Regelungen („Constraints“) über die Models (Objekt-Schablonen) in Ruby on Rails definiert, sodass Eingabedaten validiert und gegebenenfalls unschädlich gemacht werden, um beispielsweise SQL-Injections zu vermeiden (siehe NFA #2 Seite 12). Auch kann über diese Maßnahmen die Konsistenz des Datenmodells gewährleistet werden (siehe auch Abbildung 6.5, Seite 27).

6.1.2 Layout

Für die Gestaltung der Benutzer-Oberfläche wurde mit Bootstrap von Twitter gearbeitet [2]. Da das Online-Portal lediglich für Administratoren und Mitarbeiter gebraucht wird, ist sichergestellt, dass ein gewisses Mindestmaß an Usability und Übersichtlichkeit gewährleistet wird und die Navigation klar erkennbar ist. Darüber hinaus gehende Spielereien sind nicht Bestandteil der Oberfläche.

Zur Realisierung genannter Usability-Anforderungen finden einige der zahlreichen CSS-Klassen von Bootstrap Verwendung. Konsistente Layouts und Farbkombinationen sorgen für einen Wiedererkennungswert, der die Navigation erleichtert. Des Weiteren sind einige Icons eingebaut, die weiterhin eine intuitive Bedienung erleichtern sollen (siehe NFA #7, Seite 13). Beispielsweise beim E-Mail-, Datei-Upload- oder Preis-Eingabefeld.

6.1.3 Technische Implementierung

Die Programmierung am Server berücksichtigt das „convention over configuration“-Paradigma von Ruby, das beispielsweise für Java-Entwickler eine Umgewöhnung darstellt [18]. Erfreulicherweise aber unterstützt das Paradigma auch die in der Informatik gewünschten Prinzipien KISS (zu deutsch etwa: „Mache es sicher und einfach.“) und DRY („Don’t repeat yourself.“), welche Redundanzen vermeiden und die Übersichtlichkeit des Programmiercodes fördern sollen. In diesem Kapitel soll zunächst das Datenmodell erläutert, bevor die Algorithmen zur Verwaltung der Daten schematisch dargestellt und erklärt werden. Zum Abschluss findet sich ein Beschreibung der Schnittstelle.

Datenmodell

Als wichtige Grundlage für die Programmierung dient ein logisch durchdachtes Datenmodell und die Beziehungen der einzelnen Datenobjekte untereinander. Um dies zu verdeutlichen zeigt Abbildung 6.3 wie Kategorien (hier: Groups) und Produkte intern definiert sind und in welcher Beziehung sie zueinander stehen. Die Darstellung zeigt, dass eine Kategorie entweder Unterkategorien oder Produkte zugeordnet bekommen kann, wenn ihr Wert für „has_items“ *false* oder *true* ist. Dabei kann eine Kategorie entweder beliebig viele zugeordnete Produkte (über den

Fremdschlüssel „group_id“) oder andere Kategorien (über den Fremdschlüssel „parent_id“) haben. Allerdings kann es keine Produkte geben, wenn sie keiner Kategorie zugeteilt sind. Beide Objekte können über den Wert von „visible“ für Benutzer der mobilen Anwendung sichtbar oder unsichtbar gemacht werden. Es ist zu beachten, dass Ruby on Rails automatisch einen Primärschlüssel speichert, der in diesen beiden Fällen gemäß des „convention over configuration“-Paradigmas „group_id“ und „product_id“ heißt. Aus diesem Grund sind Primärschlüssel nur angegeben, wenn sie zum Verständnis notwendig sind.

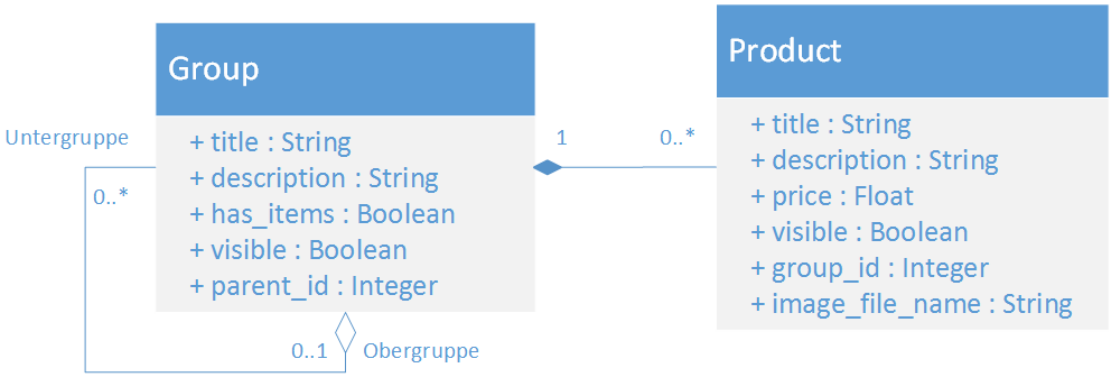

Abbildung 6.3: Modell und Beziehung von Kategorien und Produkten.

Eine weitere wichtige Objekt-Klasse sind natürlich die Beacons, die sich allgemein über ihre *UUID* („Universally Unique Identifier“), ihren *Major-* und ihren *Minor-Wert* definieren. Die UUID soll einen allgemein eindeutigen Identifikator darstellen, sodass beispielsweise Beacons mit der gleichen UUID eindeutig einem Kontext zugeordnet werden können, wie eben einer Gastronomie. Der Major-Wert dient wiederum einer Kontextzuordnung in der Gastronomie, wie beispielsweise der Zuteilung zu Innenraum oder Terrassenbereich. Der Minor-Wert schließlich identifiziert Beacons innerhalb des Kontextes einer Major-Wert-Gruppe.

Darüber hinaus sind den Beacons zur besseren Verwaltung interne „beacon_id“ zugeordnet und besitzen einen für Menschen lesbaren *Titel*. Das Objekt „DeviceAndBeacon“ repräsentiert eine Information, die mit einem Beacon und einem Android Gerät (eindeutig identifiziert über „android_device_id“) verknüpft ist. Dieses Objekt sagt beispielsweise aus, welches Gerät an einem Beacon einen Zahlungswunsch sendet. Dazu mehr im Kapitel 6.2.3, *Upload von Informationen*.

Schließlich sagt das Objekt „BeaconNeighbours“ aus, welche Beacons sich in unmittelbarer Nähe zueinander befinden. Dies wird dafür verwendet, um verbrauchte Batterien automatisch zu erkennen. Die Anwendung soll also „wissen“, welche Signale von Nachbar-Beacons sie außerdem empfangen muss. Tut sie das nicht, soll sie diese Information an den Server senden, der diese auswertet und gegebenenfalls auf der Benachrichtungsseite eine entsprechende Mitteilung ausgibt. Folgende Abbildung 6.4 illustriert alle drei Objekte und ihre Beziehungen zueinander.

Abbildung 6.4: Modell und Beziehungen von Beacons, Paaren und Beacons mit Ereignis.

Die erforderliche Datensicherheit und -konsistenz kann bereits zum Großteil über die Model-Constraints von Ruby on Rails realisiert werden (siehe NFA #2, Seite 12). Im folgenden Modell, das ein User-Objekt beschreibt, ist zu sehen, wie das Modell bereits mit Validierungsmethoden die Gegebenheit von Passwort und E-Mail Adresse prüft. Die impliziten Methoden *validates_presence_of* beispielsweise stellen sicher, dass entsprechende Attribute vorhanden sind. Außerdem kann mit nur einer Zeile (nämlich *validates_confirmation of :password*) die Übereinstimmung von Passwort und Passwort-Wiederholung geprüft werden. Die Klasse beinhaltet darüber hinaus die Methode *encrypt_password*, die gegebene Passwörter salzt und hashed, bevor sie in Kontakt mit der Datenbank geraten [13]. Die Methode *self.authenticate* der Klasse bearbeitet hingegen einen Loginprozess (siehe Abbildung 6.5).

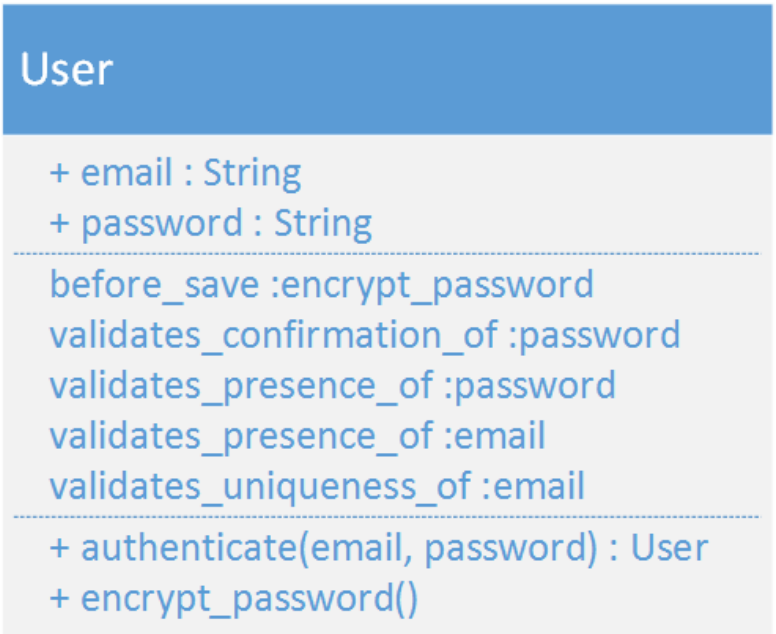

Abbildung 6.5: Klassenmodell eines Users mit Constraints.

Sollten hierbei Fehler auftreten, werden diese direkt an die Bedienoberfläche weitergegeben (siehe Abbildung 6.6). Entsprechende, leicht anwendbare Funktionen werden von Ruby on Rails zur Verfügung gestellt und erlauben eine einheitliche Erscheinung von Fehlermeldungen auf der gesamten Website.

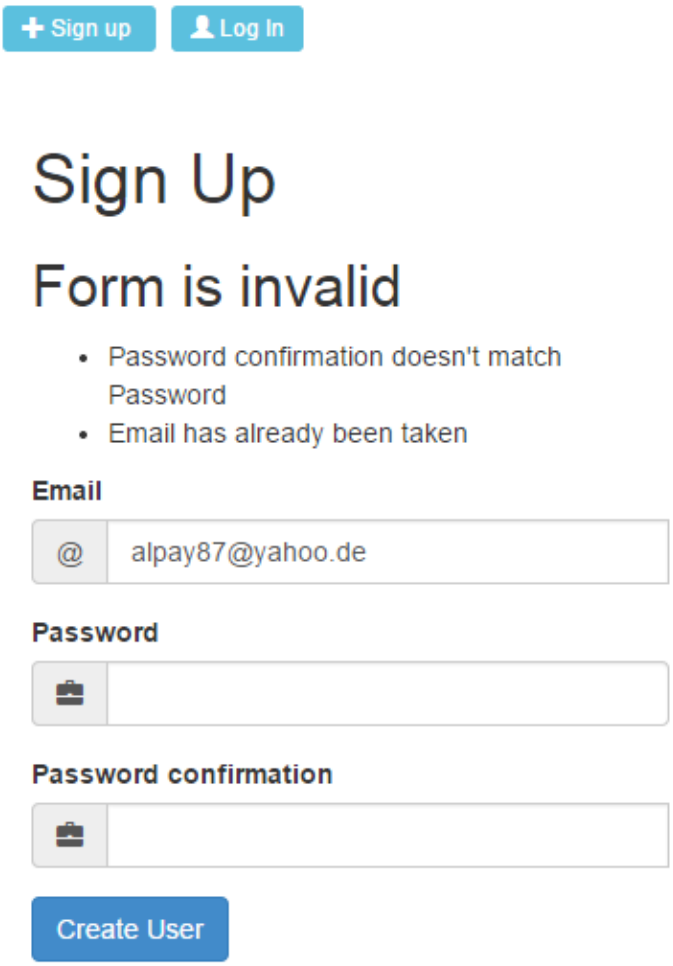

Abbildung 6.6: Fehler in Datenüberprüfung.

Verwaltung

Um den Benutzer zu unterstützen und ihn vor möglicherweise unbeabsichtigen Aktionen zu bewahren, erscheint ein Pop-Up, falls eine Kategorie, die entweder zugeordnete Produkte *oder* Unterkategorien besitzt, bearbeitet oder gelöscht werden soll. Dieser Fall ergibt sich also nur, wenn Abhängigkeiten von Produkten oder Unterkategorien gegenüber der zu löschenden oder zu bearbeitenden Kategorie vorhanden sind. Der hierfür verwendete JavaScript-Algorithmus wird im Aktivitätsdiagramm in Abbildung 6.7 veranschaulicht.

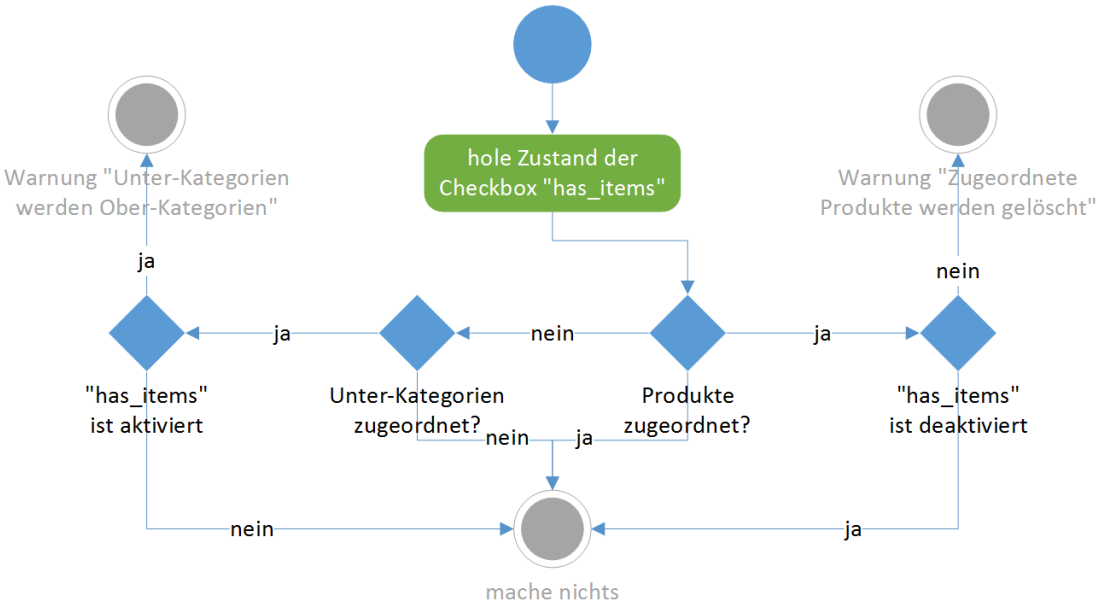

Abbildung 6.7: Schematische Darstellung des Warn-Algorithmus'.

Will der Benutzer nun die Bearbeitung einer Kategorie abschließen, wird die entsprechende Methode, die in Ruby programmiert ist, aufgerufen (siehe Abbildung 6.8). Die Methode prüft zunächst, ob der Benutzer angemeldet ist und dahingehend überhaupt die Befugnis besitzt. Ist dies nicht der Fall, erscheint eine Fehlermeldung auf dem Bildschirm und der Benutzer muss den Vorgang nach erfolgreicher Anmeldung wiederholen.

Andernfalls prüft der Algorithmus danach, ob der Kategorie Produkte zugeordnet sind, sie nun aber ab sofort keine mehr zugeteilt haben kann. In diesem Fall werden alle zugeordneten Produkte gelöscht und die Versions-Nummer für die Produkt-Datenbank erhöht. Die Veränderung an der Datenbank durch das Entfernen von Produkten führt dazu, dass die mobile Anwendung bei der nächsten Anfrage die neueste Version der Produkt-Datenbank herunterlädt.

Im anderen Fall, dass es zugeordnete Unterkategorien gibt, die aber anschließend wegen ihrer Abhängigkeit nicht existieren dürften, wird bei allen Unterkategorien die Zugehörigkeit gelöscht, was sie zu hierarchisch obersten Kategorien macht. Die korrekte Zuordnung dieser betroffenen Kategorien kann danach separat vorgenommen werden, falls dies nicht bereits im Vorfeld geschehen ist. Kann die bearbeitete Kategorie dann erfolgreich abgespeichert werden, erhöht sich auch hier die Versionsnummer für Kategorien, was zu einer Aktualisierung in der mobilen Anwendung führt.

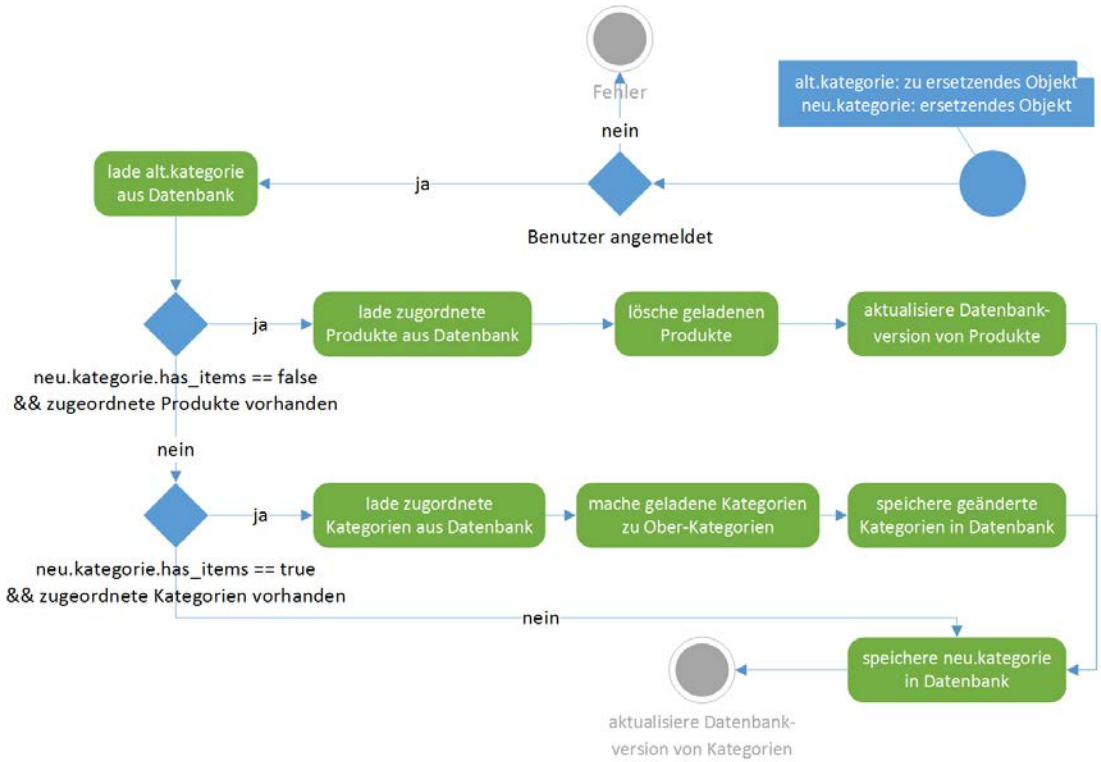

Abbildung 6.8: Schematische Darstellung des Aktualisierungs-Algorithmus' für Kategorien.

Schnittstelle

Um die Kommunikation zwischen mobiler Anwendung und Server zu ermöglichen, bedarf es einer Schnittstelle (siehe Server FA #7, Seite 12). Die hier verwendete Schnittstelle ist REST-konform und versendet Daten im JSON Format [10]. REST-konform (für „Representational State Transfer") bedeutet, dass eine Web-Adresse (URI) die Repräsentation von genau einem Seiteninhalt darstellt und dass der Server auf Anfragen mit derselben Web-Adresse auch immer mit demselben Web-Inhalt antwortet.

JSON (für „JavaScript Object Notation") ist ein Dateiformat, das für Menschen normalerweise lesbarer ist als beispielsweise das gängige XML-Format und darüber hinaus im Regelfall einen geringeren Overhead (keine effektiven Nutzdaten) verursacht als letztgenannter. JSON ist für diesen Anwendungszweck auch deswegen geeignet, weil es nicht für die Übermittlung von Binärdatenmengen vorgesehen ist. Auszug 6.1 aus einer JSON-Datei beispielsweise erhält die mobile Anwendung, wenn sie nach den Versionen der verschiedenen Datenbanktabellen fragt (mehr dazu im Kapitel 6.2.3, *Inhaltssynchronisierung*).

Abbildung 6.9 zeigt die möglichen Aufrufe, die ein Browser an die Schnittstelle stellen kann. Dabei sind HTTP-Requests des Typs GET für Aufrufe einer Website vorgesehen, während HTTP-Requests des Typs POST, PUT oder DELETE einen direkten Aufruf einer Methode zum Erstellen, Bearbeiten oder Löschen eines Datenobjekts auf dem Server darstellen. Bei Aufrufen, die

Listing 6.1: Auszug aus JSON-Datei.

```
[
    {
        "id":18,
        "uuid":"cf57d150-eb4b-d898-8d5c-89008394612f",
        "major":1,
        "minor":6,
        "title":"Eingang"
    },
    {
        "id":19,
        "uuid":"cf57d150-eb4b-d898-8d5c-89008394612f",
        "major":1,
        "minor":4,
        "title":"Tisch 1"
    }
]
```

eine Änderung in der Datenbank zur Folge haben, wird auf die Existenz eines Authentifizierungs-Token geprüft, um die Berechtigung des Anfragestellers zu validieren.

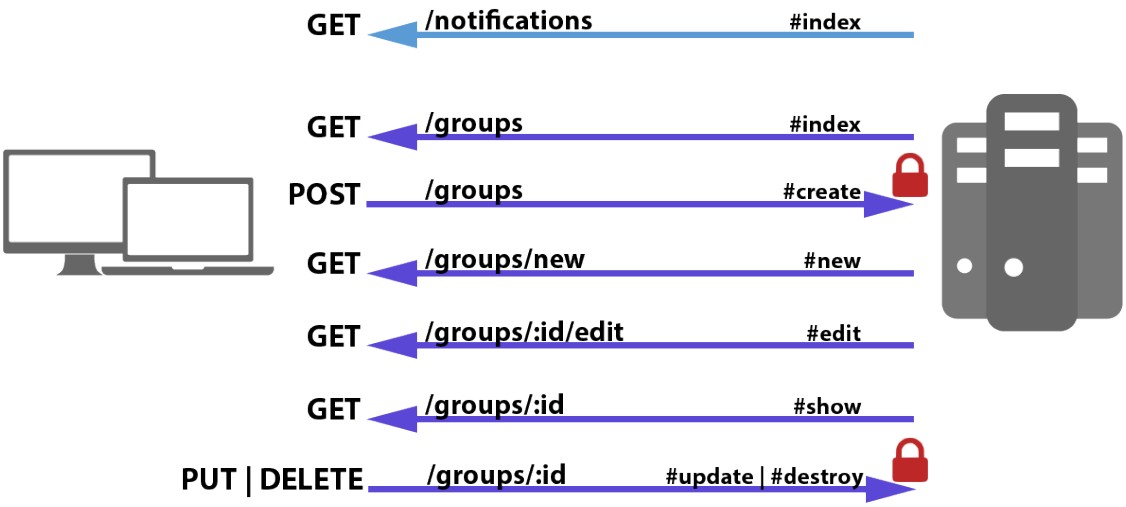

Abbildung 6.9: REST-konforme Schnittstelle für Online-Portale.

6.2 Mobile Anwendung

Anders als der Server, ist die mobile Anwendung die Komponente des Systems, die von Gästen einer Gastronomie verwendet werden kann. Hier gelten, insbesondere in Bezug auf Optik und Usability, andere Maßstäbe. Funktionsweise und Umsetzung, sowie einige nennenswerte Besonderheiten, sollen im Folgenden erläutert werden.

6.2.1 Architektur

Zur Erfüllung der Anforderungen ist die Einhaltung des MVC-Prinzips besonders wichtig (siehe NFA #3, Seite 12). In diesem Sinne bilden die Datenbank mit ihrer dazugehörigen Logik, die Benutzeroberfläche und alle Methoden, die zur Steuerung und Kommunikation genannter Komponenten dienen, jeweils eine Einheit, die untereinander wiederum in einzelne Komponenten aufgeteilt sein können. Da die mobile Anwendung ihre Inhalte zu einem großen Teil durch den Server bezieht, stellt dieser eine elementare Komponente der Architektur dar.

Im Detail in Abbildung 6.10 ist zu sehen, dass der Controller Anfragen an die REST-konforme Schnittstelle senden kann, während ein JSON-Parser die erhaltenen Daten verarbeitet und wiederum an die bereits bestehende Architektur weitergibt . Darüber hinaus bildet der Controller die Schnittstelle zur Google-API, die eine ressourcenschonende Lieferung von Ortungsdaten gewährleisten kann und für die Funktionen mit dem Geofencing eine wichtige Rolle spielt.

Wegen des breiten Spektrums an BLE- und Geofence-Funktionalitäten, bilden der Bluetooth- und Geofence-Controller jeweils eine eigene Komponenteneinheit, die wiederum über Schnittstellen mit dem Controller kommunizieren.

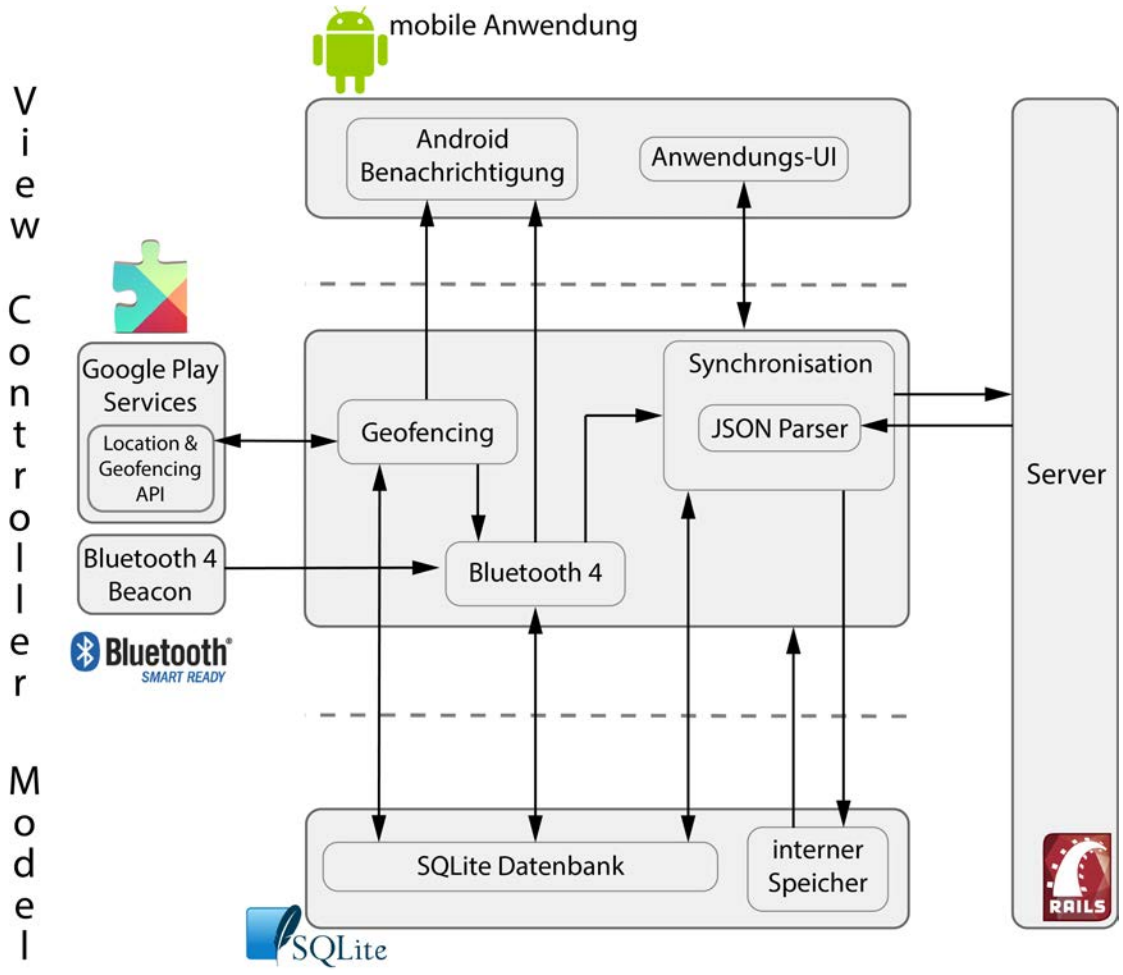

Abbildung 6.10: Architektur der mobilen Anwendung gemäß MVC-Prinzip.

Als Persistenz-Strategie fällt die Entscheidung auf die SQLite Datenbank, die leicht wegen ihrer Integration in Android verwendet werden kann. SQLite ist eine Open-Source-Datenbank, die wenig Speicher während der Laufzeit benötigt. Für die Sicherung von Bildern wird ein der Anwendung zur Verfügung stehender Speicherplatz auf dem Gerät verwendet. Die Verwaltung dieser Bilder regelt der Controller.

6.2.2 Layout

Für die Gestaltung der Benutzer-Oberflächen wurde mit XML, Standard in Android, gearbeitet. Android Studio bietet dafür, sehr ähnlich wie zuvor schon Eclipse mit ADT-Plug-in, ein übersichtliches Ordnersystem, welches in dieser Anwendung dazu genutzt wird, dauerhaft benötigte Bilder effizient abzulegen.

Zur Realisierung des Menüs kommt die *Slideholder-Klasse* zum Einsatz [19]. Nach Modifikation und Anpassung der Klasse bietet das Menü nun neben ansprechendem Design auch logisches Verhalten bei Interaktionsgesten. Für die Darstellung aller Speise- und Getränkekategorien findet der von Google zur Verfügung gestellte *FragmentPagerAdapter* Verwendung. Durch diesen kann der Benutzer intuitiv mit seitlichen Wischgesten durch die einzelnen Kategorien wechseln. Über die Menü-Schaltfläche auf der linken Seite der Actionbar kann das Menü zu jedem Zeitpunkt aufgerufen werden.

Auf der rechten Seite der Actionbar signalisieren blinkende Icons den Status wichtiger, nicht aktivierter Funktionen, nämlich der Netzwerkverbindung und der Bluetooth-Verfügbarkeit.

Auf dem Hauptbildschirm befindet sich ein Countdown-Timer, der zur nächsten Happy-Hour hochzählt und später die Restzeit herunterzählt. Mit Wischgesten nach oben kann der weitere Inhalt in den Sichtbereich gescrollt werden. Dort befinden sich allerdings nur Platzhalter, da hier keine für diese Arbeit wichtigen Funktionen dort untergebracht werden sollen.

6.2.3 Technische Implementierung

Da Android die Zielplattform darstellt, liegt der Code in Java vor, weil eine native Programmierung bevorzugt wird. Als integrierte Entwicklungsumgebung findet Android Studio, die neue, offizielle Entwicklungsumgebung von Google, Verwendung. Insbesondere wegen der vollständig funktionsfähigen Funktionalitäten von BLE auf allen Android Geräten, ist die mobile Anwendung mit allen Android Versionen ab 5.0, also API-Level 20, kompatibel (siehe NFA #1, Seite 12) und kann daher nur auf entsprechend aktuellen Geräten installiert werden.

Geofence Funktionalität

Für eine korrekte Geofence-Funktionalität ist es zunächst notwendig, dass entsprechende Geofences definiert werden [17]. Anders als in der Anforderungsanalyse gefordert, bezieht die mo-

bile Anwendung ihre Geofences nicht dynamisch vom Server, wo sie verwaltet werden, sondern sind aus Zeitgründen hard-coded (siehe Server FA #6, Seite 12).

Im Sequenzdiagramm 6.11 ist zu sehen, wie der Algorithmus Geofences definiert, sich mit dem Google Play Service verbindet und die Geofences an diesem dann anmeldet. Wird ein Umschaltvorgang („Transition“, siehe auch Abbildung 2.1 auf Seite 3) mit einem Geofence erkannt, so wird *onHandleIntent()* durch den Google Play Service via *Callback* aufgerufen.

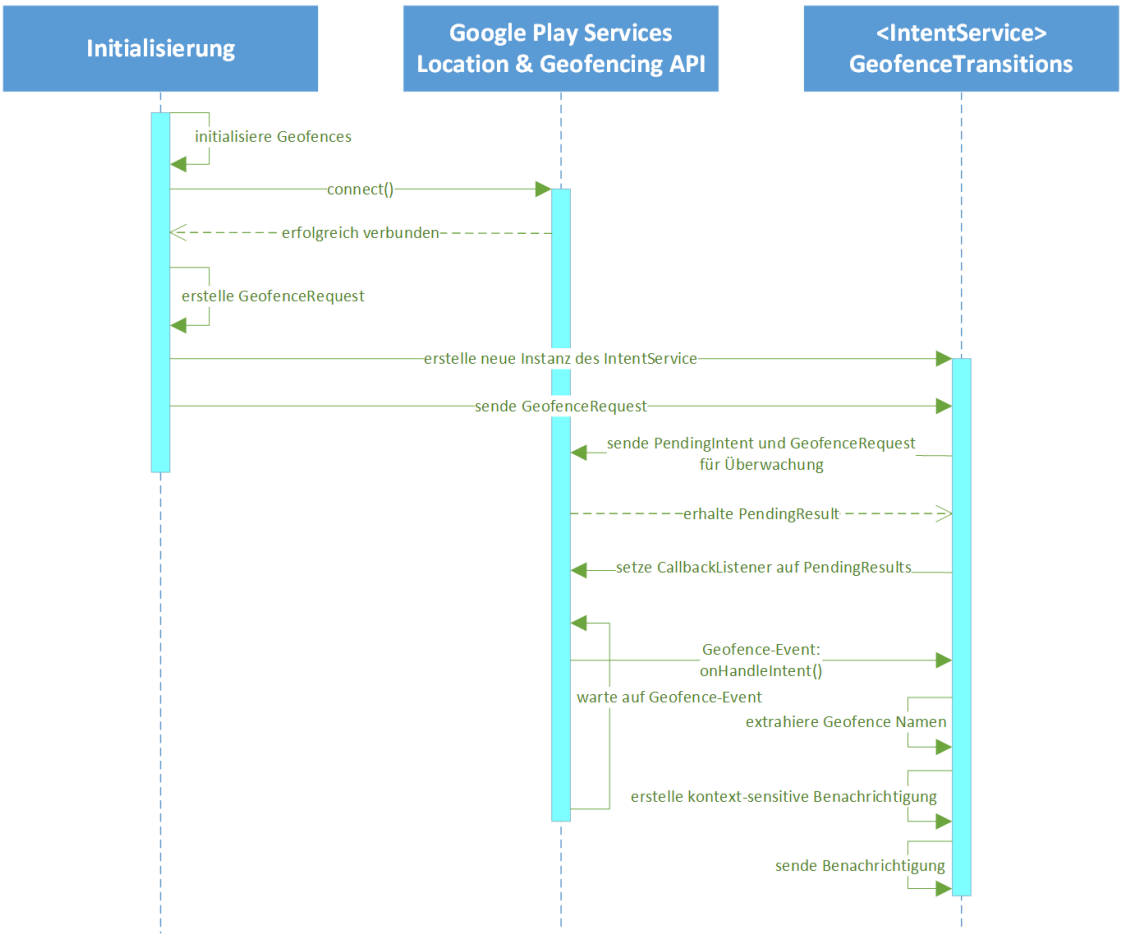

Abbildung 6.11: Vereinfachtes Sequenzdiagramm der Anmeldung, Überwachung und Behandlung von Geofences.

Um den Energieverbrauch der Anwendung zu optimieren, wird die Bluetooth-Funktionalität erst aktiviert, wenn das Gerät sich in unmittelbarer Nähe des Lokals mit den Beacons befindet. Um dies zu realisieren, wird ein kleineres Geofence mit einem Radius von etwa 45 Metern definiert. Da die Anwendung nicht die genauestmöglichen Standortdaten abfragt, muss das Geofence eine gewisse Mindestgröße besitzen, damit es überhaupt eine Aktion auslösen kann. Konkret bedeutet dies, dass die aktuellen Einstellungen eine Genauigkeit von etwa 100 Metern erreichen [4]. Dieser Kompromiss zwischen Energieverbrauch und Genauigkeit (mehr dazu in der Diskussion in Kapitel 8), wird mit folgender Einstellung erreicht:

Listing 6.2: Befehl zur Einstellung der Genauigkeit und des Energieverbrauchs.

```
1 mLocationRequest.setPriority(LocationRequest.
    PRIORITY_BALANCED_POWER_ACCURACY);
```

In der Praxis zeigt sich aber, dass in Gegenden mit Wi-Fi-Signalen üblicherweise eine Genauigkeit von 40 Metern erreicht wird. Daher muss der Radius des Geofence hier mindestens 40 Meter betragen. Spätestens beim Betreten des Lokals also, das einen drahtlosen Router betreibt, wird die Blueooth-Funktionalität der Anwendung aktiviert, wie in Abbildung 6.12 dargestellt. Verlässt der Benutzer dieses Geofence, so deaktivieren sich die Bluetooth-Funktionen wieder.

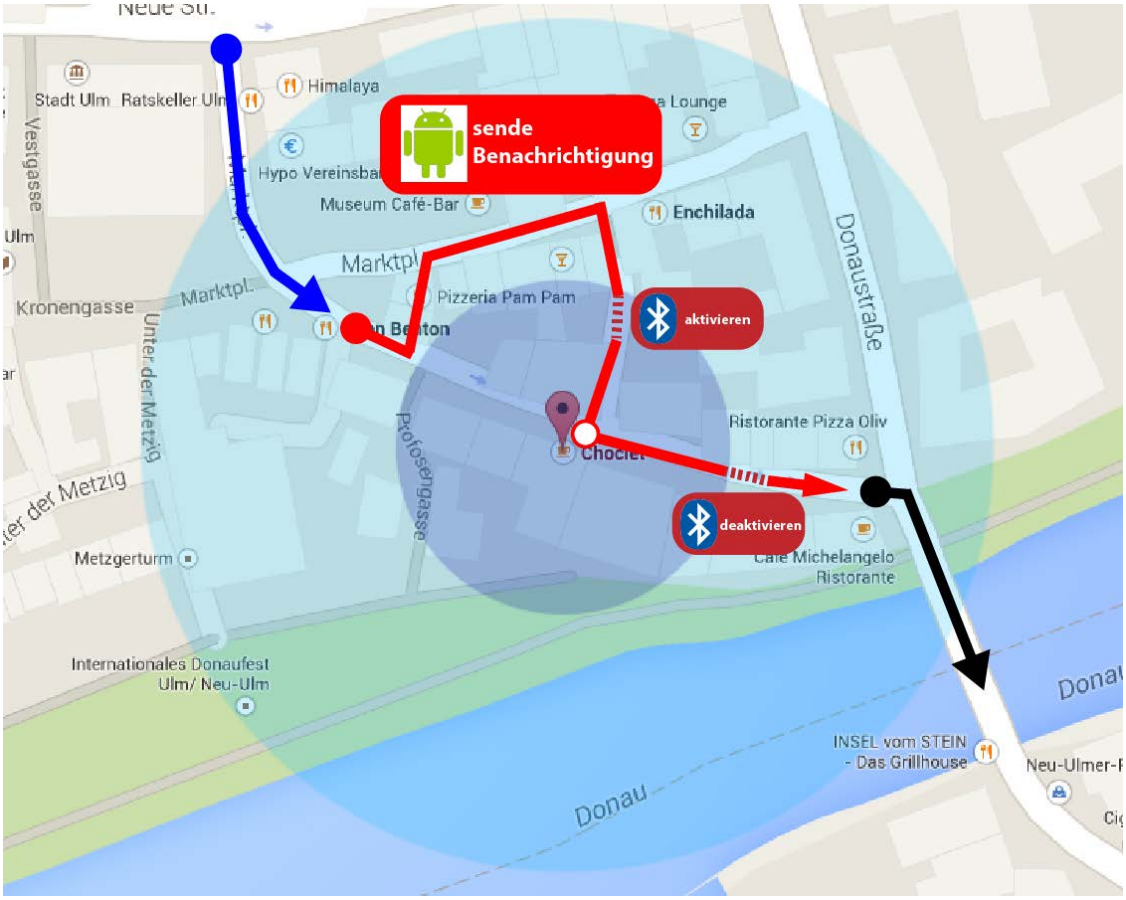

Abbildung 6.12: Schema der ausgelösten Events durch Geofence mit Steuerung der Bluetooth-Funktionen.

Bluetooth Funktionalität

Für die Anwendungsszenarien, die Gebrauch der BLE Funktion machen, wird die freie *AltBeacon-Bibliothek* verwendet [1], die alle nötigen Schnittstellen komfortabel zugänglich macht. Lediglich in Bezug auf Ressourcen-Gebrauch müssen im Sinne der Effizienz einige Modifikationen unternommen werden [8]. Da es sich außerdem um eine freie Bibliothek handelt, für diese Arbeit aber ausschließlich Beacons mit Apples iBeacons-Konfiguration verwendet werden, muss der Code

entsprechend angepasst werden, damit nur besagte Beacons von der Anwendung erkannt werden [14]:

Listing 6.3: Einstellung des Parsers für Beacons mit Apples iBeacon-Konfiguration.

```
1 beaconManager.getBeaconParsers().clear();
2 beaconManager.getBeaconParsers().add(new BeaconParser().
3   setBeaconLayout("m:0-3=4c000215,i:4-19,i:20-21,i:22-23,p:24-24"));
```

Beacons senden nur ein Signal aus, das prinzipiell nichts anderes beinhaltet als Informationen über das Beacon. Dazu gehören die Universally Unique Identifier („UUID“), sowie die Major- und Minor-Werte (siehe Erklärung in Kapitel 6.1.3, Seite 26). Wegen der einheitlichen UUIDs in diesem Kontext, können fremde Beacons schnell aussortiert werden und müssen nicht weiter die Rechenleistung des Gerätes beanspruchen. Da zur eindeutigen Identifizierung in unserem Kontext also nur noch die Major- und Minor-Werte verbleiben, stellt sich die Frage nach einer effizienten Datenstruktur, die aus einer Menge von Beacons auf performante Weise ein bestimmtes ausgeben kann.

Eine effiziente Methode, um dies zu bewerkstelligen sind *HashMaps* aus Java. HashMaps sind eine Menge von Schlüssel- und Wert-Paaren. Der Wert stellt hier das Beacon dar. Der Schlüssel ist die Kombination aus Major- und Minor-Wert. Um die Wert-Kombination nun kompatibel für HashMaps zu machen, wurde eine neue Java-Klasse implementiert. Diese Klasse stellt ein Index-Objekt dar, welches lediglich die Major- und Minor-Werte speichert. Dieses Objekt ist damit das für die HashMap benötigte Schlüssel-Objekt, wie Abbildung 6.13 zeigt.

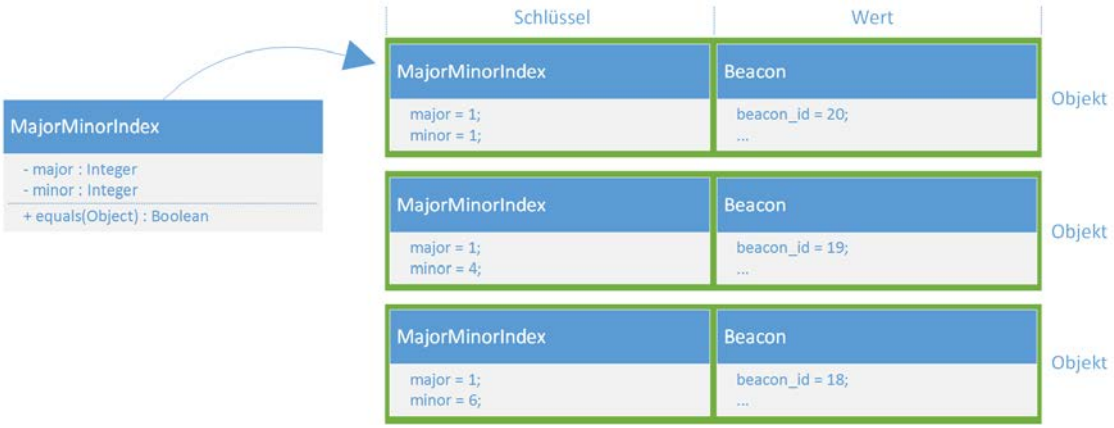

Abbildung 6.13: Modell eines MajorMinorIndex-Objekts und Schema einer Beacons-HashMap.

Mit diesem Wissen kann nun ein Blick auf die Methode geworfen werden, die aufgerufen wird, sobald ein Scanvorgang beendet wird und der interne Service des Geräts eine Menge an Beacons liefert. Ein Scan dauert hier vier Sekunden und pausiert dann für fünf Sekunden. Läuft die Anwendung nicht im Vordergrund, beträgt die Pause sogar zwei Minuten, um den Energieverbrauch stark zu senken.

In der HashMap *foundBeacons* sammelt und verwaltet die Anwendung alle Beacons, die zum Anwendungskontext gehören und sich in Reichweite des Gerätes befinden. Die folgende Methode wird für jedes aus dem Scan stammende Beacon sukzessive ausgeführt. Dabei wird mit der Datenbank abgeglichen, ob das Beacon vom Server definiert ist oder nicht. Erst dann wird es der HashMap mit allen Beacons aus dem letzten Scan hinzugefügt.

Listing 6.4: Ausschnitt aus Methode zur Filterung von Beacons nach einem Scanvorgang.

```
1  public static void processFoundBeacon(org.altbeacon.beacon.Beacon
       beacon){
2    if (hasSameUuid(beacon)){ // Nur fuer Beacons mit richtiger UUID
3      CustomBeacon newBeacon = beaconExistsLocally(beacon);   //
         Datenbankabgleich, ob Beacon gueltig
4      if(newBeacon != null){    // Beacon laut Datenbank gueltig
5        MajorMinorIndex index = new MajorMinorIndex(beacon.getId2().
           toInt(), beacon.getId3().toInt());    // Erstelle Index
6        foundBeacons.put(index, newBeacon);   // Fuege Beacon, mit Index
           , in Map hinzu
7        setLastSeenBeacon(foundBeacons.get(index)); // Mache Beacon fuer
            andere Methoden aufrufbar
8      }
9    }
10 }
```

Wurden die Beacons aus dem letzten Scan gefiltert, soll aus der Restmenge an gültigen Beacons nun der nächstgelegene herausgesucht werden. Der folgende Algorithmus soll dabei unter Berücksichtigung möglicher Ungenauigkeiten mit einem gewissen Maß an Gewissheit das Beacon ausfindig machen, an welchem der Benutzer der Anwendung gerade tatsächlich verweilt. Dies geschieht dadurch, dass ein Beacon zunächst 16 Sekunden lang bei jedem Scan als das nächstgelegene erkannt werden muss (dies entspricht drei Scanvorgängen). Erst dann wird der Ort des Beacons als Sitzplatz akzeptiert. Aufgrund der Zeitspanne werden Beacons als das nächstgelegene Beacon zwischengespeichert.

Hier muss beachtet werden, dass das nächstgelegene Beacon als Variable nicht zwangsläufig das selbe ist, wie die Beacon-Variable des Sitzplatzes. Konkret stelle man sich vor, wie man mit dem Smartphone durch das Lokal geht, was zur Folge hat, dass in diesem Zeitraum mehrere Beacon nacheinander als Sitzplatz erkannt werden würden, würde man nicht eine gewisse Verweildauer an einem Beacon voraussetzen. Abbildung 6.14 illustriert diesen Algorithmus, allerdings – zur besseren Übersicht – ohne die Verweildauer-Komponente.

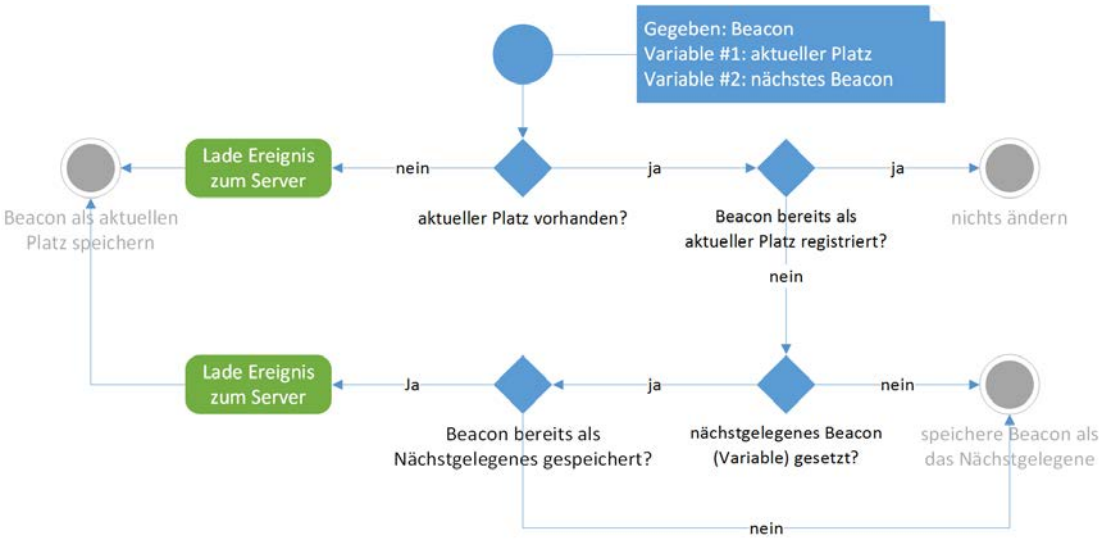

Abbildung 6.14: Aktivitätsdiagramm zeigt Algorithmus zur Sitzplatz-Bestimmung.

Inhaltssynchronisierung

Um die Inhalte auf der mobilen Anwendung mit denen vom Server zu synchronisieren und dabei möglichst wenig die Netzwerkverbindung zu belasten, werden zunächst die Versionsnummern der Kategorien, Produkte und Beacons vom Server geladen und mit den lokalen Nummern verglichen. Zum Verständnis illustriert die Abbildung 6.15 die Schnittstelle für die mobile Anwendung. Wie darauf zu sehen ist, gibt es für die mobile Anwendung nur eine POST-Anfrage an den Server, der zu einer Änderung an der Datenbank führt. Dazu aber mehr im anschließenden Kapitel auf Seite 40. Alle anderen GET-Anfragen dienen der Aktualisierung der lokalen Datenbank und der lokalen Produktbilder, wobei auffällt, dass die Adressen für die GET-Anfragen auf „.json“ enden. Dies hat den Grund, dass ein Aufruf ohne diese Endung sonst nur eine HTML-Seite liefern würde, wie sie die Websites der Administratoren brauchen.

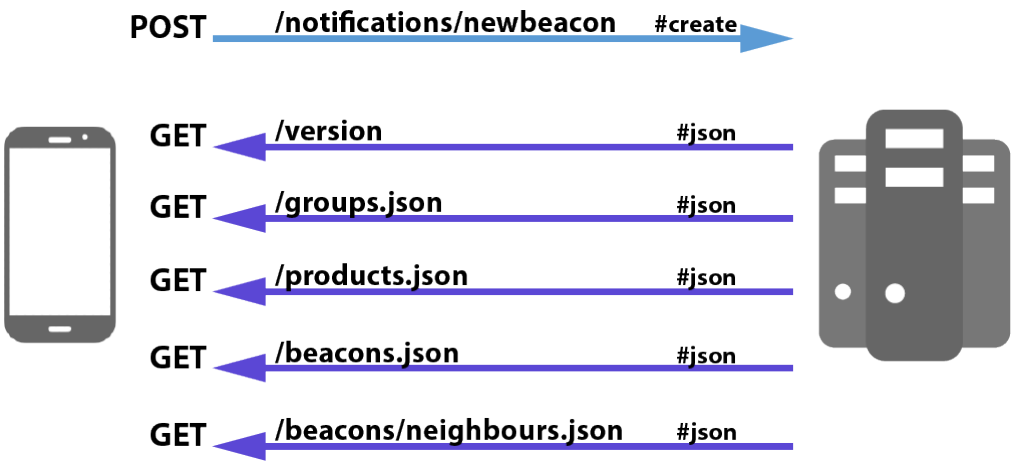

Abbildung 6.15: REST-konforme Schnittstelle für die mobile Anwendung.

Das Sequenzdiagramm aus Abbildung 6.16 zeigt, wie zunächst die notwendigen Versions-Informationen als JSON-Datei vom Server geladen werden. Anschließend werden die Versionsnummern verglichen und im Falle eines notwendigen Updates jeweils ein *AsyncTask* für Kategorien, Produkte und Beacons gestartet, welches im Hintergrund die Aktualisierung für die Datenbank lädt.

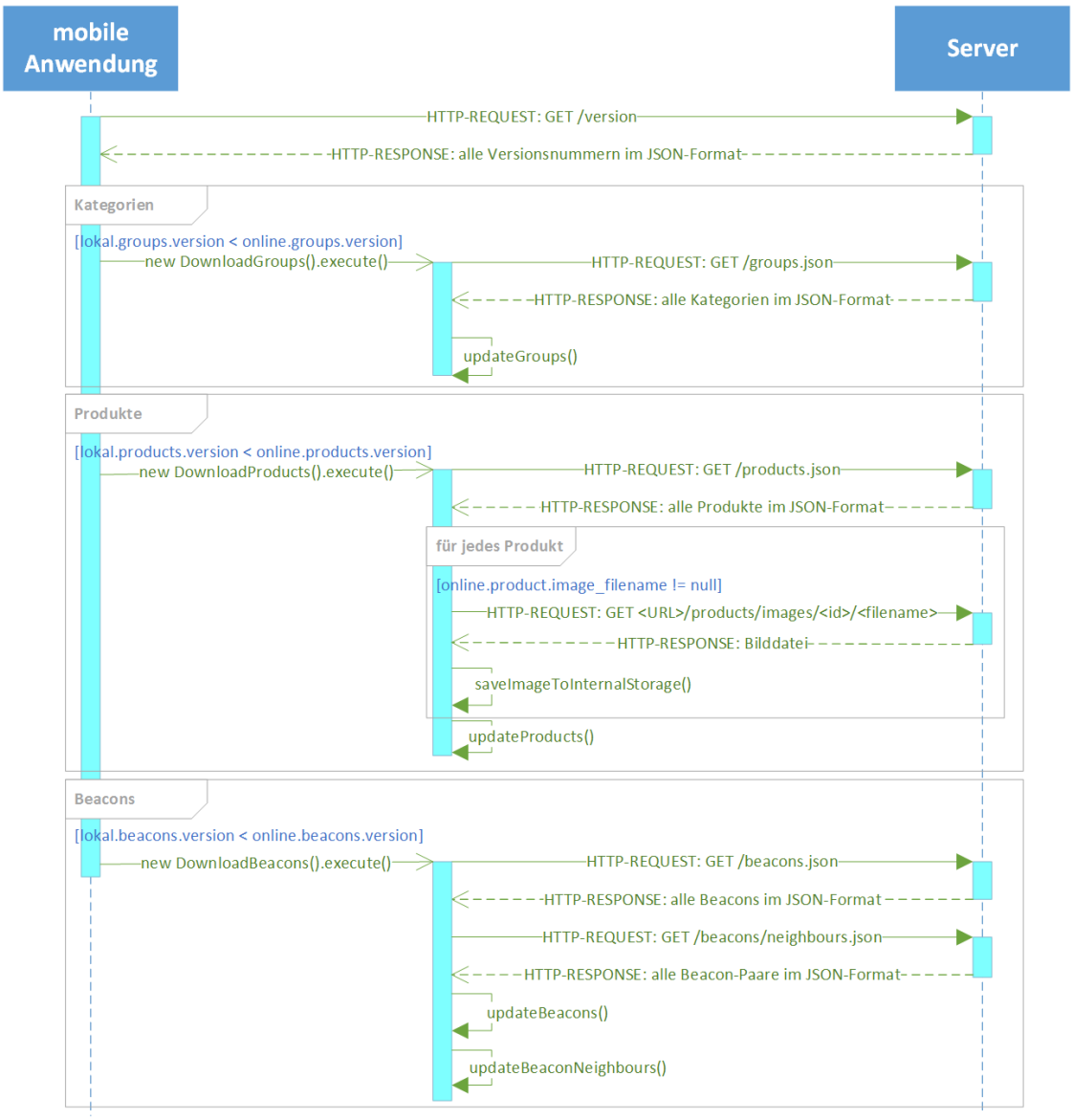

Abbildung 6.16: Sequenzdiagramm der vollständigen Synchronisation.

Sollen beispielsweise die lokalen Produkte einer Aktualisierung unterzogen werden, dann erhält das *AsyncTask* erneut ein JSON-Objekt, aus dem es die Informationen extrahieren und Produkt-Objekte bilden muss (siehe Ausschnitt 6.5). Eine Besonderheit hierbei ist, dass Produkte zugehörige Bilder haben können. In diesem Fall wird die Bild-Datei zunächst in Original-Größe heruntergeladen und anschließend in zwei Bildgrößen skaliert. Eine größere für eine Detail-Ansicht und eine kleinere für die Listenübersicht der Produkte. Diese werden dann lokal mit unterschiedlichen Dateinamen-Präfixen abgelegt.

Listing 6.5: Ausschnitt aus Methode zum Download von Produkten und Produktbildern.

```
JSONArray jArray = new JSONArray(result); // mache JSON-Datei lesbar
for(int i=0;i<jArray.length();i++){
  JSONObject temp = jArray.getJSONObject(i);
  Product p;  // erstelle Produkte aus JSON-Datei
  if(temp.getString("image_file_name").equals("null")){
    p = new Product(temp.getInt("id"), temp.getString("title"), temp.
        getString("description"), (float) temp.getDouble("price"),
        temp.getBoolean("visible"), temp.getInt("group_id"));
  } else {
    p = new Product(temp.getInt("id"), temp.getString("title"), temp.
        getString("description"), (float) temp.getDouble("price"),
        temp.getBoolean("visible"), temp.getInt("group_id"), temp.
        getString("image_file_name"));
  }
  if(p.getImage_file() != null){
    try {
      loadAndSaveImage(p.getProduct_id(), p.getImage_file());
    } catch (Exception e){
      p.setImage_file(null);
    }
  }
  products.add(p);
}
DataCenter.getDataSource().updateProducts(products, version.getNumber
    ());
```

Upload von Informationen

Für die Fälle, bei denen die mobile Anwendung einen Zahlungswunsch, eine Bestellbereitschaft oder Informationen über den Sitzplatz an den Server sendet, kommt die hierfür vorgesehene Klasse *UploadNewBeacon* zum Einsatz, welches ebenfalls als *AsyncTask* arbeitet, um im Hintergrund laufen zu können. Der Aufbau des Nachrichtenrumpfs („Body“) eines solchen HTTP-Request ist dabei immer gleich, da es stets aus vier Wert- und Schlüssel-Paaren besteht. Zum einen sind das die einzigartige Android Device ID und die Identifikationsnummer des als Sitzplatz erkannten Beacons. Darüber hinaus gibt es den „method_title“, dem nur eine Integer-Zahl zugeordnet ist. „2“ steht für Bestellbereitschaft, „1“ für Zahlungswunsch und „0“ repräsentiert nur Änderungen am Sitzplatz. Der Wert für „method_value“ gibt kontext-abhängig Auskunft über die letztendlich zu übertragende Information. Je nach Integer-Wert kann dies beim Zahlungswunsch Bar- oder Kartenzahlung bedeuten. Abbildung 6.17 verdeutlicht dies.

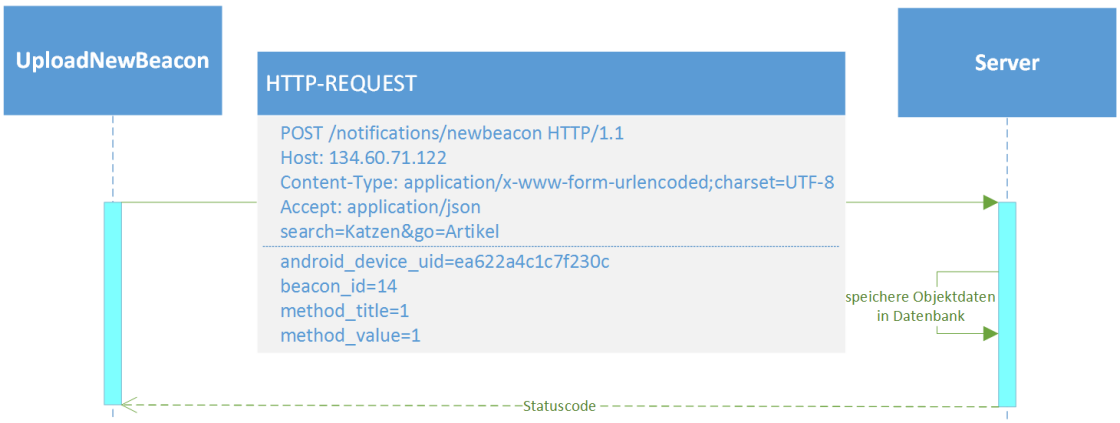

Abbildung 6.17: Sequenzdiagramm des Uploads eines Beacons mit Eventinformationen.

7 Anforderungsabgleich

Die mobile Anwendung zeigt sich nach Fertigstellung stabil und funktionsfähig. Zur Feststellung dieser Eigenschaften dienen verschiedene Android-Geräte als Test-Plattform, die über unterschiedliche Hardware und Hardware-Konfigurationen verfügen und sich in den Software-Versionen unterscheiden. Konkret wurde die Anwendung mit einem HTC Sensation, HTC One (M7) und HTC One Mini getestet, woraus sich keine Komplikationen ergaben. Beim HTC Sensation erscheint erwartungsgemäß eine Warnung, dass die Anwendung mangels Systemfunktionen nicht verwendet werden kann. Durch Berücksichtigung der wichtigsten Android Design Guide-Lines ist zudem für ein in sich stimmiges Gesamtbild gesorgt [12].

7.1 Funktionale Anforderungen

Im Folgenden sollen alle Anforderungen (siehe Kapitel 3, Seite 11) auf ihre Erfüllung hin analysiert werden und ihre Umsetzung in Prozentangaben bewertet werden. Sind Anforderungen nicht vollständig implementiert oder umgesetzt worden, gibt es außerdem eine Erläuterung diesbezüglich. Zunächst werden die Anforderungen für die mobile Anwendung abgeglichen, bevor der Abgleich für Anforderungen des Servers erfolgt.

7.1.1 Mobile Anwendung

Vollständig umgesetzt sind die Funktionen bezüglich der Bluetooth-, der Geofence- und der Ortungsfunktionalitäten. Diese funktionieren weitestgehend zuverlässig, allerdings sind die Benachrichtigungen bei Geofences nicht über den Server einstellbar, sondern sind fest im Quellcode programmiert. Die Reservierungen mit der exakten Angabe des Tisches wurden nicht implementiert, da aufgrund des Mangels an Zeit und zugunsten einer qualitativ besser umgesetzten Anwendung ein Teil der Anforderungen weggelassen werden muss. Die Entscheidung gegen die exakte Tisch-Reservierung fällt aufgrund der Bewertung auf Seite 6.

Tabelle 7.1: Anforderungsabgleich für funktionale Anforderungen der mobilen Anwendung

Anforderung	Umsetzung	Kommentar
#1: Geofence	100%	–
#2: Kontext-sensitive Benachrichtigung	80%	Benachrichtigungen sind nicht über den Server einstellbar, sondern hard-coded.
#3: Begrüßung durch Anwendung	100%	–
#4: Indoor-Ortung	100%	–
#5: Mitteilungen an Server via Anwendung	100%	–
#6: Mitteilung über Zahlungswunsch	100%	–
#7: Menükarte	100%	–
#8: Digitaler Notizzettel mit Bereitschafts-Ruf	100%	–
#9: Reservierung mit Tischnummer	0%	Aufgrund von Zeitmangel nicht umgesetzt.
#10: Einstellungen	100%	–

7.1.2 Server

Die wichtigsten Funktionen sind am Server vollständig implementiert, wenngleich einige Details und Verbesserungen nicht umgesetzt sind (mehr dazu in Kapitel 9.2, *Ausblick*). Die anfangs gewünschte Website zur Verwaltung der Geofences ist hingegen nicht programmiert. Da eine solche Website nur Sinn ergibt, wenn die Bedienbarkeit mit Integration einer Landkarte intuitiv gestaltet ist, ergab sich ein Problem mit dem Zeitrahmen, sodass die Anforderung zugunsten der Gesamtqualität fallengelassen wurde.

Tabelle 7.2: Anforderungsabgleich für funktionale Anforderungen des Servers

Anforderung	Umsetzung	Kommentar
#1: Konten-System	70%	Benutzerverwaltung nicht vorhanden, sodass sich jeder registrieren kann.
#2: Kategorien	90%	Es ist möglich, bei der Zuordnung von Ober- und Untergruppen einen geschlossenen Kreis zu bilden.
#3: Produkte	100%	–
#4: Beacons	100%	–
#5: Benachrichtigungen	100%	–
#6: Geofence	0%	Wegen des Aufwandes wurde keine Website zur Verwaltung der Geofences erstellt.
#7: RESTful API und JSON	100%	–

7.2 Nicht-Funktionale Anforderungen

Die nicht-funktionalen Anforderungen, die die Eigenschaften der Anwendung beschreiben, sind zum Großteil erfüllt. Insbesondere Stabilität und Robustheit, die besonders wichtig sind, konnten in hohem Maße erzielt werden. Dank des Web-Frameworks Ruby on Rails und eine ebenso strikte Einhaltung mit dem Android Studio kann das MVC-Prinzip besonders gut umgesetzt werden. Lediglich das Passwort, welches der Server für die Anmeldung an der Website benötigt, sollte zur Verbesserung der Datensicherheit verschlüsselt werden, bevor es an den Server gesendet wird, anstatt es nachträglich zu verschlüsseln.

Wegen Kompatibilitätsproblemen musste die Kompatibilität der mobilen Anwendung eingeschränkt werden, um eine einwandfreie Funktionsweise garantieren zu können. Mehr dazu im Kapitel 9.1, *Hürden und Herausforderungen*.

Tabelle 7.3: Anforderungsabgleich für funktionale Anforderungen der mobilen Anwendung

Anforderung	**Umsetzung**	**Kommentar**
#1: Hohe Kompatibilität	80%	Der geforderte, minimale API-Level 18 musste auf API-Level 20 angehoben werden.
#2: Datensicherheit	90%	Das vom Benutzer gesendete Passwort sollte bereits gehashed an den Server gesendet werden.
#3: Einhaltung des MVC-Prinzips	100%	–
#4: Plattform-Kompatibilitäten	100%	–
#5: Performanz und Effizienz	90%	Verbesserungen möglich bei Versionierung und Synchonisation.
#6: Stabilität und Robustheit	95%	Trotz nicht aufgetretener Fehler können unerwartete Instabilitäten nicht ausgeschlossen werden.
#7: Intuitive und funktionale Benutzeroberfläche	99%	Perfekte Bedienbarkeit ist nicht erreichbar.
#8: Ansprechendes Design	99%	Kein Design ist perfekt.

8 Diskussion

In diesem Kapitel sollen einige schwierige oder kontroverse Themen diskutiert werden, die in Verbindung mit dieser Arbeit stehen.

8.1 Nativ oder Hybrid?

Am Anfang jeder Entwicklung mobiler Anwendungen stellt sich die Frage nach der Plattform-Kompatibilität. Prinzipiell muss man hier zwischen einer nativen und einer hybriden Entwicklung unterscheiden. Eine native Entwicklung bedeutet, dass ein Programm direkten Zugriff auf alle Funktionen eines Betriebssystems, wie Kamera oder Standortbestimmung, besitzt. Dahingehend muss man für eine native Anwendung für Android beispielsweise in Java, für iPhones in Objective-C (bzw. Swift) und für Windows Phone in C# programmieren [6]. Möchte man also wie im Beispiel einer mobilen Anwendung für die Gastronomie möglichst viele Benutzer erreichen können, sollte die Anwendung auf möglichst allen gängigen mobilen Betriebssystemen verfügbar sein.

Weil aber wegen unterschiedlicher Programmiersprachen die Anwendung mehrfach von Grund auf programmiert werden muss, ist mit einem enorm hohen Aufwand zu rechnen. Zusätzlich ist der Aufwand für Wartung und Aktualisierungen sehr hoch, was oft mit wirtschaftlichen Interessen kollidiert.

Nativ: Vor- und Nachteile

+ Zugriff auf alle Funktionen eines Betriebssystems.

+ Effizienteste Nutzung von Ressourcen.

+ Beste Anpassbarkeit an technische Gegebenheiten.

– Hohe Arbeitsaufwand für Programmierung mehrerer Betriebssysteme.

– Hoher Aufwand für Aktualisierung einer Anwendung, auch bedingt durch Aktualisierungen des Betriebssystems.

– Ähnliche Gestaltung für verschiedene Plattformen schwierig.

Im Gegensatz hierzu stehen hybride Anwendungen, die prinzipiell nur eine Web-Anwendung beinhalten, die mit HTML5, CSS und JavaScript programmiert ist. Für jedes Betriebssystem existiert dann eine native Anwendung, die letztendlich nur wie ein Browser funktioniert und

die Web-Anwendung anzeigt. Allerdings kann auf diverse Geräte-Funktionalitäten zugegriffen werden, wenn man für diese nativen Rahmen-Anwendungen eine entsprechende Schnittstelle programmiert oder eines der bereits erhältlichen Frameworks wie *PhoneGap* einsetzt. Da die eigentlichen Funktionen der mobilen Anwendung aber in der Web-Anwendung stecken, ist eine hybride Programmierung nicht so performant und schnell wie native Lösungen, da hier quasi über den integrierten Browser kommuniziert werden muss.

Andererseits ist die Aktualisierung der mobilen Anwendungen verhältnismäßig einfach und schnell, weil oftmals lediglich nur die Web-Anwendung geändert werden muss, welche von den nativen Anwendungen geladen wird.

Hybrid: Vor- und Nachteile

+ Deutlich geringerer Aufwand der Programmierung für mehrere Plattformen.
+ Erheblich leichtere Wartung und Pflege.
+ Optisch identische Benutzeroberflächen.
– Deutlich eingeschränkter Zugriff auf Geräte-Funktionen.
– Eingeschränkte Möglichkeiten für Offline-Nutzung.

Fazit

Es zeigt sich, dass es keine eindeutige Antwort auf diese Problematik gibt. Je nach Anwendungskontext muss abgewogen werden, welcher Ansatz in Frage kommt und sinnvoll erscheint. Im Fall dieser Gastronomie-Anwendung muss insbesondere wegen der Bluetooth- und Geofence-Funktionalitäten nativ entwickelt werden, da die Möglichkeiten bei hybriden Anwendungen zu eingeschränkt und nicht im Sinne einer ansprechenden Anwendung sind.

Allerdings muss man ein Augenmerk auf sogenannte *cross compiler*, wie dem von *Xamarin*, richten. Auf diese Weise kann man mit der Programmiersprache C# Anwendungen für Android, iOS und Windows Phone schreiben, die dann den gemeinsamen Quellcode verwenden. Der Entwickler entwickelt also ein Programm in einer Sprache, das der Compiler dann in die für alle Plattformen notwendige Sprache übersetzt. Lediglich die Benutzeroberfläche muss separat gestaltet werden.

Angesichts der immensen Verbreitung von Smartphones und Tablets erscheint ein solcher Ansatz wenn zumindest vielleicht nicht als endgültige Lösung des Problems, dann zumindest doch als vielversprechender Ansatz für die zukünftige Entwicklung.

8.2 Genauigkeit gegen Energieverbrauch

Bei der Entwicklung mit Systemfunktionen, die Gebrauch der Gerätekomponenten und -hardware machen (wie Blueooth, GPS oder Netzwerk-Konnektivität), stellt sich im Laufe der Programmie-

rung die Frage, ob man zugunsten besserer Ergebnisse und genauerer Daten oder im Sinne der Akkulaufzeit entwickelt.

Natürlich ist für den Besitzer eines Smartphones nicht sofort ersichtlich, wieviel Ressourcen eine Anwendung benötigt, sodass es sich Entwickler auch einfach machen können und für die bestmögliche Benutzer-Erfahrung rücksichtslos auf die Ressourcen zugreifen. Allerdings ist das nicht nur schlechter Programmierstil, sondern trägt auch dazu bei, dass der Schaden groß ist, wenn der Benutzer den enormen Energieverbrauch bemerkt. Im besten Fall wird nur der Benutzer und keiner seiner Bekannten die Anwendung wieder deinstallieren.

In dieser Arbeit beispielsweise sind energielastige Funktionen deaktiviert, bis sie tatsächlich gebraucht werden. Konkret wird die Bluetooth-Funktionalität erst dann aktiv, sobald man in die Nähe von Beacons kommt. Dies lässt sich über Geofences erkennen. Der Verbrauch kann also nicht nur über Veränderungen von Parametern, sondern auch durch strukturelle Mechanismen gesteuert und gesenkt werden.

Es gilt also den Mittelweg zu finden, sich genau soviele Zugriffe auf Systemfunktionen zu „gönnen“, dass die Benutzer-Erfahrung noch ein gewünschtes Mindestmaß erreicht. Eine große Verbesserung dieses Umstands ist beispielsweise, wie über den Google Play Service Ortungsdaten abgerufen werden können. Anstelle mehrerer Anwendungen, die alle Gebrauch der GPS-Funktion machen, ist es möglich, über genannten Service die Ortungsdaten zu verwenden, die von allen installierten Anwendungen gesammelt werden. So kann der Zugriff der Anwendung auf Hardwarekomponenten drastisch gesunken werden, solange es andere Anwendungen gibt, die in kleineren Intervallen Ortungsdaten beziehen.

8.3 Datenschutz und Privatsphäre

Ein heikles Thema bei Anwendungen, die Informationen über den Benutzer, wie z.B. seine Position, sammeln, ist auch immer die Frage der Privatsphäre und des Datenschutzes. In erster Linie weiß ein Benutzer natürlich nicht, wie eine Anwendung mit seinen Daten umgeht und wie sicher sie sind. Im schlimmsten Fall ist das Misstrauen so groß, dass die Anwendung deinstalliert wird. Für Entwickler ist es natürlich ein großer Vorteil, die Anwendung zu personalisieren, indem auf Informationen des Benutzers zurückgegriffen wird. Gerade dabei aber kann ein Benutzer sehen, welche Informationen er preisgeben muss, um eine Anwendung verwenden zu können. Der Nutzen einer Anwendung oder Funktion muss also für den Benutzer groß genug sein, damit er die Preisgabe seiner Daten toleriert.

Das ändert natürlich nichts an der Tatsache, dass es die Pflicht des Entwicklers ist, die verwendeten Daten zu schützen und beispielsweise nicht unverschlüsselt über ein Netzwerk zu versenden. Die Offenlegung von Informationen, wie die Daten des Benutzers verwendet und geschützt werden, schafft Transparenz, was wiederum Vertrauen schafft. Und eine vertrauenswürdige Anwendung ist besser als eine nicht vertrauenswürdige Anwendung.

In dieser Arbeit wurde das Problem mit der Privatsphäre so gelöst, dass empfindliche Daten, wie dem Standort, nur lokal verwendet und bearbeitet werden. Lediglich die Android Device ID, die bei der erstmaligen Inbetriebnahme des Smartphones erstellt wird, wird über das Internet zum Server gesendet. Aus dieser Identifikationsnummer können aber keine Rückschlüsse auf den Besitzer des Smartphones gezogen werden. So ist auch die Zuordnung auf dem Server von Android Device ID zu einem Beacon relativ unproblematisch.

9 Zusammenfassung

Nachdem nun die praktischen, theoretischen und technischen Aspekte umfassend erläutert sind, sollen in diesem Kapitel die Schwierigkeiten während der Entwicklung geschildert und eventuelle Lösungen erklärt werden. Anschließen soll ein Ausblick über die mögliche, zukünftige Weiterentwicklung gegeben werden bevor diese Arbeit in einem persönlichen Fazit ihren Abschluss findet.

9.1 Hürden und Herausforderungen

Die Auseinandersetzung mit BLE, aber auch mit Geofences war zwar herausfordernd, allerdings war die Arbeit an der Umsetzung der Anwendungsfälle sehr spannend und interessant, insbesondere wenn es darum ging, Prozesse effizienter und sparsamer zu gestalten. Problematisch war allerdings die Feststellung, dass die in der Version 4.3, also API-Level 18, eingeführte Plattform-Unterstützung für Bluetooth 4.0 mit Low-Energy nicht für alle Smartphone-Geräte einwandfrei funktioniert. Die Verbesserungen in der darauffolgenden Version 5.0, respektiv API-Level 20, waren für das Weiterarbeiten während dieser Arbeit unerlässlich und erleichterten außerdem dank vereinfachter Schnittstellen den Umgang mit BLE. Dennoch war die Recherche für Problemlösungen vor der Erhöhung des notwendigen API-Levels zeitaufwendig. Darüber hinaus funktionieren die Beacons nicht immer zuverlässig und die Signalqualität in der Anwendung erlaubt oftmals keine gute Distanzschätzung, was neben der teilweise komplexen Gestaltung der API der Hardware ein großer Kritikpunkt ist. Die Verwendung einer Bibliothek schafft zwar Abhilfe, um die wichtigsten Funktionen leicht verwendbar zu machen, allerdings schränkt dies auch die Möglichkeiten ein, um beispielsweise *RejectedExecutionExceptions* zu behandeln, wenn zu viele Tasks im Hintergrund arbeiten müssen. Hier gibt es noch Spielraum für Optimierungen am Quellcode.

Als Herausforderung entpuppte sich auch die Portierung des Projekts von Eclipse zu Android Studio, welches im Laufe der Abschlussarbeit als stabile Version veröffentlicht wurde und zu dessen Wechsel ausdrücklich von Google empfohlen wird. Da das Projekt bereits fortgeschritten war, waren einige Einstellungen für einen compilierbaren Code und eine Einarbeitung in die Werkzeuge von Android Studio notwendig.

Auch die Lernzeit für eine neue Programmiersprache, nämlich Ruby, offenbarte sich als anstrengend, zumal das Konzept des „convention over configuration“ in Kombination mit dem Web-Framework Ruby on Rails einiges an Gewöhnung erforderte, was auch für Aufbau und Struktur von Ruby on Rails selbst gilt.

9.2 Ausblick

In dieser Arbeit werden drei Beacons verwendet, um die Anwendung testen und programmieren zu können. Trotz großer Bemühungen im Sinne der Performanz, muss in der Praxis geprüft werden, wie sich die Anwendung bei einer Zahl von bis zu 50 Beacons in einem Lokal verhält und welche Optimierungen umsetzbar sind. Schon jetzt aber ist klar, dass sich die Belastung der Netzwerkfunktion des Smartphones noch weiter verbessern lässt, wenn man die Versionierung der Datenbank auf dem Server feinkörniger gestaltet. Anstelle der Aktualisierung von Tabellen, ist es also erstrebenswert, eine Aktualisierung einzelner Datenbankeinträge zu ermöglichen.

Auch sollte der Mechanismus, der leere Batterien von Beacons erkennt und an den Server sendet, fertig implementiert werden. Die Datenstruktur ist bereits gegeben (siehe Abbildung 6.4, Seite 27), allerdings muss die Logik noch programmiert werden. Es ist davon auszugehen, dass ein solcher Mechanismus enormen Wert für den Betriebsablauf darstellt.

Was die digitale Menükarte betrifft, so gibt es sicherlich noch Spielraum für Verbesserungen. Beispielsweise sollte es unterbunden werden, dass Kategorien hinsichtlich ihrer Beziehungen untereinander einen Kreis bilden können, was sonst unter Umständen zu unerwarteten Daten-Inkonsistenzen führen kann. Auch die Modularität von realen Speise- und Getränkekarten sollte besser abgebildet werden können.

Nicht zuletzt muss auch die Benutzerverwaltung fertig gestellt werden. Auch wenn dies nicht für die geforderten Anwendungsszenarien unbedingt notwendig ist, sollte es bei einer Weiterentwicklung eine große Priorität genießen, um das gesamte System praxistauglich zu machen.

9.3 Fazit

Die Arbeit an einer mobilen Anwendung für die Gastronomie hat gezeigt: Der Markt ist da, die Anbieter überschaubar und die sinnvollen Anwendungsszenarien vorhanden. Nicht nur war die Beschäftigung mit Geofences und der Bluetooth-Technologie sehr spannend, auch den Mehrwert einer solchen Anwendung in der Praxis zu erfahren, entpuppte sich als ein erfrischendes Erlebnis. Zwar war es mühsam, eine funktionierende Grundlage zu schaffen und sich in eine neue Programmiersprache, ein unbekanntes Web-Framework und eine neue Entwicklungsumgebung einzuarbeiten, doch das Ergebnis und die gewonnenen Erfahrungen entschädigen. Persönlich sehr schade allerdings finde ich es, dass der Mechanismus zur automatischen Erkennung von leeren Batterien in den Beacons nicht fertig implementiert werden konnte. Ob die Theorie in der Praxis funktionieren würde, wäre für mich sehr interessant gewesen.

Was allerdings auch nicht vernachlässigt werden darf, ist, dass auch Thematiken ans Tageslicht traten, die zu einer kontroverse Auseinandersetzung führen können. Beispielsweise muss man als Entwickler einer mobilen Anwendung für die Gastronomie auch wissen, dass die Integration von digitalen Bestellungen Arbeitsplätze für Menschen gefährdet. Gerade in dieser Branche arbeiten wegen guter Bezahlung und flexibler Arbeitszeiten viele Studenten und finanzieren damit

ihren Alltag. Auch wenn es technisch gesehen mit Beacons allein gar nicht erst einfach umgesetzt werden kann, gelangt man früher oder später mit der Verbesserung der Technik mit dieser Frage an seine Grenze der gesellschaftlichen Verantwortung und der Moral.

Abbildungsverzeichnis

Literaturverzeichnis

[1] *AltBeacon - The Open and Interoperable Proximity Beacon Specification*. http://altbeacon.org/. – (zuletzt besucht am 19.04.2015)

[2] *Bootstrap · The world's most popular mobile-first and responsive front-end framework.* http://getbootstrap.com/

[3] *Creating and Monitoring Geofences | Android Developers*. http://developer.android.com/training/location/geofencing.html. – (zuletzt besucht am 15.04.2015)

[4] *LocationRequest | Android Developers*. https://developer.android.com/reference/com/google/android/gms/location/LocationRequest.html. – (zuletzt besucht am 21.04.2015)

[5] *Smartphone OS sales market share | Kantar Worldpanel ComTech*. http://www.kantarworldpanel.com/smartphone-os-market-share/. – (zuletzt besucht am 19.04.2015)

[6] BEIER, Andreas ; LINKE, Andreas ; SCHULZ, Hajo: Die eigene App: Entwickeln für Android, iPhone, WebOS, Symbian, Blackberry und Windows Mobile. In: *c't magazin* 16 (2010), S. 90–95

[7] BRÜNKEN, Roland ; SEUFERT, Tina ; JÄNEN, Inge: Multimodales Lernen. In: *Pädagogische Psychologie in Theorie und Praxis*. Hogrefe Verlag, 2008, S. 135–138

[8] FALK, Michael: Messen und Beeinflussen des Energieverbrauchs von Android-Systemen. In: *Energy-Efficient Applications* (2012), S. 33

[9] GOMEZ, Carles ; OLLER, Joaquim ; PARADELLS, Josep: Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. In: *Sensors* 12 (2012), Nr. 9, S. 11734–11753

[10] KALALI, Masoud ; MEHTA, Bhakti: *Developing RESTful Services with JAX-RS 2.0, WebSockets, and JSON*. Packt Publishing Ltd, 2013

[11] NAMIOT, Dmitry ; SNEPS-SNEPPE, Manfred: Geofence and network proximity. In: *Internet of Things, Smart Spaces, and Next Generation Networking*. Springer, 2013, S. 117–127

[12] NOHEJL, Petr: *Android Cheatsheet for Graphic Designers*. http://petrnohejl.github.io/Android-Cheatsheet-For-Graphic-Designers. – (zuletzt besucht am 07.03.2015)

[13] PANG, Wei: *Web Application Development: Advanced Ruby on Rails - User authentication*. http://homepages.abdn.ac.uk/pang.wei/pages/WAD/Practicals/userauthentication/Web%20Application%20Development%20%20Advanced%20Ruby%20on%20Rails%20-%20User%20authentication.html. – (zuletzt besucht am 19.04.2015)

[14] PHILIPP, Martin: *easiBeacons with AltBeacons Lib*. http://www.martin-philipp.de/2014/10/29/easibeacons-with-altbeacons-lib/. – (zuletzt besucht am 19.04.2015)

[15] PRYSS, Rüdiger ; MUSIOL, Steffen ; REICHERT, Manfred: Integrating Mobile Tasks with Business Processes: A Self-Healing Approach. In: *Handbook of Research on Architectural Trends in Service-Driven Computing*. 2014, S. 103–135

[16] SCHICKLER, Marc ; PRYSS, Rüdiger ; SCHOBEL, Johannes ; REICHERT, Manfred: An Engine Enabling Location-based Mobile Augmented Reality Applications. Version: 2015. http://dbis.eprints.uni-ulm.de/1137/. In: *Web Information Systems and Technologies - 10th International Conference, WEBIST 2014, Barcelona, Spain, April 3-5, 2014, Revised Selected Papers*. Springer, 2015 (LNBIP)

[17] SCHMITZ, Lars ; TEGEDER, Alexander: Mobile Kunden mit ortsbezogenen Nachrichten bewerben. In: *Marktplätze im Umbruch*. Springer, 2015, S. 133–140

[18] THOMAS, Dave ; HANSSON, David H.: *Agile Web Development with Rails (2nd edition)*. Pragmatic Bookshelf, 2006

[19] ZAITSEV, Dmitry: *dmitry-zaitsev/AndroidSideMenu*. https://github.com/dmitry-zaitsev/AndroidSideMenu. – (zuletzt besucht am 19.04.2015)

Name: Alpay Artun Matrikelnummer: 778552

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Alpay Artun