HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

BTZ

BTZ Reservations-App

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2012

Projekttitel: BTZ Reservations-App
Autoren: Patrizia Heer und Simon Stäheli
Betreuer: Prof. Dr. Markus Stolze, IFS
Projektpartner: Super Computing Systems AG, Zürich [SCS]
Industriepartner: Behinderten-Transporte Zürich [BTZ]

Inhaltsverzeichnis

Kapitel 1

Allgemein

1.1 Erklärung über die eigenständige Arbeit

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.
- dass wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Rapperswil, 21.12.2012

Patrizia Heer

Simon Stäheli

1.2 Aufgabenstellung

Aufgabenstellung Studienarbeit Abteilung I, HS 2012/13 Patrizia Heer & Simon Stäheli

BTZ Reservations-App

1. Auftraggeber und Betreuer

Praxispartner und Auftraggeber diese Studienarbeit ist

SCS (Super Computing Systems)
Technoparkstrasse 1
8005 Zürich
Schweiz

Ansprechpartner Auftraggeber:

Julian Heeb
julian.heeb@scs.ch

Betreuer HSR:

Prof. Dr. Markus Stolze, Institut für Software mstolze@hsr.ch
Sebastian Hunkeler, Assistent, Institut für Software, (iOS Engineering) shunkele@hsr.ch

2. Ausgangslage

Die Abkürzung BTZ steht für die Stiftung Behinderten-Transporte Zürich. Die BTZ transportiert mobilitätsbehinderte Menschen aus dem Kanton Zürich.

Die SCS steht mit dem BTZ in Kontakt, da Julian Heeb selbst Benutzer des BTZ Services ist. Der Wunsch nach einer iPhone App wurde ursprünglich von Julian Heeb geäussert, womit er bei der BTZ auf offene Ohren stiess. Da die BTZ eine Stiftung ist und nur bedingt finanzielle Mittel vorhanden sind, kam die Idee für eine solche App in Zusammenarbeit mit einer FH als Semesterarbeit zu realisieren.

3. Ziele der Arbeit

Im Rahmen dieser Studienarbeit sollen die folgenden Aufgaben bearbeitet werden:

- Die Bedürfnisse von direkten und indirekten Nutzern der BTZ App sollen analysiert werden. Insbesondere sind die Rollen von SCS und BTZ im Projekt klar zu definieren. Zudem muss der Arbeitsfluss der Reservationsentgegennahme (aktuell und zukünftig) in verschiedenen konkreten Beispielen dokumentiert werden
- Ein Interaktionskonzept (UI Design) für die BTZApp ist zu entwickeln. Dieses sollte auf die identifizierten Bedürfnisse von Nutzern ausgelegt sein, und die Möglichkeiten von iOS sinnvoll nutzen. Das heisst, es muss plausibel argumentiert werden, warum die Lösung einen Vorteil gegenüber der aktuellen Lösung (Bestellen mit dem Telefon) bringt. Das vorgeschlagene Interaktions-Konzept sollte mit möglichst einfachen Mitteln getestet werden und mit den Auftraggebern und Kunden ein zu implementierendes UI-Design beschlossen werden (z.B. Paper Prototyping).
- Das UI Design ist mit Hilfe des iOS SDK zu implementieren. Das interne Design und die Architektur des Systems sollen nachvollziehbar aus den Benutzerbedürfnissen und Technologie-Constraints abgeleitet werden. Hierbei sollte nicht nur eine hohe Benutzbarkeit des Systems für die Benutzer-Zielgruppe angestrebt werden, sondern auch eine angemessene Stabilität und Wartbarkeit sicher gestellt werden.
- Dem Auftraggeber ist ein lauffähiges und getestetes System, inklusive Dokumentation für die Wartung und Installationsanleitung abzuliefern.
- Für die HSR ist eine Dokumentation zur Studienarbeit (siehe unten) zu erstellen, sowie ein Video.

Die folgenden Software Komponenten sollen implementiert werden:

- Eine iPhone App (iOS 6) für das iPhone 3GS und besser.
- Eine korrespondierende Server Komponente, welche Reservationen von den iPhone Clients entgegen nimmt und in einer Datenbank verfügbar macht.
- [Optional] Call Center Software, welche die Sichtung und Abarbeitung der eingegangenen Reservationen ermöglicht.
- [Optional] Push Benachrichtigungen über bestätigte oder abgelehnte Reservationen

4. Zur Durchführung

Mit dem HSR-Betreuer finden in der Regel wöchentliche Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf durch die Studierenden zu veranlassen.

Alle Besprechungen sind von den Studenten mit einer Traktandenliste vorzubereiten und die Ergebnisse in einem Protokoll zu dokumentieren, welches im Projekt-Repository stets zugreifbar ist.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen (gemäss Projektplan) sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben. Über die abgegebenen Arbeitsresultate erhalten die Studierenden ein Feedback. Eine definitive Beurteilung erfolgt aufgrund der am Abgabetermin abgelieferten Dokumentation. Die Evaluation erfolgt aufgrund des separat abgegebenen Kriterienkatalogs in Übereinstimmung mit den Kriterien zur SA Beurteilung. Es sollten hierbei auch die Hinweise aus dem abgegebenen Dokument „Tipps für die Strukturierung und Planung von Studien-, Diplom- und Bachelorarbeiten" beachtet werden.

5. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen. Die zu erstellenden Dokumente sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollen den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Die Dokumentation ist vollständig auf CD/DVD in 2 Exemplaren abzugeben.

Zudem ist eine kurze Projektresultatdokumentation im öffentlichen Wiki von Prof. M. Stolze zu erstellen. Diese muss einen Link auf ein öffentlich zugängliches (z.B. YouTube) „Video Poster" enthalten, welche das Resultat der Arbeit dokumentiert (das Video sollte auch im Original auf der CD/DVD enthalten sein). Dieses Video sollte nicht mehr als 2 Minuten lang sein. Das Video enthält einen Intro-Screen mit HSR-Logo, Titel der Arbeit, Namen der Studenten und Namen des Betreuers/Dozenten und Industriepartner (Standbild PPT Folie im HSR Corporate Design, 5 sec). Im ersten Inhaltsteil des Videos (30 sec) wird erklärt warum das gelöste Problem relevant ist: Warum braucht es das System, warum ist die Arbeit ohne System mühsam oder unmöglich? Der zweite Inhaltsteil des Videos (max. 50 sec) gibt eine Systemübersicht und zeigt anhand eines Einsatzbeispiels was das Resultat dieser Arbeit wem bringt. Abschluss mit Zusammenfassung der "Nutzer-Benefits" auf PPT Standbild. Im dritten Inhaltsteil (max. 30 sec) werden die „Errungenschaften" der Arbeit prägnant dargestellt. Insbesondere werden gemeisterte technische und organisatorische Herausforderungen aufgelistet (z.B. Architektur auf PPT Folie im HSR Corporate Design). Im Abspann (5 sec) sollte der Intro-Screen wieder eingeblendet werden. Das Material für die Video-Erstellung wird von der HSR gestellt (Multimediastelle). Das Video ist mit Untertiteln auszustatten. Auf eine Tonspur sowie Musik ist zu verzichten. Beispiele von HSR-Videos zu verschiedenen Arbeiten finden Sie bei YouTube (z.B. Kinect Bodyscanner http://www.youtube.com/watch?v=Q1ngxAkiaRg)

6. Termine

Siehe auch Terminplan auf
https://www.hsr.ch/Termine-Diplom-Bachelor-und.5142.0.html

17.09.2012 Beginn der Studienarbeit, Ausgabe der Aufgabenstellung durch die Betreuer.

21.12.2012 Abgabe des Berichtes an den Betreuer bis 12.00 Uhr.

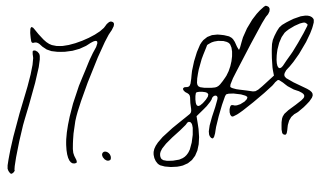
7. Beurteilung

Eine erfolgreiche SA zählt 8 ECTS-Punkte pro Studierenden. Für 1 ECTS Punkt ist eine Arbeitsleistung von ca. 25 bis 30 Stunden budgetiert. Dies entspricht ungefähr 17h pro Woche (auf 14 Wochen) und damit ca. 2 Tage Arbeit pro Woche.

Für die Beurteilung ist der HSR-Betreuer verantwortlich.

Die Bewertung der Arbeit erfolgt entsprechend der verteilten Kriterien-Liste.

Rapperswil, den 17. September 2012



Prof. Dr. Markus Stolze
Institut für Software
Hochschule für Technik Rapperswil

1.3 Abstract

Die Stiftung Behinderten-Transporte Zürich, nachfolgend BTZ genannt, ist eine kantonale Stiftung, die Transportaufträge für gebehinderte Personen durchführt. Solche Transportaufträge sollen in Zukunft elektronisch entgegengenommen und verarbeitet werden können. Die individuellen Bedürfnisse einer solchen Institution an eine Softwarelösung erfordern ein massgeschneidertes Produkt. Hinzu kommt, dass eine Non-Profit Organisation wie die BTZ nicht im Stande ist, ein individuell angepasstes Softwareentwicklungsprojekt in dieser Grösse eigenständig zu finanzieren.

In dieser Arbeit geht es um die Entwicklung einer Referenzimplementation eines solchen Reservierungssystems, dass bei Bedarf zur Durchführung eines "In-Market" Experimentes verwendet werden kann. Als mobiles Endgerät wurde im Rahmen dieses Projektes die Plattform iOS vorgegeben. Die Lösung sollte sicherstellen, dass auf das Backend der Referenzimplementation von allen Smartphonetypen her zugegriffen werden kann. Weiter soll das neue System in die bereits vorhandene Microsoft-Infrastruktur integriert werden, stabil funktionieren und möglichst einfach zu nutzen sein. Zusätzlich stellt die korrekte Synchronisation der Daten zwischen dem Server und dem Smartphone eine weitere Herausforderung dar. Die Daten sollen auf dem mobilen Endgerät zwischengespeichert werden um das Datentransfervolumen möglichst gering zu halten. Die Erhaltung der Datenkonsistenz darf dabei auf keinen Fall vernachlässigt werden.

Mit den evaluierten Anforderungen als Vorlage wurde ein System, bestehend aus Smartphone App, Webservice und Call-Center Client, entwickelt. Die Kommunikation zwischen den einzelnen Komponenten erfolgt über den offenen Standard HTTP[1]/JSON[2]. Der Webservice und die Call-Center Applikation wurden aufgrund der Anforderungen an die Infrastruktur mit .NET Technologien realisiert. Die Synchronisation der Daten zwischen dem Webservice und dem mobilen Endgerät erfolgt mit dem open source Framework RestKit, dass in Kombination mit dem Persistenzframework Core Data von Apple für eine einfach Handhabung der Datensynchronisation sowie des lokalen Zwischenspeichers sorgt.

Ein grosser Vorteil von diesem System ist die nahezu unbegrenzte Möglichkeit für Erweiterungen. Ist die Akzeptanz einer solchen Smartphone App bei der Kundschaft vorhanden, wäre in einem weiteren Schritt die Integration in das bestehende Dispositionssystem möglich. Dies wiederum würde die internen Abläufe der BTZ stark vereinfachen und entsprechend die Flexibilität in der Disposition erhöhen. Neue Aufträge müssten nicht mehr per Telefon entgegengenommen werden, sondern könnten innerhalb einer definierten Zeitlimite elektronisch verarbeitet werden. Um so wichtiger ist deshalb, dass die Smartphone App einfach und intuitiv zu bedienen ist. Deswegen wurde bei der Entwicklung das Augenmerk im Speziellen auf GUI Guidelines und auf die Usability gelegt. Durch die Wahl einer offenen Kommunikationsschnittstelle wird die zukünftige Kompatibilität mit weiteren Smartphonetypen gewährleistet.

[1] `http://tools.ietf.org/html/rfc2616`
[2] `http://www.json.org/`

1.4 Management Summary

1.4.1 Ausgangslage

Die Stiftung Behinderten-Transporte Zürich, nachfolgend BTZ genannt, ist eine kantonale Stiftung, die Transportaufträge für gehbehinderte Personen durchführt. Fahraufträge werden bisher über konventionelle Schnittstellen wie Telefon, Fax oder Webformular entgegengenommen, wobei letztere beide nur einen minimalen Anteil sämtlicher eingehenden Reservierungen ausmachen. Ein Anruf mit einer Bestellung muss von einem Disponent bei den BTZ direkt verarbeitet werden. Dies bedeutet, dass ein Fahrzeug gesucht wird, das zur gewünschten Abholzeit frei ist und sich idealerweise in der Nähe des Abholortes befindet. Der Fortschritt der Technik in den letzten Jahren im Bereich der mobilen Endgeräte ermöglicht diesbezüglich völlig neue Szenarien. Auch ältere Personen, welche den Grossteil der BTZ Kundschaft ausmachen, sind sich je länger je mehr den Umgang mit Smartphones gewohnt. Ein elektronisches Reservierungssystem inklusive Smartphone App als entsprechende Softwarelösung bietet sich deshalb geradezu an.

Die Lösung soll bestmöglich in die bestehende “Microsoft Windows” Infrastruktur integriert werden können. Was die Smartphone App betrifft, soll in einem ersten Schritt das iPhone von Apple fokussiert werden.

1.4.2 Vorgehen

Um den Ablauf einer Reservierung zu verstehen, konnten wir bei einem Besuch in den Büros der BTZ einen Einblick in die verschiedenen Abläufe erhalten. In einer weiteren Sitzung wurden die Anforderungen besprochen und spezifiziert. Das auf der Webseite der BTZ vorhandene Reservierungsformular[3] diente bei der Definition der Anforderungsspezifikationen als Referenz. Mit den gesammelten Informationen wurden Paper-Prototypes erstellt und damit die Grundlage für erste Usability Tests gelegt, die mit einem potentiellen zukünftigen Benutzer durchgeführt wurden. Die nächsten Schritte umfassten die Evaluation der zu verwendenden Technologien und die Implementierung eines Architekturprototypen. Die Paper Prototyps und die Usability Tests haben geholfen, grobe Fehler im GUI Design zu verhindern. Mit dem Architekturprototypen wurden die zu verwendenden Technologien evaluiert.

Für ein geordnetes Projektmanagement sorgte das Vorgehen nach RUP.

[3] `http://btz.ch/bestellen/`

1.4.3 Ergebnis

Aus den Anforderungen ist eine Referenzimplementation des Reservierungssystem bestehend aus drei Komponenten entstanden:

1. iPhone App als Schnittstelle zum Endbenutzer
2. Webservice als Kommunikations- und Datenpersistenzkomponente
3. Call-Center Client mit dem die Disposition eingehende Aufträge verarbeitet

Dabei ist anzumerken, dass der Call-Center Client - als optionaler Bestandteil des Projektes deklariert - nur in seiner absoluten Grundfunktionalität implementiert wurde.

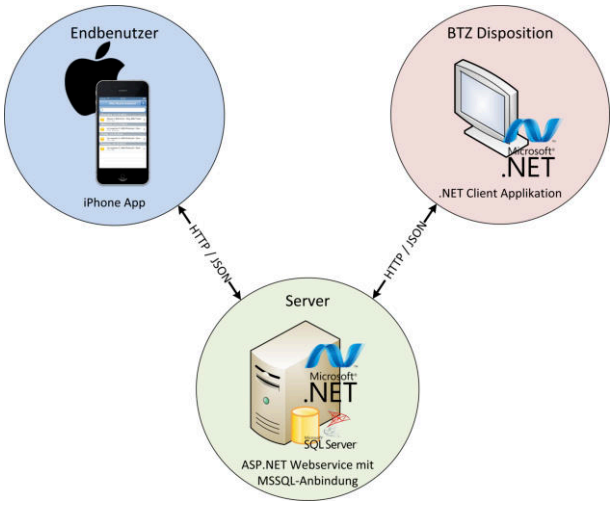


Abbildung 1.1: Architekturübersicht sämtlicher Komponenten

Aufgrund der vorhandenen Infrastrukturanforderungen wurden beim Webservice und dem Call-Center Client auf die Microsoft-Technologie .NET gesetzt. Die Kommunikation erfolgt über den offenen Standard JSON / HTTP, was die Anbindung weiterer Smartphonetypen problemlos ermöglicht.

Die App ermöglicht es dem Kunden primär neue Fahraufträge der BTZ-Zentrale zu übermitteln. Ferner können Fahraufträge, unter Einhaltung der Geschäftsregeln, beliebig angepasst werden. Übermittelte Aufträge und eventuelle Anpassungen werden von der BTZ Zentrale über den Call-Center Client entweder akzeptiert oder abgelehnt. Jede Mutation eines Auftrages bringt eine Statusänderung mit sich, welche in einem zum Auftrag gehörenden Logbuch vermerkt wird.

1.4.4 Ausblick

Um das System in Betrieb zu nehmen, fehlen noch folgende Punkte:

1. Implementation von Push-Notifications auf dem iPhone bei Statusänderungen von Reservierungen
2. Verbesserung des Call-Center Clients bezüglich Usability und Funktionalität
3. Submission der iPhone App in den Apple Store

Das entworfene System erlaubt diverse Weiterentwicklungsmöglichkeiten:

- Unterstützung der Funktionalität auf weiteren Plattformen (z.Bsp. Android, Windows Phone)
- Integration des entworfenen Systems in die aktuell verwendete Dispositionssoftware, die ebenfalls auf .NET Technologien und Microsoft SQL Server basiert
- Erweiterung der App um weitere Funktionalitäten (z.Bsp. das Erfassen von Daueraufträgen)

.

Kapitel 2

Technischer Bericht

2.1 Anforderungsspezifikationen

2.1.1 Aktuelle Situation

Folgende zwei Abschnitte zeigen einen kurzen Einblick in die konkreten Arbeitsläufe im Tagesgeschäft der BTZ.

2.1.1.1 In den Arbeitsprozess involvierte Personen

Im BTZ Hauptquartier an der Badenerstrasse 627, 8048 Zürich arbeiten in der Regel 4-5 Personen. Es folgt eine kurze Beschreibung der Arbeitsplätze.

First Level Disponent

Der First Level Disponent verwaltet Bestellungen für den gleichen Tag, sowie kurzfristige Änderungen. Der Arbeitsplatz ist im Schichtbetrieb von 07:00 - 21:30 Uhr besetzt. Der First Level Disponent arbeitet parallel mit zwei Arbeitsstationen:

- Dispositions Arbeitsstation mit 2 Bildschirmen - Betrieb der Dispositionssoftware
- Internet / E-Mail Arbeitsstation mit 1 Bildschirm - Empfang von Reservationen, die über das Webformular getätigt werden, via E-Mail

Second Level Disponent

Der Second Level Disponent verwaltet Bestellungen für die kommenden Tage. Der Arbeitsplatz ist zu den üblichen Bürozeiten besetzt. Der Second Level Disponent arbeitet an einer Arbeitsstation.

Third Level Disponent / Aushilfe

Wird als Joker bei übermässigem Arbeitsaufwand, beispielsweise zu Spitzenzeiten, eingesetzt.

Buchhaltung

Verantwortlich für die komplexen Abrechnungen.

Geschäftsleitung

Ansprechpartner für Projekte.

2.1.2 Ablauf Reservation

Möchte ein Kunde eine Taxifahrt reservieren, so erfolgt dies auf eine der folgenden 3 Arten:

- Telefonisch (sehr häufig) - Termin und alle Details werden direkt im Gespräch ausgehandelt.
- Via Fax (selten, je länger je weniger, vor allem für längerfristige Aufträge) - Interaktion mit dem Kunden erfolgt über Fax / Telefon / E-Mail
- Via Webformular (selten, je länger je mehr, vor allem für längerfristige Aufträge) - Interaktion mit dem Kunden erfolgt über E-Mail

Bei einer Reservation, unabhängig vom Medium, müssen folgende Angaben gemacht werden:

- Vorname, Nachname
- Telefon (Notfallnummer)
- E-Mail (nur Webformular)
- Abholadresse, Abholdatum & Abholzeit
- Ankunftsadresse & gewünschte Ankunftszeit
- Handelt es sich bei der Reservation um eine Fahrt zu einen Termin? [Ja / Nein] Trifft dies zu, so werden zusätzlich folgende Angaben aufgenommen:
  - Beginn Termin
  - Falls Retourfahrt ab derselben Adresse erwünscht ist:
    * Abholzeit definitiv aufnehmen oder
    * Abholzeitpunkt per Telefon spontan melden
- Treppe oder andere Hindernisse vorhanden [Ja / Nein]
- Bemerkungen
  - Angaben zum Stockwerk
  - Gibt es Gepäckstücke zu transportieren
  - Kurze Unterbrechung während der Fahrt (z.B. Apotheke)
  - ...

2.1.2.1 Daueraufträg

Parallel zu einer einfachen Reservation können telefonisch für regelmässige Sitzungen, Therapieaufenthalte oder ähnliches auch Daueraufträge erfasst werden.

2.1.2.2 Darstellung eines Ablaufs einer Reservation

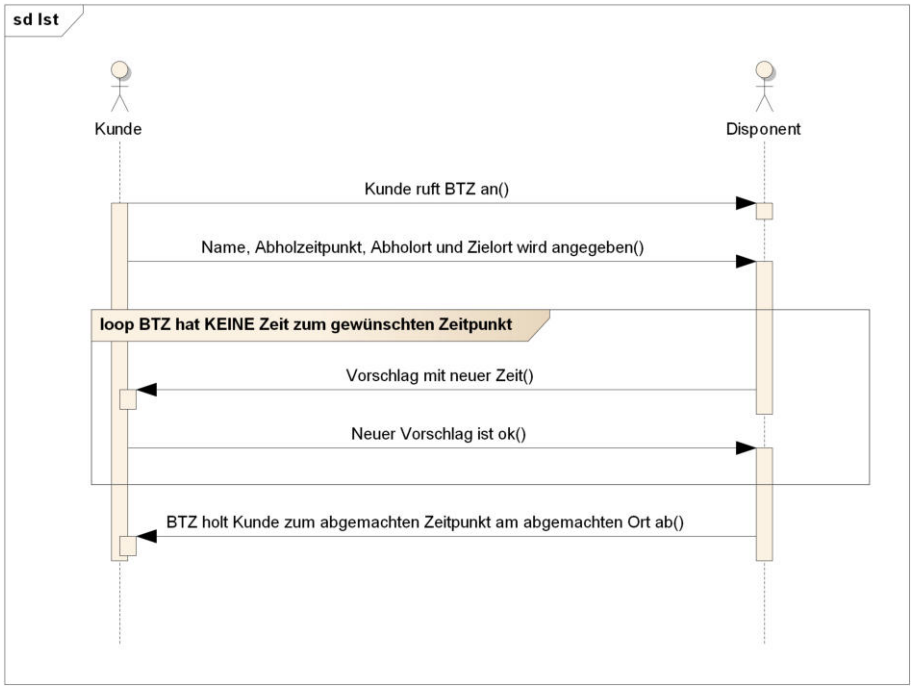


Abbildung 2.1: Aktueller Ablauf einer Reservation per Telefon

2.1.3 Projektumfang

Neben den im Kapitel 2.1.2 Ablauf Reservation beschriebenen Reservationsarten “Telefon, “Fax” und “Webformular” soll eine vierte Reservationsmöglichkeit “Smartphone Applikation”, nachfolgend “App” genannt, entwickelt werden. Die App hat primär zwei Ziele:

1. Die Disposition vor allem bezüglich längerfristigen Aufträgen entlasten.
2. Die Benutzer der BTZ nutzen vermehrt Smartphones. Wieso also nicht eine Reservationsmöglichkeit auf diesem modernen Kanal anbieten?

Der Reservationsablauf über die App soll möglichst nahe am Ablauf einer Reservation via Webformular sein. Im Unterschied zum Webformular besteht jedoch die Möglichkeit dem Kunden direkt innerhalb der App eine Rückmeldung bezüglich dem Reservationsstatus zu geben. Mit dieser Zusatzfunktionalität der Rückmeldung wird innerhalb der BTZ teilweise Neuland betreten. Neuland insofern, dass noch keine exakten Abläufe für entsprechende Szenarien bestehen. Ein konkretes Beispiel sind die Zeitlimiten sowie die Rückmeldungen an den Kunden:

1. Wieviel vorher kann eine Reservierung erstellt werden?
2. Wieviel vorher darf eine Reservierung noch verändert werden?
3. Wie wird ein Kunde über eventuelle Änderungen (Zeitplan kann nicht eingehalten werden) informiert?
4. Wie schnell muss der Auftrag eines Kunden bestätigt / abgelehnt werden?
5. ...

Solche allfälligen Geschäftsregeln werden durch die BTZ im Verlaufe des Projektes definiert und sollen auch nach Beendigung des Projektes aufgrund gesammelter Erfahrungswerte noch angepasst werden können. Nicht zum Projektumfang gehört alles, was mit Berechnung der Fahrpreise, sowie Daueraufträgen zu tun hat.

2.1.3.1 Softwarekomponenten

Das Projekt “BTZ Reservations Software” besteht grundlegend aus drei Komponenten:

1. Smartphone Client Applikation zum Erfassen von Aufträgen
2. Webservice mit Datenbankanbindung zur Kommunikation und Speicherung der Daten
3. Call-Center Applikation zur Verwaltung der Aufträge

2.1.4 Funktionale Anforderungen

2.1.4.1 Akteure

2.1.4.1.1 Disponent

Ein BTZ Mitarbeiter verwaltet die aufgegebenen Reservationen (annehmen / ablehnen) und koordiniert die Fahrer.

2.1.4.1.2 Kunde

Ein Kunde nimmt die Dienste der BTZ in Anspruch und ist im Besitz eines iPhones. Mit der BTZ Reservations App macht der Kunde Reservationen bei der BTZ und ändert / löscht diese bei Bedarf. Er ist technisch nicht versiert.

2.1.4.2 Use Case Diagram

Folgendes Diagramm zeigt eine grobe Übersicht über die zu implementierenden Anwendungsfälle (engl. Use Cases). Die Systemgrenze des Use Case Diagrammes stellt die BTZ Reservations Software als komplettes System dar. Eine Aufteilung der einzelnen Komponenten wie in Kapitel 2.1.3.1 Softwarekomponenten aufgelistet, erfolgt im Deployment Diagramm.

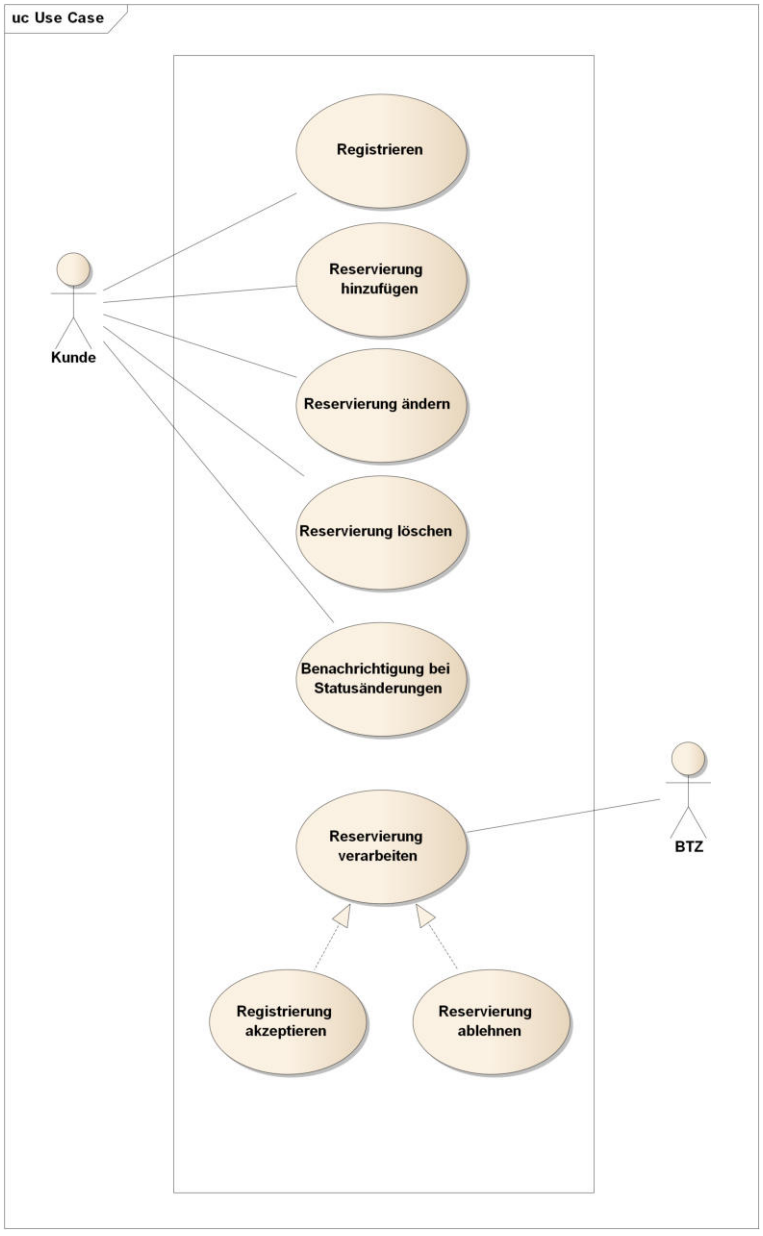


Abbildung 2.2: Use Case Diagram

2.1.4.3 Use Cases

2.1.4.3.1 UC1: Registrieren

Use Case Nummer	1
Use Case Name	Registration
Akteur	Kunde
Beschreibung	Beim ersten Smartphone Applikations Start wird der Kunde aufgefordert seine persönlichen Daten (Name, Vorname, Mobilenummer) anzugeben.
Vorbedingung	*keine*
Nachbedingung	Kunde ist in der Datenbank erfasst
Priorität	Hoch

Tabelle 2.1: UC1: Registrieren

2.1.4.3.2 UC2: Reservierung hinzufügen

Use Case Nummer	2
Use Case Name	Reservierung hinzufügen
Akteur	Kunde
Beschreibung	Der Kunde kann mit der Smartphone Client Applikation neue Reservationen hinzufügen, sofern sie mindestens 4 Stunden im voraus getätigt wird. Dazu müssen folgende Felder ausgefüllt werden (* = Optionale Angaben)
Vorbedingung	Benutzer muss angemeldet sein, bzw. die Applikation mindestens einmal gestartet haben.
Nachbedingung	Reservierung erscheint in der Übersicht
Standardablauf	1. «+» Button drücken, wodurch sich die Maske «Reservierung hinzufügen» öffnet 2. Folgende Felder müssen ausgefüllt werden: - Vorname, Nachname - Telefon - Abholadresse, Abholdatum & Abholzeit - Ankunftsadresse & gewünschte Ankunftszeit - Termin? [Ja / Nein] - Treppe vorhanden [Ja / Nein] - Bemerkungen 3. Reservierung speichern. 4. a) Netzwerkverbindung vorhanden: Reservierung wird an den Server übertragen. b) Keine Netzwerkverbindung vorhanden: Reservierung kann nicht gespeichert werden. Der Benutzer kann entweder abbrechen oder den Speichervorgang wiederholen.
Priorität	Hoch

Tabelle 2.2: UC2: Reservierung hinzufügen

2.1.4.3.3 UC3: Reservierung ändern

Use Case Nummer	3
Use Case Name	Reservierung ändern
Akteur	Kunde
Beschreibung	Der Kunde kann mit der Smartphone Applikation die zuvor erstellte Reservationen bearbeiten, sofern eine bestimmte Zeitlimite nicht unterschritten wurde.
Vorbedingung	Benutzer muss angemeldet sein, bzw. Applikation mindestens einmal gestartet haben. Reservierung muss erstellt sein
Nachbedingung	geänderte Reservierung muss gespeichert sein
Standardablauf	1. Reservierung öffnen 2. Gewünschte Änderungen vornehmen 3. Reservierung speichern 4. a) Netzwerkverbindung vorhanden: Reservierung wird an den Server übertragen. b) Keine Netzwerkverbindung vorhanden: Reservierung kann nicht gespeichert werden. Der Benutzer kann entweder abbrechen oder den Speichervorgang wiederholen.
Priorität	Hoch

Tabelle 2.3: UC3: Reservierung ändern

2.1.4.3.4 UC4: Reservierung löschen

Use Case Nummer	4
Use Case Name	Reservierung löschen
Akteur	Kunde
Beschreibung	Der Kunde kann mit der Smartphone Applikation zuvor erstellte Reservationen wieder löschen, sofern eine bestimmte Zeitlimite nicht unterschritten wurde.
Vorbedingung	Benutzer muss angemeldet sein, bzw. Applikation mindestens einmal gestartet haben.
Nachbedingung	Reservierung ist von der Übersichtsseite entfernt
Priorität	Hoch

Tabelle 2.4: UC4: Reservierung löschen

2.1.4.3.5 UC5: Benachrichtigung bei Statusänderungen

Use Case Nummer	5
Use Case Name	Benachrichtigung bei Statusänderung
Akteur	Kunde
Beschreibung	Bearbeitet der Disponent die Reservierung, so muss der Kunde danach über die Statusänderung informiert werden.
Vorbedingung	Reservierung wurde erfasst
Nachbedingung	-
Priorität	Optional

Tabelle 2.5: UC5: Benachrichtigung bei Statusänderungen

2.1.4.3.6 UC6: Reservierung verarbeiten

Use Case Nummer	6
Use Case Name	Reservierung verarbeiten
Akteur	Disponent
Beschreibung	Eine getätigte Reservierung muss von einem BTZ Mitarbeiter verarbeitet werden.
Vorbedingung	Reservierungsdetails angezeigt
Nachbedingung	Nachricht an Kunde
Priorität	Hoch

Tabelle 2.6: UC6: Reservierung verarbeiten

2.1.4.3.7 UC6.1: Reservierung bestätigen

Use Case Nummer	6.1
Use Case Name	Reservierung bestätigen
Akteur	Disponent
Beschreibung	Der Disponent kann eine von einem Kunden erstellte Reservation bestätigen. Der Kunde wird über die Bestätigung der Reservation informiert.
Vorbedingung	-
Nachbedingung	Nachricht an Kunde
Priorität	Hoch

Tabelle 2.7: UC6.1: Reservierung bestätigen

2.1.4.3.8 UC6.2: Reservierung ablehnen

Use Case Nummer	6.2
Use Case Name	Reservierung ablehnen
Akteur	Disponent
Beschreibung	Der Disponent kann eine von einem Kunden erstellte Reservation ablehnen. Der Kunde wird über die Ablehnung der Reservation informiert.
Vorbedingung	-
Nachbedingung	Nachricht an Kunde
Priorität	Hoch

Tabelle 2.8: UC6.2: Reservierung ablehnen

2.1.4.4 Ablauf Reservation

Folgendes Sequenzdiagramm stellt den typischen Ablauf einer Reservierung eines Kunden mit allen beteiligten Komponenten dar.

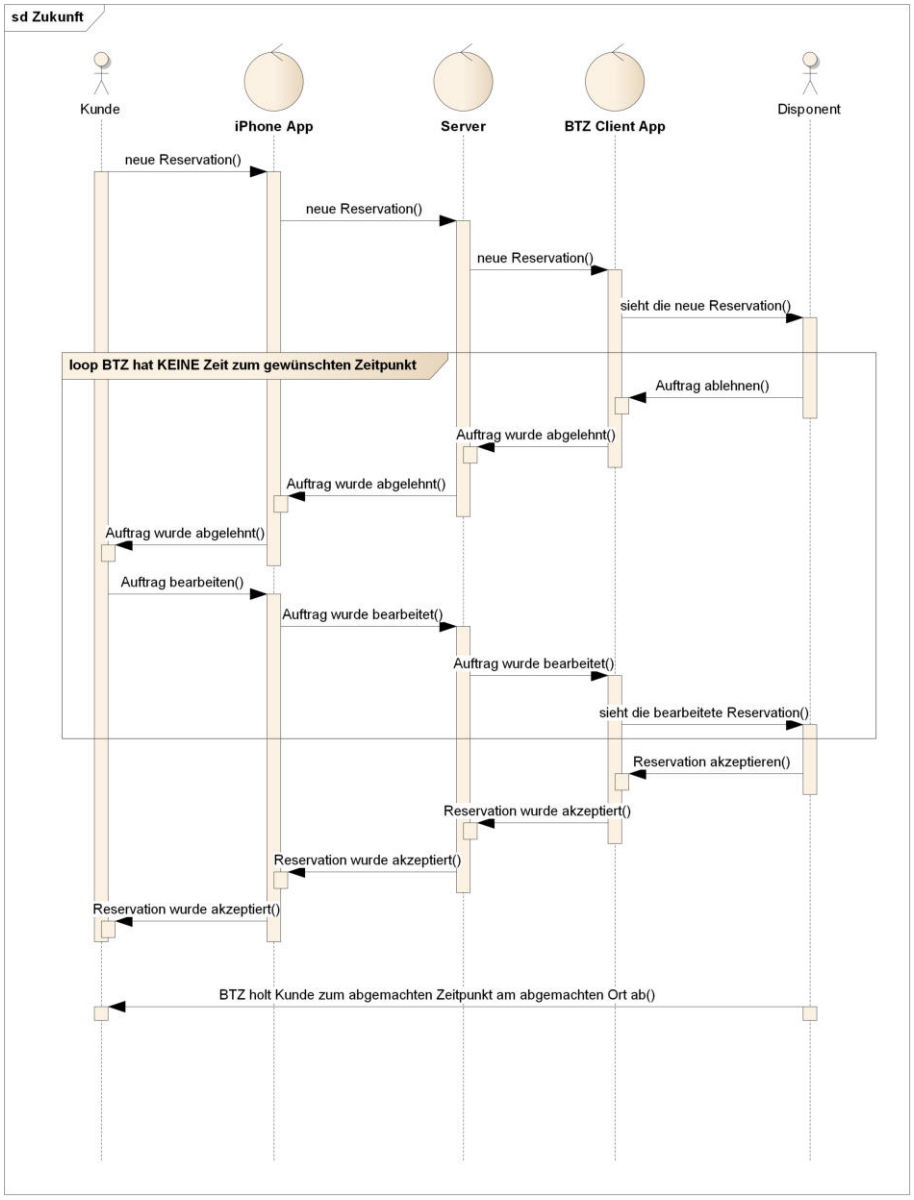


Abbildung 2.3: Ablauf einer Reservierung

2.1.5 Nichtfunktionale Anforderungen

2.1.5.1 Technologien

Folgende Technologien wurden durch die BTZ / SCS festgelegt und sind Teil der Aufgabenstellung:

Komponente	**Technologie**
Smartphone	iPhone 3GS / iOS6 / native App (Objective-C)
Datenbank	Microsoft SQL Express
Disponenten Client	C# .NET Applikation
Kommunikation	ASP .NET Webservice

Tabelle 2.9: Technologien

2.1.5.2 iPhone Applikation Qualitätsmerkmale

2.1.5.2.1 Funktionalität

- Alle Reservierungen werden fehlerfrei auf dem GUI dargestellt und komplett an den Server übermittelt.
- Der Kunde wird informiert, falls es Fehler während der Übertragung gibt.
- Reservierungen, Reservierungsänderungen etc. müssen nicht gecacht werden, wenn sie nicht übertragen werden können. Der Benutzer muss jedoch über die fehlerhafte Durchführung der Operation informiert werden um die Operation ggf. zu einem späteren Zeitpunkt erneut durchführen zu können.
- Um das Transfervolumen des mobilen Datenverkehrs möglichst gering zu halten, sollen die übertragenen Daten lokal zwischengespeichert und damit auch nach einem App Neustart lokal zur Verfügung stehen.
- Die Übertragung einer neuen Reservation dauert bei gutem Netzempfang (WLAN, 3G) nicht länger als 2 Sekunden.
- Die Benachrichtigung von Statusänderungen einer Reservierung wird per Push-Notification übertragen.

2.1.5.2.2 Benutzbarkeit

- Die grafische Oberfläche der App wurde nach den Apple iOS Human Interface Guidelines and Principles entwickelt um ein entsprechendes Mass an Standardverhalten zu gewährleisten.
- Papierprototypen wurden mit potentiellen Nutzern durchgeführt um allfällige Designprobleme bereits während der Entwicklung zu beseitigen.
- Eine neue Reservierung kann innerhalb von 2 Minuten erfasst werden.

2.1.5.2.3 Sicherheit

- Ein Benutzer registriert sich bei der erstmaligen Anwendung mit seiner Telefonnummer beim Webservice und wird über diese auch identifiziert. Da keine sensitiven Daten übertragen werden, wird auf ein Login mit Passwort verzichtet.
- Auf eine Verschlüsselung des Datenverkehrs darf explizit verzichtet werden.

2.1.5.3 Webservice Qualitätsmerkmale

2.1.5.3.1 Funktionalität

- Reservierungen werden während dem Schreib- und Lesevorgang nicht modifiziert. Die Eingabe entspricht der Ausgabe. Der Erhalt der Datenkonsistenz muss gegeben sein.
- Die Webservice Schnittstelle basiert auf einem offenen Standard. Damit können zu einem späteren Zeitpunkt allfällige Apps auf weiteren Smartphonetypen (z.Bsp. Android) angebunden werden.

2.1.5.4 Call-Center Applikation Qualitätsmerkmale

2.1.5.4.1 Funktionalität

- Reservierungen werden fehlerfrei auf dem GUI dargestellt. Änderungen in den Reservierungen im BTZ werden fehlerfrei an den Webservice übermittelt.
- Neue Reservierungen werden mit einem Intervall von 1 Minute aktualisiert (Synchronisation mit dem Datenbankserver).

2.1.5.4.2 Benutzbarkeit

- Die Bedienbarkeit des Clients soll möglichst einfach gehalten werden und daher nur die benötigten Funktionen zur Verfügung stellen.
- Der Disponent wird aktiv über neue Reservierungen informiert.

2.1.6 Übersicht Funktionalität

2.1.6.0.3 iPhone Applikation

Feature	Priorität	Anforderung
Kernfunktionalität - Anzeigen, Erstellen und Bearbeiten von Aufträgen	1	obligatorisch
Adressfavoriten bereits genutzer Adressen	2	obligatorisch
GeoLocation Funktionalität als Unterstützung beim Erfassen von Adressen	2	freiwillig
Push-Notifications bei Statusänderungen von Aufträgen	3	freiwillig

Tabelle 2.10: Übersicht Funktionalittä: iPhone Applikation

2.1.6.0.4 Webservice

Feature	Priorität	Anforderung
Offene, nicht-proprietäre Webservice Schnittstelle	1	obligatorisch
Datenpersistenz mit Microsoft SQL Express Datenbank	1	obligatorisch

Tabelle 2.11: Übersicht Funktionalität: Webservice

2.1.6.0.5 Call-Center Applikation

Feature	Priorität	Anforderung
Anzeigen von Aufträgen	3	freiwillig
Bearbeiten von Aufträgen	3	freiwillig
Verwalten der Kunden	3	freiwillig

Tabelle 2.12: Übersicht Funktionalität: Call-Center Applikation

2.2 Software Analyse

2.2.1 Domain Model

Die Domäne besteht aus vier Hauptkonzepten, die in Relation zueinander stehen.

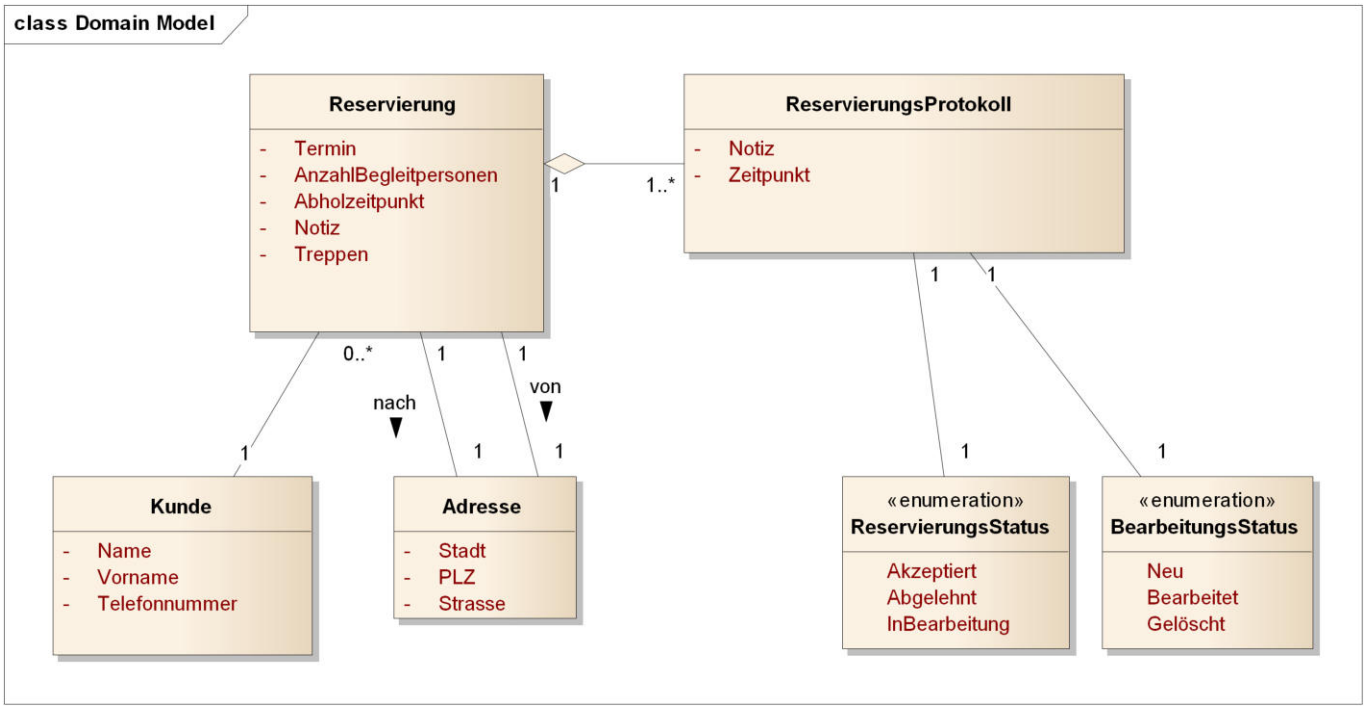


Abbildung 2.4: Domain Model

Reservierung

Das Konzept «Reservierung» bildet den Kern des Systems. Sie beinhaltet alle Daten, die das BTZ benötigt, um einen Fahrauftrag auszuführen.

Kunde

Das Konzept «Kunde» beschreibt den Kunden. Er wird anhand der Telefonnummer eindeutig identifiziert.

Adresse

Das Konzept «Adresse» stellt den Abholort oder Zielort dar.

Reservierungs Protokoll

Das ReservationsLog stellt den Verlauf der Statusänderungen einer Reservierung dar. Mit jeder Statusänderung wird ein neuer ReservationLog-Eintrag erstellt. Ein «ReservierungsProtokoll» hat zwei Hilfskonzepte «BearbeitungsStatus» und «ReservierungsStatus», die Auskunft über Statusänderungen von Seiten des Kunden sowie des BTZ geben.

Bearbeitungs-Status

Beim Bearbeitungs Status handelt es sich um ein Hilfskonzept des «ReservationLog» . Es gibt Auskunft darüber, wann ein Benutzer eine Reservierung erstellt, ändert oder löscht.

Reservierungs-Status

Beim Reservierungs Status handelt es sich um ein Hilfskonzept des «ReservationLog» . Es gibt Auskunft darüber ob bzw. inwiefern eine Reservierung vom BTZ bereits verarbeitet wurde.

2.2.2 State Machine Diagram

Für die Reservation gibt es 8 verschiedene Zustände. Das Call Center verursacht Zustandsänderungen nach rechts, wohingegen der Kunde die Zustandsänderungen gegen links und nach unten verursacht.

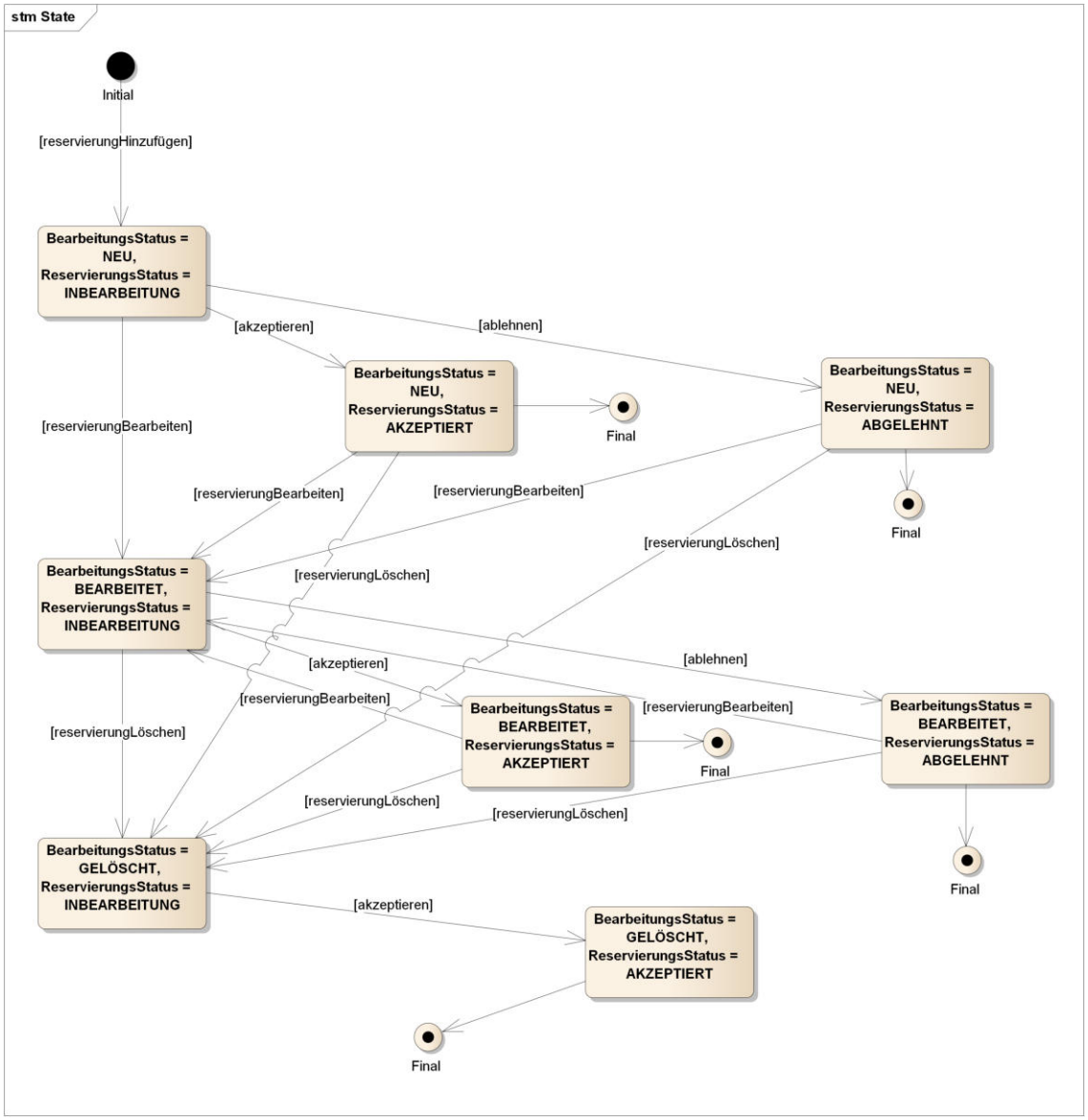


Abbildung 2.5: Zustandsänderungen Reservationsstatus

2.2.3 Data Model

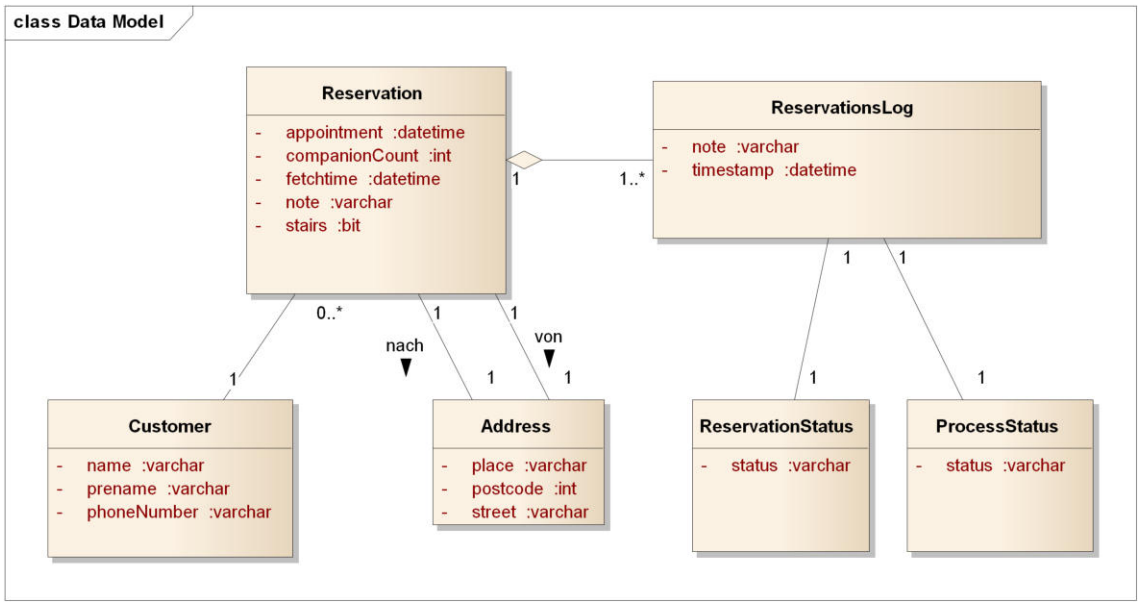


Abbildung 2.6: Data Model

Auf Basis des Domain Models wird das Data Model erstellt. Es zeigt wie die Daten in der Datenbank gespeichert werden. Das Attribut «id» wurde der einfachheitshalber weggelassen. Zur eindeutigen Identifizierung erhält jede Entität eine «id» in Form eines Integers.

2.2.4 System Sequenz Diagram

Im folgenden Abschnitt werden die zwei komplexesten System-Sequenzdiagramme beschrieben. Da alle weiteren System-Sequenzdiagramme ähnlich oder gleich aufgebaut sind, haben wir auf eine explizite Darstellung aller Diagramme verzichtet.

2.2.4.1 addReservation(reservation: Reservation)

Folgendes Diagramm zeigt den Standardablauf vom Hinzufügen einer Reservierung durch einen Kunden bis zur Bestätigung der Reservierung durch den Disponenten.

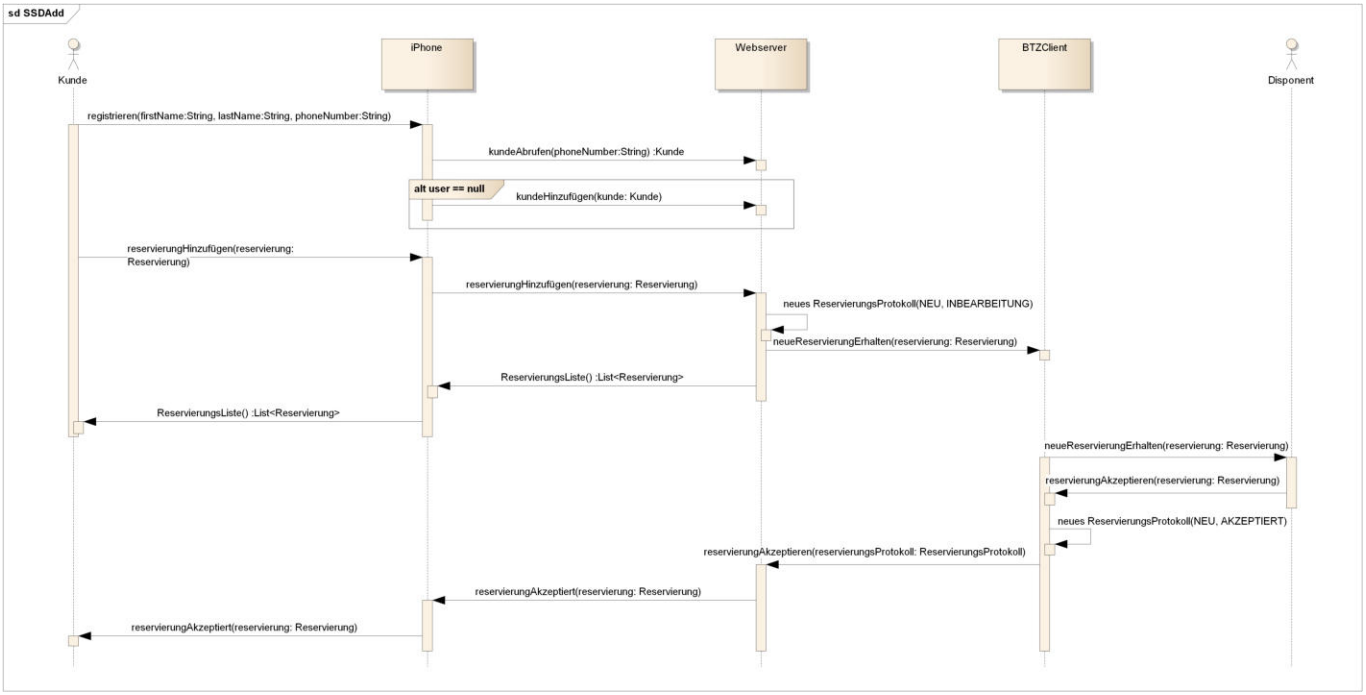


Abbildung 2.7: System Sequenz Diagram: addReservation

2.2.4.2 changeReservation(reservation: Reservation)

Dies zeigt den Standardablauf beim Ändern einer Reservierung bis zur erneuten Bestätigung durch den Disponenten.

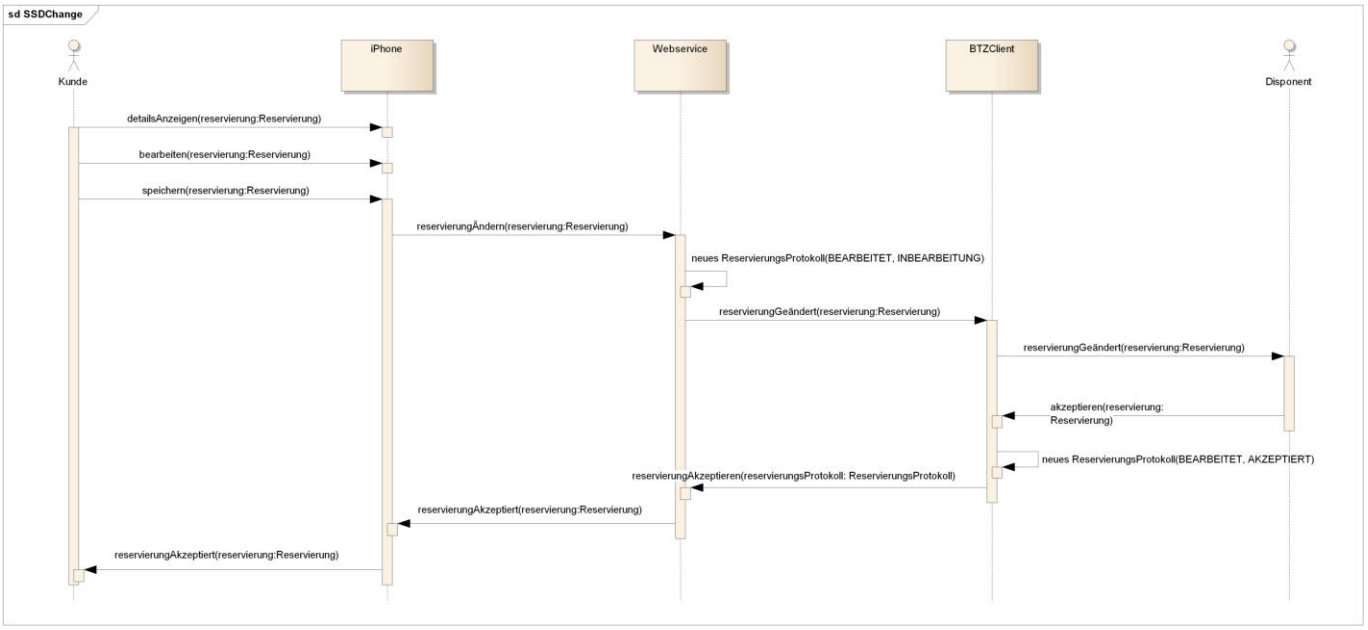


Abbildung 2.8: System Sequenz Diagram: changeReservation

2.2.5 Business Rules

Geschäftsregeln gibt es vor allem bei der Einhaltung von Zeitlimiten und dem Ändern des Status.

1. Eine Reservierung für eine Fahrt muss mindestens 4 Stunden vor ihrer Durchführung erfasst werden.
2. Eine Reservierung kann bis maximal 4 Stunden vor ihrer Durchführung vom Kunden geändert oder gelöscht werden.
3. Eine Reservierung kann bis maximal 1 Stunde vor ihrer Durchführung von der BTZ akzeptiert oder abgelehnt werden.
4. Eine von BTZ akzeptierte Reservierung kann von BTZ nicht mehr abgelehnt werden.

Das Erstellen, Ändern und Löschen von Reservierungen, welche die obengenannte Geschäftsregeln verletzen, müssen telefonisch direkt mit dem Disponenten bearbeitet werden.

2.2.6 Operation Contracts

Die folgenden Operation Contracts beziehen sich auf die oben definierten System-Sequenzdiagramme.

2.2.6.1 addReservation

Operation Name	addReservation(reservation: Reservation)
Cross Reference	UC 2: Reservierung hinzufügen
Precondition	• Der Benutzer wurde identifiziert • *Customer*-Instanz wurde erstellt
Postcondition	• eine *Reservierung*-Instanz *res* wurde erstellt • eine *ReservationsLog*-Instanz *resLog* wurde erstellt • *res* wurde mit *resLog* verknüpft • *res* wurde mit dem aktuellen Customer verknüpft • zwei Address-Instanzen *toAdr*, *fromAdr* wurden erstellt • *res* wurde mit *toAdr* und *fromAdr* verknüpft • *res* wurde auf der Datenbank gespeichert

Tabelle 2.13: addReservation Contract

2.2.6.2 changeReservation

Operation Name	changeReservation(reservation: Reservation)
Cross Reference	UC 3: Reservierung ändern
Precondition	• Reservierung wurde erstellt und auf der Datenbank gespeichert
Postcondition	• *ReservationsLog*-Instanz *resLog* mit ProcessStatus Changed wurde erstellt • bestehende Reservierung wurde in der Datenbank aktualisiert

Tabelle 2.14: changeReservation Contract

2.3 Software Architektur

2.3.1 Architekturentscheide

Ein Teil der Architekturentscheide wurde bereits durch den Industriepartner Super Computing Sytems in Zusammenarbeit mit den BTZ aufgrund deren bestehenden IT-Infrastruktur festgelegt.

2.3.1.1 Architekturvorgaben

Call Center Client

Komponente	**Verwendete Technologie**
Betriebssystem	Microsoft Windows XP oder höher
Client Technologie	.NET 4.0 Applikation

Tabelle 2.15: Architekturvorgaben Call-Center Client

BTZ Webservice

Komponente	**Verwendete Technologie**
Betriebssystem	Microsoft Windows Server 2003 oder höher
Webserver	IIS 6.0 oder höher
Framework	ASP.NET 4.0 Web API
Datenbank	Microsoft SQL Server Express

Tabelle 2.16: Architekturvorgaben Webservice

BTZ iPhone App

Komponente	**Verwendete Technologie**
Betriebssystem	iOS 6.0

Table 2.17: Architekturvorgaben iPhone App

2.3.1.2 Zu evaluierende Architekturentscheide

Grundsätzlich gibt es folgende Architekturentscheide zu evaluieren:

- Möglichkeit um auf dem iPhone Daten über eine RESTful Webservice Schnittstelle zu beziehen
- Möglichkeit die bezogenen Daten auf dem iPhone permanent zu speichern (lokaler Cache)
- Offene Schnittstelle (Repräsentation der Daten) zur Übertragung der Daten um die Kompatibilität zu weiteren Smartphonetypen zu gewährleisten

2.3.1.3 Evaluation: Repräsentation der Daten

Vorgegeben ist der Einsatz eines RESTful Webservices. Per Definition sind RESTful-Schnittstellen jedoch nicht an ein konkretes Übertragungsformat gebunden. Dies bestätigt das folgende Zitat:

> *«REST provides a hybrid of all three options by focusing on a shared understanding of data types with metadata, but limiting the scope of what is revealed to a standardized interface. REST components communicate by transferring a representation of a resource in a format matching one of an evolving set of standard data types, selected dynamically based on the capabilities or desires of the recipient and the nature of the resource. « [Fie00]*

Für den Einsatz in diesem Projekt bieten sich als Übertragungsformate XML[1] und JSON[2] an. Beide Formate sind Standardisiert und kennen eine breite Unterstützung. Dies ist ein Hauptkriterium bezüglich der angestrebten Kompatibilität mit anderen Smartphonetypen. JSON gilt im Bereich der http-Datenübertragung aufgrund der geringeren Datenmenge und der besseren Lesbarkeit als de facto Standard in diesem Gebiet. Auf eine Evaluation in Form einer Entscheidungsmatrix wird daher verzichtet. Der Architekturentscheid fällt daher auf die Verwendung von JSON.

2.3.1.4 Evaluation: lokales Caching

Als lokale Speicherung der übertragenen Daten bietet sich das Core Data Framework[3] von Apple geradezu an. Es basiert auf einer lokalen SQLite[4] Datenbank die die Daten persistiert und nach einem App-Neustart wieder zur Verfügung stellt. Weitere Features wie zum Beispiel das automatische Aktualiseren von TableViews bei Datenänderungen machen das Framework umso attraktiver. Es gilt gegenwärtig als «Best Practice» in der iOS-Entwicklung weshalb hier ebenfalls auf eine Evaluation verschiedener Produkte verzichtet wird.

2.3.1.5 Evaluation: RESTful Schnittstelle

Ziel der Evaluation ist ein Framework, dass einen Webservice ansprechen, JSON Daten empfangen und diese gegebenenfalls interpretieren kann. Um die verschiedenen Produkte korrekt evaluieren zu können wurden mehrere Kriterien inkl. Gewichtung definiert. Die Kriterien wurden zudem in «zwingend» und «optional» unterteilt. Produkte die «zwingende» Kriterien nicht berücksichtigen werden in der Entscheidungsmatrix gar nicht erst berücksichtigt.

[1] Extensible Markup Language
[2] JavaScript Object Notation
[3] `http://developer.apple.com/library/ios/#documentation/DataManagement/Conceptual/iPhoneCoreData01/Introduction/Introduction.html`
[4] `http://www.sqlite.org`

2.3.1.5.1 Zwingende Kriterien [Gewichtung]

- High-level HTTP request / response system [2]
- Möglichkeit um Server / Environments (produktiv / entwicklung) zu wechseln [1]
- Unterstützung von JSON (Parsing) [2]

2.3.1.5.2 Optionale Kriterien [Gewichtung]

- Object-Mapping System (JSON > Objekte) [2]
- Möglichkeit zur Serialisierung von Objekten (Objekte > JSON) [2]
- Core Data Unterstützung [2]
- HTTPS Unterstützung [1]

Folgende Produkte wurden genauer untersucht:

RestKit 0.10.3

Ein Objective-C Framework für iOS um einfach mit RESTful Webservices zu interagieren. Open-Source Projekt auf GitHub.

NSUrlRequest (erfüllt zwingende Kriterien nicht)

Bestandteil des Foundation Framework und somit der iOS Core Libraries. Sehr einfacher HTTP-Client um simple HTTP-Request abzusetzen.

ASIHTTPRequest (erfüllt zwingende Kriterien nicht)

Wrapper-Klasse um CFNetwork (iOS low level network library) um mit RESTful basierten Webservices zu interagieren.

AFNetworking

Open-Source Framework auf Basis der iOS network library.

Entscheidungsmatrix

		RestKit 0.10.3	AFNetworking
zwingend	High-level HTTP request / response system	**2**	2
	Möglichkeit um Server / Environments (produktiv / entwicklung) zu wechseln	**1**	1
	Unterstützung von JSON (Parsing)	**2**	2
optional	Object-Mapping System (JSON > Objekte)	**2**	0
	Möglichkeit zur Serialisierung von Objekten (Objekte > JSON)	**2**	0
	Core Data Unterstützung	**2**	0
	HTTPS Unterstützung	**1**	1
	Total Punkte	**12**	6

Tabelle 2.18: Entscheidungsmatrix

Aufgrund der offensichtlichen Überlegenheit von RestKit, wurde dieses als Client-Schnittstelle zum Webservice verwendet.

2.3.1.5.3 Datenbankzugriff Webservice

Für den Zugriff auf die Datenbank gibt es zwei verschiedene Möglichkeiten. Entweder man benutzt das Entitiy Framework[**?**] oder man schreibt die Queries und macht auch den Zugriff auf die Datenbank selber. Da beim Entitity Framework alles bereits integriert und einfach zu handhaben ist, kommt die Lösung mit den selbstgeschriebenen SQL Statements nicht in Frage. Das Entitiy Framework bietet zwei verschiedene Ansätze: Code First und Database First. Bei Code First werden zuerst Modellklassen erstellt, welche dann mit Annotations als Tabellen gekennzeichnet werden. Daraus lässt sich dann die Datenbank kreieren. Wir haben uns jedoch für die Database First Variante entschieden, da man mit den Annotations bei Code First viel beachten muss, damit diese dann richtig funktioniert. Bei Database First wird die bestehende Datenbank abgefragt und daraus wird in einem Designer die Datenbank mit ihren Tabellen und Assoziationen dargestellt. Im Hintergrund dieses Designers werden für alle Tabellen Klassen generiert, sowie einen Datenbank Kontext, welcher die Daten der Datenbank und der App zusammenhält und es ermöglicht, die Daten wieder zurück in die Datenbank zu speichern. Das Entitiy Framework ist ein sehr nützliches Framework, das viel Arbeit abnimmt und hilft, gröbere Fehler zu umgehen.

2.3.2 Architekturübersicht

Eine grobe Architekturübersicht bietet folgendes Deployment Model.

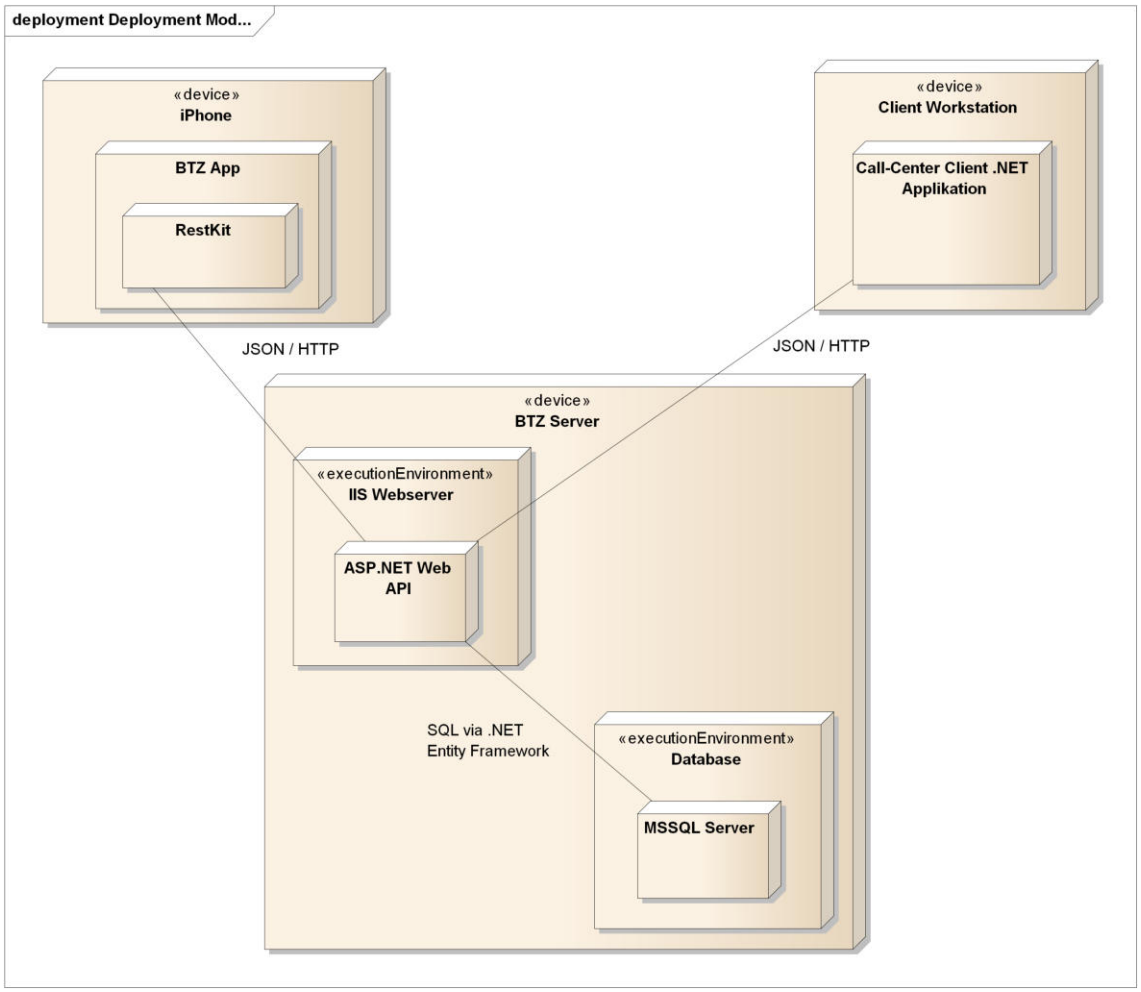


Abbildung 2.9: Deployment

Das Gesamtsystem beinhaltet 2 Clients, welche auf einen Webservice zugreifen, welcher wiederum auf eine Datenbank zugreift.

2.3.3 Architektur iOS App

2.3.3.1 Logical View

Die untenstehende Grafik stellt den logischen Aufbau der iOS App unterteilt in ihre Schichten dar. Auf die einzelnen Schichten wird im Kapitel Development View genauer eingegangen.

Abbildung 2.10: Schichtenmodell iOS App

2.3.3.2 GUI

Die GUI Komponenten werden grundsätzlich über das in der IDE Xcode integrierte Storyboard designt. Das Storyboard generiert XML Code. Über die ViewController werden die GUI-Elemente mit Daten versehen und damit auch das MVC-Pattern umgesetzt. Detaillierter wird im Kapitel 2.3.6 Externes Design auf das externe Design eingegangen.

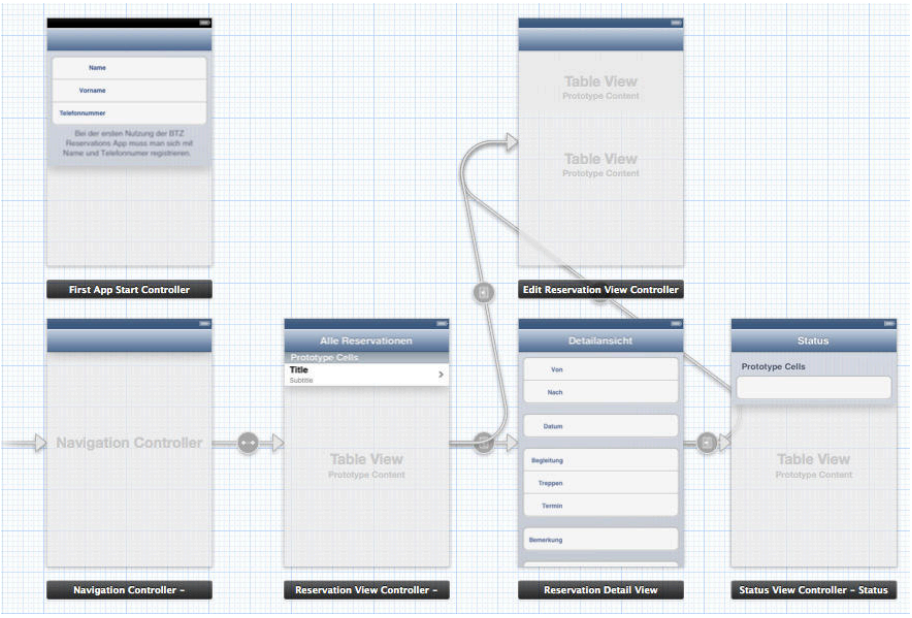


Abbildung 2.11: Übersicht Storyboard

2.3.3.3 ViewController

Für jede View wird ein ViewController benötigt. Damit wird das Model-View-Control Pattern umgesetzt. Die verschiedenen ViewController werden vom Apple Framework UIKit verwaltet. UIKit ist beispielsweise für die Navigation zwischen den verschiedenen Views verantwortlich.

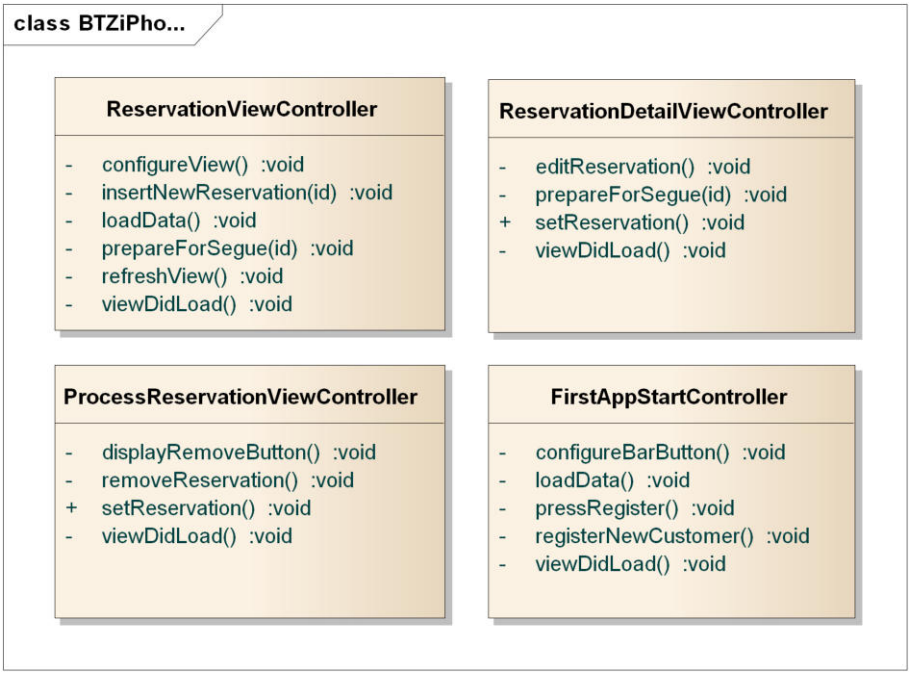


Abbildung 2.12: ViewController

2.3.3.4 Models

Folgendes Diagramm stellt die Umsetzung des Domain Models innerhalb des iPhone App dar. Auf die Auflistung der getter() / setter() Methoden der einzelnen Attribute wird absichtlich verzichtet.

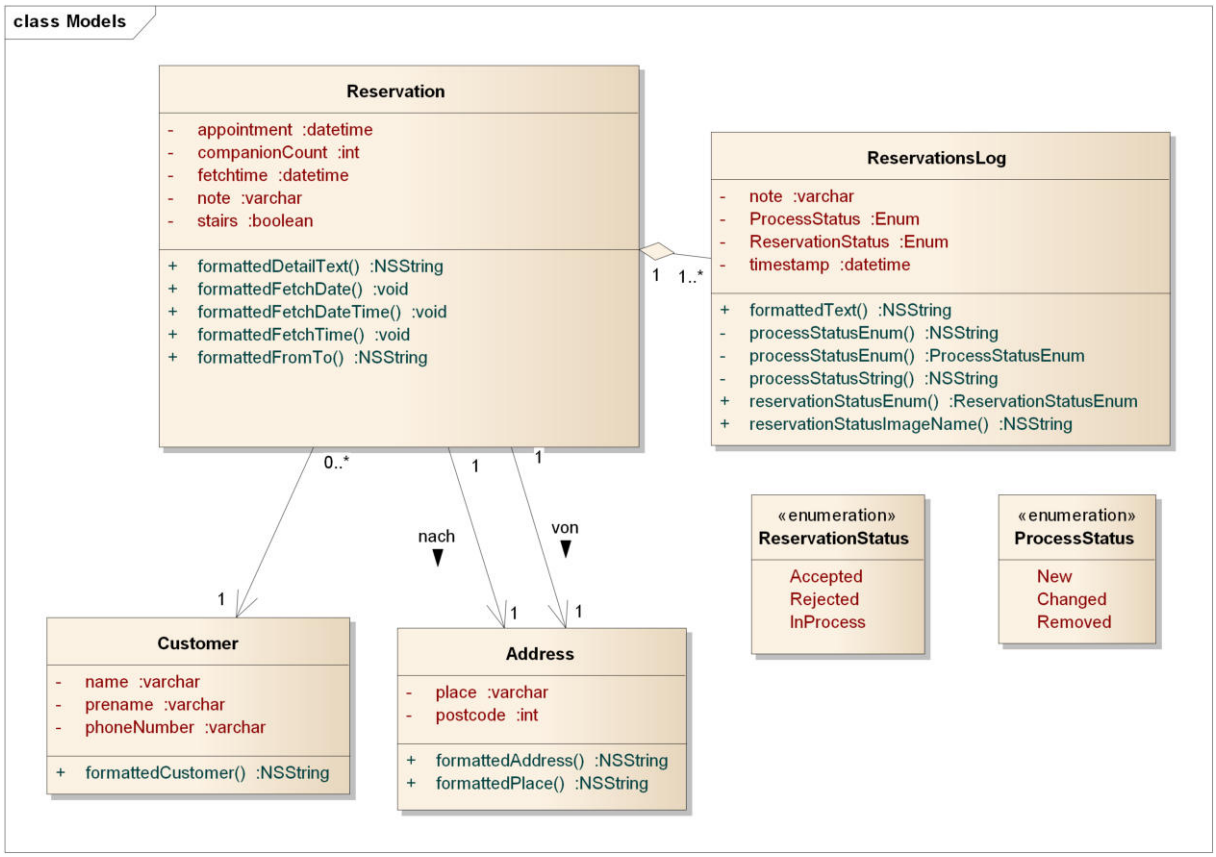


Abbildung 2.13: Übersicht Domain Klassen

2.3.3.5 Services

2.3.3.5.1 RestKitController

Der RestKitController ist für die Initialisierung des RestKit Frameworks verantwortlich.

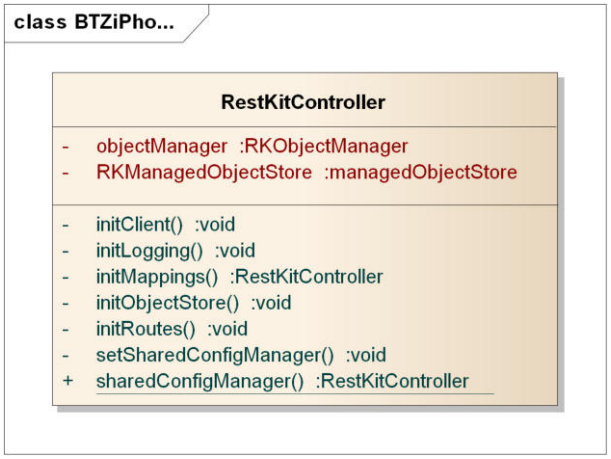


Abbildung 2.14: RestKitController

2.3.3.5.2 CoreData

Der lokale Datencache wird über das Core Date Framework von Apple realisiert. Dazu wird ein Datenmodell analog zum Domain Model erstellt, woraus dann die einzelnen Model Klassen generiert werden. Die Interaktionen zwischen RestKit und CoreData werden automatisch durch das RestKit Framework verwaltet.

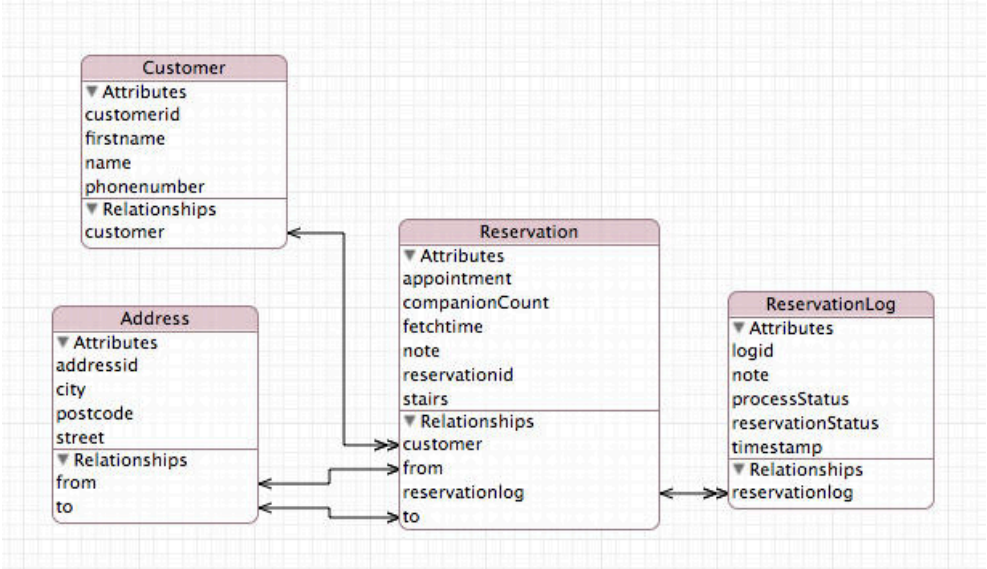


Abbildung 2.15: Daten Modell - lokaler Cache

2.3.4 Architektur Webservice

2.3.4.1 Logical View

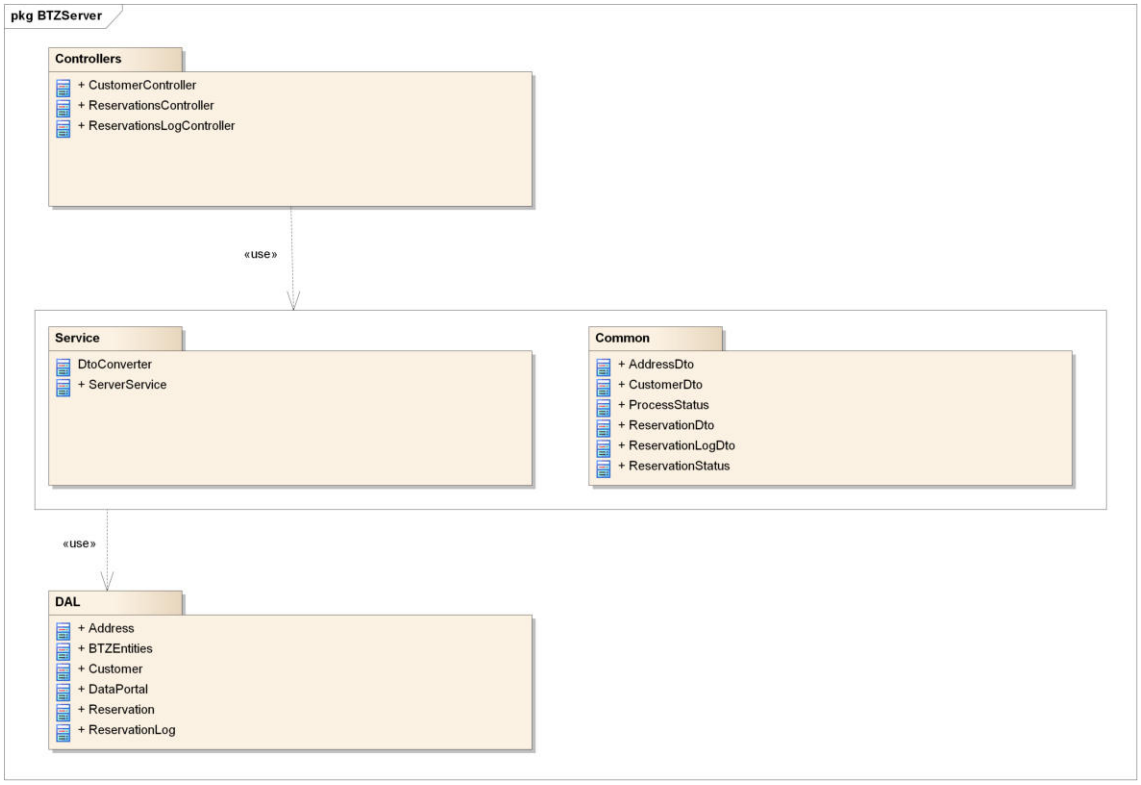


Abbildung 2.16: Logical View

Diese Grafik zeigt die logische Sicht des Servers unterteilt in Schichten. Die oberen Schichten können jeweils nur auf darunterliegende Schichten zugreifen.

2.3.4.2 Controller

Die Controller sind über verschieden URL's erreichbar. Man kann mit den üblichen Http Actions wie GET oder POST auf die Controller zugreifen. Der Output von den GET Actions ist ein JSON. Bei allen anderen kommt ein HTTP Response Code[Fie99c] zurück.

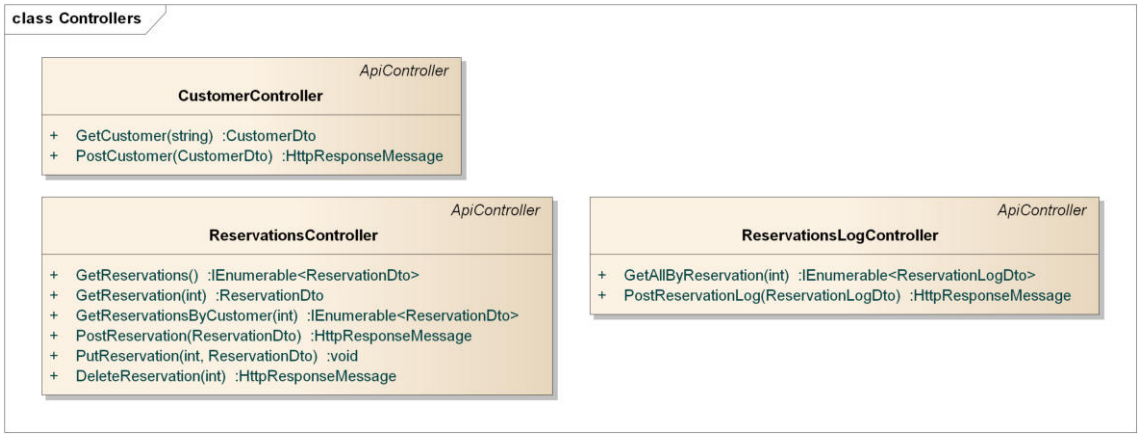


Abbildung 2.17: Controller

2.3.4.2.1 Reservations Controller

Der Reservations Controller stellt alle Aktionen, die mit einer Reservation zu tun haben, zur Verfügung.

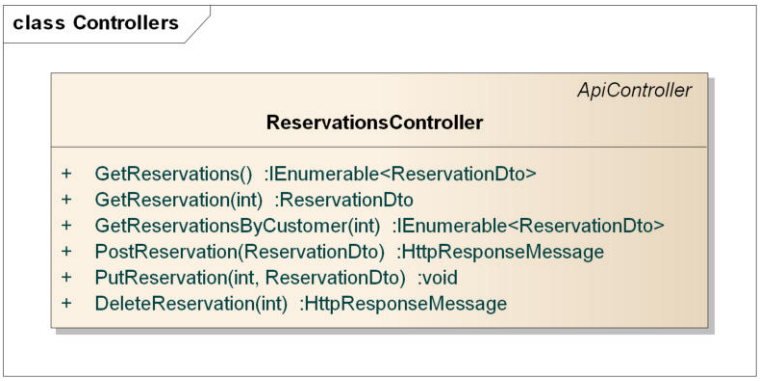


Abbildung 2.18: ReservationsController

2.3.4.2.2 URL's

Über die folgenden URL's kann man auf die Reservationen zugreifen:

Aktion	**URL**
Alle Reservationen	GET /api/reservations
Reservation mit bestimmter id	GET /api/reservations/id
Alle Reservationen von einem Kunden mit der id	GET /api/reservations/?customerId=id
Reservation hinzufügen	POST /api/reservations
Reservation ändern	PUT /api/reservations

Tabelle 2.20: Url's für Reservationen

2.3.4.2.3 Output

Hier ein Beispieloutput, für die URL /api/reservations/1


```
{
    "Id": 1,
    "Appointment": "2012-12-16T22:39:00",
    "CompanionCount": 2,
    "Fetchtime": "2012-12-16T21:39:00",
    "Note": "-",
    "Stairs": false,
    "ToAddress": {
        "Street": "Dammweg 3000",
        "Postcode": 1234,
        "City": "Root",
        "Id": 120
    },
    "FromAddress": {
        "Street": "Bahnhofstrasse",
        "Postcode": 7890,
        "City": "Rapperswil",
        "Id": 119
    },
    "Customer": {
        "Name": "Muster",
        "Firstname": "Hans",
        "Phonenumber": "0791234567",
        "Id": 1
    },
    "ReservationLogDtos": [
        {
            "Timestamp": "2012-12-13T16:39:18.76",
            "Note": null,
            "fkReservationStatus": 3,
            "fkProcessStatus": 1,
            "Id": 124
        }
    ]
}
```


Abbildung 2.19: JSON Output: Reservation mit der id 1

2.3.4.2.4 Constraints

Für die Felder einer Reservation sind folgende Werte vorgesehen:

Feld	Typ	NULL	Länge
Id	int	NOT NULL	
Appointment	DateTime	NULL	
CompanionCount	int	NULL	
Fetchtime	DateTime	NOT NULL	
Note	String	NULL	200
Stairs	bool	NOT NULL	

Tabelle 2.21: Reservation Felder

Bei einer Adresse sind die Werte wie folgt vorgesehen:

Feld	Typ	NULL	Länge
Id	int	NOT NULL	
Street	String	NOT NULL	100
PostCode	int	NOT NULL	
City	String	NOT NULL	100

Tabelle 2.22: Adress Felder

2.3.4.2.5 Customer Controller

Der Customer Controller stellt alle Aktionen, die mit einem Kunden zu tun haben, zur Verfügung.

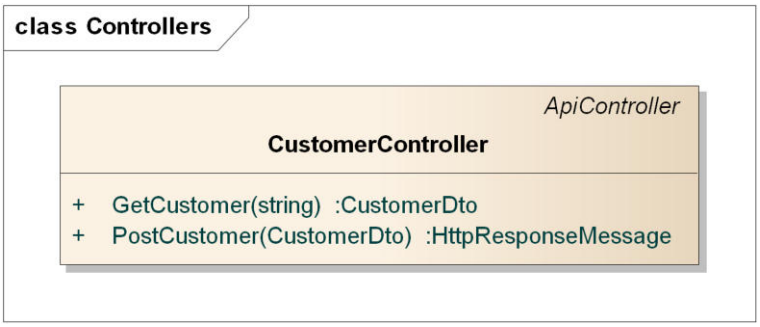


Abbildung 2.20: Customer Controller

2.3.4.2.6 URL's

Über die folgenden URL's kann man auf die Kunden zugreifen:

Aktion	**URL**
Ein Kunde anhand seiner Telefonnummer	GET /api/customer/?phoneNumber=Telefonnummer
Kunde hinzufügen	POST /api/customer

Tabelle 2.24: Url's für den Kunden

2.3.4.2.7 Output

Unten sieht man den Output beim GET Request auf die URL /api/customer/?phoneNumber=0795653820

```
{
    "Name": "Muster",
    "Firstname": "Hans",
    "Phonenumber": "0791234567",
    "Id": 1
}
```

Abbildung 2.21: JSON Output: Kunde mit der Telefonnummer 0795653820

2.3.4.2.8 Constraints

Bei dem Kunden sind die Constraints folgende:

Feld	**Typ**	**NULL**	**Länge**
Id	int	NOT NULL	
Name	String	NOT NULL	100
Firstname	String	NOT NULL	100
Phonenumber	String	NOT NULL	10

Tabelle 2.25: Kunde Felder

2.3.4.2.9 Reservations Log Controller

Der Reservations Log Controller stellt alle Aktionen, die mit einem Reservation Log zu tun haben, zur Verfügung.

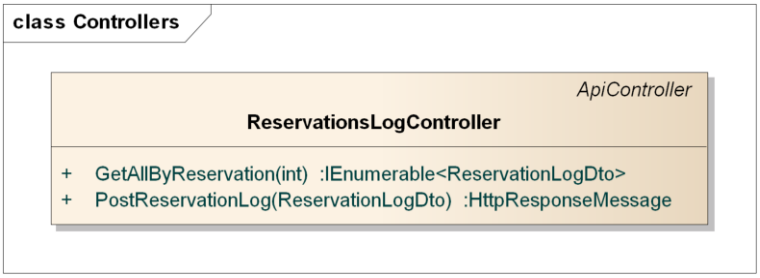


Abbildung 2.22: Reservations Log Controller

2.3.4.2.10 URL's

Über die folgenden URL's kann man auf die Reservationen zugreifen:

Aktion	**URL**
Alle Reservation Logs zu einer Reservation mit der id	GET /api/reservationslog/?reservationId=id
Reservation Log hinzufügen	POST /api/reservationslog

Tabelle 2.27: Url's für Reservations Logs

2.3.4.2.11 Output

Der Output für die URL /api/reservationslog/?reservationId=1 sieht folgendermassen aus:


```
[
        {
                "Timestamp": "2012-12-13T16:39:18.76",
                "Note": null,
                "fkReservationStatus": 3,
                "fkProcessStatus": 1,
                "Id": 124
        },
        {
            "Timestamp": "2012-12-13T16:39:30.683",
            "Note": "Erst ab 21.30 Uhr mglich",
            "fkReservationStatus": 2,
            "fkProcessStatus": 1,
            "Id": 125
        }
]
```


Abbildung 2.23: JSON Output: Reservation Logs für die Reservation mit der id 1

2.3.4.2.12 Constraints

Für die Reservation Logs JSON gelten folgende Constraints:

Felder	**Typ**	**NULL**	**Länge**
Id	int	NOT NULL	
Timestamp	DateTime	NOT NULL	
Note	String	NULL	200
fkChangedStatus	int	NOT NULL	
fkReservationStatus	int	NOT NULL	

Tabelle 2.28: ReservationLogs Felder

2.3.4.3 Service/Common

2.3.4.3.1 Service

Abbildung 2.24: Service

Die Service Schicht stellt den Controllern die Daten Transfer Objekte zur Verfügung und ist für deren Erstellung mittels dem DtoConverter zuständig. Um diese zurückzuspeichern, werden die Dto's wieder zurückkonvertiert in Entity Objekte.

2.3.4.3.2 Common

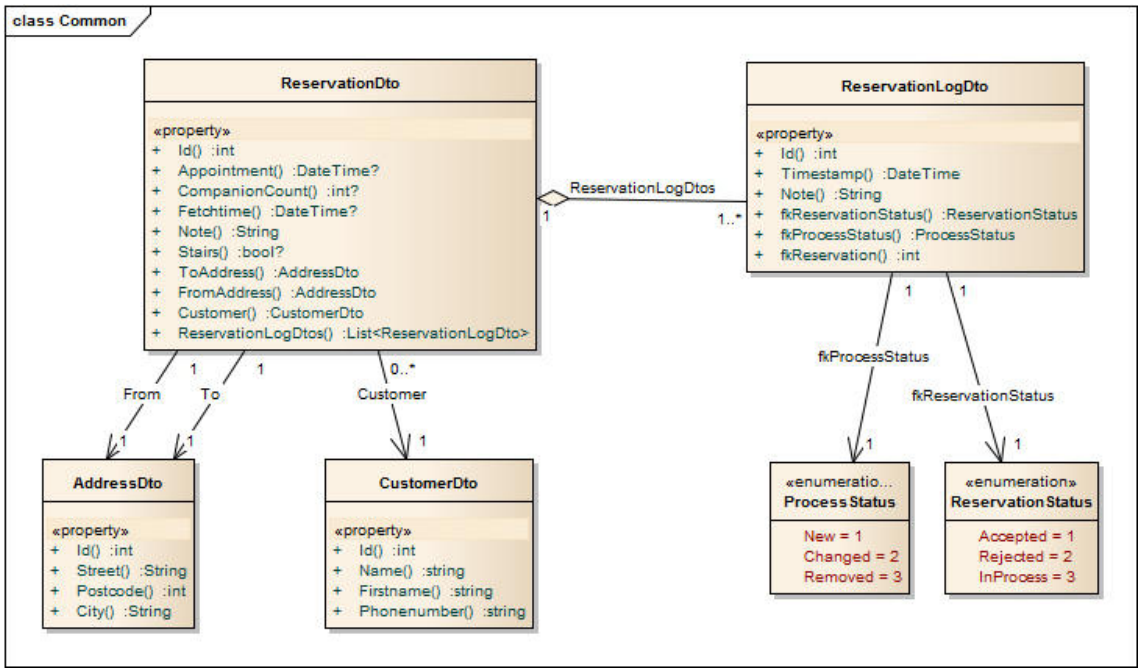


Abbildung 2.25: Common

Das Common Paket beinhaltet alle Daten Transfer Objekte. Die Daten Transfer Objekte sind dazu da, die Daten aus der Datenbank aufzubereiten und für die IOS App in die richtige Schreibweise zu bringen.

2.3.4.4 DAL

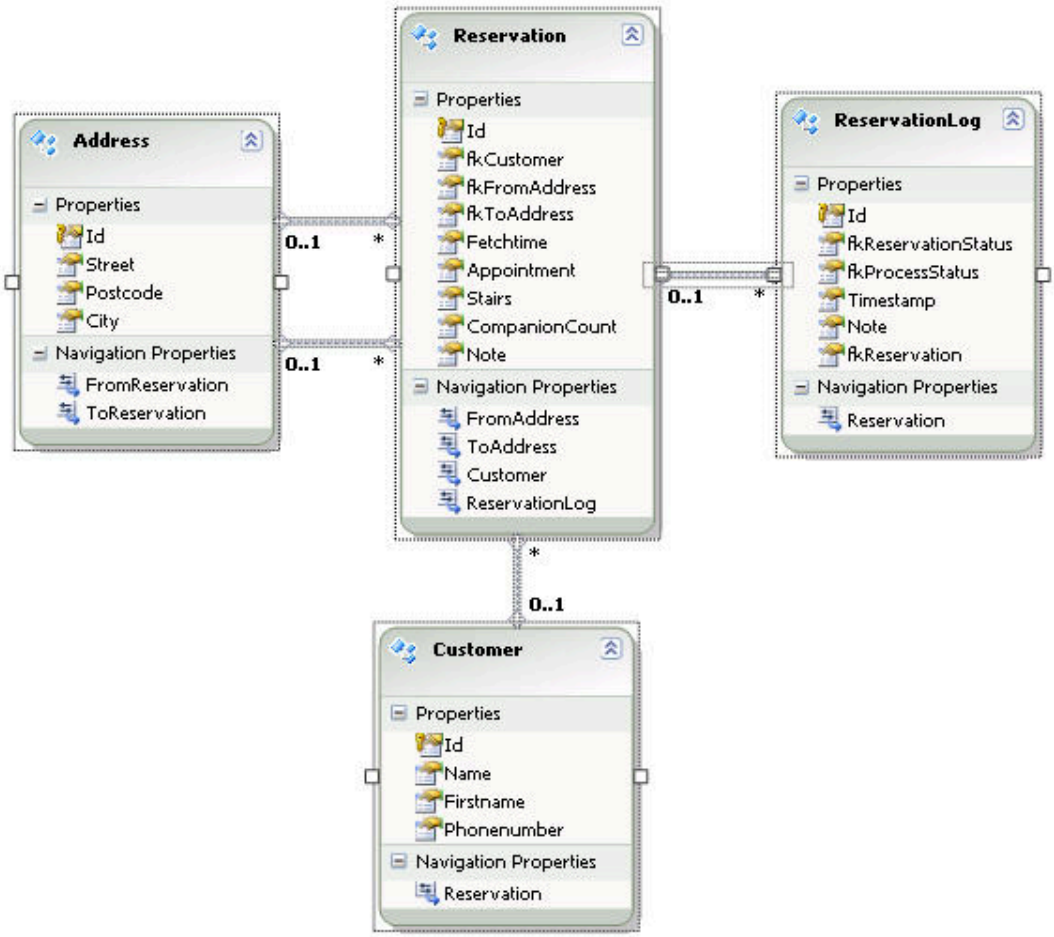


Abbildung 2.26: Data Access Layer

Der Data Access Layer implementiert das Entity Framework. Das oben abgebildete Modell stellt die Tabellen und Assoziationen der Datenbank dar. Im Hintergrund dieses Modells wird Code generiert. Zum einen werden die Entities geniert, zum anderen einen Context, welcher alle Daten in einer Liste pro Tabelle zusammenhält. Über diesen Context geschieht der Zugriff auf die Daten vom DataPortal aus.

2.3.4.4.1 Data Portal

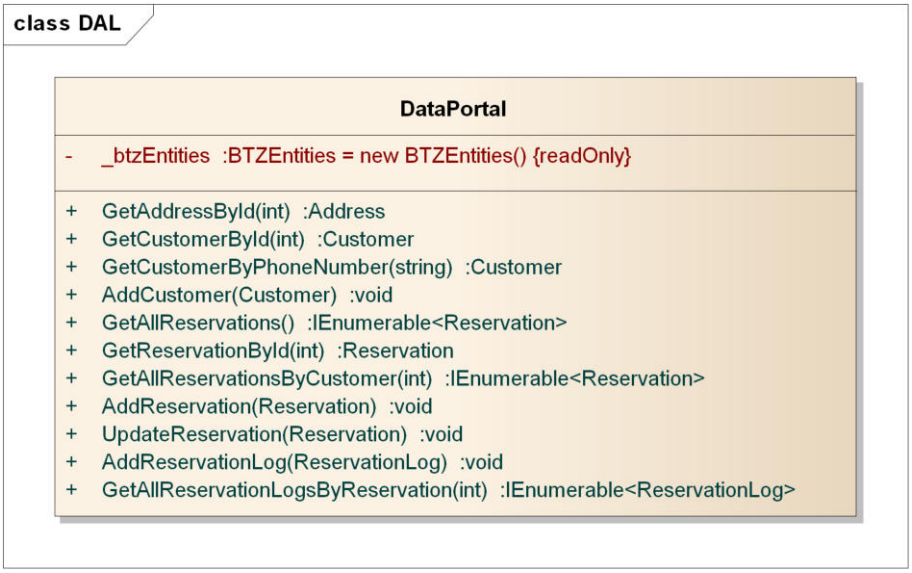


Abbildung 2.27: Business Layer

Das DataPortal macht den Zugriff auf die Datenbank über den Context des Entityframeworks und stellt für die oberen Schichten über Methoden alle Daten zur Verfügung, die gebraucht werden.

2.3.4.5 Sequenzdiagram

2.3.4.5.1 Reservation hinzufügen

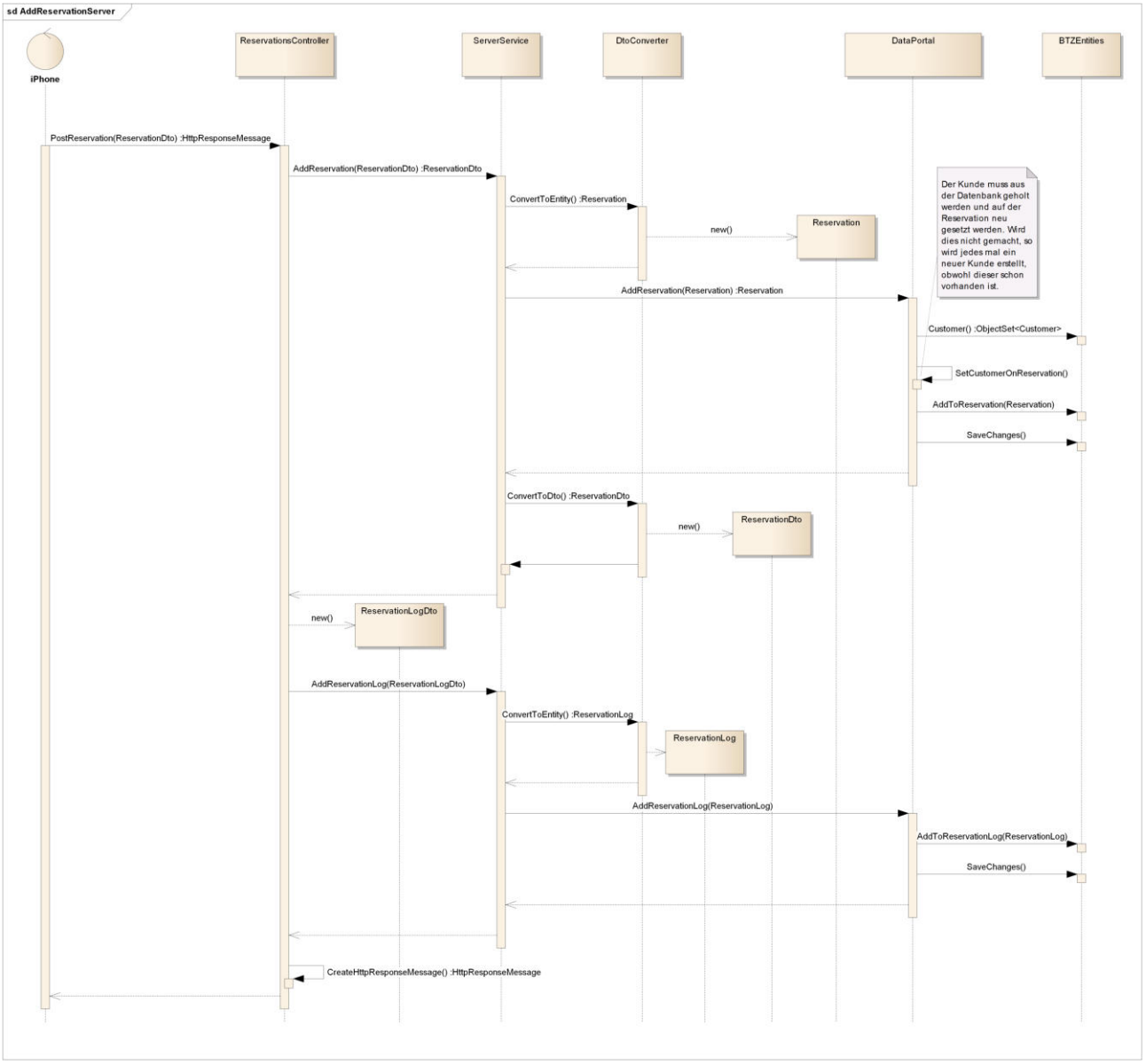


Abbildung 2.28: Reservation hinzufügen

2.3.5 Architektur Call-Center Client

2.3.5.1 Layer

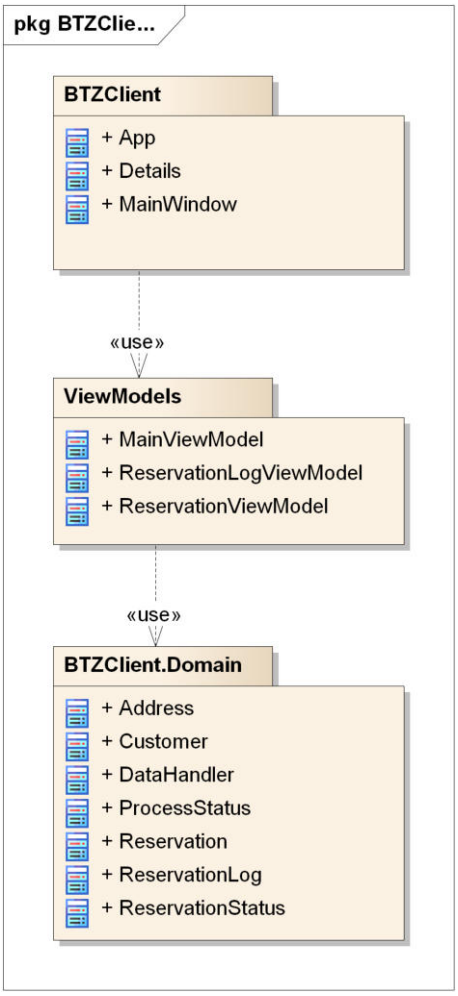


Abbildung 2.29: Layer

Das BTZClient Package ist der Startpunkt der Applikation und beinhaltet alle Gui Ansichten erstellt mit XAML [Mac06], sowie die für die Anzeige benötigten ViewModels. Im BTZClientDomain ist die Domain enthalten sowie der DataHandler Service, welcher die Daten vom Server abholt.

2.3.5.2 BTZClient

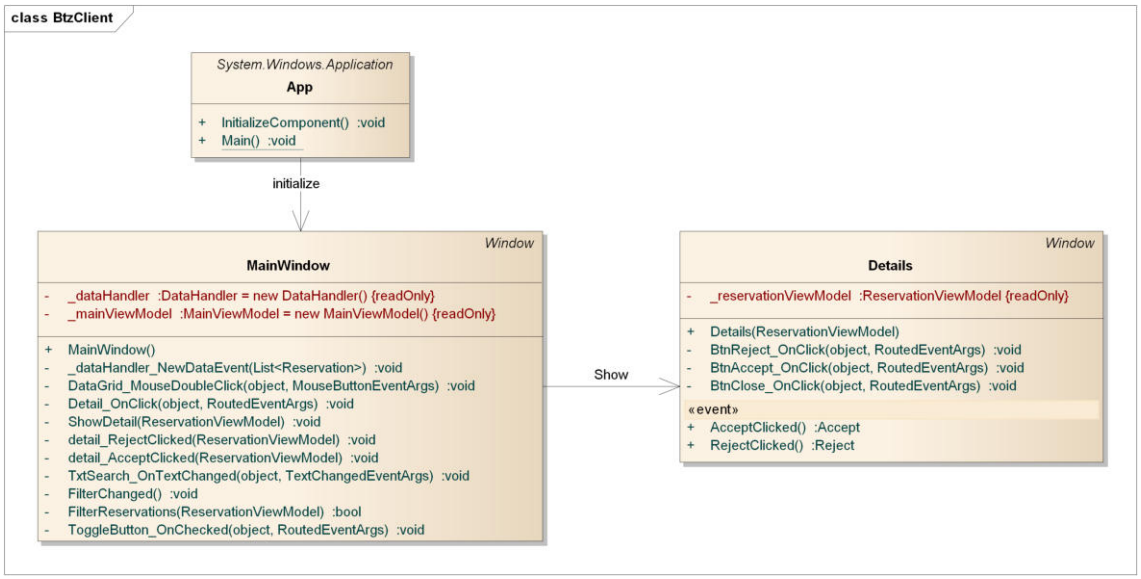


Abbildung 2.30: BTZClient

Das MainWindow ist die Hauptansicht des Clients. Es zeigt eine Übersicht über alle Reservationen mit Filtermöglichkeiten an. Man hat die Möglichkeit aus dem MainWindow eine Detailansicht zu öffnen, wobei eine Details View instanziert und angezeigt wird. Wird in der Detailansicht eine Reservation akzeptiert oder abgelehnt, so wird ein Event abgefeuert, damit die MainWindow die Aktion an den DataHandler und somit an den Server weiterleiten kann.

2.3.5.3 ViewModels

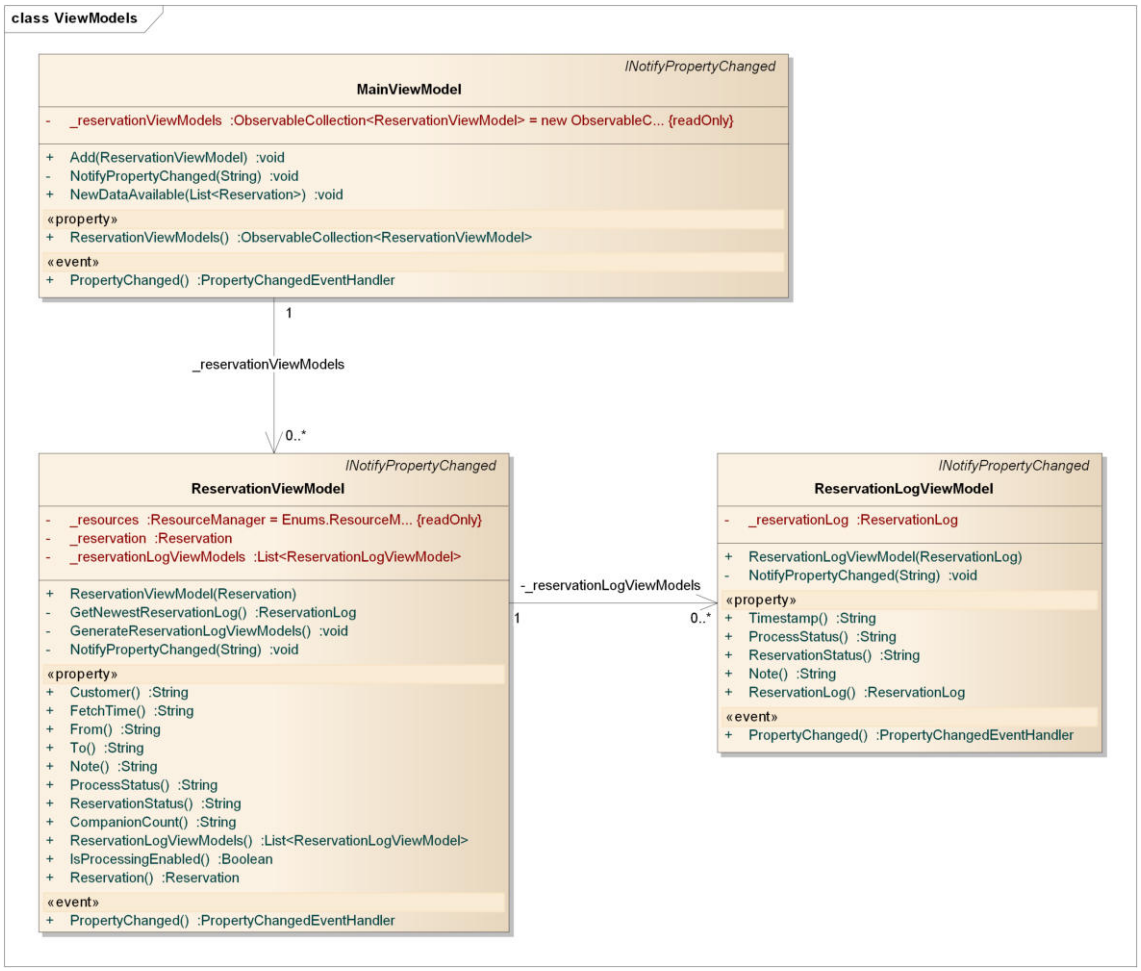


Abbildung 2.31: View Models

Die View Models wissen, wie die Domain Daten angezeigt werden müssen. Sie implementieren alle das INotifyPropertyChanged Interface, was eine einfache Handhabung bei Änderungen gewährleistet.

2.3.5.3.1 MainViewModel

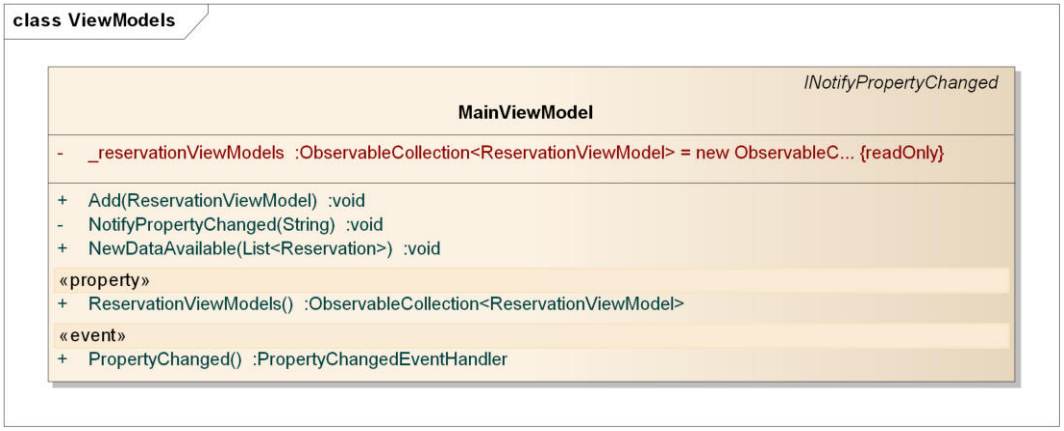


Abbildung 2.32: Main View Model

Das MainViewModel hält alle Reservationen zusammen. Werden neue Daten aus der Datenbank ausgelesen, so werden diese im MainViewModel mit den bestehenden zusammengefügt oder die bestehenden geändert.

2.3.5.4 BTZClientDomain

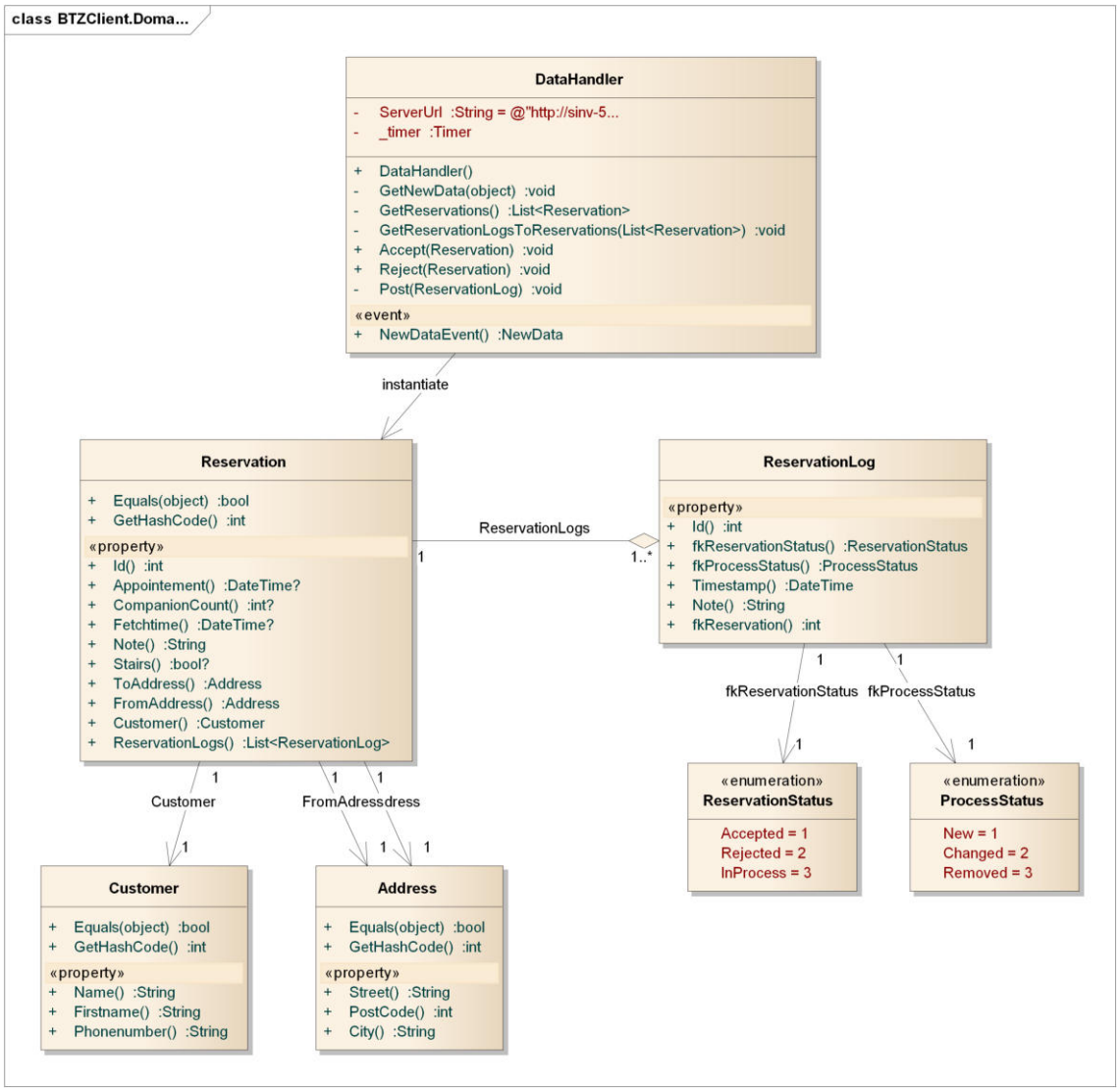


Abbildung 2.33: Domain

Die BTZClientDomain speichert die Daten, welche vom Webserver kommen. Die Properties müssen daher mit den Felder im JSON übereinstimmen.

2.3.5.5 Data Handler

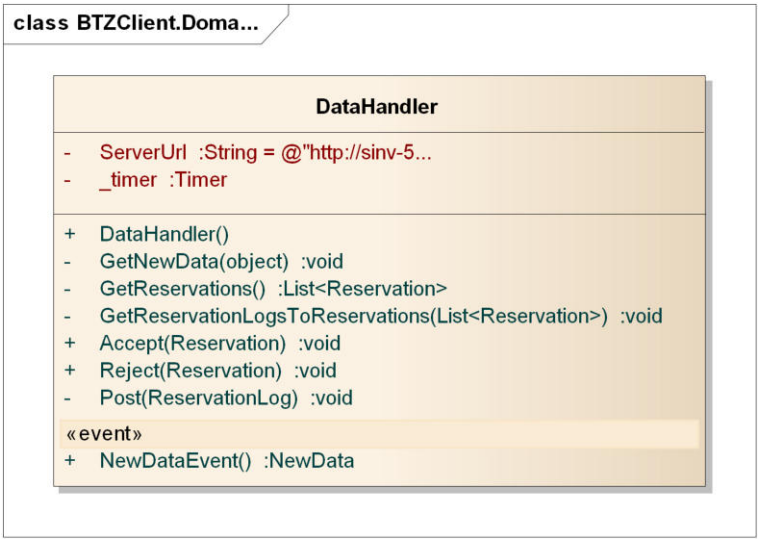


Abbildung 2.34: Data Handler

Der DataHandler macht die Abfrage zum Server und via die externe Library JSON.Net [Jso] wird der JSON String in die Objekte deserialisiert. Um eine Reservation zu akzeptieren oder abzulehnen sendet der DataHandler einen Post Request an den Server. Ein Timer, welcher in einem eigenen Thread läuft, holt dauernd die Daten vom Server und feuert danach einen Event, so dass die Daten verfügbar sind. Auf diesen Event hört das MainViewModel.

2.3.6 Externes Design

Folgendes Kapitel beschreibt das Graphical User Interface der iPhone App. Das externe Design der iPhone App wurde aufgrund der erstellten Papierprototypen und der durchgeführten Usability Tests umgesetzt. Um die Applikation möglichst intuitiv zu gestalten, ist das Design möglichst nahe an die iOS Human Interface Guidelines von Apple[5] gehalten.

Für den in den Anforderungsspezifikationen als optional definierten Call-Center Client wurde kein externes Design erstellt, da der Client nur in Form eines Prototypen umgesetzt wurde.

2.3.6.1 Navigationsübersicht iPhone App

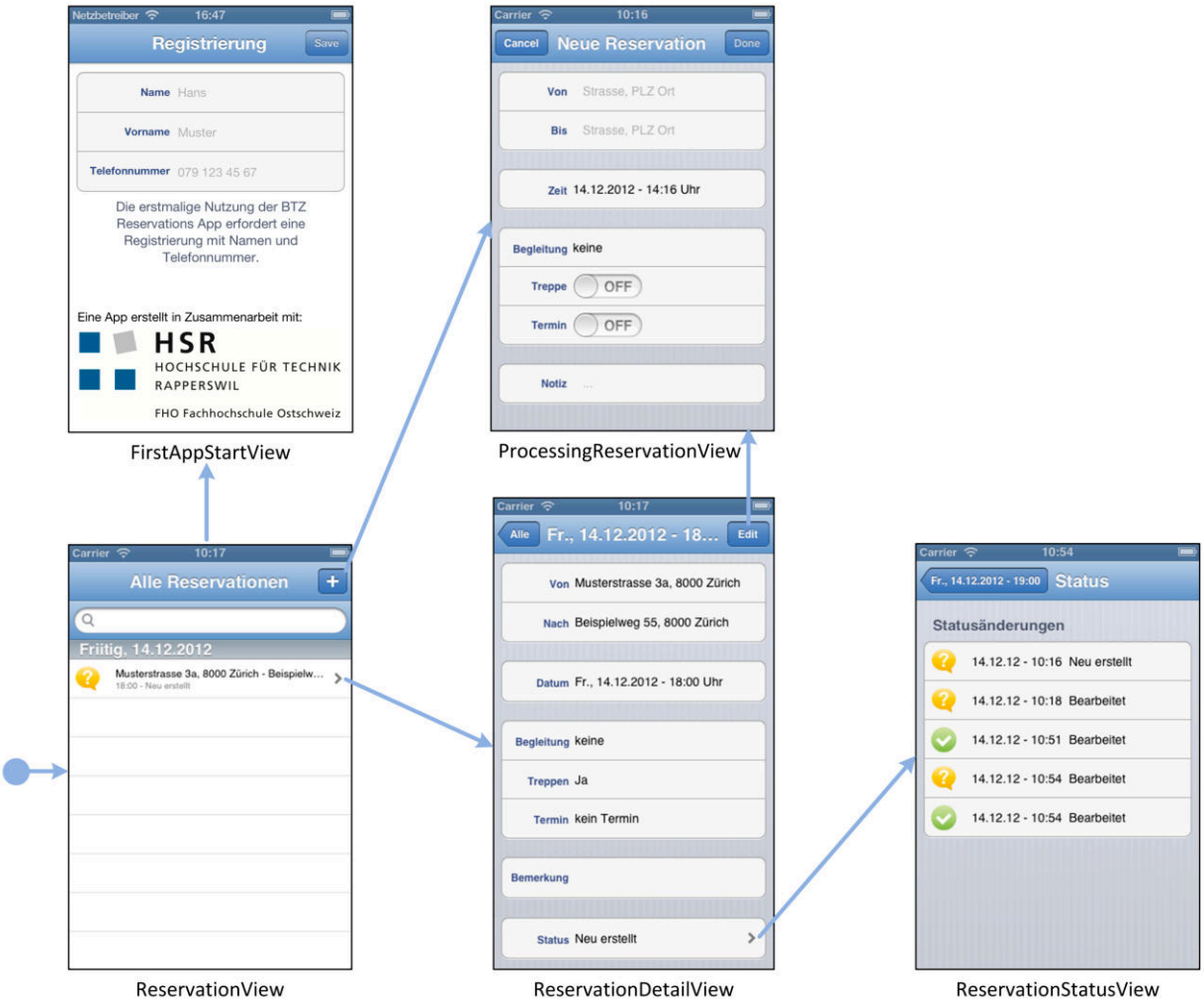


Abbildung 2.35: Navigationsübersicht iPhone App

[5] http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html

2.3.6.2 FirstAppStartView

Die FirstAppStartView wird ausschliesslich beim ersten Start der App angezeigt. Sie dient dazu, den Benutzer am System einmalig zu registrieren. Über den Button «Speichern» wird die Registrierung durchgeführt, beziehungsweise die erforderlichen Daten (Name, Vorname & Telefonnummer) an den Webservice übertragen. Ist die Registrierung erfolgreich, so werden die Daten gleichzeitig im Cache des mobilen Endgerätes abgelegt, wo sie beispielsweise für die Erstellung von Reservierungen wiederverwendet werden können. Anhand der Telefonnummer die angegeben wird, wir der Benutzer eindeutig identifiziert. Wird das Telefon gewechselt, die Telefonnummer jedoch behalten, so wird der Benutzer als derselben erkannt und die bereits getätigten Reservierungen werden automatisch abgerufen.

Abbildung 2.36: FirstAppStartView

2.3.6.2.1 Designentscheide

- Einfach gehaltene Gesamtoberfläche, die über kein Navigationselement verlassen werden kann, da die Registrierung zwingend durchgeführt werden muss.
- Beschreibung «weshalb» diese Registrierung durchgeführt werden muss.
- Anzeige des «HSR Logos» gemäss Aufgabenstellung.

2.3.6.2.2 Einschränkungen & Regeln zum Verhalten der GUI Elemente

- Beschränkung der maximalen Textlänge in den Textfeldern auf 25 Zeichen.
- Standardtastatur bei den Textfeldern «Name» und «Vorname».
- Zahlentastatur beim Textfeld «Telefonnummer».

2.3.6.3 ReservationView

Abbildung 2.37: ReservationView

Nach der erfolgreichen Registrierung erscheint die Hauptübersicht der App. Sie besteht aus einer Tabelle mit allen bereits getätigten Reservationen. Wurden noch keine Reservierungen erfasst, so wird ein entsprechender Text angezeigt. Anhand der Symbole ist erkennbar, ob der Auftrag vom Call-Center bereits akzeptiert, abgelehnt oder noch nicht behandelt wurde. Auf die Interpretation der einzelnen Symbole wird im Kapitel StatusView weiter eingegangen.

2.3.6.3.1 Designentscheide

- Möglichst einfach gestaltete Gesamtoberfläche mit lediglich 3 Funktionalitäten:
  - Detailansicht Reservierung
  - Hinzufügen einer neuen Reservierung
  - Durchsuchen von Reservierungen
- Gruppierung sämtlicher Reservierungen nach Datum
- Sortierung sämtlicher Gruppen nach Datum in aufsteigender Reihenfolge
- Die wichtigsten Informationen jeder Reservierung (Von, Nach, Datum, Status) auf einen Blick
- Berücksichtigung «Empty Cell Case»
- Hilfetext bei der ersten Verwendung der App in Kombination mit dem «Empty Cell Case»

2.3.6.3.2 Einschränkungen & Regeln zum Verhalten der GUI Elemente

- Standardtastatur für die Verwendung der Suchleiste

2.3.6.4 ProcessingReservationView

Abbildung 2.38: ProcessingReservationView

Die ProcessingReservationView ist die CRUD-View. Sie dient zum Hinzufügen von neuen, sowie dem Editieren von vorhandenen Reservierungen. Erreichbar ist sie entweder über die Hauptübersicht «ReservationView» mit der Bestätigung auf den «+» Button oben rechts oder über die Detailansicht einer Reservierung über den «Edit» Button. Die ProcessingView hat damit faktisch zwei Zustände:

1. Reservierung hinzufügen
2. Reservierung editieren

2.3.6.4.1 Designentscheide «Reservierung hinzufügen»

- Dem Benutzer möglichst wenig Raum für Fehleingaben bieten, indem Textfelder nach Möglichkeit vermieden werden und GUI Elemente wie Switcher, DatePicker und Picker verwendet werden.
- View möglichst analog zur ReservationDetailView (Gruppierte Tabelle)
- Die Adresse muss im Format «STRASSE, PLZ ORT» angegeben werden. Die Idee einer zusätzlichen SubView über die Strasse, die PLZ und den Ort in separaten Textfeldern angegeben werden kann, wurde aus Usability Gründen wieder verworfen. Etwas unschön ist dennoch die Aufforderung zur Eingabe von redundanten Daten (PLZ und Ort).
- Wird die Adresse in einem falschen Format eingegeben, wird der Benutzer über eine AlertView über den Fehler informiert und aufgefordert, das Adressfeld komplett zu löschen oder das fehlerhafte Adressformat zu berichtigen.
- Um die Eingabe der Adresse zu vereinfachen, werden über eine «Auto-Suggestion-List» bereits verwendete und somit im lokalen Cache vorhandene Adressen vorgeschlagen.

2.3.6.4.2 Ausbaupotential

- Verwendung von Lokalisierungsdiensten (WLAN, GPS, Zellinformationen) zur Ortung und Ermittlung der aktuellen Adresse.
- Auto-Suggestion-List ergänzen mit Adressvorschlägen aus dem Apple Maps oder einem ähnlichen Framework.

2.3.6.4.3 Einschränkungen & Regeln zum Verhalten der GUI Elemente (Reservierung hinzufügen)

- «Fertig» Button erscheint nur, wenn mindestens «Von», «Bis» und «Zeit» ausgefüllt wurden.
- Beschränkung der Maximallänge der Adressen Textfelder auf 50 Zeichen.
- Beschränkung der Maximallänge des Notiz-Textfeldes auf 100 Zeichen.

2.3.6.4.4 Einschränkungen & Regeln zum Verhalten der GUI Elemente (Reservierung editieren)

- «Fertig» Button erscheint nur, wenn mindestens ein Attribut eine Änderung erfahren hat.

Abbildung 2.39: Auto-Suggestion-List

Abbildung 2.40: Picker-View

Auto-Suggestion-List

Um die Eingabe der Adresse zu vereinfachen, hat man eine Liste mit Vorschlägen zur Verfügung. Darin werden alle Adressen angezeigt, die bereits einmal gebucht wurden.

PickerView

Für die Eingabe des Abholzeitpunkts wird in Form einer SubView eine sogenannte PickerView zur Verfügung gestellt, die eine einfache, fehlerfreie Eingabe gewährleistet.

2.3.6.5 ReservationDetailView

Abbildung 2.41: ReservationDetailView

Mit einem Klick auf eine Reservation in der ReservationView werden die Details der ausgewählten Reservation mit Hilfe der ReservationDetailView angezeigt. In dieser Ansicht kann die Reservation mit einem Klick auf Edit bearbeitet werden. Es wird dann die gleiche Ansicht geöffnet, wie wenn man eine neue Reservation erstellt mit der Ausnahme, dass bereits alle Felder ausgefüllt sind.

2.3.6.5.1 Designentscheide

- Gruppierte Tabelle mit allen anzuzeigenden Attributen, wie es auch die Apple Human Interface Guidelines vorschreiben
- Anzeige des aktuellsten Status. Ein Klick auf den Status öffnet die «StatusView» mit einer Übersicht über alle Statusänderungen

2.3.6.5.2 Einschränkungen & Regeln zum Verhalten der GUI Elemente

- «Edit» Button ist nur verfügbar, sofern die Geschäftsregeln bezüglich dem Ändern einer bestehenden Reservierung nicht verletzt werden.

2.3.6.6 StatusView

Abbildung 2.42: StatusView

Jede Zeile stellt einen Statuswechsel dar. Dieser wird entweder durch den Benutzer der App oder den Disponenten mit dem Call-Center Client durchgeführt. Dem Disponenten ist es zudem möglich, dem Status eine Notiz hinzuzufügen. Diese wird unterhalb der Statusbezeichnung angezeigt.

2.3.6.6.1 Designentscheide

- Einfache, nicht veränderbare Statusübersicht.
- Einträge werden nach dem Datum aufsteigend sortiert.

2.3.6.6.2 Statussymbole

In der Detailansicht kann man den aktuellen Status anwählen, um eine Übersicht über den ganzen Verlauf der Reservation zu erhalten. Die verschiedenen Icons bedeuten folgendes:

Die Reservierung wurde akzeptiert

Die Reservierung wurde abgelehnt

Die Reservierung wurde noch nicht bearbeitet

2.4 Realisierung

Dieses Kapitel zeigt auf, welche Probleme während der Realisierung aufgetreten sind und welche Lösungswege aus welchen Beweggründen angestrebt wurden.

2.4.1 iPhone App

2.4.1.1 Verwendete Nachschlagwerke

Als grundsätzliche Nachschlagwerke dienten während der Implementation der iPhone App folgende Quellen:

- Standford University lecture CS 193P "iPhone developing"[6]
- Buch: "iOS programming : the big nerd ranch guide" [Con11]
- Apple Developer Portal [7]
- Stackoverflow, Keyword "restkit"[8]
- Google Newsgroup "RestKit"[9]

2.4.1.2 RestKit - Mappin Problematik

Was das Framework RestKit an einem sehr umfangreichen Funktionsumfang bietet, fehlt im Gegenzug leider in der Dokumentation. Ebenfalls auffallend ist der schnell Release-Zyklus (drei neue Releases während 14 Wochen) und die stetig ändernde API. Dies stellt gerade bei der Suche nach Hilfestellungen eine besondere Herausforderung dar.
Konkrete Probleme gab es beim vom .NET Webservice generierten JSON. Das JSON ist zwar valid enthält jedoch keinen RootKeyPath. RestKit verwendet für das JSON <-> Objekt-Mapping jedoch primär diesen sogenannten RootKeyPath. Das Vorhandensein dieses RootKeyPaths vereinfach das Objekt-Mapping stark. Sind die Attribute im JSON und die lokalen Model-Klassen in der Namensgebund identisch kann das Mapping automatisch aufgrund der RootKeyPaths durchgeführt werden.

```
1 {
2     "Customer": {
3         "Id": 1,
4         "Name": "Muster",
5         "Firstname": "Hans",
6         "Phonenumber": "079.."
7     }
8 }
```

JSON mit RootKeyPath

```
1 {
2     "Id": 1,
3     "Name": "Muster",
4     "Firstname": "Hans",
5     "Phonenumber": "079.."
6 }
```

JSON ohne RootKeyPath

Selbstverständlich lässt sich das Objekt-Mapping auch für JSON ohne RootKeyPath realisieren. Dabei muss jedoch jedes einzelne Attribut gemappt werden und das gesamte Mapping dann einem KeyPath zugewiesen

[6] http://www.stanford.edu/class/cs193p/cgi-bin/drupal/
[7] https://developer.apple.com/
[8] http://stackoverflow.com/questions/tagged/restkit
[9] https://groups.google.com/forum/?fromgroups#!forum/restkit

werden. Beim Senden eines HTTP-Requests (z.Bsp. einer POST-Requests) muss das Mapping dann explizit angegeben werden.

```
1  RKManagedObjectMapping *reservationMapping = [RKManagedObjectMapping
       mappingForClass:[Reservation class] inManagedObjectStore:objectManager.
       objectStore];
2      [reservationMapping mapKeyPathsToAttributes:@"Id", @"reservationid", nil];
3      [reservationMapping mapKeyPathsToAttributes:@"Appointment", @"appointment",
           nil];
4      [reservationMapping mapKeyPathsToAttributes:@"CompanionCount", @"
           companionCount", nil];
5      [reservationMapping mapKeyPathsToAttributes:@"Fetchtime", @"fetchtime", nil
           ];
6      [reservationMapping mapKeyPathsToAttributes:@"Note", @"note", nil];
7      [reservationMapping mapKeyPathsToAttributes:@"Stairs", @"stairs", nil];
8      reservationMapping.primaryKeyAttribute = @"reservationid";
9
10 [...]
11
12 [objectManager.mappingProvider setObjectMapping:reservationMapping forKeyPath:@"
       Reservation"];
```

Codebeispiel: Mapping jedes einzelnen Attributes inkl. KeyPath Zuweisung

```
1  [[RKObjectManager sharedManager] postObject:resLog usingBlock:^(RKObjectLoader *
       loader) {
2          RKObjectMapping *mapping = (RKObjectMapping*)[[[RKObjectManager
               sharedManager] mappingProvider] mappingForKeyPath:@"
               ReservationLogDtos"];
3          loader.objectMapping = mapping;
4          loader.serializationMIMEType = RKMIMETypeJSON;
5      loader.delegate = self;
6  }];
```

Codebeispiel: Explizites definieren des Mappings bei einem POST-Request eines “ReservationLog”-Objektes

2.4.2 Webservice

Während der Entwicklung des Webservices gab es zwei erwähnenswerte grössere Probleme:

- Bei der Aktualisierung (PUT Request) eines Objektes.
- Konzeptioneller Überlegungsfehler beim “Löschen” von Reservationen

Wurde beispielsweise ein “Reservation”-Objekt aktualisiert so wurde die zum “Reservation”-Objekt gehörende Adresse (“Address”-Objekt) neu instanziiert und in die Datenbank eingefügt. Der Grund dafür liegt im Entity-Framework. Ein “Reservation”-Objekt enhält neben der ID des “Address”-Objektes auch eine Instanz des

“Adress”-Objektes. Das Problem wurde gelöst, indem die Objektreferenzen auf die Relationsobjekte, in diesem Fall das “Address”-Objekt gelöscht und nur noch die Id der Relationen in der Reservation behalten wurden.

Einen sehr guten Input hat das Review des Webservices durch Julian Heeb von der SCS gegeben. Das Löschen einer Reservierung wurde in einer ersten Implementation über einen HTTP DELETE Request realisiert. Der HTTP DELETE Request ist im RFC2616 des HTTP Standards wie folgt definiert:

> *The DELETE method requests that the origin server delete the resource identified by the Request-URI.* [**?**]

Die Löschung einer Reservierung löscht jedoch nicht die Ressource “Reservierung”, sondern fügt der Ressource “Reservierung einen neuen Log-Eintrag in Form eines “ReservationLog”-Objektes hinzu. Aus diesem Grund ist an dieser Stelle nur ein HTTP POST Request gemäss RFC2616 des HTTP Standards korrekt:

> The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line. [Fie99b]

Die Implementation wurde entsprechend angepasst.

2.4.3 Call-Center Client

Der Call-Center Client wurde als optional definiert. Daher wurden für diesen Teil keine Architekturentscheide gemacht, das bedeutet, dass keine GUI Technologien evaluiert wurden. Dies führte dazu, dass das GUI zweimal gemacht werden musste. Zuerst wurde als Technologie Windows Forms[10] benutzt. Dies führte zu einigen Problemen. Beim sogenannten DataGrid, welches alle Reservierungen anzeigen sollte, war die Datenanbindung sehr mühsam. Es war zwar möglich, eine Liste anzuhängen, diese musste jedoch zwischengespeichert werden. Diese zwischengespeicherte Liste wurde bei der Selektion einer Zeile wieder benötigt, da aus dem Grid nicht direkt auf die Objekte zugegriffen werden konnte. Auch um auf den selektierten Index zuzugreifen benötigte es viele Schritte. Anhand diesen Indexes konnte aus der zwischengespeicherten Liste das Objekt herausgelesen werden. Dies ist einer der Gründe, weshalb für das GUI die Markup Language XAML[11] verwendet wurde. Das DataGrid im XAML ist sehr einfach zu handhaben und bietet alle Funktionen die wir brauchen, auf einfache Weise.

[10] Windows Forms ist der Name einer Programmierschnittstelle zur Erstellung graphischer Benutzeroberflächen (GUIs)

[11] Extensible Application Markup Language ist eine von der Software-Firma Microsoft entwickelte allgemeine Beschreibungssprache für die Oberflächengestaltung.

2.5 Testing

2.5.1 Usability Tests

Während der Studienarbeit wurden 2 Usability Tests durchgeführt. Zum einen am Anfang der Arbeit und zum anderen der zweite zum Schluss. Die Usability Tests am Anfang haben uns geholfen das Risiko einzugrenzen und gröbere Fehler im GUI Design zu vermeiden. Ein reales Szenario wurde anhand Paper Prototypes durchgespielt. Die verwendeten Paper Prototypes können im Anhang im Kapitel 4.1 Usability Tests betrachtet werden. In einem ersten Schritt wurden Paper Prototypes mit dem Mockup-Tool Balsamiq erstellt. Um dem realen Look & Feel gerecht zu werden wurden in einem zweiten Schritt die erstellten Mockups dann in Photoshop mit iOS GUI Elementen[12] erstellt. Für den ersten Durchgang der Usability Tests wurde die zweite Version verwendet.

Abbildung 2.43: Paper Prototypes Version #1

Abbildung 2.44: Paper Prototypes Version #2

[12] http://apfeltech.net/2012/04/entwickler-bietet-ios-gui-elemente-als-psd-datei-an-download/

Zum Schluss der Arbeit wurde mit der fertigen App erneut Usability Tests durchgeführt. So konnte einerseits verifiziert werden, dass das GUI intuitiv bedienbar ist und andererseits eventuelle Fehlfunktionen eruiert werden. Ohne jegliche Erklärungen hat sich der Benutzer dem vorgegeben Szenario angenommen und eine Reservierungen vorgenommen.

2.5.1.1 Szenarios

1. Um ein BTZ Taxi zu reservieren, haben Sie bis jetzt immer zum Telefon gegriffen. Neu gibt es dafür eine App. Heute möchten Sie Ihre erste Reservierung mit der neuen App durchführen. Die Reservierung ist für den 15. Oktober 2012. Nach der Arbeit möchten Sie um 15.30 Uhr an den Hauptbahnhof Zürich. Tätigen Sie diese Reservierung mit der App.
2. Einige Tage und Reservierungen später möchten Sie eine Reservation tätigen, um sich nach einer Sitzung um 15.30 Uhr vom Hauptbahnhof abholen zu lassen und sich ins Büro zurückbringen zu lassen. Erstellen Sie diese Reservierung für den 20. Oktober 2012.
3. Einen Tag nachdem die Reservierung getätigt wurde, wird die Sitzung vom 20. Oktober 2012 eine Stunde nach hinten geschoben. Ändern Sie die Reservierung entsprechend.

2.5.1.2 Testperson

Die Usability Tests wurden mit Julian Heeb von der SCS durchgeführt. Julian Heeb ist selbst Kunde bei der BTZ und macht vom Transportservice täglich gebrauch.

Abbildung 2.45: Julian Heeb

2.5.1.3 Feedback von Usability Tester aufgrund der Paper Prototypes

Die Usability Tests wurden mit Julian Heeb, einem zukünftigen Benutzer der iPhone App, durchgeführt. Hier seine Inputs:

- Labelbeschriftung “Start” durch “Zeit” ersetzen
- Der nächste Fahrauftrag soll zuoberst in der Liste erscheinen (Hauptübersicht)

2.5.1.4 Feedback vom Betreuer aufgrund der Paper Prototypes

Aufgabenstellung ändern, man sollte den Benutzer nicht sagen, dass er die Reservierung ändern möchte, sondern dass er z.B. eine Stunde später ins Büro fahren möchte.

2.5.1.5 Feedback von Usability Tester aufgrund der fertigen iPhone App

Beim zweiten Usability Test anhand der fertigen iPhone App haben sich folgende Resultate ergeben:

- Beim Tippen auf den Pfeil "Neue Reservation hinzufügen" wird die Tastatur ein- und ausgeblendet.
- Für eine spätere Weiterentwicklung der App wären Vorschläge bei der Eingabe der Adresse nützlich, da ich vielleicht nicht immer die genaue Postleitzahl weiss. Für den Moment ist es absolut ok, auch die Vorschläge bereits eingegebener Adressen ist nützlich.
- Der Begriff "Status" verwirrt mich ein wenig. Der Text, der im Feld angezeigt wird, ist eher die "Letzte vorgenommene Aktion". Evtl. wäre es sinnvoll den Text des Icons hinzuschreiben statt die Benutzeraktion.

2.5.1.6 Verarbeitung des Feedbacks

Die Resultate des ersten Usability Tests wurden ausgewertet und während der Entwicklung der externen Architektur berücksichtigt.
Das Feedback für den zweiten Usability Test anhand der fertig App konnten nicht mehr komplett verarbeitet werden. Folgende Entscheide wurden getroffen:

- Der erwähnte Bug betreffend "Neue Reservation hinzufügen" wurde behoben.
- Automatische Adressvorschläge würden den grundsätzlichen Funktionsumfang der App erweitern und stellen keine einfachen GUI-Anpassungen dar. Das Feature war optional geplant, wurde aus Zeitgründen jedoch nicht mehr umgesetzt. In einem weiteren Release wäre eine Implementation jedoch durchaus denkbar.
- Das Konzept der Anzeige der verschiedenen Reservationsstatus müsste ggf. nochmals überdenkt werden. Dies wird auf Zeitgründen jedoch auf einen späteren Release verschoben.

2.5.2 Unit Testing

2.5.2.1 Webservice

Auf dem Server wurden die Layer Service und DAL getestet. Im Data Access Layer wird jedoch nur die Klasse DataPortal getestet, da alle anderen Klasse keine Logik enthalten.
Um den Datenbankzugriff wahrheitsgetreu zu testen, wurde eine Testdatenbank erstellt. Die Unittests greifen auf diese BTZTest Datenbank zu. Vor jedem Test wird diese gelöscht und alle Daten neu eingefügt.
Folgende Unittest Klassen mit den entsprechenden Unittests sind vorhanden:

DataPortalTest	
	TestAddCustomer
	TestAddReservation
	TestGetAllReservationsForCustomer
	TestUpdateReservation
DtoConverterTest	
	TestConvertToDtoAddress
	TestConvertToEntityAddress
	TestConvertToDtoAndEntitiyAddressNull
	TestConvertToDtoCustomer
	TestConvertToEntitiyCustomer
	TestConvertToDtoAndEntitiyCustomerNull
	TestConvertToDtoReservation
	TestConvertToEntitiyReservation
	TestConvertToDtoAndEntitiyReservationNull
	TestConvertToDtoReservationLog
	TestConvertToEntityReservationLog
	TestConvertToDtoAndEntityReservationLogNull
	TestConvertToListNull
ServiceTest	
	TestAddCustomer
	TestAddReservation
	TestAddReservationLog
	TestGetAllReservationsForCustomer
	TestUpdateReservation

Tabelle 2.29: Übersicht Unit Tests

2.5.2.1.1 Code Coverage

Hier wurde eine Code Coverage Analyse erstellt. Im Folgenden sieht man, dass die zu testenden Klassen eine Test Abdeckung von 100% haben.

Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
Administrator@SINV-56020 2012-12-12 11:33:34	122	18.63%	533	81.37%
BTZServer.DAL.dll	122	27.35%	324	72.65%
{} BTZServer.DAL	122	27.35%	324	72.65%
Address	25	49.02%	26	50.98%
BTZEntities	20	50.00%	20	50.00%
Customer	16	38.10%	26	61.90%
DataPortal	0	0.00%	127	100.00%
Reservation	36	29.27%	87	70.73%
ReservationLog	25	39.68%	38	60.32%
BTZServer.Service.dll	0	0.00%	209	100.00%
{} BTZServer.Service	0	0.00%	209	100.00%
DtoConverter	0	0.00%	168	100.00%
ServerService	0	0.00%	41	100.00%

Abbildung 2.46: Code Coverage Ergebnis

2.5.3 Metriken

2.5.3.0.2 Softwaremetriken

Softwaremetriken standardisiert nach IEEE 1061 [74998] dienen zur Überprüfung der Codequalität. Werden sie regelmässig überprüft, können Abweichungen von vorgegebenen Qualitätszielen frühzeitig erkannt und behoben werden.

Um die Qualität des Servercodes anhand eines interpretierbaren Wertes zu testen, haben wir zwei Metriken eingesetzt, die direkt mit den Visual Studio mitgeliefert werden. Zum einen wurden die Klassen nach ihrer Wartbarkeit, Komplexität und der Vererbungstiefe analysiert und zum anderen die Testabdeckung.

2.5.3.0.3 Code Metrics

Die Code Metrics haben einen sehr guten Wert geliefert, wie man anhand der unter gezeigten Grafik erkennen kann. Keine Klasse hat mehr als 100 Codezeilen. Die Vererbungstiefe beträgt maximal 3 Vererbungen, was jedoch vom Entity Framework abhängt. Auch die Komplexität der Methoden wurde möglichst klein gehalten. Lange Methoden wurden in kleinere, besser testbare Methoden unterteilt. Die Maintainability ist ein Index, der anzeigt wie gut wartbar der Code ist aufgrund aller Zahlen. Auch hier sind die Werte sehr gut.

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Lines of Code
BTZServer (Debug)	82	33	3	74
BTZServer	84	16	2	25
BTZServer.Controllers	80	17	3	49
CustomerController	73	4	2	12
HomeController	96	2	3	3
ReservationsController	73	8	2	25
ReservationsLogController	77	3	2	9
BTZServer.Common (Debug)	93	93	2	86
BTZServer.Common.DataTransferObjects	93	93	2	86
AddressDto	89	14	2	16
CustomerDto	89	14	2	16
DtoBase	95	3	1	3
ProcessStatus	100	0	1	0
ReservationDto	88	40	2	31
ReservationLogDto	89	22	2	20
ReservationStatus	100	0	1	0
BTZServer.Dal (Debug)	79	116	3	299
BTZServer.DAL	79	116	3	299
Address	82	17	3	37
BTZEntities	82	15	2	26
Customer	82	14	3	33
DataPortal	65	11	1	81
Reservation	82	39	3	76
ReservationLog	82	20	3	46
BTZServer.Service (Debug)	73	33	1	85
BTZServer.Service	73	33	1	85
DtoConverter	70	22	1	52
ServerService	76	11	1	33
BTZServer.Testing (Debug)	68	43	1	188
BTZServer.Testing	68	43	1	188
DataPortalTest	72	7	1	23
DtoConverterTest	67	14	1	73
ServiceTest	67	9	1	35
TestEnvironmentHelper	64	13	1	57

Abbildung 2.47: Code Metrics Ergebnisse

2.5.4 Systemtests

2.5.4.1 Testumgebung

Die Testumgebung umfasst folgende Komponenten:

- Server: Virtueller HSR Entwicklungsserver mit leerer Testdatenbank
- Call-Center Client: Studienarbeits Lab-PC mit Windows 7 und installiertem Call-Center Client
- iPhone App: Privates iPhone 3GS / iOS6 / BTZ-App

2.5.4.2 Vorgehen

Bei der Entwicklung der Testfälle haben wir in einem ersten Schritt sämtliche funktionalen Anforderungen berücksichtigt. In einem zweiten Schritt wurde das State Machine Diagramm aus Kapitel xyz beigezogen und die Testfälle soweit ergänzt, dass sämtliche Statusübergänge abgedeckt werden. In einem dritten Schritt wurden dann noch die nicht funktionalen Anforderungen beigezogen.

Als Ausgangslage wird eine leere Testdatenbank verwendet, die während den Tests kontinuierlich mit Testdaten aufgefüllt wird. Die eingegebenen Daten werden jeweils separat notiert, über die jeweilige Komponente (beispielsweise die iPhone App) ins System eingepflegt und in den anderen Komponenten (beispielsweise Datenbank, Call-Center Client) auf ihre Konsistenz überprüft. So wird die korrekte Übertragung der Daten über die HTTP/JSON Schnittstelle gleich mitgeprüft.

Anzumerken ist, dass auf Tests der optionalen Call-Center Client Applikation verzichtet wurde. Der Client ist zwar Teil der Testumgebung, jedoch nur um die Handhabung des Testens zu vereinfachen.

Use Case #	**Use Case**	**Test #**	**Bemerkung**
UC1	Registrieren	1	
UC2	Reservierung hinzufügen	2	
UC3	Reservierung ändern	4	
UC4	Reservierung löschen	5	
UC5	Benachrichtigung bei Statusänderung	-	optional, nicht implementiert
UC6	Reservierung verarbeiten	2	
UC6.1	Reservierung bestätigen	2	
UC6.2	Reservierung ablehnen	3	

Tabelle 2.30: Testabdeckung funktionale Anforderungen (Use Cases)

2.5.4.2.1 Resultate

Die Resultate sind wie folgt zu interpretieren:

- Testfälle 1 - 14, Überprüfung der funktionalen Anforderungen.
- Testfälle 15 - 18, Überprüfung der nicht funktionalen Anforderungen.

#	Datum	Testablauf	Resultat
1	12.12.2012	1. Registrierung beim ersten App-Start - Kunde registriert sich mit Vorname, Nachname und Telefonnummer, Daten werden korrekt an den Webservice übertragen und in der Datenbank gespeichert.	PASS
2	12.12.2012	1. Reservierung hinzufügen (iPhone) 2. Reservierung akzeptieren (Call-Center Client)	PASS
3	12.12.2012	1. Reservierung hinzufügen (iPhone) 2. Reservierung ablehnen (Call-Center Client)	PASS
4	12.12.2012	1. Reservierung hinzufügen (iPhone) 2. Reservierung ablehnen (Call-Center Client) 3. Reservierung bearbeiten (iPhone) 4. Reservierung akzeptieren (Call-Center Client)	PASS
5	12.12.2012	1. Reservierung hinzufügen (iPhone) 2. Reservierung akzeptieren (Call Center Client) 3. Reservierung löschen (iPhone) 4. Reservierung (Löschung) akzeptieren (Call-Center Client)	PASS
6	12.12.2012	1. Reservierung hinzufügen (iPhone) 2. Reservierung bearbeiten (iPhone) 3. Reservierung akzeptieren (Call-Center Client)	PASS
7	12.12.2012	1. Reservierung hinzufügen (iPhone) 2. Reservierung ablehnen (Call-Center Client) 3. Reservierung bearbeiten (iPhone) 4. Reservierung (Bearbeitung) ablehnen (Call-Center Client)	PASS

Tabelle 2.31: Übersicht Systemtests 1-7

#	Datum	Testablauf	Resultat
8	12.12.2012	1. Reservierung hinzufügen (iPhone) 2. Reservierung ablehnen (Call-Center Client) 3. Reservierung bearbeiten (iPhone) 4. Reservierung (Bearbeitung) ablehnen (Call-Center Client) 5. Reservierung bearbeiten (iPhone) 6. Reservierung akzeptieren (Call-Center Client)	PASS
9	12.12.2012	1. Reservierung hinzufügen (iPhone) 2. Reservierung akzeptieren (Call Center Client) 3. Reservierung bearbeiten (iPhone) 4. Reservierung akzeptieren (Call Center Client) 5. Reservierung bearbeiten (iPhone) 6. Reservierung akzeptieren (Call Center Client)	PASS
10	12.12.2012	1. Reservierung hinzufügen 2. Reservierung ablehnen 3. Reservierung löschen 4. Reservierung (Löschung) akzeptieren	PASS
11	12.12.2012	1. Reservierung hinzufügen 2. Reservierung akzeptieren 3. Reservierung löschen 4. Reservierung (Löschung) akzeptieren	PASS
12	12.12.2012	1. Reservierung hinzufügen (iPhone) 2. Reservierung bearbeiten (iPhone) 3. Reservierung löschen (iPhone) 4. Reservierung akzeptieren (Call-Center Client)	PASS

Tabelle 2.32: Übersicht Systemtests 8-12

#	Datum	Testablauf	Resultat
13	12.12.2012	1. Reservierung hinzufügen (iPhone) 2. Reservierung ablehnen (Call-Center Client) 3. Reservierung bearbeiten (iPhone) 4. Reservierung (Bearbeitung) ablehnen (Call-Center Client) 5. Reservierung löschen (iPhone) 6. Reservierung (Löschung) akzeptieren (Call-Center Client)	PASS
14	12.12.2012	1. Reservierung hinzufügen (iPhone) 2. Reservierung ablehnen (Call-Center Client) 3. Reservierung bearbeiten (iPhone) 4. Reservierung (Bearbeitung) akzeptieren (Call-Center Client) 5. Reservierung löschen (iPhone) 6. Reservierung (Löschung) akzeptieren (Call-Center Client)	PASS
15		Datenkonsistenz ist während der Übertragung gewährleistet. 1. Wurde bereits in den Systemtests #1 - 14 überprüft, indem bei der manuellen Eingabe jeweils sämtliche eingegebenen Daten notiert und nach der Verarbeitung überprüft wurden.	PASS
16	13.12.2012	Datenmigration von einem iPhone auf ein anderes. 1. FirstAppStart View erscheint 2. Registrierung mit identischen Benutzerdaten 3. Bereits erstellte Reservierungen werden heruntergeladen	PASS
17	13.12.2012	Fehlerbenachrichtigung bei Verbindungsabbruch 1. AlertView benachrichtigt den Benutzer, dass die Übertragung nicht stattgefunden hat	PASS
18	13.12.2012	Dauer Interaktion mit dem Server (Reservierung erstellen, ändern,..) bei guter Netzverbindung 1. Dauer < 1 sec	PASS

Tabelle 2.33: Übersicht Systemtests 13-18

Kapitel 3

Projektmanagement

3.1 Projektorganisation

Das Projekt wurde nach dem Softwareentwicklungsvorgehensmodell [IBM12] durchgeführt.

3.1.1 Organisationsstruktur Projekt

Die RUP Disziplin "Implementierung" des Projektes wurde in zwei grobe Teilbereiche unterteilt: ".NET Entwicklung" und "iOS Entwicklung", dem jeweils ein/e Hauptverantwortliche/r zugeteilt wurde. Die restlichen Disziplinen (Anforderungsspezifikationen, Software Analyse, Software Design,..) wurden gemeinsam bewältigt.

- **Simon Stäheli** iOS Entwicklung
- **Patrizia Heer** .NET Entwicklerung

3.1.2 Externe Partner

- **Beni Ammeter** BTZ[1], Industriepartner
- **Julian Heeb** SCS[2], Projektpartner
- **Markus Stolze** HSR[3], Betreuer
- **Sebastian Hunkeler** HSR, Betreuer Assistent

3.2 Projektverlauf

3.2.1 Meilensteine

Folgende Meilensteine wurden im Projektplan während der Planungsphase definiert. Diese konnten bis in die Mitte des Projektes auch wie geplant eingehalten werden. Aufgrund des erhöhten Zeitaufwandes während der Implementierungsphase gab es in der zweiten Hälfte des Projektes Verzögerungen. Weitere Informationen bezüglich Zeitauswertung finden sich im Kapitel 3.2.3.2 Analyse der Zeitauswertung

[1] Behinderten-Transporte Zürich
[2] Super Computing Systems AG
[3] Hochschule für Technik, Rapperswil

Meilenstein	**Geplant**	**Durchgeführt**	**Inhalt**
MS1	Woche 1	Woche 1	• Kickoff • Vorgehensmodell definiert
MS2	Woche 1-2	Woche 2	• Use Case Diagramm • Projektplan aktualisiert
MS3	Woche 3	Woche 3	• Paper Prototypes 1. Version
MS4	Woche 4	Woche 4	• Domain Models • Review Inhaltsverzeichnis
MS5	Woche 5	Woche 5	• Paper Prototypes 2. Version
MS6	Woche 6	Woche 6	• Paper Prototypes Final Version • Feature Sets festgelegt • Benutzerbeobachtung / Befragung beendet
MS7	Woche 7	Woche 9	• Prototyp Demo • Projektplan aktualisiert
MS8	Woche 8-12	Woche 12	• Feature freeze • Projektplan aktualisiert
MS9	Woche 13	Woche 14	• Draft Wikipage • Draft Video
MS10	Woche 14	Woche 14	• Abgabe

Table 3.1: Meilensteine

3.2.2 Release

Release	Zeitpunkt	Inhalt
R1	02.11.2012	Architekturprototyp
R2	26.11.2012	Bereit für Code Review
R3	21.12.2012	Abgabe

Table 3.2: Release Übersicht

3.2.3 Zeitauswertung

Es folgt eine Auswertung der benötigten Arbeitsstunden pro definiertes Arbeitspaket.

Arbeitspaket	Arbeitsphase	SOLL	IST
Projekt Management		26	25.5
	Projekt Inhalt definieren	2	5
	Zeitplan erstellen / ändern	4	7.5
	Coding Guidelines	4	2
	Dokumenten Vorlage	4	1
	Evaluation Technologien	4	6
	Risiken evaluieren	8	4
Einarbeitung		19	43
	iOS	11	34
	Webservice	8	9
Requirements		10	7.5
	Anforderungsspezifikation	10	7.5
Analyse		25	18
	iOS Client Domainmodell	7	7
	Server Domainmodell	7	6
	Call-Center Domainmodell	5	2
	Use Case Modell	6	3

Table 3.4: Arbeitspakete Teil 1

Arbeitspaket	**Arbeitsphase**	**SOLL**	**IST**
Design		44	46
	iOS Client Klassendiagramm	6	1
	Server Klassendiagramm	6	9
	Call-Center Klassendiagramm	6	9
	Datenbank Modell	4	5
	Paper Prototypes	10	8
	Architekturdokument	12	14
Architekturprototyp		92	127
	iOS Client Implementation	46	57
	Server Implementation	26	39
	Call-Center Implementation	16	19
	Datenbank	4	12
iOS Client Implementation		76	116
	Serverzugriff	23	20
	GUI	37	95
	*Push Notification	16	1
Server Implementation		34	22
	Webservice	20	14
	Datenbankzugriff	14	8
Call-Center Client		26	20
	Serverzugriff	14	11
	GUI	12	9
Datenbank		4	3
	Modell implementieren	4	3

Table 3.6: Arbeitspakete Teil 2

Arbeitspaket	**Arbeitsphase**	**SOLL**	**IST**
Testing		44	44
	Server Unittest	12	15
	BTZ Client Unittest	12	4
	Usability Tests	8	13
	System Tests	12	12
Infrastruktur		4	11
	Git einrichten	2	3
	IDE/Workstation/Server konfigurieren	2	8
Organisatorisches		80	113
	Betreuer	28	15
	Review	6	5
	Nachbeabreitung Review	12	5
	BTZ Meetings	8	5
	Protokoll schreiben	14	7
	Video erstellen	12	8
	Dokumentation		68
Total aufgewendete Stunden		**484**	**596**

Table 3.8: Arbeitspakete Teil 3

3.2.3.1 IST / Soll Vergleich

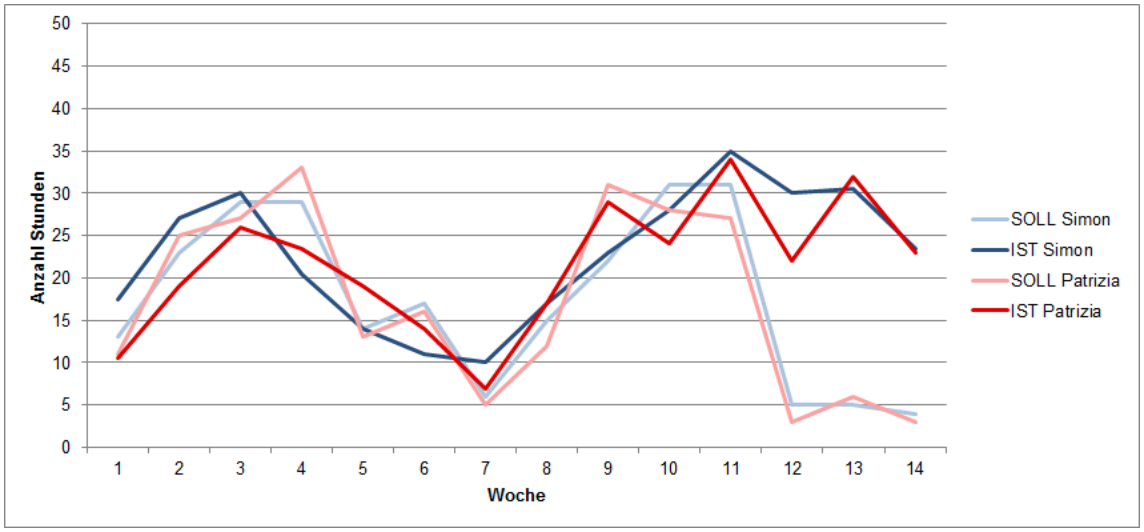


Figure 3.1: IST / SOLL Stundenvergleich pro Person

3.2.3.2 Analyse der Zeitauswertung

Die erste Hälfte des Projektes und damit die Software Analyse und Teile des Software Designs konnten zeitlich ziemlich genau nach Projektplan durchgeführt werden. Ab Woche 7 gabs es dann einige ungeplanten Mehraufwände, die zu allgemeinen Verzögerungen führten. Ein erstes Problem war das schlecht dokumentierte Framework RestKit, dass wir in der iPhone App verwendet haben. Hinzu kam ein zeitintensiver Release in einem anderen Projekt, der zwar vorhersehbar war (sichtbar an den SOLL-Graphen in Woche 7, in Komibination mit den RestKit-Problemen), dann aber zu Verzögerungen führte.
Etwas unterschätzt wurde zudem die Gestaltung des GUI der iPhone App. Das Grundgerüst der App Stand zwar erstaunlich schnell, Kleinigkeiten wie Textfeldvalidierungen, voneinander abhängige GUI Komponenten und andere Perfektionierungen nahmen dann aber mehr Zeit in Anspruch als geplant. Dies führte einerseits zu einem Mehraufwand bezüglich der gearbeiteten Stunden und andererseits zum Weglassen von Funktionen in der iPhone App die gemäss Kapitel 2.1.6 Übersicht Funktionalität als optional definiert wurden.
Die Implementation des Webservices benötigte weniger Zeit als geplant, was die zusätzliche Entwicklung eines Prototypen des Call-Center Clients ermöglichte. Dieser wurde jedoch im Status eines Prototypen belassen, da wir uns beide ab Woche 9 schliesslich auf die iPhone App konzentrierten.
Die eingeplanten zwei Wochen Pufferzeit zum Ende des Projektes wurde schlussendlich für die im Verlauf der Arbeit etwas vernachlässigte Dokumentation verwendet. So konnte das Projekt dennoch fristgerecht abgechlossen werden.

3.3 Infrastruktur

Es folgt eine kurze, tabellarische Dokumentation über die verwendeten Werzeuge und Hardwarekomponenten, die während dem Projekt verwendet wurden.

3.3.1 Hardware

Anzahl	Bezeichnung	Betriebssystem
2x	iMac	OSX 10.8
2x	MacBookPro	OSX 10.8
2x	HSR Workstation	Windows 7
1x	Virtueller HSR Server	Windows Server 2008
1x	iPhone 4	iOS 6
1x	iPhone 3GS	iOS 6

Table 3.9: Hardware Übersicht

3.3.2 Software

3.3.2.1 Entwicklungs Tools

Software	Zweck
XCode 4.x	Entiwcklungsumgebung für iPhone App
Restkit 0.10.3	Restful Webservice Zugriff
Core Data [iOS 6]	Caching in der iPhone App
Visual Studio 2010	Entwicklungsumgebung
ASP.NET MVC 4	Webservice Framework
Entitiy Framework 4	Datenbank Zugriff Webservice
XAML	GUI Framework
Resharper 7	Refactoring und Produktivitäts Tool für Visual Studio
JSON.Net	JSON Parser für Visual Studio 2010

Table 3.10: Entwicklungswerkzeuge Übersicht

3.3.2.2 Projektmanagement Tools

Software	**Zweck**
git	Versionsmanagement
Lyx	Projektdokumentation
Jira	Task-/Bugtracking
Excel	Zeiterfassung

Table 3.11: Projektmanagementwerkzeuge Übersicht

3.3.2.3 Modellierung / Bildbearbeitung Tools

Software	**Zweck**
Enterprise Architect	Modellierung
Visio	Übersichtsgrafiken
Adobe Photoshop 6	Bildbearbeitung
Adobe Illustrator 6	App Icon Erstellung

Table 3.12: Bildbearbeitungswerkzeuge Übersicht

Kapitel 4

Anhang

4.1 Usability Tests

4.1.0.4 Erster App Start

Abbildung 4.1: Registrierung

Beim ersten öffnen der App muss man sich registrieren.

Abbildung 4.2: Registrierung ausgefüllt

Sind alle Daten eingegeben, sieht das folgendermassen aus:

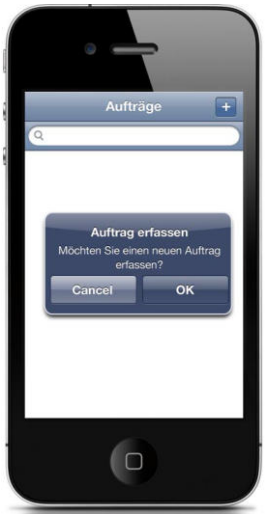


Abbildung 4.3: Erstes Öffnen

Danach erscheint eine leere Tabelle mit einem Hinweis, wie man weiterfahren kann:

4.1.0.5 Erste Reservation tätigen

Abbildung 4.4: Eingabemaske

Abbildung 4.5: Adresseingabe

Um die Adresse optimal eingeben zu können, erscheint unter dem Textfeld eine Eingabehilfe:

Abbildung 4.6: Reservation eingegeben

Die ausgefüllte Reservation sieht so aus.

Abbildung 4.7: Reservations Übersicht vor dem Akzeptieren

Auf der Übersichtsseite erscheint die noch nicht akzeptierte Reservation.

Abbildung 4.8: Reservations Übersicht nach dem Akzeptieren

Auf der Übersichtsseite erscheint die noch nicht akzeptierte Reservation.

Abbildung 4.9: Reservations Details

In der Detail Ansicht der Reservation, kann man nun auch sehen, dass sich der Status geändert hat.

Abbildung 4.10: Status Details

Was mit dem Auftrag bereits alles passiert ist,
kann man in den Status Details sehen.

4.1.0.6 Weitere Reservierungen wurden getätigt

Abbildung 4.11: Reservations Übersicht

4.1.0.7 Neue Reservation

Abbildung 4.12: Eingabemaske

Abbildung 4.13: Reservation eingegeben

Folgend sind die Daten der neuen Reservation eingegeben.

Abbildung 4.14: Reservations Übersicht vor dem akzeptieren

In der Übersicht erscheint die neue Reservation als in Bearbeitung.

Abbildung 4.15: Reservations Übersicht nach dem akzeptieren

Nach dem das BTZ die Reservation akzeptiert hat wird der Status auf der Übersichtsseite geändert.

Abbildung 4.16: Details der Reservation

Auch in der Detailansicht hat sich der Status geändert.

Abbildung 4.17: Status Details

Und zuletzt kann man die Statusänderung auch in den Details des Status sehen.

4.1.0.8 Reservierung bearbeiten

Abbildung 4.18: Reservation geändert

Für die zuletzt getätigte Reservation wird die Abholzeit geändert.

Abbildung 4.19: Übersicht nach der Bearbeitung, vor dem akzeptieren

Der Status wird wieder auf In Bearbeitung gesetzt.

Abbildung 4.20: Übersicht nach der Bearbeitung, nach dem akzeptieren

Abbildung 4.21: Detailansicht

Nach dem das BTZ die Änderung akzeptiert hat, ändert sich wiederum überall der Status.

Abbildung 4.22: Status Details

Abbildungsverzeichnis

Literaturverzeichnis

[74998] Ieee standard for a software quality metrics methodology. *IEEE Std 1061-1998*, page i, dec. 1998.

[Con11] Joe Conway. *iOS programming : the big nerd ranch guide*. big nerd ranch, Atlanta GA, 2. print. edition, 2011.

[Fie99a] Roy Thomas Fielding. Http-delete. http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.7, 1999.

[Fie99b] Roy Thomas Fielding. Http-post. http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.5, 1999.

[Fie99c] Roy Thomas Fielding. Http status codes. http://www.w3.org/Protocols/rfc2616/rfc2616.html, 1999.

[Fie00] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures, 2000.

[IBM12] IBM. Rup. http://www.ibm.com/software/awdtools/rup/, 2012.

[Jso] Json.net. http://json.codeplex.com/.

[Lar04] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Addison Wesley Professional, third edition, 2004.

[Mac06] Lori A. MacVittie. *XAML - in a nutshell: a desktop quick reference*. 2006.

[Mica] Microsoft. Code metrics. http://msdn.microsoft.com/en-us/library/bb385914.aspx.

[Micb] Microsoft. Microsoft sql server express. http://www.microsoft.com/sqlserver/en/us/editions/2012-editions/express.aspx.

[Micc] Microsoft. Mvc4. http://www.asp.net/mvc/mvc4.

[WK10] James Wightman and S Klein. *Pro Entity Framework 4.0*. Pro. Springer, Dordrecht, 2010.

Glossar

BTZ Behinderten-Transporte Zürich
Der Mobilitätsservice mit Spezialfahrzeugen ist für Mobilitätsbehinderte im Rollstuhl reserviert. Die Stiftung wurde 1990 durch eine Volksabstimmung in der Stadt Zürich gegründet. Die Stadt finanziert das Defizit der Stiftung mit Steuereinnahmen. Ebenfalls unterstützt wird BTZ von ProMobil.

IIS Internet Information Services
Webserver der Firma Microsoft der den Betrieb von Weservices auf Basis von .NET Technologien ermöglicht.

JSON JavaScript Object Notation
Ein Datenformat in für Mensch und Maschine lesbarer Textform für den Datenaustausch zwischen Anwendungen.

XAML Extensible Application Markup Language
Beschreibungssprache für die Oberflächengestaltung in .NET.

Reservation Ausdruck der Schweizer Standardsprache.
Gleichbedeutend zum Ausdruck “Reservierung”.