Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Hoang Do

Konzeption und Implementierung einer mobilen Anwendung zur vereinfachten Zeitbuchung

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Hoang Do

Konzeption und Implementierung einer mobilen Anwendung zur vereinfachten Zeitbuchung.

Bachelorarbeit eingereicht im Rahmen Bachelorprüfung

im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Stefan Sarstedt
Zweitgutachter : Prof. Dr. Wolfgang Gerken

Abgegeben am 15.11.2013

Hoang Do

Thema der Arbeit
Konzeption und Implementierung einer mobilen Anwendung zur vereinfachten Zeitbuchung

Stichworte
Mobile Anwendung, Android, iPhone, Window Phone, mobile Browser, jQuery, jQueryMobile, Sorfware Engineering, Teamwarp, Zeitbuchung.

Kurzzusammenfassung
Die Erfassung der Arbeitszeiten spielt in der Betriebswirtschaft eine sehr wichtige Rolle. Durch die geleisteten Arbeitszeiten kann der Aufwand einer Aufgabe oder eines ganzen Projekts ermittelt werden. Im Rahmen dieser Bachelorarbeit soll eine mobile Anwendung zur Vereinfachung der Erfassung der Arbeitszeiten realisiert werden. Diese Anwendung sollte einen Teil des traditionellen Buchungssystems ersetzen. Mittels dieser Anwendung sollte Mitarbeiter darüber verfügen, die Arbeitszeiten an sogenannte generische Tätigkeiten besser zu ermitteln. Unter einer mobilen Anwendung versteht man ein Softwareprogramm, das auf mobilen Geräten bzw. unter mobilen Betriebssystemen ausgeführt wird. Es werden die möglichen Technologien zur Umsetzung einer mobilen Anwendung vorgestellt. In dieser Arbeit werden die einzelnen Entwicklungsphasen wie Analyse, Spezifikation, Entwurf, Implementierung und Test nach den Standards des Software Engineerings erläutert. Auf die beiden Phasen Spezifikation und Entwurf werden Fokus gesetzt. In der Spezifikation werden Konzeptionen bestimmt, welche zu einer guten mobilen Anwendung beiträgt. Der Entwurf legt die eingesetzte Entwicklungstechnologie fest und behandelt die Architektur des Systems. Nach der Implementierung sollte die Evaluierung erläutert werden. Dabei sollte die Anwendung getestet und die Testergebnisse ausgewertet werden.

Hoang Do

Title of the paper
The conception and implementation of a mobile Application on simplifying the time-booking

Keywords
Mobil application, Android, iPhone, Window Phone, Mobile Browser, jQuery, jQueryMobile, Software Engineering, Time-booking

Abstract

The collection of working hours plays a very important role in the business management. Through the worked hours it is possible to determine the effort of a

task or an entire project. The purpose of this thesis is to show the implementation of a mobile application which facilitates the collection of said working hours. This application aims to replace a part of the traditional accounting system. This application should allow co-workers to determine their working hours to so called "generic tasks". A mobile application is a software program that runs on mobile devices or under a mobile operating system. The possible technologies for the implementation of a mobile application will be presented. This thesis covers every development phase in accordance to the software engineering, like analysis, specification, design, implementation and testing. The thesis focuses on both the specification and design. The specification offers some conceptions for a good mobile application. The design establishes the used development technology and handles the architecture of the system. The evaluation should be explained after the implementation. The application should then be tested and the results evaluated.

Inhaltsverzeichnis

Literaturverzeichnis

Adamski, Bernhard. 1998. *Praktisches Arbeitszeitmanagement.* Solingen : Raimund Roth GmbH, 1998. 3-89577-086-8.

Apple. iOS Developer Program. [Online] [Zitat vom: 13. 03 2013.] https://developer.apple.com/programs/ios/.

Franke, Florian und Ippen, Johannes. 2012. *Apps mit HTML5 und CSS3 : für iPad, iPhone und Android.* s.l. : Galileo Computing, 2012. 3-8362-1848-8.

Funk, Marcus. 2011. Hybrid App Entwicklung. [Online] 2011. [Zitat vom: 10. 06 2013.] http://www.flyacts.com/hybrid-app-entwicklung.

Galileo. 2013. Entwicklung und Funktion hybrider Apps. [Online] 2013. [Zitat vom: 14. 06 2013.] http://www.galileo-webdesign.de/app-entwicklung/hybride-apps.html.

Glanzkinder GmbH & Co. KG. 2012. [Online] 2012. [Zitat vom: 10. 06 2013.] http://web-app.de/vor-und-nachteile/.

Goll, Joachim und Dausmann, Manfred. 2013. *Architektur- und Entwurfsmuster der Softwaretechnik.* Wiesbaden : Springer Vieweg, 2013. 978-3-8348-2431-8.

Google. Entwickler-Registrierung bei Google Play. [Online] [Zitat vom: 13. 06 2013.] https://support.google.com/googleplay/android-developer/answer/113468?hl=de.

Mobafone. 2013. Native Apps und WebApps. [Online] 2013. [Zitat vom: 10. 06 2013.] http://www.smartphone-app-entwicklung.de/index.php/de/native-apps-and-webapps.

poqstudio. 2012. Hybrid apps vs native apps – what does it all mean for ecommerce? [Online] 2012. [Zitat vom: 14. 06 2013.] http://poqstudio.com/2013/05/hybrid-apps-vs-native-apps-what-does-it-all-mean-for-ecommerce/.

Seven, Doug. 2012. What is a Hybrid Mobile App? [Online] 14. 06 2012. [Zitat vom: 17. 06 2013.] http://www.icenium.com/blog/icenium-team-blog/2012/06/14/what-is-a-hybrid-mobile-app-.

Steyer, Ralph. 2013. *Apps mit PhoneGap entwickeln.* Deutschland : Hanser, 2013.

—. 2013. *Apps mit PhoneGap entwickeln : universelle Web-Apps plattformneutral programmieren.* München : Hanser, 2013. 3-446-43510-7.

Würstl, Daniel. 2013. Unterschiede und Vergleich native Apps vs. Web Apps. [Online] 21. 02 2013. [Zitat vom: 10. 06 2013.] http://www.app-entwickler-verzeichnis.de/faq-app-entwicklung/11-definitionen/107-unterschiede-und-vergleich-native-apps-vs-web-apps.

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

1.1 Motivation

Die Personalzeiterfassung ist die Erfassung von Arbeitszeiten des Arbeitsnehmers und spielt in der Betriebswirtschaft eine wichtige Rolle. In vielen Unternehmen werden mittlerweile nicht nur die allgemeinen Arbeitszeiten erfassen. Oft kommt auch eine projektspezifische bzw. tätigkeitbezogene Zeiterfassung zum Einsatz. (Adamski, 1998 S. 28)

Während diese in kleinen Unternehmen meistens mit Formularen handschriftlich erfasst oder in Excellisten eingetragen werden, nutzen mittlere und große Unternehmen oft spezielle Software-Lösungen. Bei der Projektzeiterfassung beispielweise werden Arbeitszeiten erfasst und zugleich Projekten zugeordnet. Dies ist besonders für das Projektcontrolling und die Projektabrechnung wichtig, da so Auswertungen über den erfolgten Arbeitsaufwand ermöglicht werden. (Adamski, 1998)

Je nachdem, welche Daten erfasst werden, kann man Tätigkeitsanalysen der Mitarbeiter durchführen sowie die Wirtschaftlichkeit der Projekte berechnen. Letztendlich macht es vor allem Sinn wenn für das Projekt eine feste Stundenanzahl pauschal festgelegt wird. Oft werden die geleisteten Stunden den Kunden in Rechnung gestellt. (Adamski, 1998)

Trotz der modernen Buchungsmethoden werden die Arbeitszeiten im Allgemein noch ungenau von Mitarbeitern selber eingegeben, was zu einer fehlerhaften Aufwandsrechnung führen kann. Die Firma mgm-technology partners stellt hierfür ein Beispiel dar. Bei mgm-tp benutzen Mitarbeiter eine web-basierte Anwendung, um die Arbeitszeiten den Aufgaben zuzuordnen. Wegen der hohen Komplexität des Dialogs, der zahlreichen Buchungskonten und der fehlenden Kopplung mit dem Ticketsystem, indem die Aufgaben organisiert sind, ist die Personalzeiterfassung durch die web-basierte Buchungsanwendung zeitaufwändig und fehleranfällig.
Um die Genauigkeit des Zeitbuchungsvorgangs zu verbessern, wird im Rahmen dieser Arbeit eine mobile Anwendung entwickelt. Diese Anwendung, mit dem Name „iWarp“, soll

das bisherige Zeiterfassungssystem nicht vollständig ersetzen, sondern nur den wichtigsten Teil der Buchung durch eine exakte Zeitmessung sowie eine Kopplung mit dem externen Ticketsystem verbessern und vereinfachen.

1.2 Ziele der Arbeit

Das Unternehmen mgm-technology partners stellt den Mitarbeitern ein mordenes Zeiterfassungssystem zur Verfügung. Um dieses System zu verbessern, soll in dieser Arbeit eine mobile Anwendung erstellt werden, welche den Buchungsvorgang wesentlich vereinfacht.
Zu den wichtigsten Zielen der Arbeit gehört es, die Vorrausetzung zu schaffen, dass sich die Mitarbeiter schnell mit dem neuen Buchungssystem zurechtfinden und effizient damit arbeiten.
Die Nutzung soll möglichst einfach sein und mit wenigen Berührungen erfolgen. Es wird erwartet, dass die Benutzerschnittstelle einfach gestaltet ist, sodass sich die Benutzer bei der erstmaligen Nutzung orientieren können, wie sie ihre Ziele erreichen können.
Weiterhin wäre es wünschenswert, wenn das im Rahmen dieser Arbeit realisierte Buchungssystem einen Teil des traditionellen Systems ersetzen könnte. Um dieses absoluten Ziel zu erreichen, sind eine sorgfältige Anforderungsanalyse, eine gut strukturierte Architektur sowie eine saubere Implementierung Voraussetzung.

1.3 Aufbau der Arbeit

Die Arbeit gliedert sich in acht Kapiteln. Das erste Kapitel gibt eine Einführung in das Thema der Arbeit und das Ziel, welches in der Arbeit erreicht werden soll. Das zweite Kapitel gibt einen Überblick über die möglichen Technologien, die zur Erstellung einer mobilen Anwendung eingesetzt werden können. Im darauffolgenden Kapitel werden die Anforderungen analysiert. Dabei wird auch das traditionelle Buchungssystem Teamwarp und seine Direct-Access-Interface vorgestellt. Der Großteil des Kapitels Analyse behandelt die Anforderungen des Auftraggebers. Die Spezifikation wird im vierten Kapitel erbracht. Das fachliche Datenmodell, die Funktionalitäten, die Anwendungsfälle sowie die Dialoge und ihre Abfolge werden nacheinander vorgestellt. Das fünfte Kapitel handelt vom Entwurf der Anwendung. Es wird die Entscheidung der eingesetzten Technologie sowie Frameworks getroffen. Hauptbestandteile des Kapitels sind die Architektur des Systems und die Abläufe der Anwendung. Im sechsten Kapitel wird die Implementierung vorgestellt und beschreibt wie die Funktionalitäten und Anwendungsfälle realisiert werden. Das siebte Kapitel gibt einen kleinen Einblick in die erste qualitative Evaluierung der Anwendung, wobei ein Testszenario durchgeführt und die Ergebnisse ausgewertet wird. Das letzte Kapitel schließt die Arbeit mit einer Zusammenfassung der Ergebnisse und einem Ausblick auf die Zukunft ab.

2 Technologien

Nach dem heutigen Stand der Technik gibt es mehrere Wege, eine mobile Anwendung, im Allgemeinen auch als App bezeichnet, umzusetzen. Insgesamt gibt es drei Alternativen:

- Native mobile Anwendung
- Mobile Web-Anwendung
- Mobile Hybrid-Anwendung

In diesem Kapitel werden die drei Entwicklungstechnologien für Apps vorgestellt. Jede Variante wird in einem eigenen Abschnitt mit Definitionen, Vor- und Nachteilen vorgestellt. (Franke, et al., 2012)

2.1 Native mobile Anwendung

2.1.1 Definition

Unter nativen Apps werden im Allgemeinen Anwendungen verstanden, die für bestimmte Plattformen entwickelt werden. Sie sind in der für die jeweilige Plattform spezifischen Programmiersprache geschrieben und werden für den Compiler der Plattform optimiert. Solche Anwendungen müssen vor der Nutzung auf dem Endgerät installiert werden. (Franke, et al., 2012 S. 21)

2.1.2 Vorteile

- **App-Store**
  Der erste Vorteil von nativen Apps ist der App-Store. „App-Store“ ist die allgemeine Bezeichnung für eine digitale Vertriebsplattform von Anwendungssoftware. (beispielsweise der App-Store von Apple oder Google‘s Play Store). Der Service ermöglicht es Benutzern, Software in einem Anwendungskatalog zu suchen und herunterzuladen.

Sobald eine App im App-Store verfügbar ist, steht sie Millionen zahlungsfähigen Benutzern zur Verfügung. Die Betreiber kümmern sich um die komplette Abwicklung vom Angebot bis zum Verkauf. (Franke, et al., 2012 S. 22)

- **Hardware des Gerätes**
  Mit nativen Apps ist der komplette Zugriff auf die Hardware des Endgerätes wie beispielweise Kamera, Lautsprecher und Sensoren möglich. Wenn eine App eine solche Funktion wie beispielweise Fotoaufnahme benötigt, ist es unumgänglich, die App als eine native App zu entwickeln. (Franke, et al., 2012 S. 22)

- **Plattform-spezifische Features**
  Mit nativen Apps ist es auch leicht, die Plattform-spezifischen Informationen auszulesen, wie zum Beispiel Kontakte und Kalender. Es ist manchmal sinnvoll, den eigenen Kalender mit einer App zu synchronisieren, um zum Beispiel einen Plan zu erstellen. (Franke, et al., 2012 S. 22)

- **Software Development Kits (SDKs)**
  Die Hersteller der jeweiligen Plattform stellen den Entwicklern umfangreiche SDKs in der entsprechenden Programmiersprache für ihre Plattform zur Verfügung. Diese beinhalten neben umfangreichen Entwicklungswerkzeugen auch Bibliotheken, um vollen Zugriff auf die Hardware und die plattform-spezifische Features zu ermöglichen. (Franke, et al., 2012)

- **Ladezeit und Geschwindigkeit der Ausführung**
  Native Apps sind beim Start und bei der Ausführung schnell, da die Prozesse in der Plattform selbst ablaufen und sich die meisten Dateien im Speicher des Geräts befinden, sodass Dateien nicht aus dem Internet geladen werden müssen. (Mobafone, 2013)

- **Offline-Nutzung**
  Je nach Umsetzung können alle Dateien auf dem mobilen Gerät gespeichert werden, so dass die App auch ohne Datenverbindung nutzbar ist. (Glanzkinder GmbH & Co. KG, 2012)

2.1.3 Nachteile

- **Plattform-Abhängigkeit**
  Eine App muss in der spezifischen Programmiersprache einer Plattform geschrieben werden. Die Portierung einer auf einer Plattform funktionierenden App auf eine andere Plattform ist sehr aufwändig. Für den Entwickler ist es unumgänglich, die App in der Programmiersprache der neuen Zielplattform zu schreiben.
  Sollte es Änderungen am Quellcode geben, beispielweise beim Update, ist eine Portierung der Änderungen auf die verschiedenen Plattformen erforderlich. Dadurch entsteht eine zusätzliche Fehlerquelle. (Franke, et al., 2012 S. 28)

- **App-Store**
  Ein App-Store hat nicht nur Vorteile für den Entwickler. Die Entwickler sind komplett von einer Redaktion, welche einen App-Store betreut, abhängig. Die Redaktion entscheidet, welche App im App-Store veröffentlicht wird und welche nicht. Das gilt auch für Programmaktualisierungen und die Behebung von Fehlern. Der Vorgang der Veröffentlichung kann einige Woche oder sogar Monate dauern. Viele Apps werden gar nicht veröffentlicht. Im schlimmsten Fall war die Entwicklung einer App komplett umsonst. (Franke, et al., 2012)

- **Kosten für Entwickler**
  Abgesehen vom Zugang zu den Entwicklungswerkzeugen ist die Entwicklung einer App in der Regel kostenlos. Sobald der Entwickler seine App veröffentlichen möchte, muss er mit einer kostenpflichte Registrierung für einen App-Store rechnen[1]. Der Dienst eines App-Stores ist dabei nicht kostenlos[2].
  Für die Entwicklung einer App ist je nach Zielplattform eine bestimmte Hard- und Software nötig. Für die Entwicklung von Apps für iOS stellt Apple die Entwicklungsumgebung Xcode zur Verfügung. Das Programm ist jedoch nur unter Mac OS, einem von Apple entwickelten Betriebssystem, lauffähig. Für den Entwickler ist es dann unumgänglich, einen Apple-Computer zu erwerben, wenn er diese Entwicklungsumgebung (IDE) benutzen will.

- **Aufwand für Entwickler**

[1] Die Teilnahme an Apple's iPhone Developer Program kostet jährlich 99$ (Apple). Bei Google sind es einmalig 25$ (Google).
[2] Apple und Google müssen mit 30% am Umsatz beteiligt werden.

Eine neue Plattform bedeutet eine neue Programmiersprache. Um eine native App für verschiedene Plattformen zu entwickeln, müssen verschiedene Programmiersprache erlernt werden. (Franke, et al., 2012)

2.2 Web-App-Entwicklung

2.2.1 Definition

Im Gegensatz zu nativen Apps sind Web-Apps nicht in einem App-Store verfügbar und müssen auch nicht vor der Nutzung auf dem Smartphone installiert werden. Sie sind eigentlich Webseiten, die für den mobilen Zugriff durch Smartphones optimiert werden und können über einen URL in einem Web-Browser aufgerufen werden. Die Benutzeroberfläche von Web-App ist an native Apps angelehnt, so dass die Benutzer den Eindruck haben sie würden eine native App bedienen. (Franke, et al., 2012)

Smartphones besitzen heutzutage immer leistungsfähigere Browser, die mit Browsern von Computern vergleichbar sind. Mit den Standard-Web-Technologien wie HTML5, CSS3 und Javascript ist die Realisierung umfangreicher und modern gestalteter Apps möglich. (Franke, et al., 2012)

2.2.2 Vorteile

- **Plattform-Unabhängigkeit**
  Einer der Vorteile der Web-App-Entwicklung ist die Plattform-Unabhängigkeit. Im Gegensatz zu native Apps sind Web-Apps nicht für eine bestimmte Zielplattform entwickelt. Sie sind mit Web-Technologien erstellt, so dass nur ein Browser benötigt wird, um sie auszuführen. Die Entwickler müssen in der Regel jede Web-App nur einmal mit Web-Technologien umsetzen, und die App kann auf jeder Plattform ausgeführt werden. (Franke, et al., 2012)

- **Keine Installation nötig**
  Eine Web-App ist in technischer Hinsicht auch eine Webseite, sie muss deshalb nicht auf Endgeräten installiert werden. Web-Apps können über Suchmaschinen im Internet gefunden und direkt genutzt werden. Wenn der Nutzer eine Web-App als Lesezeichen speichert, ist diese App - wie eine native App - auf dem Hauptbildschirm des Geräts als ein Icon verfügbar. (Franke, et al., 2012)

- **Unabhängigkeit von App-Store**

Eine Web-App kann in Sekundenschnelle veröffentlicht und aktualisiert werden, da sie keinen Zulassungsprozess durchlaufen muss und von keiner Organisation abhängig ist. Daher ist auch keine Registrierung an einem App-Store erforderlich, sodass die Gebühr für einen App-Store entfällt. (Würstl, 2013)
Bei kostenpflichten Apps muss keine Provision an den App-Store abgeführt werden. (Mobafone, 2013)

- **Keine Synchronisation nötig**
  Updates, Erweiterungen und Fehlerkorrekturen werden auf dem eigenen Server veröffentlicht. Die Daten-Synchronisation findet automatisch beim neuen Laden einer App statt, so dass sich die Benutzer auf manuelle Updates verzichten können. (Mobafone, 2013)

- **Umwandlung in native Apps möglich**
  Vorausschauend programmierte Web-Apps können in native Apps umgewandelt werden (siehe hybride Apps) und damit einige Vorteile von nativen Apps erzielen. (Steyer, 2013)

2.2.3 Nachteile

- **Keine tiefe Integration in das Endgerät**
  Im Gegensatz zu einer nativen App verfügt eine Web-App über einen sehr begrenzten Zugriff auf die Hardware des Geräts wie beispielweise Kamera, Lautsprecher und Sensoren. Eine Web-App ist nicht in der Lage, plattform-spezifische Features wie zum Beispiel Kontakte und Kalender auszulesen. (Franke, et al., 2012)

- **Dauerhafte Internetverbindung erforderlich**
  Da eine Web-App in der Regel eine Webseite ist, ist eine Datenverbindung während der Nutzung notwendig. (Steyer, 2013)

- **Keine direkte Datenspeicherung auf Endgerät**
  Im Gegensatz zu nativen Apps wird die direkte Speicherung der Daten auf Endgeräten von einer Web-App nicht unterstützt. Die indirekte Datenspeicherung ist zwar grundsätzlich möglich, jedoch werden dafür spezielle Techniken wie beispielweise der sogenannte lokale Speicher, den Speicherplatz eines Webbrowsers benötigt. (Steyer, 2013)

- **Performance**
  Da eine Web-App nicht direkt auf der Plattform läuft sondern in einem Webbrowser, ist die Ausführungsgeschwindigkeit deutlich langsamer als bei einer nativen App. Außerdem befinden sich fast alle Daten einer Web-App auf dem

Webserver, was bedeutet, dass die Ladezeiten wesentlich länger sind. (Franke, et al., 2012)

- **Kein App-Store**
  Die Tatsache, dass Web-Apps nicht von einem App-Store abhängig sind, ist gleichzeitig ein Nachteil für die Entwickler. Im Gegensatz zu einer nativen App muss ein Entwickler seine Web-App selber bewerben. Außerdem müssen die Bezahlungsverfahren bei kostenpflichtigen Apps selbst umgesetzt werden.

2.3 Hybrid-App-Entwicklung

2.3.1 Definition

Wie in den zwei vorhergehenden Abschnitten erläutert, ist häufig ein Vorteil von nativen Apps wiederum ein Nachteil von Web-Apps und umgekehrt. Um die Vorteile von nativen Apps und Web-Apps zu kombinieren und die Nachteile der beiden Ansätze abzuschwächen wird ein Lösungsansatz, die sogenannte Hybrid-App-Entwicklung, eingesetzt. (Steyer, 2013)

Mit der Hybrid-App-Entwicklung ist es möglich, eine Web-App in eine native App umzuwandeln, die App kann danach ganz normal über einen App-Store veröffentlicht werden. Hybrid Apps verknüpfen die Vorteile der nativen und Web orientierten Programmierung und ermöglichen den Entwicklern, unabhängig und zugleich effizient mobile Anwendungen zu entwickeln. (Funk, 2011)

Die Entwickler müssen den Kern der App nur einmal mit Web-Technologien wie HTML5, CSS3, Javascript und möglichweise einem oder mehreren Frameworks umsetzen. Einige Richtlinien, die die Hybrid-App-Entwicklung voraussetzt, müssen dabei berücksichtigt werden. Aus der implementierten Web-App kann danach mit Hilfe von einem mobile Entwicklungsframework wie beispielsweise PhoneGap[3] eine Anwendung, die einer nativen App ähnlich ist, erstellt werden. Die resultierte Hybrid-App ist keine reine native App, da alle Darstellungen, statt mit dem nativen UI-Framework der Plattform, über das sogenannte Web-Steuerelement wiedergegeben werden. Eine Hybrid-App ist auch keine Web-App, denn sie kann in App-Stores veröffentlicht werden und verfügt über einen Zugriff auf die nativen APIs der Endgeräte. (Steyer, 2013)

Hybrid-Apps bestehen eigentlich aus zwei Komponenten: Der native Teil wird auf das Gerät des Nutzers installiert und greift auf das sogenannte Web-Steuerelement zu, die auf Basis von Webtechnologien die eigentlichen Inhalte und Funktionen bereitstellen. Der native Teil der Hybrid-App kann beispielweise in Objective-C (für iOS) oder Java (für Android)

[3] PhoneGap ist ein kostenloses und quelloffenes Framework, das die Hybrid-Entwicklung unterstützt (http://phonegap.com).

implementiert oder mit Hilfe spezieller Frameworks wie PhoneGap entwickelt werden. (Galileo, 2013)

Das Web-Steuerelement[4] ist eigentlich der Container, der den Webbrowser darstellt. Hybrid-Apps nutzen dieses Web-Steuerelement um mit Hilfe der nativen Browser-Wiedergabemaschine[5] die HTML und Javascript Dateien im Vollbild-Format zu präsentieren. Die Browser-Wiedergabemaschine ist nicht der Browser selbst, es ist ein Modul der Plattform, der die HTML, Javascript Dateien einliest und zu einer Darstellung verarbeitet. (Seven, 2012)

Hybrid-Apps setzen eine Abstraktionsschicht um, die den Zugriff auf die native API der Geräte ermöglicht. Durch diese Abstraktionsschicht wird ein gemeinsamer Satz von APIs in Javascript zur Verfügung gestellt. Diese Javascript APIs funktionieren auf jeder Plattform, die vom jeweiligen Framework unterstützt wird. Die Umsetzung der Abstraktionsschicht ist bei der Implementierung von Web-Apps unmöglich, da der der Browser und die APIs der Geräte aus Sicherheitsgründen getrennt sind. (Seven, 2012)

2.3.2 Vorteile

- **Entwicklungskosten sparen**
  Einer der größten Vorteile einer Hybrid-App ist die Zeit- und Geldersparnis. Der Kern der Anwendung muss nur einmal implementiert werden, was den Aufwand deutlich reduziert. (Steyer, 2013)

- **App-Stores**
  Eine Hybrid-App kann wie eine native App über den App-Store veröffentlicht werden und von den Vorteilen des App-Stores profitieren. (Steyer, 2013)

- **Zugriff auf die Hardware und die plattform-spezifische Features des Gerätes**
  Durch die Abstraktionsschicht lassen sich die Hardware der mobilen Geräte sowie die plattform-spezifische Features in Hybrid-Apps benutzen. (Steyer, 2013)

2.3.3 Nachteile

- **Performance**
  Die Geschwindigkeit ist für mobile Apps von Bedeutung. Eine gute Performance wird von den Benutzern verlangt. Laut einer Umfrage der Seite

[4] UIWebview auf iOS, Webview auf Android und andere.
[5] Das sogenannte WebKit ist die Browser-Wiedergabemaschine, die auf iOS, Android, Blackberry und andere Plattformen verwendet wird.

http://www.compuware.com erwarten 80% der Benutzer von einer mobilen Anwendung eine Ladezeit von drei Sekunden oder weniger (poqstudio, 2012). Eine native App ist immer schneller beim Laden und bei der Ausführung als eine Hybrid-App.

- **Reaktionsfähigkeit**
  Native Apps reagieren unverzüglich auf Fingerberührungen, zum Beispiel wenn ein Finger über den Bildschirm gezogen wird, um das nächste Foto zu sehen. Diese Reaktionsfähigkeit kann eine Hybrid-App noch nicht erreichen. (poqstudio, 2012)

- **Design und Animationen**
  Die Animationen und Bewegungen in einer nativen App werden flüssiger dargestellt. Sie sehen schöner aus und fühlt sich flüssiger an als bei einer Web-App oder einer Hybrid-App. (poqstudio, 2012)

Es wurden die drei Entwicklungs-Technologien für mobile Anwendungen erläutert. Die Vor- und Nachteile von den einzelnen Entwicklungswegen sind Argumente, die bei der Entscheidung über die Technik für die Umsetzung eine wichtige Rolle spielen.

3 Analyse

In diesem Kapitel werden die Anforderungen vom Auftraggeber analysiert. Es wird am Anfang des Kapitels das bisherige Zeiterfassungssystem „Teamwarp“ und seine Stärken und Schwächen vorgestellt. Im Kapitel 3.2 wird das bereitgestellte Direct-Access-Interface von Teamwarp betrachtet. Im Abschnitt 3.3 werden die Ziele der Anwendung, die im Rahmen dieser Arbeit entwickelt wird, erläutert und weiter begründet, weshalb die Anwendung im Kontext des Unternehmens geeignet ist. Im nächsten Abschnitt 3.4 werden die Anforderungen des Auftraggebers analysiert und priorisiert. Dabei wird ein Anwendungsfalldiagramm ausgeführt, das auf den Anforderungen basiert. Das Kapitel 3.5 behandelt die Abgrenzung des Kontexts der Arbeit und legt fest, was im Rahmen der Arbeit nicht umgesetzt werden muss.

3.1 Das vorliegende System – Teamwarp

Wie im ersten Kapitel beschrieben benutzen die Mitarbeiter von mgm-technologies partners eine web-basierte Anwendung, das sogenannte Teamwarp, um Arbeitszeiten zu erfassen und zu buchen. In diesem Abschnitt wird Teamwarp selbst mit einigen wesentliche Funktionalitäten, seine Stärken sowie Schwächen vorgestellt.

Abbildung 1: Teamwarps monatliche Übersicht

Um Teamwarp zu benutzen muss sich jeder Benutzer mit eigenen Zugangsdaten einloggen. Nach dem Einloggen wechselt Teamwarp zu seiner monatlichen Übersicht. Auf dieser Seite befinden sich allgemeine Informationen über die geleisteten Arbeitszeiten im Monat, welche durch die Start-, Ende-und Pausenzeiten sowie den Arbeitszeiten in Stunden an einzelnen Tagen (unter „Begin“, „End“, Break“ und „Hour“) definiert sind. In Abbildung 1 ist ein Abschnitt der Seite dargestellt. Unter „Discrep“ stehen die Abweichungen zwischen der gesamten Arbeitszeit und der Aufgaben zugeordneten Arbeitszeit an einzelne Tage. Es wird erwartet, dass diese Abweichungen am Ende des Monats leer bzw. 0:00 sind.

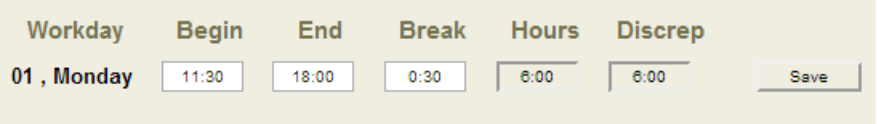

Abbildung 2: Tages Arbeitszeit

Wenn der Benutzer auf einen Arbeitstag („Workday“) klickt, gelangt Teamwarp auf die Übersicht des ausgewählten Tages. Auf dieser Seite können die Start-, Ende- sowie Pausenzeit des Tages eingegeben und gespeichert werden. Wie in Abbildung 2 gezeigt, wird die geleistete Arbeitszeit automatisch berechnet und angezeigt, sobald alle drei Datenfelder ausgefüllt sind.

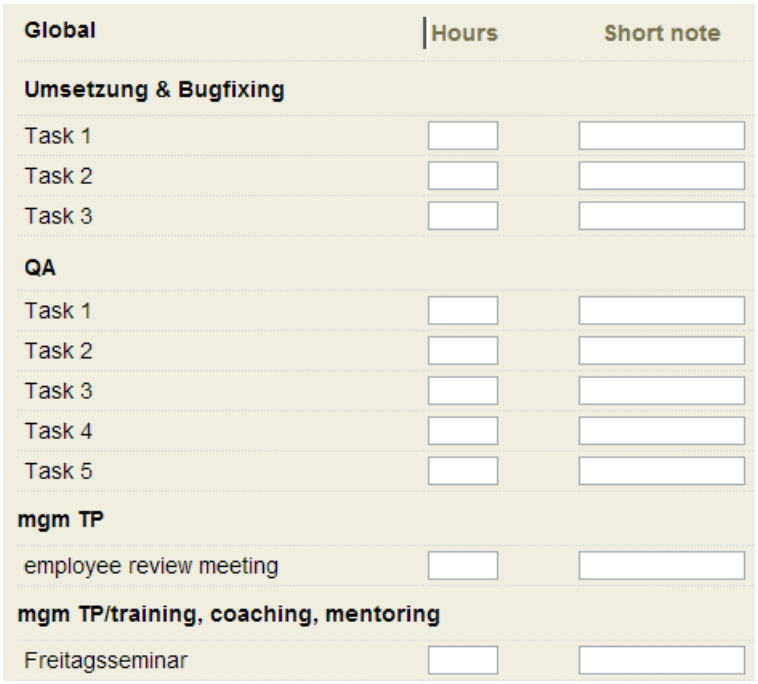

Abbildung 3: Tätigkeiten und daran gearbeitete Zeiten

Damit die Abweichung der Arbeitszeit („Discrep“) 0 ist, muss die geleistete Arbeitszeit des Tages einer oder mehreren Aufgaben zugeordnet werden. In der Abbildung 3**Error! Reference source not found.** ist eine Übersicht über die Tätigkeiten („Task“), die Arbeitszeiten von einzelnen Tätigkeiten und dazugehörige Notizen dargestellt. Jede Zeile wird als ein Teamwarp-Knoten bezeichnet. Außer den festgelegten Tätigkeiten wie beispielweise Freitagsseminar, Standup-Meeting und Einarbeitung ist eine Menge von „Tasks“ (siehe Abbildung 3) zu beachten, die auf keine bestimmte Aufgabe verweisen. Dieser Abschnitt ermöglicht die sogenannten generischen Zeitbuchungen, welche die Hauptrolle der Zeiterfassung an einem Tag spielen. Neben den gearbeiteten Zeiten sind die Notizen bei der generischen Zeitbuchung besonders wichtig. Über die Notizen wird festgelegt, auf welches JIRA-Ticket die Arbeitszeit gebucht werden soll. Deswegen ist es nötig, dass die eingegebenen Notizen eindeutig sind und auf richtige Tickets referenzieren. An dieser Stelle ist noch ein sogenannter Buchungskontext zu beachten. In der ist ein Buchungskontext durch die Zeile: „QA“ bezeichnet. Dieser Buchungskontext besagt, unter welchem Kontext ein Mitarbeiter arbeitet und entscheidet, auf welches Konto die Buchung geschickt wird.

JIRA ist ein internes Ticket-System von mgm-technologies partners. In jedem Ticket ist eine Aufgabe mit Titel, Beschreibung, Priorität usw. gekapselt. Jedes Ticket hat einen eindeutigen Name, der aus den dazugehörigen Projektname und einer Nummer besteht. Wenn ein Mitarbeiter neue Aufgaben bzw. eine neue Teilaufgabe entdeckt, erstellt er ein neues Ticket im JIRA-System. Sollte diese Aufgabe bzw. diese Teilaufgabe bearbeitet werden, trägt der Mitarbeiter den Name des entsprechenden Tickets als eine Notiz in Teamwarp ein. Zusammen mit der eingetragenen Arbeitszeit bilden sie eine generische Zeitbuchung.
In Teamwarp werden in der Regel sowohl Arbeitszeiten als auch JIRA-Tickets manuell von Mitarbeitern eingegeben. Wenn die ganze Arbeitszeit eines Tages vollständig an einen oder mehreren Tasks verteilt ist, beträgt die Abweichung der Arbeitszeit unter „Discrep“ 0. Die Tagesbuchung gilt also als richtig und kann gespeichert werden.

Im Folgenden werden die Vor- und Nachteile von Teamwarp analysiert.

- **Vorteile:**
    - Teamwarp ist eine web-basierte Anwendung, die für einen PC-Browser optimiert ist. Deshalb können Mitarbeiter Teamwarp auf dem Rechner benutzen, mit dem sie arbeiten. Dadurch lässt sich Zeit sparen, weil kein zweites Gerät verwendet werden muss.
    - Da Mitarbeiter auf demselben Rechner arbeiten und die Zeitbuchung durchführen, kann die Eingabe des JIRA-Tickets durch einfaches Kopieren und Einfügen erfolgen. Dies garantiert die Korrektheit der Namen von Tickets und spart zugleich Zeit.
- **Nachteile:**

- Wie in Abbildung 3 gezeigt stehen zehn Tasks für die generischen Buchungen zu Verfügung. Diese zahlreichen Tasks sind häufig überflüssig. Ein Mitarbeiter bearbeitet normalerweise nicht zehn verschiedene Tasks am Tag, deshalb braucht er nicht so viele Felder.
- Die Arbeitszeit, die ein Mitarbeiter für eine Tätigkeit verbraucht, muss manuell eingegeben werden, so dass die Genauigkeit der Zeitangabe nicht gewährleistet werden kann.

Teamwarp ist mit einer Staging-Datenbank verbunden, die von Mitarbeitern erzeugten JIRA-Tickets mit allen Informationen wie beispielweise Ticketzustand, Tickettitel und Ticketbeschreibung beinhaltet. Die Staging-Datenbank wird jede Nacht aktualisiert. Das bedeutet, dass wenn ein Ticket in JIRA neu erstellt wird, dieses Ticket erst am nächsten Tag in der Staging-Datenbank vorhanden ist.

3.2 Direct-Access-Interface von Teamwarp

Dieser Abschnitt behandelt die vom Auftraggeber zur Verfügung gestellte direkte Schnittstelle zu Teamwarp. Diese Schnittstelle bietet Dienste an, die für die Entwicklung von iWarp benötigt werden. Die Schnittstelle ermöglicht die Kommunikation zwischen Teamwarp und externen Systemen. Das bedeutet, dass ein externes System in der Lage ist, Daten von Teamwarp auszulesen bzw. eigene Daten an Teamwarp zu schicken.
Die Schnittstelle ist eine http-Schnittstelle und wird unter einer Sub-URL zur Verfügung gestellt.
Die Zugangsdaten von Mitarbeitern werden bei jedem http-Request benötigt. Als Antwort von Teamwarp werden Texte zurückgegeben. Im folgende werden drei von der Schnittstelle angebotene Dienste vorgestellt, welche für iWarp besonders interessant sind.

- **Generische Buchung** (Name des Diensts: GENERIC)**:**
  Mit diesem Dienst kann ein äußeres System seine Daten an Teamwarp schicken, um eine generische Buchung auszuführen. Die Eingabeparameter sind:
  - Datum: Der Tag, an dem die Buchung erfolgt.
  - Buchungskontext-ID: Identifikation des Kontexts, unter dem die Buchung erfolgen soll.
  - JIRA-Ticket: Der Name des Tickets, welches bearbeitet wurde.
  - Arbeitszeit: Die geleistete Arbeitszeit für das Ticket.

  Dieser Dienst nutzt den folgenden Algorithmus: nehme den Ticketname als einen eindeutigen Schlüssel für den Tag und den gegebenen Buchungskontext. Wenn ein Eintrag mit diesem Schlüssel bereits existiert, wird er mit der gegebenen Arbeitszeit aktualisiert. Ansonsten wird der nächste freie Eintrag mit der Arbeitszeit und den Ticketname gefüllt (siehe Abbildung 3). Wenn entweder der Buchungskontext-ID keinen Kontext identifiziert oder keine generische Knote

mehr zur Verfügung steht, gilt die Buchung als nicht erfolgreich und es wird ein Fehler zurückgegeben.

- **Liste der Teamwarp-Knoten** (Name des Diensts: LIST)**:**
  Eine Teamwarp-Knoten ist eine Zeile in der Seite der Tagesbuchung in Teamwarp (siehe Abbildung 3). Dieser Dienst erlaubt es, Informationen über belegte sowie unbelegte Teamwarp-Knoten von einem Tag abzufragen. Der Eingabeparameter ist:
    - Datum: Der Tag, an dem Informationen über Teamwarp-Knoten abgefragt werden soll.

  Als Antwort gibt Teamwarp eine Liste von Teamwarp-Knoten mit den jeweiligen zusätzlichen Informationen zurück. Die Informationen jedes Knotens enthalten:
    - Projekte, die einem Mitarbeiter zugeordnet sind.
    - Kontexte, unter denen ein Mitarbeiter arbeiten kann.
    - Angabe, ob der Knoten schon belegt ist.
    - Die Arbeitszeit des Knotens, wenn sie schon belegt ist.
    - Die Notiz des Knotens, wenn sie schon belegt ist.

- **Liste der Tickets** (Name des Dienst: ISSUE)**:**
  Mit diesem Dienst „Liste der Tickets“ kann ein äußeres System eine Liste von gefilterten JIRA-Tickets abfragen. Die Eingabeparameter des Diensts sind:
    - JIRA-Projekt: das Projekt, dessen Tickets abgefragt werden sollen.
    - Nummer: Jedes Ticket hat eine Nummer, die aus mehreren Ziffern besteht. Nach diesem Parameter filtert die Schnittstelle die von Teamwarp zurückzugebenden Tickets, indem es auf die Tickets einschränkt wird, deren Nummer mit den angegebenen Ziffern beginnen.

3.3 Ziele der Anwendung

Im vorherigen Abschnitt wurden die wesentlichen Funktionalitäten sowie Vor- und Nachteile des Zeiterfassungssystems Teamwarp, betrachtet. In diesem Kapitel werden die Ziele der im Rahmen dieser Arbeit zu implementierenden Anwendung vorgestellt.

Wie bereits beschrieben sind die generischen Buchungen im betriebswirtschaftlichen Kontext des Unternehmens von besonderer Bedeutung. Einer der wichtigsten Aspekte von Zeitbuchungen ist die exakte Messung der Zeit. Der Aufwand für die Zeitmessung bei einzelnen Buchungen ist hoch und es gibt dafür keine Unterstützung in Teamwarp. Die Arbeitszeit kann zwar mit dem PC oder einer Uhr gemessen werden, jedoch muss der Messwert danach manuell in Teamwarp übertragen werden. Mit Hilfe eines mobilen Geräts sollte es einfach sein, die geleistete Arbeitszeit exakt zu bestimmen und anschließend den Messwert ohne Aufwand für die Buchung zu verwenden.

Ein weiterer Aspekt der generischen Zeitbuchungen ist die Angabe eines JIRA-Tickets. Es wird erwartet, dass in die Notizfelder der generischen Zeitbuchungen nur Angaben über JIRA-Tickets eingetragen werden. Jedoch ist die Eingabe der Notizenfelder nicht auf Ticketnamen begrenzt, so dass es Teamwarp ermöglicht beliebigen Text in die Notizenfelder einzugeben. Im mobilen Buchungssystem soll diese Begrenzung realisiert werden, so dass bei generischen Buchungen tatsächlich nur Ticketnamen als Notizen eingetragen werden können.

Die entwickelte Anwendung „iWarp“ soll Teamwarp nicht komplett ersetzen, sondern als ein alternatives Interface für Teamwarp dienen und um die fehlende Unterstützung bei der generischen Buchung erweitern. Indem die generischen Zeitbuchungen in einer mobilen Anwendung umgesetzt werden, schafft iWarp einen neuen Weg zur Zeiterfassung von ticketbezogenen „Tasks“. Mitarbeiter könnten ihre Arbeitszeiten mit iWarp auf ihren mobilen Geräten erfassen, wobei nur die hierfür nötigen Dialoge und Felder zu sehen sind. Dadurch sind die Zeitbuchungen konsistent und weniger aufwändig.

Um die oben beschriebenen Ziele zu erreichen, sind mehrere Anforderungen vom Auftraggeber festgelegt worden, die im nächsten Abschnitt vorgestellt werden.

3.4 Anforderungsanalyse

In diesem Abschnitt werden die Anforderungen vom Auftraggeber ausgeführt und analysiert. Die Anforderungen teilen sich in zwei Gruppen ein: fachliche und technische Anforderungen. In diesem Kapitel wird der Begriff „Task“ eingeführt, der nicht eine Buchungszeile (wie in Abbildung 3) bezeichnet, sondern ein Ticket, das einem Buchungskontext zugeordnet ist.

3.4.1 Fachliche Anforderungen

Die fachlichen Anforderungen werden nach Prioritäten in zwei Gruppen geteilt.

1) **Sehr wichtig:**
   - **A1.1:** Benutzer müssen sich jedes Mal authentisieren bevor sie die Anwendung nutzen können.

   - **A1.2:** Benutzer können JIRA-Tickets selektieren, an denen sie arbeiten wollen.
     Für eine generische Buchung ist die Angabe eines Ticketnamens notwendig. In der Anwendung sollen Mitarbeiter alle Tickets der Staging-Datenbank sehen und auswählen können.

- **A1.3:** Benutzer können einen Buchungskontext für ein JIRA-Ticket auswählen, um einen Task zu erzeugen.
  Ein Task ist ein Ticket, das einem bestimmten Buchungskontext zugeordnet ist. Jeder Mitarbeiter hat mindestens einen Buchungskontext. Sollte es nur einen Kontext geben, ist die Auswahl des Kontexts nicht nötig. Wenn es aber mehr als einen Kontext gibt, ist die Auswahl des Kontexts zu beachten. Die Angabe über den Buchungskontext spielt bei einer generischen Buchung eine Rolle.

- **A1.4:** Benutzer können alle generischen Buchungen sehen, die sie am jeweiligen Arbeitstag mit der Anwendung gemacht haben.

- **A1.5:** Benutzer können den Beginn und das Ende der Arbeit an ein Ticket erfassen.
  Wie oben beschrieben sind Zeitangaben der Hauptteil von jeder generischen Buchung. Teamwarp bietet keine Funktion für die Messung von Arbeitszeiten. Mitarbeiter müssen entweder Zeiten abschätzen oder andere Werkzeuge verwenden, um die geleisteten Zeiten an einzelne Tickets zu messen. Mithilfe der Erfassung vom Beginn und Ende einer Tätigkeit kann die geleistete Arbeitszeit berechnet werden. Arbeitszeiten sollen automatisch eingetragen werden, sodass die generischen Buchungen wesentlich verbessert werden können.

- **A1.6:** Benutzer können Buchungen an Teamwarp schicken.
  In iWarp findet eigentlich keine wirkliche Buchung statt. Die Buchungen oder genauer, die berechneten Arbeitszeiten für einzelne Tasks, welche in iWarp erstellt werden, müssen an Teamwarp geschickt werden. Teamwarp aktzeptiert Arbeitszeiten nur in 15-Minuten-Blöcken. Deshalb sollen die Arbeitszeiten in iWarp gerundet werden, bevor sie an Teamwarp geschickt werden.

2) Wichtig:

- **A1.7:** Benutzer können die Anzahlen der Projekte bzw. der Buchungskontexte, die ihnen angezeigt werden, einschränken.
  Einem Mitarbeiter können mehrere Projekte bzw. Buchungskontexte zugewiesen sein. Es ist möglich, dass ein Projekt, welches schon vergangen ist oder an dem der Mitarbeiter nicht mehr arbeitet, noch zur Verfügung steht. In diesem Fall wird erwartet, dass ein Mitarbeiter dieses Projekt herausfiltern kann, so dass das Projekt nicht mehr auswählbar ist. Das gleiche gilt für die Buchungskontexte, unter denen der Mitarbeiter nicht mehr arbeitet. Durch den Ausschluss der unnötigen Projekte bzw. Kontexte können die generischen Zeitbuchungen vereinfacht werden.

- **A1.8:** Ein Benutzer kann die von ihm in der Vergangenheit ausgewählten Tickets sehen.
  Es kommt häufig vor, dass ein Mitarbeiter über mehrere Tage an einem Ticket arbeitet. Deswegen wird erwartet, dass dieses Ticket schnell gefunden werden kann, da es vorgeschlagen wird.

- **A1.9:** Benutzer können ein Ticket neu erstellen, das noch nicht in der Staging-Datenbank vorhanden ist.
  Die Staging-Datenbank wird nur in der Nacht aktualisiert. Deshalb kann ein Mitarbeiter ein Ticket nicht finden, welches er gerade neu in JIRA erstellt hat. Aus diesem Grund ist es nötig, dass ein Ticket innerhalb von iWarp neu erstellt werden kann. Benutzer müssen darauf hingewiesen werden, dass das in iWarp neu erstellte Ticket außerhalb von iWarp nicht existiert und es nur für die Zeitbuchung vorgesehen ist. Das Ticket muss zusätzlich manuell in JIRA erstellt werden.

- **A1.10:** Benutzer können Zusatzinformationen von Tickets sehen.
  Um das Risiko zu vermindern, dass der Benutzer ein falsches Ticket selektiert, sollen Zusatzinformationen zusammen mit den Tickets angezeigt werden. Zusatzinformationen können beispielweise der Zustand und der Titel des Tickets sein. Mithilfe des Zustands bzw. des Titels von Tickets wird der Selektionsvorgang eines Tickets wesentlich erleichtert.

- **A1.11:** Benutzer können sich vom System ausloggen.
  Damit keine Fremde die Anwendung benutzen kann, sollten Benutzer die Möglichkeit haben, sich vom System abzumelden.

3.4.2 Technische Anforderungen

- **A2.1:** Die Anwendung soll sowohl für die iOS, Android und andere Plattforme verfügbar sein.
- **A2.2:** Die Kommunikation zu Teamwarp soll mittels des Direct-Access-Interfaces implementiert werden.
- **A2.3:** Die Anwendung sollte einfach und intuitiv zu bedienen sein. Die Benutzung sollte dabei leicht verständlich und zum größten Teil selbsterklärend sein.

3.4.3 Anwendungsfalldiagramm

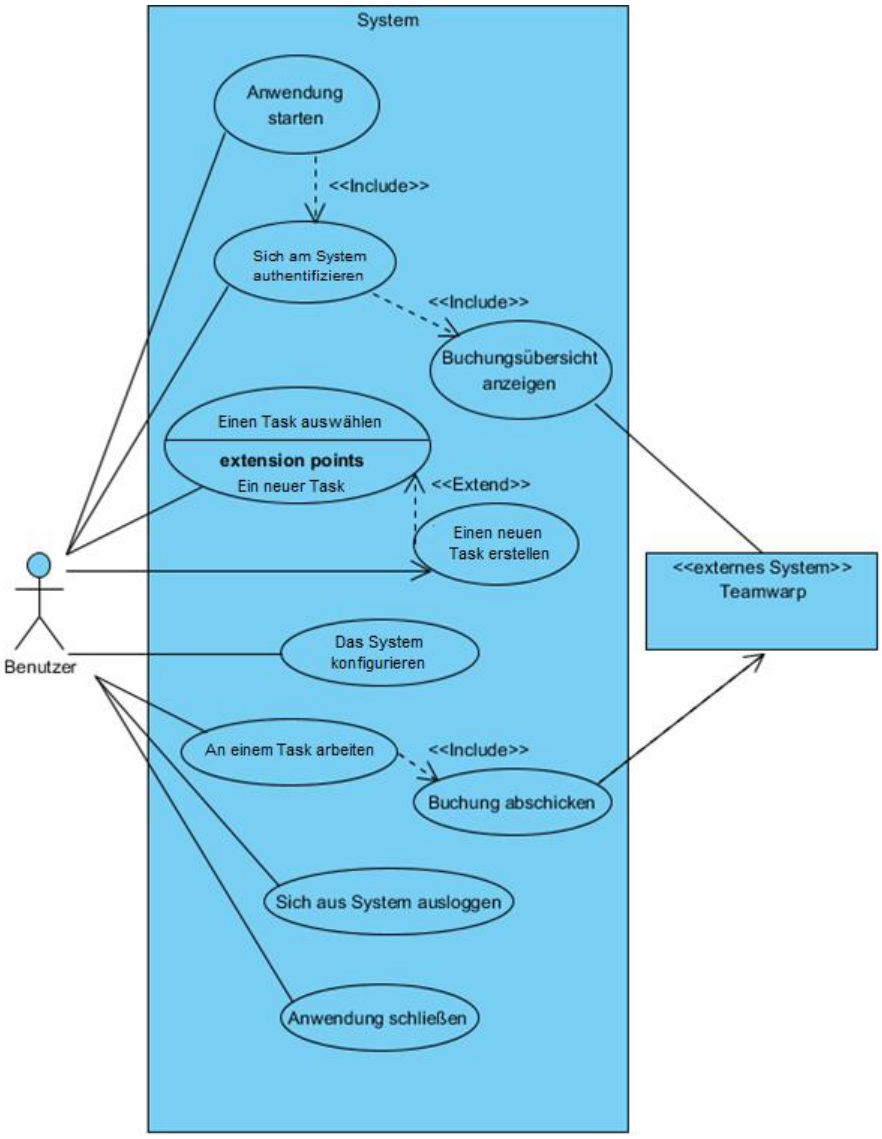

Abbildung 4: Anwendungsfalldiagramm

Das Anwendungsfalldiagramm in Abbildung 4 zeigt die wichtigsten fachlichen Anforderungen (im Kapitel 3.4.1) mit den zwei beteiligten Akteuren.
Der Benutzer ist der Akteur, der mit dem System interagieren will. Der andere Akteur ist Teamwarp, welches die Buchungen vom System annimmt und speichert.

Der Benutzer startet die Anwendung. Um die Anwendung zu benutzen muss der Benutzer sich beim System authentifizieren. Nachdem die Authentifikation erfolgt ist, wird dem Benutzer eine Übersicht über die generischen Zeitbuchungen angezeigt. Im nächsten Schritt wählt der Benutzer den Task, an dem er arbeiten will. Der Benutzer kann dann an dem ausgewählten Task arbeiten. Nachdem der Benutzer die Arbeit an einem Task beendet hat, kann die Buchung an Teamwarp geschickt werden. Der Benutzer kann die Anwendung schließen wenn er sie nicht mehr benutzen will.

3.5 Kontextabgrenzung

Der vorherige Abschnitt hat die fachlichen und technischen Anforderungen erläutert, die im Rahmen dieser Arbeit zu realisieren sind. In diesem Abschnitt geht es um die Abgrenzung der Arbeit. Es wird vorgestellt, was innerhalb der Arbeit nicht umgesetzt werden soll. Die Abgrenzung teilt sich in zwei Unterpunkte ein: fachliche und technische Abgrenzung.

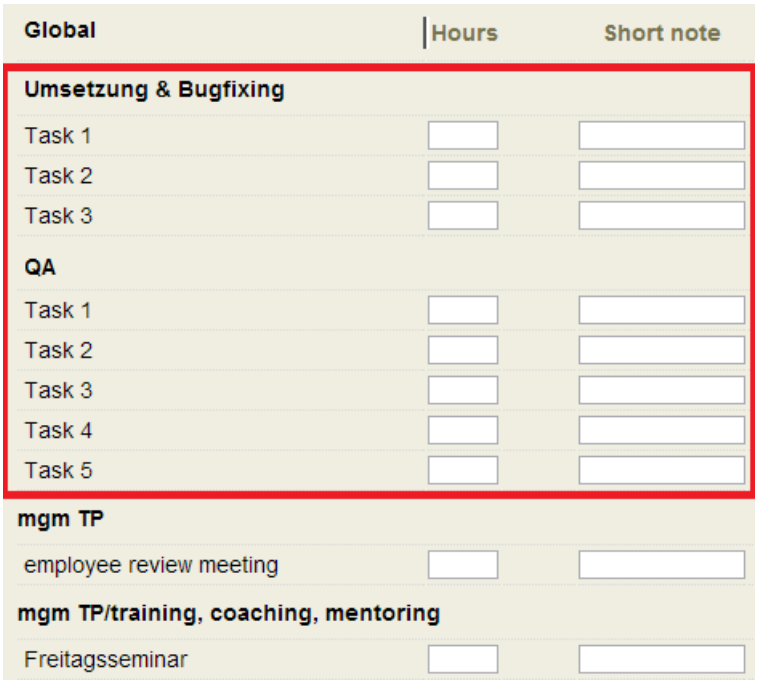

Abbildung 5: Fachliche Kontextabgrenzung

- **Fachliche Abgrenzung:**
  - Die monatliche Übersicht.
    In iWarp ist eine monatliche Übersicht nicht umzusetzen. Es ist nicht erforderlich, dass Benutzer in iWarp die Zeiterfassungen von einzelnen Tagen sehen können wie in Abbildung 1.
  - Die allgemeine Zeiterfassung eines Tages.
    iWarp soll die Start-, Ende und Pausenzeiten von Benutzern nicht erfassen.

- Die nicht-generischen Zeitbuchungen.
  Die Zeiterfassungen von allen festgelegten Tätigkeiten sind nicht in iWarp zu realisieren. In Abbildung 5 ist eine Grenze eingezeichnet, die alle generischen Buchungseinträge umgibt. Alle Buchungseinträge, die außerhalb dieser Grenze liegen, sind für generische Zeitbuchung irrelevant und nicht in iWarp zu implementieren.

- **Technische Abgrenzung:**
  - Die Schnittstelle zu Teamwarp ist nicht im Rahmen dieser Arbeit zu erstellen. Diese wird von mgm-technologies-partners bereitgestellt.

4 Spezifikation

Im vorherigen Kapitel wurde die Analysephase, insbesondere die Anforderungen vorgestellt, die als Ausgangpunkt für die Spezifikation dient. Im folgenden Kapitel geht es um die Spezifikation der Anwendung. Das Kapitel teilt sich in mehrere Abschnitte ein. Im Abschnitt 4.1 wird das fachliche Datenmodell beschrieben, welches mithilfe eines Klassendiagramms dargestellt wird. Der Abschnitt 4.2 behandelt die Funktionalitäten der Anwendung. Diese Funktionalitäten ergeben sich aus den fachlichen Anforderungen. Im Abschnitt 4.3 werden die Anwendungsfälle auf Basis des Anwendungsdiagramms vorgestellt. Im Abschnitt 4.4 werden die Dialoge der Anwendung der Anwendung entwickelt. Es werden die entworfenen Dialoge in Form von Mockups zusammen mit Beschreibungen eingeführt. Im Abschnitt 4.5 wird der Ablauf von Dialogen erläutert. Es werden die Übergänge zwischen Dialogen ausführlich beschrieben.

4.1 Fachliches Datenmodell

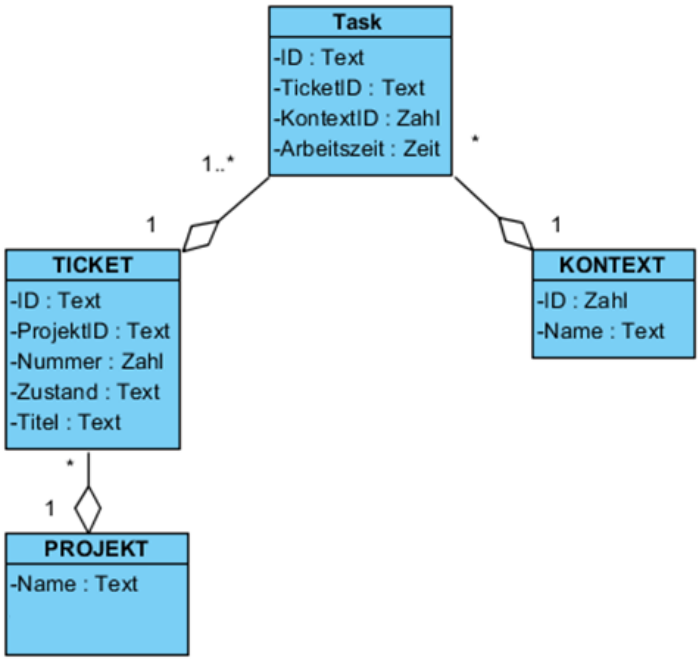

Abbildung 6: Fachliches Datenmodell

Die Abbildung 6 stellt das fachliche Datenmodell dar. Die Datentypen im Datenmodell wurden aus den Anforderungen ermittelt.

- **PROJEKT:**
  Der Datentyp bezeichnet das Projekt, an dessen Tickets der Benutzer arbeiten will. Jedes Projekt hat einen eigenen Name, die Projektnamen sind im Unternehmen eindeutig. Deswegen benötigt der Datentyp Projekt keine ID. Ein Projekt kann viele oder keine Tickets haben.
- **TICKET:**
  Der Datentyp bezeichnet das Ticket, an dem der Benutzer arbeiten will. Jedes Ticket gehört genau zu einem Projekt. Jedes Ticket hat eine Nummer, einen Zustand und einen Titel. Die ID von einem Ticket wird durch sein Projekt und seine Nummer zusammengesetzt. Ein Task wird aus einem Ticket und einem Kontext gebildet. Deshalb können aus einem Ticket ein oder mehrere Tasks erstellt werden.
- **KONTEXT:**
  Der Datentyp bezeichnet den Kontext, unter dem der Benutzer arbeiten will. Jeder Kontext hat einen Name und eine sogenannte generische ID. Unter einem Kontext können viele oder keine Tickets bearbeitet werden.
- **TASK:**

Der Datentyp bezeichnet den Task, an welchem der Benutzer arbeiten will. Jeder Task besteht aus einem Ticket und einem Kontext. Jeder Task hat ein Attribut: „Arbeitszeit“, diese Attribut drückt die Zeitspanne aus, die der Benutzer an einem Task gearbeitet hat.

4.2 Funktionalitäten

In diesem Abschnitt werden die gewünschten Funktionalitäten der Anwendung ausgeführt. Die Funktionalitäten werden im Wesentlich aus der fachlichen Anforderungen abgeleitet, sie werden nach Prioritäten in drei Gruppen verteilt.

1) **Funktionalitäten mit hoher Priorität (sehr wichtig)**
   - **F1:** Der Benutzer kann sich ins System einloggen, indem er seinen mgm-Benutzername und sein Passwort eingibt.
   - **F2:** Der Benutzer hat eine Übersicht über die Buchungen, die er mithilfe der Anwendung an dem jeweiligen Arbeitstag gemacht hat.
   - **F3:** Der Benutzer kann ein Ticket auswählen, indem er ein Projekt selektiert und die Ticketnummer eingibt oder ein vorgeschlagenes Ticket auswählt.
   - **F4:** Der Benutzer kann aus dem ausgewählten Ticket einen Task erzeugen, indem er einen Kontext für das Ticket selektiert.
   - **F5:** Der Benutzer kann einen Task starten, damit der Zähler der Arbeitszeit für den Task beginnt.
   - **F6:** Der Benutzer kann einen Task stoppen, damit der Zähler der Arbeitszeit für den Task stoppt.
   - **F7:** Der Benutzer kann die Buchungen an Teamwarp schicken.
   - **F8:** Der Benutzer kann sich aus dem System ausloggen.
2) **Funktionalitäten mit mittlerer Priorität (wichtig)**
   - **F9:** Dem Benutzer werden die von ihm zuvor eingegebenen Tickets vorgeschlagen, wenn er versucht ein Ticket auszuwählen.
   - **F10:** Wenn der Benutzer mindestens die ersten zwei Ziffern einer Ticketnummer eingibt, werden ihm alle Tickets angezeigt, welche mit diesen Ziffern beginnen.
   - **F11:** Dem Benutzer werden die Zusatzinformationen (Status und Titel) von Tickets angezeigt.
   - **F12:** Der Benutzer kann ein neues Ticket erstellen, welches nicht in der Staging-Datenbank vorhanden ist. Der Benutzer wird darauf hingewiesen, dass das erstellte Ticket nur innerhalb des Systems existiert.
   - **F13:** Wenn ein Task „aktiv“ ist, kann der Benutzer die aktuelle Arbeitszeit des Tasks sehen.
   - **F14:** Wenn ein Task gestoppt wird, werden dem Benutzer eine aufgerundete und eine abgerundete Arbeitszeit angezeigt. Der Benutzer kann dem Task eine von den zwei gerundeten Arbeitszeiten zuordnen.

- **F15:** Wenn ein Task gestoppt wurde, werden dem Benutzer sowohl die gerundete als auch die tatsächliche Arbeitszeit des Tasks angezeigt.
- **F16:** Der Benutzer kann an einem Task weiterarbeiten, indem er den Task wieder startet. Dabei wird die tatsächliche Arbeitszeit als Ausgangswert verwendet.
- **F17:** Der Benutzer kann maximal an einem Task auf einmal arbeiten. Der Benutzer wird gewarnt, wenn er einen Task startet während ein anderer Task noch „in Bearbeitung“ ist.
- **F18:** Der Benutzer kann im System festlegen, welche Projekte bzw. Buchungskontexte ihm zur Verfügung gestellt werden.

3) Funktionalitäten mit niedriger Priorität (nicht so wichtig)

Neben den Funktionalitäten, die aus den fachlichen Anforderungen abgeleitet werden, werden im Folgenden die Funktionalitäten mit niedriger Priorität erläutert. Dabei handelt es sich um Features, welche die Benutzbarkeit verbessern.

- **F19:** Der Benutzer kann einen hinzugefügten Task entfernen, an dem er noch nicht gearbeitet hat. Ein bearbeiteter Task darf nicht entfernt werden können. Denn dies führt dazu, dass der Benutzer die Übersicht über die Tagesbuchungen verlieren kann.
- **F20:** Der Benutzer kann konfigurieren, für wie lange ein eingegebenes Ticket im Speicher der Anwendung bleibt. Das bedeutet, dass der Benutzer festlegen kann, nach wie vielen Tagen ein von ihm ausgewähltes Ticket nicht mehr bei der Auswahl eines neuen Tickets vorgeschlagen wird.
- **F21:** Der Benutzer kann den Buchungskontext von einem Task ändern.

4.3 Anwendungsfälle

Im Abschnitt 3.4.3 wurde das Anwendungsfalldiagramm vorgestellt, das nur auf den sehr wichtigen Anforderungen basiert und nur das Erfolgsszenario der einzelnen Anwendungsfälle enthält. In diesem Abschnitt werden alle Anwendungsfälle ausführlicher dargestellt.

Titel	UC1: Sich am System authentifizieren
Akteur	Benutzer
Ziel	Der Benutzer kann die Anwendung nutzen
Auslöser	Der Benutzer startet die Anwendung
Erfolgsszenario	1. Der Benutzer gibt seinen eigenen Benutzernamen und sein Passwort ein. 2. Das System überprüft die Zugangsdaten. 3. Das System wechselt zur Hauptseite der Anwendung.
Fehlerfälle	2.a. Anmeldung des Benutzers schlägt fehl: System meldet einen Anmeldungsfehler und fordert zur nochmaligen Eingabe auf.

Anforderungen	A1.1

Tabelle 1: Anwendungsfall "Sich am System authentifizieren"

Titel	UC2: Buchungsübersicht anzeigen
Akteur	System
Ziel	Der Benutzer hat eine Übersicht über die Tagesbuchungen
Auslöser	Der Benutzer navigiert zur Hauptseite der Anwendung
Erfolgsszenario	1. Das System zeigt alle Tasks an, die der Benutzer am Tag hinzugefügt hat. 2. Das System zeigt die geleisteten Arbeitszeiten der Tasks an. 3. Das System zeigt den laufenden Task an.
Anforderungen	A1.4

Tabelle 2: Anwendungsfall "Buchungsübersicht anzeigen"

Titel	UC3: Einen Task auswählen
Akteur	Benutzer
Ziel	Der Benutzer kann an einem Task arbeiten
Auslöser	Der Benutzer möchte einen Task hinzufügen
Erfolgsszenario	1. Der Benutzer wählt ein Projekt aus. 2. Das System schlägt Tickets mit Zusatzinformationen, die zum Projekt gehören und schon vom Benutzer ausgewählt wurden, vor. (a)3. Der Benutzer sucht eines der vorgeschlagenen Tickets aus. (a)4. Der Benutzer wählt einen Kontext für das Ticket aus. (b)3. Der Benutzer gibt eine Nummer ein. (b)4. Das System zeigt alle Tickets mit Zusatzinformationen an, die mit der eingegebenen Nummer beginnen. (b)5. Der Benutzer wählt ein Ticket aus, an dem er arbeiten will. (b)6. Der Benutzer wählt einen Kontext für das Ticket aus, unter dem er arbeitet.
Erweiterungen	5b.a. Der Benutzer findet das gewünschte Ticket nicht. 5b.a.1. Ausführung des Anwendungsfalls: „Eine neue Task erstellen“.
Anforderungen	A1.2, A1.3, A1.8, A1.10

Tabelle 3: Anwendungsfall "Eine Task auswählen"

Titel	UC4: Einen neuen Task erstellen
Akteur	Benutzer
Ziel	Der Benutzer kann an einem Task arbeiten, dessen Ticket nicht in der Staging-Datenbank existiert.
Auslöser	Der Benutzer findet das Ticket nicht, an dem er arbeiten will.
Erfolgsszenario	1. Der Benutzer wählt ein Projekt aus. 2. Der Benutzer gibt die Ticketnummer ein.

	3. Der Benutzer erstellt das Ticket. 4. Der Benutzer sucht einen Kontext für das Ticket aus.
Anforderungen	A1.9
Akteur	Benutzer

Tabelle 4: Anwendungsfall "Einen neuen Task erstellen"

Titel	UC5: Das System konfigurieren
Akteur	Benutzer
Ziel	Der Benutzer kann Projekte bzw. Kontexte ausschließen, welche ihm nicht zur Verfügung gestellt werden sollen. Der Benutzer kann die Lebenszeit von Tickets einstellen, die entscheidet, nach wie vielen Tagen ein Ticket, das von ihm eingegeben wurde, nicht mehr vom System vorgeschlagen wird.
Erfolgsszenario	1. Der Benutzer wählt Projekte aus, die ihm zur Verfügung gestellt werden sollen. 2. Der Benutzer wählt Kontexte aus, die ihm zur Verfügung gestellt werden sollen. 3. Der Benutzer stellt die Lebensdauer von Tickets ein. 4. Das System speichert die Einstellungen.
Fehlerfälle	4.a. Der Benutzer hat kein Projekt oder keinen Kontext ausgewählt: System meldet den Fehler und fordert zum nochmaligen Auswählen auf.
Anforderungen	A1.7

Tabelle 5: Anwendungsfall "Das System konfigurieren"

Titel	UC6: An einem Task arbeiten
Akteur	Benutzer
Ziel	Der Benutzer kann an einem Task arbeiten und die geleistete Arbeitszeit erfassen.
Auslöser	Der Benutzer möchte an einem Task arbeiten.
Erfolgsszenario	1. Der Benutzer startet einen Task. 2. Das System versetzt den Task in den „laufend“ Zustand. 3. Das System zeigt die aktuelle Arbeitszeit des Tasks an. 4. Der Benutzer stoppt den Task. 5. Das System zeigt die aufgerundete und die abgerundete Arbeitszeit an. 6. Der Benutzer wählt eine der zwei gerundeten Arbeitszeiten aus. 7. Das System versetzt die Task in den Ruhezustand. 8. Das System zeigt die gerundete und die tatsächliche Arbeitszeit des Tasks an.
Fehlerfälle	1.a. Der Benutzer startet einen Task, während sich ein anderer Task im „laufenden“ Zustand befindet: Das System meldet den Fehler und

	der Task kann nicht gestartet werden. 1.b. Jeder Buchungskontext verfügt über eine begrenzte Anzahl von Buchungszeilen (siehe **Error! Reference source not found.**). Der Benutzer startet einen Task, deren Buchungskontext keine freie Buchungszeile mehr zur Verfügung stellt: Das System meldet den Fehler und der Task kann nicht gestartet werden.
Erweiterungen	6.1. Ausführung des Anwendungsfalls „Buchung schicken“.
Anforderungen	A1.5

Tabelle 6: Anwendungsfall "An einem Task arbeiten"

Titel	UC7: Buchung schicken
Akteur	System
Ziel	Die Zeitbuchungen werden in Teamwarp gespeichert.
Auslöser	Der Benutzer stoppt einen Task und wählt die gerundete Arbeitszeit aus.
Erfolgsszenario	1. Das System sammelt die Eingabewerte der Buchung: den Buchungskontext, das Ticket und die Arbeitszeit. 2. Das System schickt die Buchung an Teamwarp. 3. Teamwarp empfängt und speichert die Buchung.
Fehlerfälle	2.a. Die Buchung kommt nicht bei Teamwarp an: Das System meldet den Fehler.
Anforderungen	A1.6

Tabelle 7: Anwendungsfall "Buchung schicken"

4.4 Dialog

Dieses Kapitel behandelt die Dialoge der Anwendung auf Basis der Funktionalitäten. Mit Hilfe von balsamiq[6] werden die Dialoge entworfen. Für jeden Dialog wird jeweils ein Mockup dargestellt und die wichtigen Elemente des Mockups beschrieben. Außerdem wird bei jedem Dialog erläutert, welche Funktionalitäten er abdeckt.

[6] balsamiq ist ein Werkzeug für das Erstellen von Mockups (http://balsamiq.com)

4.4.1 LOGIN

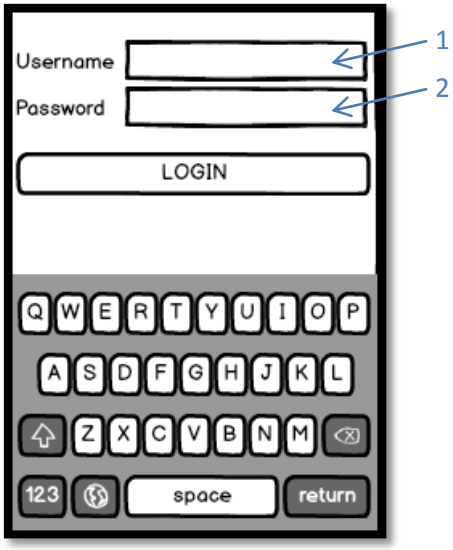

Abbildung 7: Mockup „LOGIN“

Im Dialog (Abbildung 7) können Benutzername und Passwort von einem Mitarbeiter eingegeben werden, um sich einzuloggen.

Element	Typ	Beschreibung
1	Eingabefeld	-Eingabeformat: Text -Inhalt: Benutzername
2	Eingabefeld	-Eingabeformat: Text -Inhalt: Passwort
LOGIN	Button	-Antippen, um die Zugangsdaten abzusenden und sich einzuloggen. -Wenn die Zugangsdaten korrekt sind, wechselt das System in den Dialog „YOUR BOOKING“ (**Error! Reference source not found.**). -Wenn die Zugangsdaten falsch sind, wird eine Fehlermeldung gezeigt. **Aufruf:** „YOUR BOOKING“ (Das bedeutet, das Antippen auf diesen Button diese Seite aufruft).

Tabelle 8: Mockup "LOGIN"

Abgedeckte Funktionalitäten: **F1.**

4.4.2 YOUR BOOKING

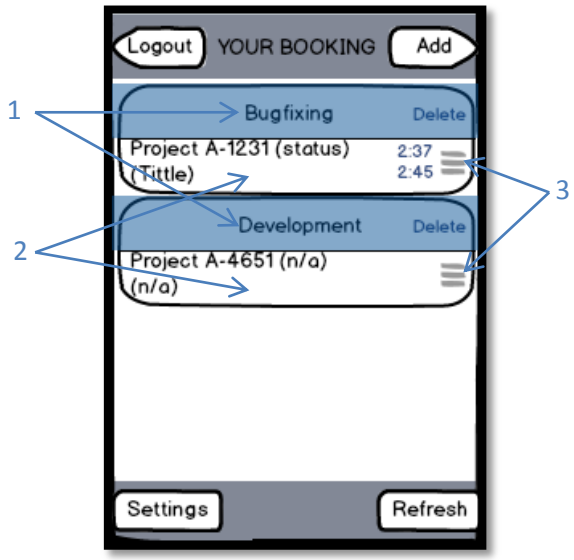

Abbildung 8: Mockup "YOUR BOOKING"

Im Dialog (Abbildung 8) können Mitarbeiter die Übersicht über die Tagesbuchungen sehen.

Element	Typ	Beschreibung
1	Listenüberschrift	Die Listenüberschriften stellen die Kontexte von Tickets dar.
2	Listenelement, Button	-Jedes Listenelement stellt ein Ticket dar, das unter einem Kontext steht. -Jedes Listenelement ist wiederum ein Button. Antippen, um den entsprechenden Task zu starten. Wenn ein anderer Task „in Bearbeitung“ ist, wird eine Fehlermeldung gezeigt. Ansonsten wird der ganze Button gelb und der Zähler der Arbeitszeit fängt an zu zählen. -Wenn auf einen „in Bearbeitung“ Task getippt wird, erscheint ein Pop-up, in dem der Benutzer die gerundete Arbeitszeit wählen können. **Aufruf:** „YOUR BOOKING-STOP“.
2:37	Text	Stellt die tatsächliche geleistete Arbeitszeit dar.
2:45	Text	Stellt die gerundete Arbeitszeit dar.
(status),(n/a)	Text	Stellt den Zustand des Tickets dar. „n/a“ bedeutet dass der Zustand des Tickets nicht verfügbar ist.

(Title), (n/a)	Text	Stellt den Titel des Tickets dar. „n/a“ bedeutet dass der Titel des Tickets nicht verfügbar ist.
3	Button	Antippen um den Kontext des Tickets zu ändern. **Aufruf:** „YOUR BOOKING-CHANGE BRANCH“.
Delete	Button	Antippen, um einen ausgewählten Task des entsprechenden Kontexts zu entfernen. Wenn auf den Button angetippt wird, kann der Benutzer wählen, welcher Task entfernt werden soll.
Logout	Button	Antippen, um sich vom System abzumelden. **Aufruf:** „LOGIN“.
Add	Button	Antippen, um einen neuen Task hinzuzufügen. **Aufruf:** „SELECT TICKET“.
Settings	Button	**Aufruf:** „SETTINGS“.
Refresh	Button	Antippen, um die hinzugefügten Tickets zu aktualisieren.

Tabelle 9: Mockup "YOUR BOOKING"

Abgedeckte Funktionalitäten: **F2, F5, F6, F8, F11, F13, F15, F16, F17, F19.**

4.4.3 YOUR BOOKING-CHANGE BRANCH

Abbildung 9: Mockup "YOUR BOOKING-CHANGE BRANCH"

Im Dialog (Abbildung 9) kann der Benutzer den Kontext von einem Ticket ändern. Dieser Dialog besteht aus einem Pop-up, das über dem Dialog „YOUR BOOKING“ erscheint.

Element	Typ	Beschreibung
CHANGE BRANCH	Listenüberschrift	
Development **Bugfixing**	Listenelement, Button	-Jedes Listenelement stellt einen verfügbaren Kontext dar, der als neuer Kontext für ein Ticket ausgewählt werden kann. -Jedes Listenelement ist wiederum ein Button. Wenn auf einen Button angetippt wird, ordnet die Anwendung dem Ticket den entsprechenden Kontext zu.
CANCEL	Button	Antippen damit das Pop-up ausgeblendet wird. Dabei wird nichts ausgewählt. **Aufruf:** „YOUR BOOKING“.

Tabelle 10: Mockup "YOUR BOOKING-CHANGE BRANCH"

Abgedeckte Funktionalitäten: **F21.**

4.4.4 YOUR BOOKING-STOP

Abbildung 10: Mockup "YOUR BOOKING-STOP"

Im Dialog (Abbildung 10) können Mitarbeiter die gerundete Arbeitszeit für einen Task auswählen. Der Dialog besteht aus einem Pop-up, das über dem Dialog „YOUR BOOKING“ erscheint.

Element	Typ	Beschreibung
STOP **Please choose…**	Listenüberschrift	Stellt den Zweck des Pop-ups dar. Übersetzung: „STOP Wählen Sie bitte Ihre gerundete Arbeitszeit“.
2:30, **2:45**	Listenelement, Button	-Stellt die abgerundete oder aufgerundete Arbeitszeit dar. -Ist auch ein Button. Wenn auf den Button angetippt wird, wählt die Anwendung die gerundete Arbeitszeit für den Task. Die gewählte Arbeitszeit wird anschließend an Teamwarp geschickt. Der Zähler der Arbeitszeit hört dabei auf zu zählen. **Aufruf:** „YOUR BOOKING“.
CANCEL	Button	Antippen damit das Pop-up ausgeblendet wird. Dabei wird nichts ausgewählt. **Aufruf:** „YOUR BOOKING“.

Tabelle 11: Mockup "YOUR BOOKING-STOP"

Abgedeckte Funktionalitäten: **F7, F14.**

4.4.5 SELECT TICKET

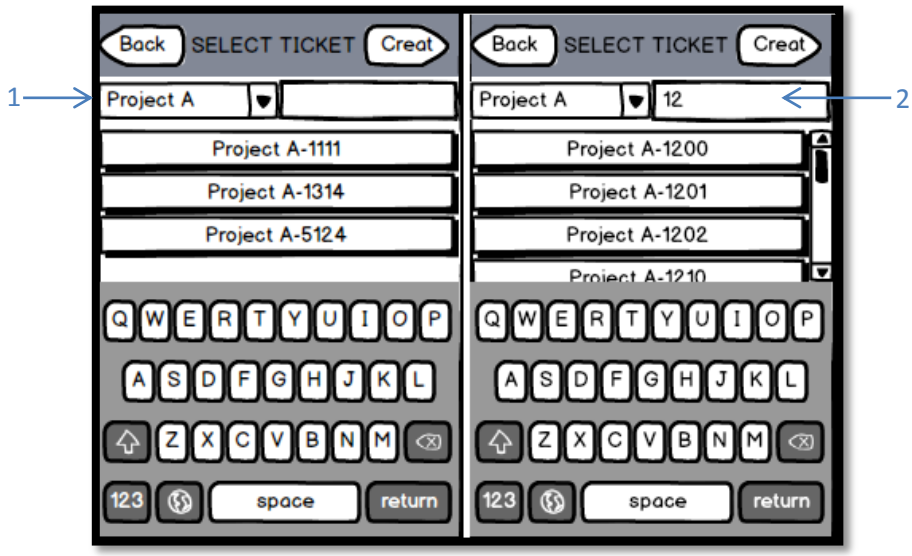

Abbildung 11: Mockup "SELECT TICKET"

In diesen Dialogen (Abbildung 11) kann der Benutzer Tickets auswählen, an denen er arbeitet. Der Benutzer kann ein Ticket, das er schon zuvor selektiert hat und von der Anwendung vorgeschlagen wird, auswählen. Alternativ wählt der Benutzer ein Ticket aus, indem er die Ticketnummer eingibt und nach dem gewünschten Ticket sucht.

Element	Typ	Beschreibung
1	Auswahlliste[7]	Stellt die verfügbaren Projekten dar, von denen der Benutzer ein Projekt auswählt.
2	Eingabefeld	Hier kann der Benutzer die Ticketnummer eingeben, um nach dem gewünschten Ticket zu suchen. Falls das Ticket mit der eingegebenen Nummer in der Staging-Datenbank noch nicht vorhanden ist, wird das Ticket vom System nicht gefunden.
Project A-1111 **Project A-1314**	Listenelement, Button	Wenn das Kästchen leer ist oder nur eine Ziffer enthält, werden die Tickets vorgeschlagen, die

[7] Eine Wahlliste stellt Optionen dar, von denen der Benutzer eine selektiert. Bei einer Auswahlliste sind nur die ausgewählte Option sichtbar.

Project A-5124 **In Abbildung 12a**		vom Benutzer bereits ausgewählt wurden. -Jedes Listenelement stellt ein vorgeschlagenes Ticket dar. -Jedes Listenelement ist auch ein Button. Wenn auf den Button getippt wird, wird das entsprechende Ticket ausgewählt. **Aufruf:** „SELECT TICKET-SELECT BRANCH“
Project A-1200 **Project A-1201** **Project A-1202…** **In Abbildung 12b**	Listenelement, Button	Wenn das Kästchen zwei oder mehr Ziffern enthält, werden Tickets vorgeschlagen die mit diesen Ziffern beginnen. -Jedes Listenelement stellt ein vorgeschlagenes Ticket dar. -Jedes Listeelement ist auch ein Button. Wenn auf den Button getippt wird, wird das entsprechende Ticket ausgewählt. **Aufruf:** „SELECT TICKET-SELECT BRANCH“
BACK	Button	**Aufruf:** „YOUR BOOKING“
CREATE	Button	**Aufruf:** „CREATE TICKET“

Tabelle 12: Mockup "SELECT TICKET"

Abgedeckte Funktionalitäten: **F3, F9, F10.**

4.4.6 SELECT TICKET-SELECT BRANCH

Abbildung 13: Mockup "SELECT TICKET-SELECT BRANCH"

Im Dialog (Abbildung 13) kann der Benutzer den Kontext für das ausgewählte Ticket selektieren. Der Dialog besteht aus einem Pop-up, das über dem Dialog „SELECT TICKET“ erscheint.

Element	Typ	Beschreibung
SELECT BRANCH	Listenüberschrift	
Development, Bugfixing	Listenelement, Button	-Jedes Listenelement stellt einen Kontext dar. -Jedes Listenelement ist wiederum ein Button. Wenn auf den Button getippt wird, ordnet die Anwendung dem Ticket den entsprechenden Kontext zu. Der resultierende Task wird zum Dialog „YOUR BOOKING“ hinzugefügt. **Aufruf:** „YOUR BOOKING“
CANCEL	Button	Antippen, damit das Pop-up ausgeblendet wird. Dabei wird nichts ausgewählt. **Aufruf:** „SELECT TICKET“.

Tabelle 13: Mockup "SELECT TICKET-SELECT BRANCH"

Abgedeckte Funktionalitäten: **F4.**

4.4.7 CREATE TICKET

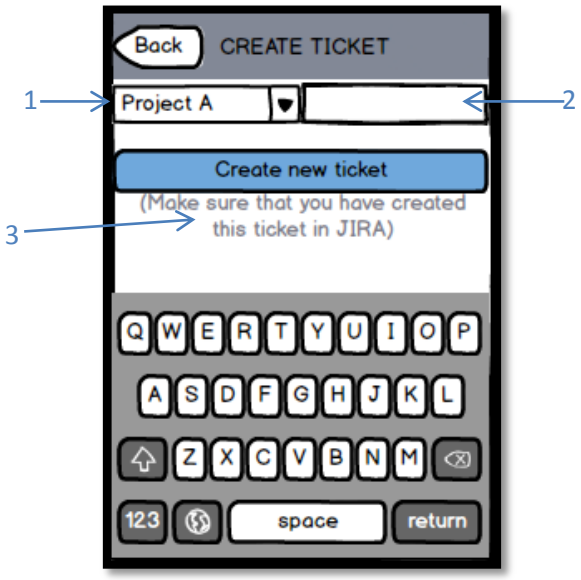

Abbildung 14: Mockup "CREATE TICKET"

Im Dialog (Abbildung 14) kann Benutzer ein Ticket neu erstellen, welches in der Staging-Datenbank noch nicht vorhanden ist. Das neu erstellte Ticket existiert nur innerhalb des Systems. Deshalb sollte in JIRA ein entsprechendes Ticket bereits erstellt sein.

Element	Typ	Beschreibung
1	Auswahlliste	Stellt die verfügbaren Projekte dar, von denen der Benutzer nur ein Projekt auswählen kann.
2	Eingabefeld	Hier kann der Benutzer die Nummer des zu erstellenden Tickets eingeben.
Create new ticket	Button	Antippen um einen Kontext für das Ticket auszusuchen. **Aufruf:** „CREATE TICKET-SELECT BRANCH“
3	Text	Ein Hinweis. Übersetzung: „Stellen Sie sicher dass Sie dieses Ticket in JIRA erstellt haben“.
Back	Button	**Aufruf:** „SELECT TICKET“.

Tabelle 14: Mockup "CREATE TICKET"

Abgedeckte Funktionalitäten: **F12.**

4.4.8 CREATE TICKET-SELECT BRANCH

Abbildung 15: Mockup "CREATE TICKET-SELECT BRANCH"

Im Dialog (Abbildung 15) kann der Benutzer einen Kontext für das neuerstellte Ticket auswählen. Der Dialog besteht aus einem Pop-up, das über dem Dialog „CREATE TICKET“ erscheint.

Element	Typ	Beschreibung
SELECT BRANCH	Listenüberschrift	Übersetzung: „Kontext wählen“.
Development, Bugfixing	Listenelement, Button	-Jedes Listenelement stellt einen Kontext dar. -Jedes Listenelement ist wiederum ein Button. Wenn auf den Button getippt wird, ordnet die Anwendung dem Ticket den entsprechenden Kontext zu. Der resultierte Task wird zum Dialog „YOUR BOOKING“ hinzugefügt. **Aufruf:** „YOUR BOOKING“.
CANCEL	Button	Antippen, damit das Pop-up ausgeblendet wird. Dabei wird nichts ausgewählt. **Aufruf:** „CREATE TICKET“.

Tabelle 15: Mockup "CREATE TICKET-SELECT BRANCH"

Abgedeckte Funktionalitäten: **F4**.

4.4.9 SETTINGS

Abbildung 16: Mockup "SETTINGS"

Im Dialog (Abbildung 16) kann der Benutzer folgende Einstellungen der Anwendung vornehmen:

- Welche Projekte in der Wahlliste zur Verfügung stehen sollen.

- Welche Kontexte im Pop-up für die Auswahl der Kontexte(wie in Abbildung 9, Abbildung 13 und Abbildung 15) wählbar sein sollen.
- Nach wie vielen Tagen ein ausgewähltes Ticket vom Speicher der Anwendung entfernt werden soll, sodass einem Mitarbeiter das Ticket nicht mehr vorgeschlagen wird.

Element	Typ	Beschreibung
PROJECTS, BRANCHES, TICKET LIFETIME	Text	Stellen die Überschriften für Projekte, Kontexte und Ticketlebensdauer dar.
Project-A, Project-B	Kontrollbox	-Jedes Kontrollboxelement stellt ein Projekt dar, das einem Mitarbeiter zugeordnet wird. -Wenn ein Kontrollboxelement angehakt wird, ist das entsprechende Projekt in der Auswahlliste der Projekte verfügbar. -Es können mehrere Elemente angehakt werden.
Development, Bugfixing, QA	Kontrollbox	-Jedes Kontrollboxelement stellt einen Kontext dar, unter dem ein Mitarbeiter arbeiten kann. -Wenn ein Kontrollboxelement angehakt wird, ist der entsprechende Kontext im Pop-up, in dem der Benutzer einen Kontext für das Ticket auswählen kann, verfügbar. -Es können mehrere Elemente angehakt werden.
Forever, 1 day, 2 days	Radiogruppe[8]	-Jedes Radiogruppenelement stellt eine mögliche Ticketlebensdauer dar. -Es kann nur eine Option ausgewählt werden.
Back	Button	Antippen, um die Änderung der Einstellung zu verwerfen. **Aufruf:** „YOUR BOOKING“.
Save	Button	Antippen, um die Änderung der Einstellung zu speichern. **Aufruf:** „YOUR BOOKING“.
Refresh	Button	Dem Benutzer kann ein oder mehrere neue Projekte sowie Kontexte zugewiesen werden. Wenn auf den Button angetippt wird, werden die Listen von möglichen Projekten und Kontexten aktualisiert.

Tabelle 16: Mockup "Settings"

Abgedeckte Funktionalitäten: **F18, F20.**

[8] Eine Radiogruppe stellt Optionen dar, von denen nur eine ausgewählt werden kann. Bei einer Radiogruppe sind alle Optionen sichtbar.

4.5 Ablauf von Dialogen

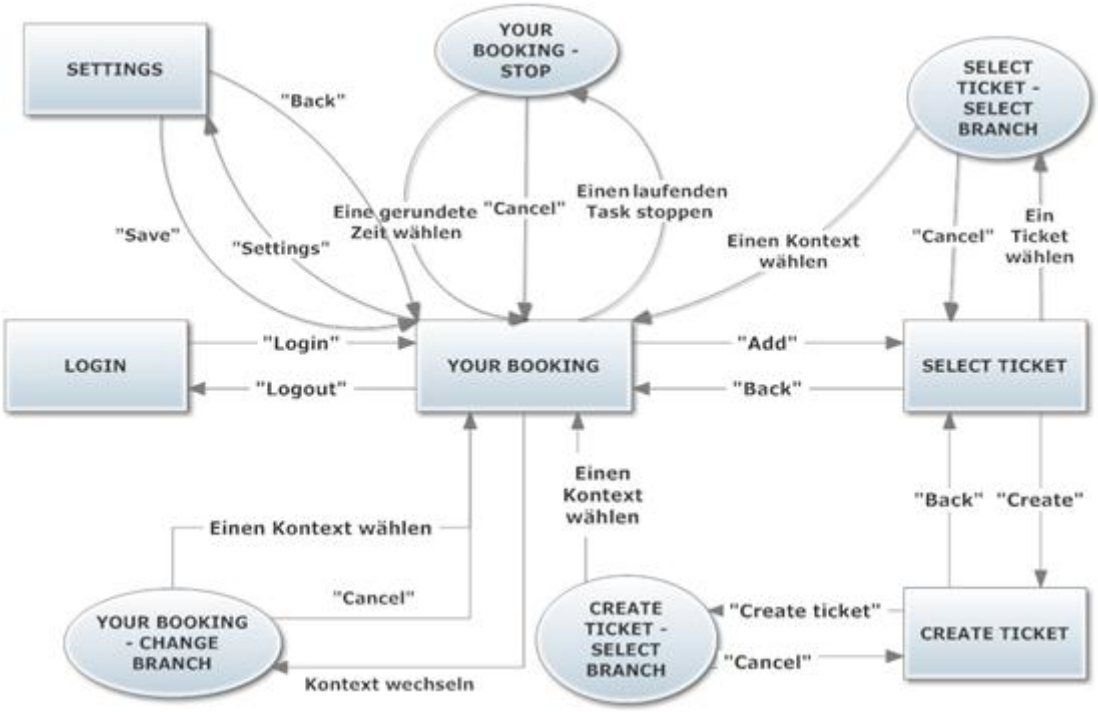

Abbildung 17: Ablauf von Dialogen

Im letzten Abschnitt wurden die Dialoge in Form von Mockups vorgestellt. In diesem Abschnitt wird die Abfolge der Dialoge (Abbildung 17) erläutert.

- **LOGIN**
  Der Benutzer beginnt mit dem Dialog „LOGIN“, bei dem er seinen eigenen Benutzername und Passwort eingeben muss um sich einzuloggen. Nachdem die Zugangsdaten erfolgreich überprüft werden, gelangt der Benutzer zum Dialog „YOUR BOOKING“.

- **YOUR BOOKING**
  Im Dialog „YOUR BOOKING“ wird dem Benutzer die Übersicht über die Tagesbuchungen angezeigt. Hier kann der Benutzer:
  - Den Kontext von einem Ticket ändern→ „YOUR BOOKING-CHANGE BRANCH“.
  - Einen Task an dem er noch nicht gearbeitet hat, von der Übersicht entfernen.
  - Einen Task starten.
  - Einen laufenden Task stoppen → „YOUR BOOKING-STOP“.

- Einen neuen Task hinzufügen → „SELECT TICKET“.
- Sich ausloggen → „LOGIN“.

- **YOUR BOOKING-CHANGE BRANCH**
  Hier kann der Benutzer:
  - Einen neuen Kontext für das Ticket wählen → „YOUR BOOKING“.
  - Den Vorgang abbrechen indem er „CANCEL“ antippt → „YOUR BOOKING“.

- **YOUR BOOKING-STOP**
  Hier kann der Benutzer:
  - Eine gerundete Arbeitszeit für die Task auswählen → „YOUR BOOKING“.
  - Den Vorgang abbrechen indem er „CANCEL“ antippt → „YOUR BOOKING“.

- **SELECT TICKET**
  Hier kann der Benutzer:
  - Ein vorgeschlagenes Ticket auswählen → „SELECT TICKET-SELECT BRANCH“.
  - Nach einem Ticket suchen und ein gefundenes Ticket auswählen → „SELECT TICKET-SELECT BRANCH“.
  - Ein neues Ticket erstellen, indem er „CREATE“ antippt → „CREATE TICKET“.
  - Zurück zu „YOUR BOOKING“ gehen, indem er „Back“ antippt → „YOUR BOOKING“.

- **SELECT TICKET-SELECT BRANCH**
  Hier kann der Benutzer:
  - Einen Kontext für das Ticket auswählen → „YOUR BOOKING“.
  - Den Vorgang abbrechen indem er „CANCEL“ antippt → „SELECT TICKET“.

- **CREATE TICKET**
  Hier kann der Benutzer:
  - Ein Ticket neuerstellen, indem er die Ticketnummer eingibt und „Create new ticket“ antippt → „CREATE TICKET-SELECT BRANCH“.
  - Zurück zu „SELECT TICKET“ gehen indem er „Back“ antippt → „SELECT TICKET“.

- **CREATE TICKET-SELECT BRANCH**
  Hier kann der Benutzer:
  - Einen Kontext für das neuerstellte Ticket auswählen → „YOUR BOOKING“.
  - Den Vorgang abbrechen indem er „CANCEL“ antippt → „CREATE TICKET“.

- **SETTINGS**
  Hier kann der Benutzer die Einstellung des Systems ändern und dann:
  - Die Änderung speichern, indem er „Save“ antippt → „YOUR BOOKING“.
  - Die Änderung verwerfen, indem er „Back“ antippt → „YOUR BOOKING“.

5 Entwurf

In diesem Kapitel handelt es sich um den Entwurf der Anwendung. Der Abschnitt 5.1 behandelt die technischen Entscheidungen. Es wird vorgestellt, welche Entwicklungstechnologie verwendet wird und warum sie ausgewählt wurde. Außerdem wird erläutert, welche Frameworks sowie Bibliotheken zur Vereinfachung der Implementierung zum Einsatz kommen sollen. Im nächsten Abschnitt 5.2 sind die Entwurfsentscheidungen. Es wird die Überlegung erläutert, ob die Anwendung wirklich einen Backend-Server benötigt und die Entscheidung wird begründet. Im Kapitel 5.3 wird die Architektur der Anwendung in Form von drei Bausteinsichten vorgestellt. Während die Bausteinsicht Level 0 einen Überblick über die Anwendung im Zusammenhang mit mobilen Endgeräten und dem externen System Teamwarp gibt, wird in der Bausteinsicht Level 1 der grobe Aufbau des internen Systems vorgestellt. Die Bausteinsicht Level 2 beschreibt ausführlich die einzelnen Bestandteile der Anwendung zusammen mit ihren Innerkomponenten sowie den Zusammenhang zwischen ihnen. Das Kapitel 5.4 behandelt die Abläufe der Anwendung. Mittels Sequenzdiagramme werden die Abfolgen der einzelnen Anwendungsfälle ausführlich dargestellt und beschrieben.

5.1 Technische Entscheidungen

Im Kapitel 2.Technologien wurden drei mögliche Entwicklungstechnologien für eine mobile Anwendung erläutert. Es soll in diesem Abschnitt entschieden und begründet werden, welcher Entwicklungsweg zum Einsatz kommen soll. Zu jedem Entwicklungsansatz gehören mehrere Frameworks, die die Programmierung vereinfachen und dabei den Aufwand der Entwicklung erleichtern. Auch müssen die Entscheidungen über die verwendete Frameworks getroffen werden.

5.1.1 Entwicklungstechnologie

Bei der Auswahl aus einer der drei Entwicklungswegen native, weborientierte und hybride Ansätzen, spielen die Anforderungen an die Anwendung sowie die Unternehmenskontext eine entscheidende Rolle. Das Ziel ist, einen Ansatz zu finden, mit dem sich die Anwendung mit einem vertretbaren Aufwand umsetzen lässt.
Wie in den Anforderungen in Kapitel 3.4.1 beschrieben sind die wichtigsten Aufgaben von iWarp: JIRA-Tickets von Teamwarp abfragen, Arbeitszeiten von Tasks berechnen und Zeitbuchungen an Teamwarp zu schicken. Diese Aufgaben benötigen in der Regel nur triviale Interaktionen. Deshalb ist kein Bedarf an hoher Performance oder anspruchsvoller Grafikdarstellung zu erwarten. Die hohe Geschwindigkeit beim Laden und Ausführen von Anwendungen sowie die Fähigkeit aufwändiger Grafikdarstellung gelten als die wichtigsten Vorteile des nativen Ansatzes (siehe das Kapitel 0). Um den Aufwand zu vermeiden, für jede Plattform eine Programmierspreche lernen zu müssen, kann aus den beiden oben beschrieben Argumenten der native Entwicklungsweg ausgeschlossen werden.
Bei der Ausführung der Anwendung werden nach den Anforderungen im Kapitel 3.4 keine Hardwarekomponenten von mobilen Geräten und keine plattformspezifischen Features benötigt. Darüber hinaus wird die Anwendung speziell für eine bestimmte Gruppe von Mitarbeitern entwickelt. Das bedeutet, dass die Anwendung nicht in App-Stores veröffentlicht werden muss. Die Anwendung kann den Benutzern auf andere Weise zur Verfügung gestellt werden, um die Gebühren für App-Stores zu sparen. Die zwei genannten Aspekte, der Zugriff auf die Hardware des Geräts und die Veröffentlichung einer Anwendung in App-Stores, sind nach Kapitel 2.3.2 die wesentlichen Vorteile des hybriden Ansatzes. Um eine Hybrid-App zu entwickeln, sind ein oder mehrere zusätzliche Frameworks wie zum Beispiel PhoneGap erforderlich. Damit dieser Aufwand umgegangen wird, kann der hybride Entwicklungsweg vernachlässigt werden.
Mitarbeiter nutzen die Anwendung nur im Büro, in dem eine Internetverbindung verfügbar ist. Diese erfüllt die Bedingung für eine Web-App, die eine dauerhafte Internetverbindung benötigt.
Aufgrund aller oben beschriebenen Faktoren lässt sich feststellen, dass der weborientierte Ansatz mit HTML, CSS und Javascript ein passender Ansatz ist.

5.1.2 Frameworks

Bei der Entwicklung dieser Webanwendung werden jQuery[9] und jQueryMobile[10] eingesetzt. jQuery ist eine kostenlose Javascript-Bibliothek, welche Funktionen zur Navigation und Manipulation von Webelementen zur Verfügung stellt. Dieses Framework vereinfacht die Programmierung von funktionalen Anforderungen.

[9] http://jquery.com/
[10] http://jquerymobile.com/

jQueryMobile ist ein Touchscreen-optimiertes weborientiertes Framework. Dieses Framework unterstützt mit HTML5 die UI-Programmierung von Webanwendungen, die zu fast allen mobilen Plattformen kompatibel[11] sind.

5.1.3 Bibliotheken

Außer jQuery und jQueryMobile werden einige weitere Javascript-Bibliotheken verwendet, um den Aufwand der Programmierung zu verringern.

- **jasmine[12]:** Ermöglicht den JUnit-Test von Javascript-Funktionen. Jasmine hat eine einfache Syntax, sodass Entwickler damit unkompliziert Test schreiben können.
- **jstorage[13]:** Vereinfacht die Speicherung der Daten in fast allen mobilen Browsern und allen Desktop-Browsern. jstorage unterstützt die Speicherungen von Texten, Zahlen, Javascript-Objekten und Arrays.
- **rc4[14]:** Stellt die Verschlüsselung und Entschlüsselung in Javascript zu Verfügung.
- **jquerytimer[15]:** Stellt zahlreiche Funktionen zur Zeitnahme bereit.

5.2 Entwurfsentscheidungen

Wie in den Anforderungen im Kapitel 3.4.1 beschrieben muss die Anwendung in der Lage sein, Zugangsdaten von Mitarbeitern zu überprüfen, die Arbeitszeit von Mitarbeitern zu berechnen und die von Mitarbeitern eingegebenen Tickets zu speichern. Dies sind die Anforderungen, die einen Backend-Server benötigen könnten. Denn die Überprüfung von Zugangsdaten benötigt eine Datenbank im Backend-Server, die Benutzernamen und Passwörter von Benutzern speichert. Die Berechnung der Arbeitszeiten könnte auch einen Backend-Server, in dem die Arbeitszeiten berechnet werden, brauchen. So würden die Arbeitszeiten weiterlaufen, auch wenn der Browser deaktiviert wird. Zur Speicherung der ausgewählten Tickets könnte auch ein Backend-Server, in dem sich eine Datenbank der Tickets befindet, nötig sein.
Im Folgenden wird begründet, ob für die Anwendung tatsächlich ein Backend-Server erforderlich ist.

- **Zugangsdaten überprüfen**
  Dank des Direct-Access-Interfaces von Teamwarp können Zugangsdaten von Mitarbeitern direkt in Teamwarp überprüft werden. Aus diesem Grund ist keine

[11] http://jquerymobile.com/gbs/
[12] http://pivotal.github.io/jasmine/
[13] http://jstorage.info/
[14] https://gist.github.com/farhadi/2185197
[15] http://jchavannes.com/jquery-timer/

Datenbank mit Benutzernamen und Passwörtern von allen Benutzern nötig. Für diese Anforderung braucht die Anwendung keinen Backend-Server.

- **Arbeitszeit berechnen**
  Wenn ein mobiler Browser aktiv ist, ist die Berechnung der Arbeitszeit ohne Backendserver problemlos möglich. Falls der Browser deaktiviert wird während ein Task läuft, ist es jedoch schwierig, die Arbeitszeit dieses Tasks weiter zu berechnen, nachdem der Browser wieder aktiviert wurde. Um diese Problem zu lösen gibt es zwei Lösungsmöglichkeiten. Eine Möglichkeit ist, die Arbeitszeiten permanent im Backend-Server zu speichern und zu berechnen. Die andere Möglichkeit ist, den Anfangszeitpunkt eines bearbeiten Tasks im System zu speichern. Mithilfe dieses Anfangszeitpunkts lässt sich jederzeit die aktuelle Arbeitszeit für einen Task berechnen, unabhängig davon, ob der Browser in der Zwischenzeit deaktiviert wurde.

- **Eingegebene Tickets speichern**
  Statt die ausgewählten Tickets von allen Benutzern in eine Datenbank im Backend zu speichern, kann die Anwendung die Daten von einem Mitarbeiter in seinem eigenen Browser speichern. Dadurch lässt sich ein Backend-Server vermeiden.

Aufgrund der obigen Überlegungen lässt sich feststellen, dass für die Umsetzung der Anforderungen kein Backend-Server nötig ist.

5.3 Architektur

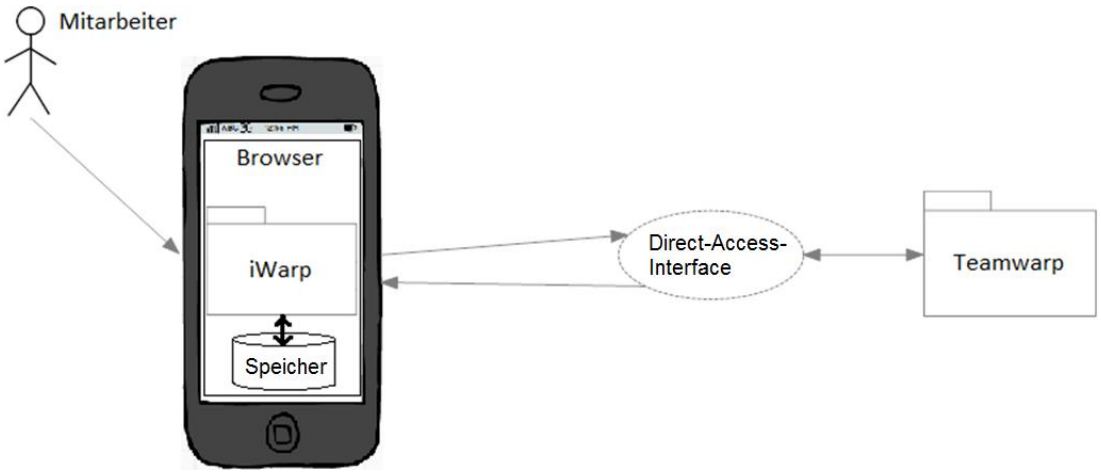

Abbildung 18: Bausteinsicht Level 0

In diesem Abschnitt wird die Architektur der Webanwendung behandelt. Wie im vorherigen Abschnitt 5.2 festgestellt wurde, soll die Webanwendung nur aus einem Frontend

bestehen, welches im Browser der Endgeräte der Benutzer ausgeführt wird und alle Funktionen bereitstellt (in Abbildung 18). Das Frontend kommuniziert mit Teamwarp durch das Direct-Access-Interface, um Daten von Teamwarp abzufragen und Buchungen an Teamwarp zu schicken. Die zu speichernden Daten werden im lokalen Speicher des Browsers abgelegt.
Im Folgenden werden die Komponenten des Frontends sowie der Zusammenhang zwischen den Komponenten beschrieben.

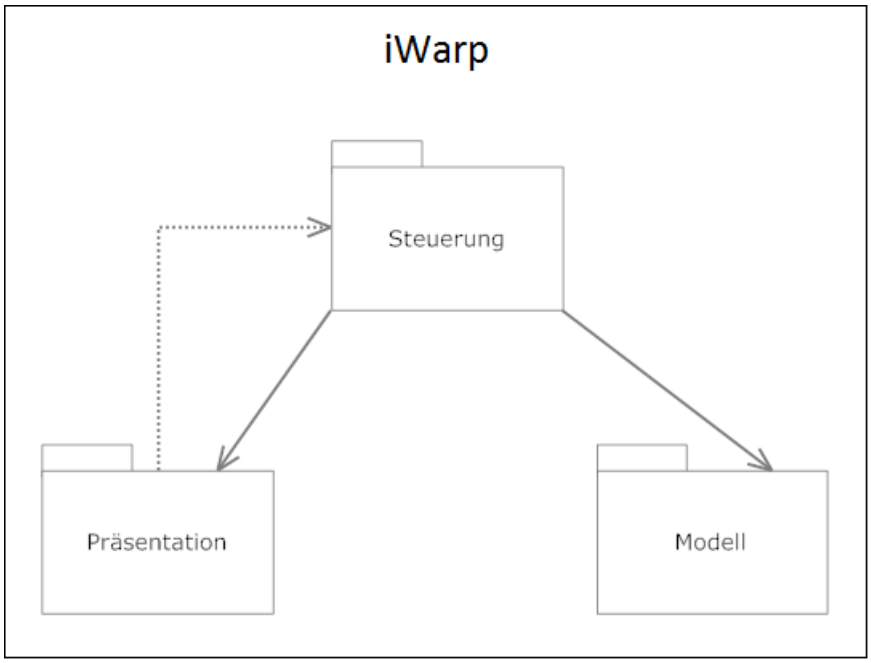

Abbildung 19: Bausteinsicht Level 1

Die Anwendung wird nach dem Entwurfsmuster MVC (Model-View-Controller) strukturiert (Abbildung 19).
MVC ist ein Muster zur Strukturierung von Software den drei Einheiten Datenmodell, Präsentation und Programmsteuerung. Ziel des Musters ist es, einen flexiblen Programmentwurf zu erstellen, der eine spätere Änderung oder Erweiterung erleichtert und die Wiederverwendbarkeit der einzelnen Komponenten ermöglicht. (Goll, et al., 2013)
Das Modell enthält die darzustellenden Daten und basiert auf dem fachlichen Datenmodell. Das Modell ist von der Steuerung und der Präsentation unabhängig.
Die Präsentation besteht aus einer oder mehreren HTML-Seiten. Sie stellt die statischen Teile der Benutzeroberfläche wie beispielweise die Seiten und die statischen Buttons bereit. Die Präsentation nimmt die Benutzeraktion entgegen und leitet diese mit Hilfe von „Beobachtern“ an die Steuerung weiter. „Beobachter“ („Action listener“) sind Codeteile, die auf Ereignisse wie zum Beispiel Seitenwechsel und das Antippen von Buttons warten.

Die Steuerung verwaltet die Präsentation und das Modell. Es ist die Aufgabe der Steuerung, sich in Kontakt mit Teamwarp zu setzen, Daten im Modell zu manipulieren und die dynamischen Teile in der Präsentation darzustellen. Die Steuerung entscheidet aufgrund der Benutzeraktion in der Präsentation, welche Daten im Modell geändert werden müssen. Außerdem verwaltet die Steuerung die Daten vom Benutzer und den Zustand der Anwendung.

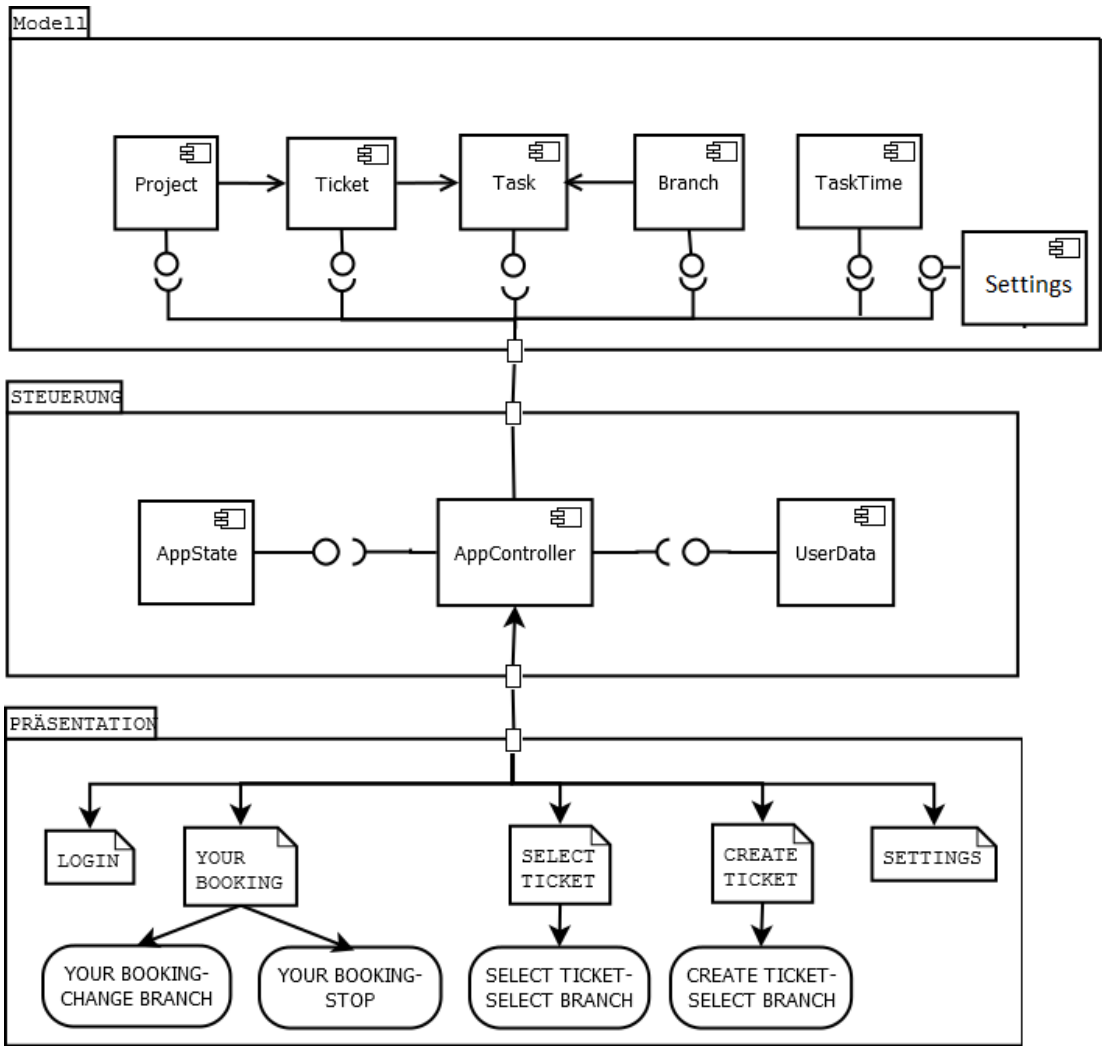

Abbildung 20: Bausteinsicht Level 2

Die Abbildung 20 stellt die einzelnen Komponenten der jeweiligen Bestandteile sowie die Zusammenhänge zwischen ihnen dar. Die Komponenten der Präsentation sind die HTML-Seiten, welche die Benutzeroberfläche darstellen. Diese Seiten werden von der Steuerung verwaltet. Sie enthalten „Beobachter“, die auf Benutzeraktionen warten und diese

Aktionen an die Steuerung weiterleiten. Das Modell bildet das fachliche Datenmodell und wurde mit einer zusätzlichen Komponente „TaskTime“ erweitert. Jede Komponente des Modells entspricht einer Klasse des fachlichen Datenmodells.
Die Komponente „Project“ beinhaltet alle zugewiesenen Projekte des Benutzers. Die Komponente unterscheidet auch zwischen darzustellenden und zu versteckenden Projekten. Die Komponenete „Branch“ enthält Informationen über die verfügbaren Buchungskontexte des Benutzers. Sie hält auch darzustellende und zu versteckende Kontexte auseinander. Ein Ticket besteht aus einem Projekt und zusätzlichen Informationen. Die Komponente „Ticket“ hat die Aufgaben, dem Benutzer Tickets vorzuschlagen, ausgewählte Tickets im Speicher abzulegen und neue Tickets zu erstellen. Ein Task besteht aus einem Ticket und einem „Branch“ (einem Kontext). Diese Komponente verwaltet die Liste der vom Benutzer erstellten Tasks. Die Komponente „TaskTime“ verwaltet die Arbeitszeiten von Tasks. Sie ist unabhängig von der Komponente „Task“. Jedes Element der Komponente „TaskTime“ ist mit der ID eines Tasks gebunden. Ein „TaskTime“-Element hat drei Attribute zu verwalten: die Startzeit des Tasks, die tatsächliche Arbeitszeit des Tasks und die auf den Task gebuchte Arbeitszeit. In der Komponente Settings wird die Ticketlebensdauer gespeichert.
Die Steuerung enthält drei Komponenten: „AppState“, „UserData“ und „AppController“. Die Komponente „AppState“ fasst die Informationen über den Zustand des Systems wie beispielweise die ID des laufenden Tasks zusammen. Die Komponente „UserData“ verwaltet Informationen über die Zugangsdaten vom Benutzer. „AppController“ ist die Hauptkomponente der Steuerung und verwaltet das gesamte System. Er nimmt die Benutzeraktionen wie zum Beispiel das Tippen auf einen Button oder die Eingaben des Benutzers aus der Präsentation entgegen und interpretiert diese. Der AppController übernimmt die Kommunikation mit Teamwarp. Eine weitere wichtige Aufgabe vom AppController ist, das Modell je nach Benutzeraktion zu manipulieren und die Präsentation entsprechend anzupassen. Ein Beispiel: Der Benutzer wählt einen Kontext für das ausgesuchte Ticket und die Präsentation leitet den Event an den AppController weiter. Der AppController fügt anschließend diesen Task in die Liste der Tasks in der Komponente Task hinzu und aktualisiert die darzustellende Tasksliste in der Präsentation.

5.4 Abläufe der Anwendung

In diesem Abschnitt werden mit Hilfe der Sequenzdiagramme die Abläufe der einzelnen Anwendungsfälle der Anwendung erläutert.

5.4.1 UC1: Sich aus System authentifizieren

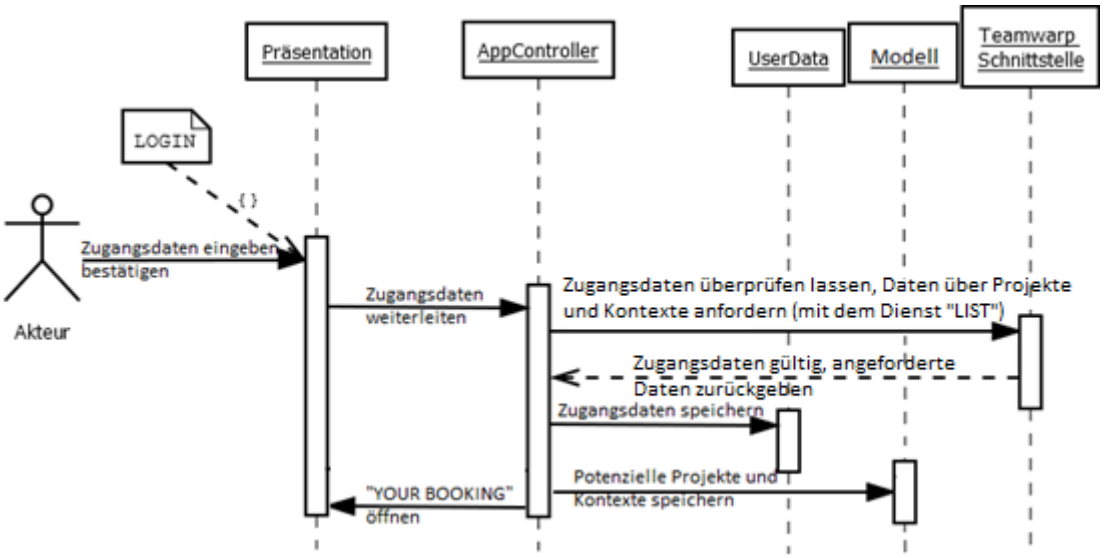

Abbildung 21: Sequenzdiagramm "Sich am System authentifizieren"

1. Auf der Seite LOGIN gibt der Benutzer seine Zugangsdaten ein und bestätigt.
2. Die Präsentation leitet die Zugangsdaten an den AppController weiter.
3. Der AppController schickt die Zugangsdaten an Teamwarp mit dem Befehl „LIST“ (siehe Kapitel 3.2), um die Zugangsdaten überprüfen zu lassen sowie die potenziellen Projekte und Kontexte anzufordern.
4. Wenn die Zugangsdaten gültig sind, gibt Teamwarp die angeforderten Daten zurück.
5. Der AppController speichert die Zugangsdaten in UserData.
6. Der AppController speichert die potenziellen Projekte und Kontexte im Projekt- und Kontextmodell.
7. Der AppController öffnet in der Präsentation die „YOUR BOOKING“-Seite.

5.4.2 UC2: Buchungsübersicht anzeigen

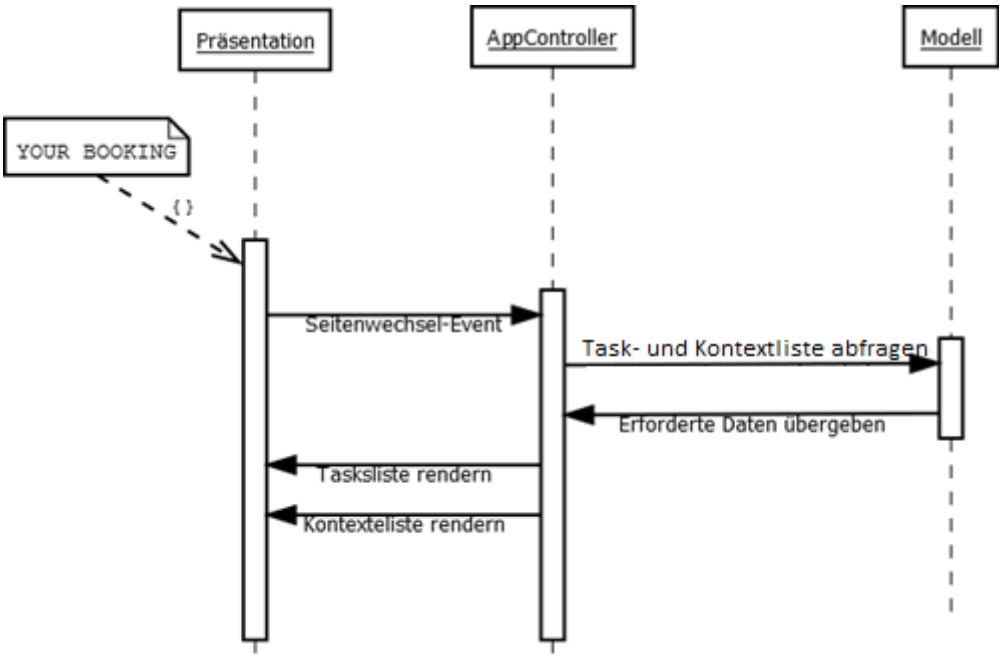

Abbildung 22: Sequenzdiagramm "Buchungsübersicht anzeigen

1. Wenn sich Benutzer eingeloggt ist wechselt das System zur Seite „YOUR BOOKING“, wird ein Seitenwechsel-Event an den AppController geschickt.
2. Der AppController ruft den aktuellen Task- und die Kontextliste für den Benutzer aus dem Task- und dem Kontextmodell ab.
3. Der AppController rendert die Taskliste in „YOUR BOOKING“.
4. Der AppController rendert die Kontextliste in „YOUR BOOKING-CHANGE BRANCH“.

5.4.3 UC3: Einen Task auswählen

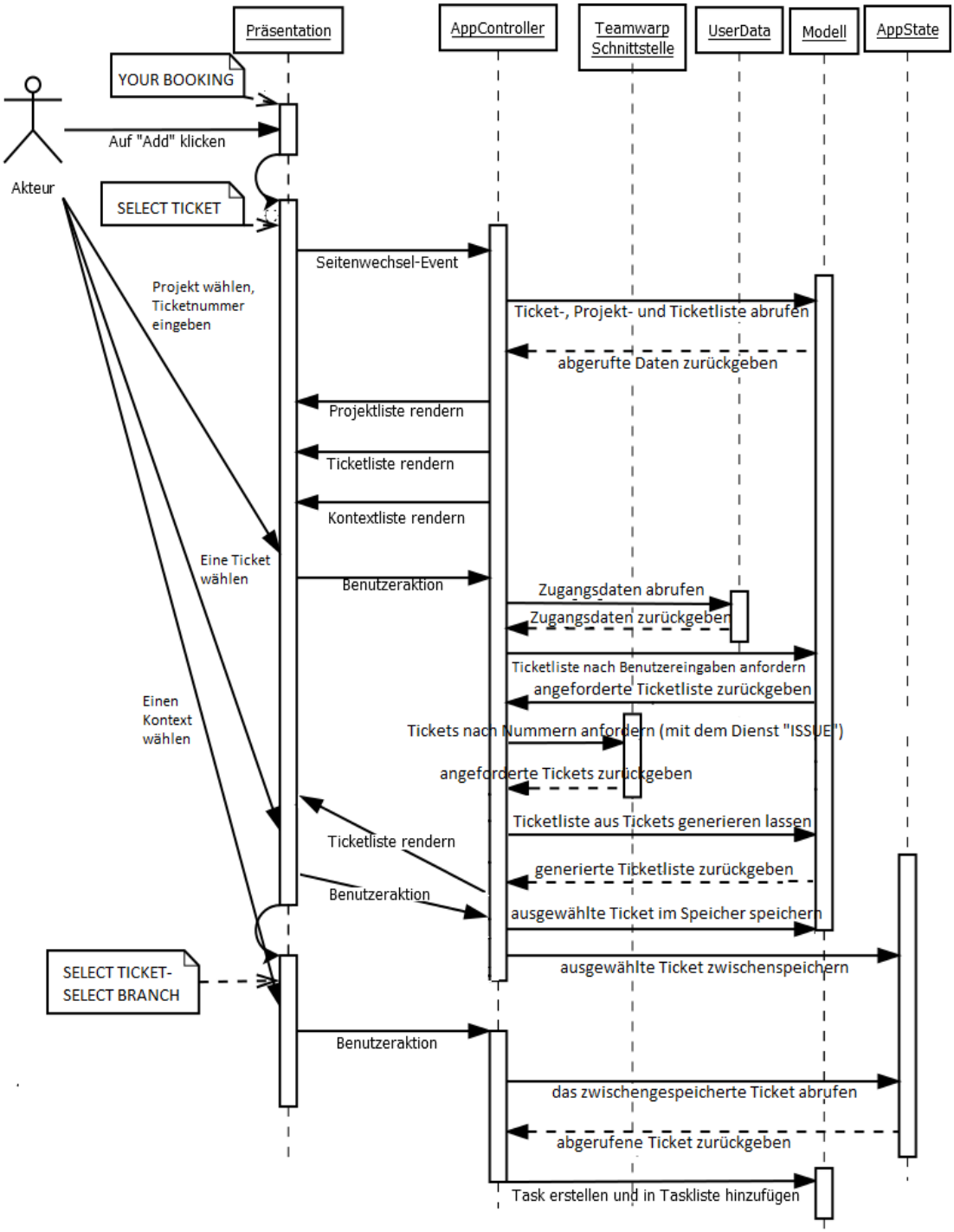

Abbildung 23: Sequenzdiagramm "Einen Task auswählen"

1. Wenn der Benutzer in „YOUR BOOKING“ auf „Add“ tippt, wechselt das System zu „SELECT TICKET“. Es wird dabei einen Seitenwechsel-Event an den AppController geschickt.
2. Der AppController fragt die aktuelle Projekt-, Ticket- und Kontextliste aus dem Projekt-, Ticket- und Kontextmodell ab.
3. Der AppController rendert die Auswahlliste der Projekte.
4. Der AppController rendert die Liste der Tickets, die vom Benutzer schon ausgewählt wurden.
5. Der AppController rendert die Kontextliste in „SELECT TICKET-SELECT BRANCH“.
6. Der Benutzer wählt ein anderes Projekt und/oder der Benutzer gibt eine Nummer ein. Die Benutzeraktionen werden an den AppController weitergeleitet.
7. Wenn die eingegebene Ticketnummer weniger als zwei Ziffern enthält:
    a. Der AppController fordert eine Ticketliste aus dem Ticketmodell nach der Benutzereingabe.
    b. Das Ticketmodell gibt eine Ticketliste, die der Benutzereingabe entspricht, zurück.
8. Wenn die eingegebene Ticketnummer zwei oder mehr Ziffern enthält:
    a. Der AppController ruft die Zugangsdaten von UserDate ab.
    b. UserData gibt die Zugangsdaten zurück.
    c. Der AppController schickt eine Anfrage, welche die Nummer in der Benutzereingabe enthält, mit dem Dienst „ISSUE“ an Teamwarp, um Tickets nach der Benutzereingabe anzufordern.
    d. Teamwarp gibt die angeforderten Tickets zurück.
    e. Der AppController leitet die zurückgegebenen Tickets an das Ticketmodell weiter, um eine Ticketliste generieren zu lassen.
    f. Das Ticketmodell gibt die generierte Ticketliste zurück.
9. Der AppController rendert die neue zurückgegebene Ticketliste in der Präsentation.
10. Der Benutzer wiederholt die Schritte ab Schritt 6 bis er das gewünschte Ticket findet.
11. Der Benutzer wählt ein Ticket. Die Benutzeraktion wird an den AppController weitergeleitet. Das System wechselt zu „SELECT TICKET-SELECT BRANCH“.
12. Der AppController speichert das ausgewählte Ticket im AppState zwischen.
13. Der Benutzer wählt einen Kontext aus.
14. Der AppController ruft das zwischengespeicherte Ticket aus dem AppState ab.
15. Der AppState gibt das abgerufene Ticket zurück.
16. Der AppController erstellt einen Task aus dem zurückgegebenen Ticket und dem ausgewählten Kontext im Taskmodell und fügt es der Taskliste hinzu.

5.4.4 UC4: Eine neuen Task erstellen

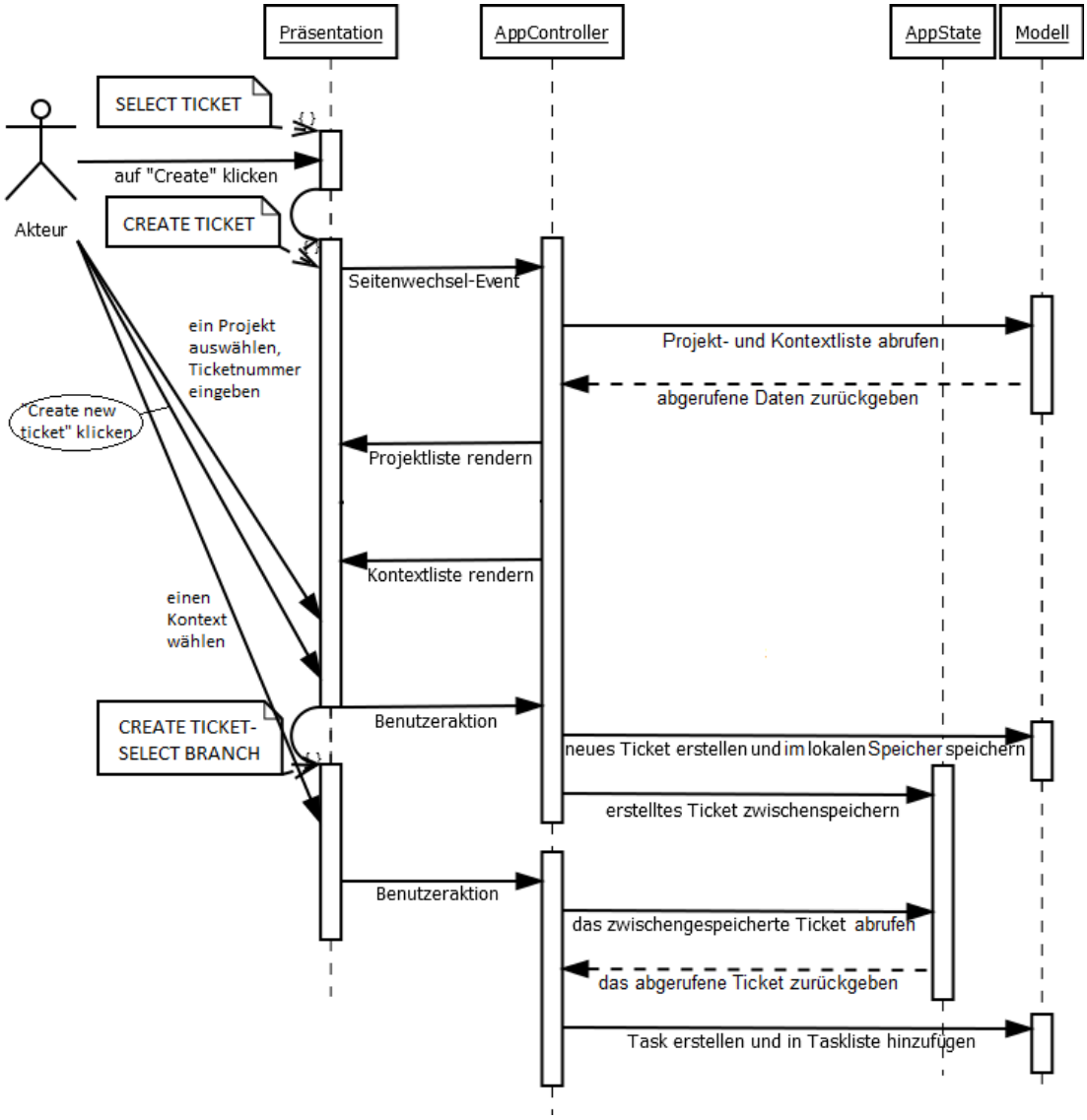

Abbildung 24: Sequenzdiagramm "Einen neuen Task erstellen“

1. Wenn der Benutzer in „SELECT TICKET“ auf „Create“ tippt, wechselt das System zu „CREATE TICKET“. Dabei wird der Seitenwechsel-Event an den AppController weitergeleitet.
2. Der AppController ruft die Projekt- und Kontextliste aus Projekt - und Kontextmodell ab.
3. Das Project- und das Kontextmodell geben die abgerufenen Daten zurück.
4. Der AppController rendert die Auswahlliste der Projekte.

5. Der AppController rendert die Kontextliste in „CREATE TICKET-SELECT BRANCH“.
6. Der Benutzer wählt ein Projekt aus und gibt die Ticketnummer ein.
7. Der Benutzer tippt auf „Create new ticket“. Die Benutzeraktion wird an den AppController weitergeleitet. Das System öffnet das Pop-up „CREATE TICKET-SELECT BRANCH“.
8. Der AppController speichert das erstellte Ticket im AppState zwischen.
9. Der Benutzer wählt einen Kontext aus. Die Benutzeraktion wird an den AppController weitergeleitet.
10. Der AppController ruft das zwischengespeicherte Ticket aus dem AppState ab.
11. Der AppState gibt das abgerufene Ticket zurück.
12. Der AppController erstellt einen Task aus dem zurückgegebenen Ticket und dem ausgewählten Kontext im Taskmodell und fügt ihn der Taskliste hinzu.

5.4.5 UC5: Das System konfigurieren

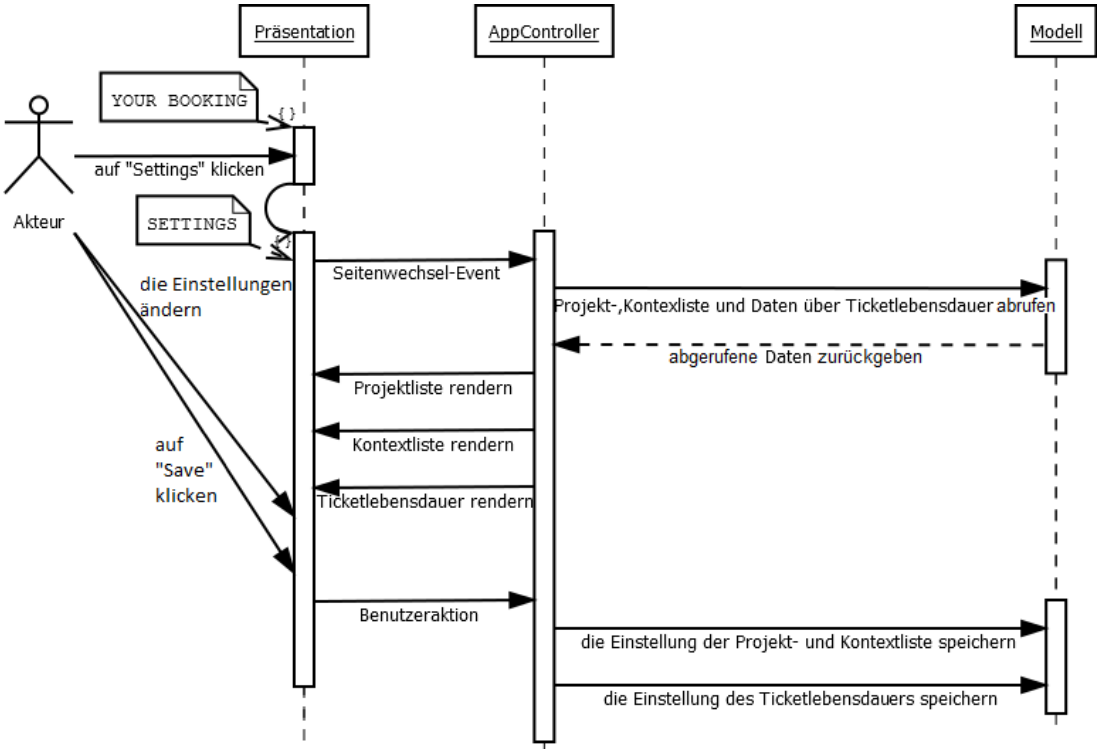

Abbildung 25: Sequenzdiagramm "Das System konfigurieren"

1. Wenn der Benutzer in „YOUR BOOKING“ auf „Settings“ tippt, wird das System zu „SETTINGS“ gewechselt. Dabei wird der Seitenwechsel-Event an den AppController weitergeleitet.
2. Der AppController ruft die Projekt-, und Kontextliste sowie die Ticketlebensdauer aus Projekt-, Kontext- und Einstellungsmodell ab.

3. Die Modelle geben die abgerufenen Daten zurück.
4. Der AppController rendert den Kontrollbox der Projekte.
5. Der AppController rendert den Kontrollbox der Kontexte.
6. Der AppController rendert die Radiogruppe der Ticketlebensdauer.
7. Der Benutzer ändert die Einstellungen.
8. Der Benutzer tippt auf „Save“. Die Benutzeraktion wird an den AppController weitergeleitet.
9. Der AppController manipuliert die Daten im Projekt- Kontext-, und Einstellungsmodell.

5.4.6 UC6: An einem Task arbeiten

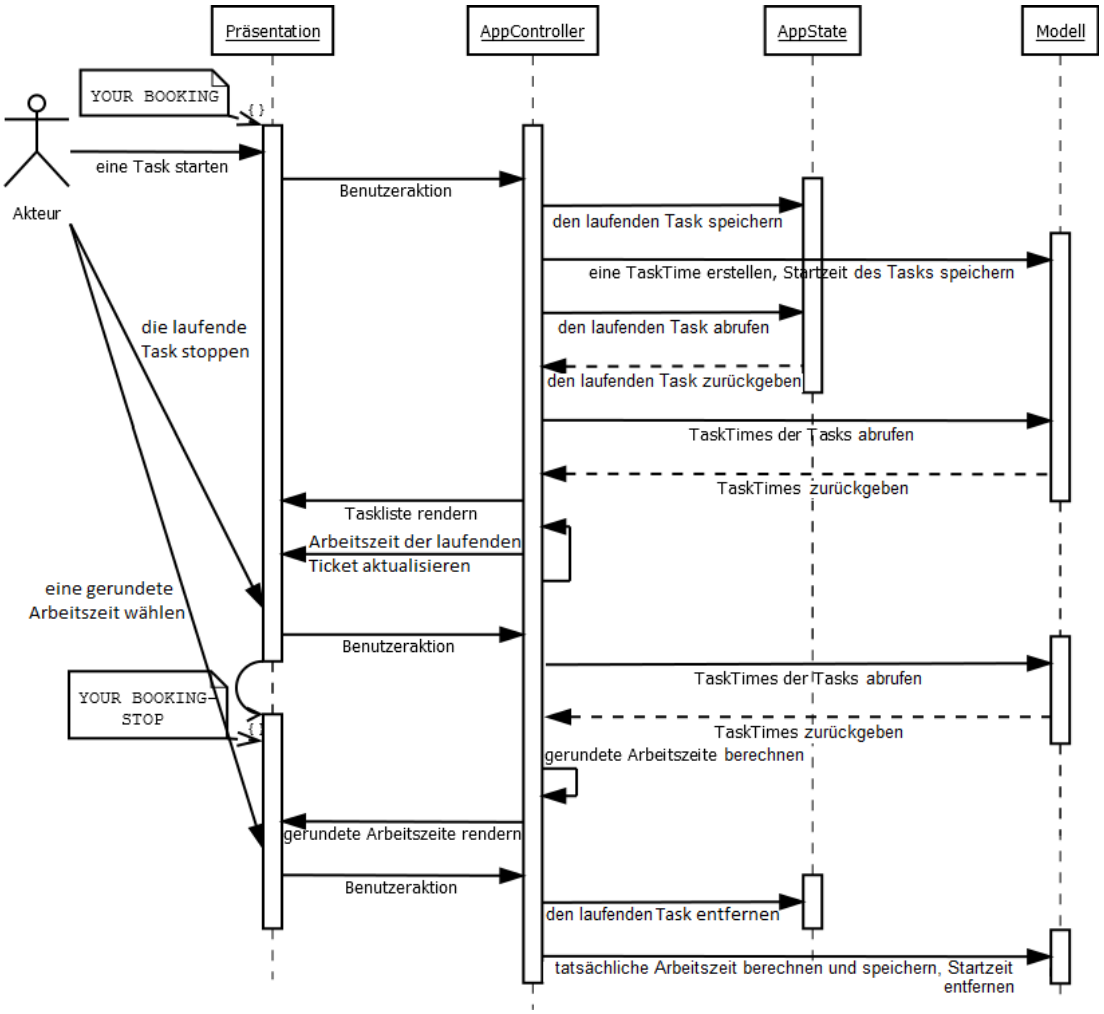

Abbildung 26: Sequenzdiagramm "An einem Task arbeiten"

1. Wenn der Benutzer in „YOUR BOOKING“ auf einen Task tippt, wird eine Benutzeraktion an den AppController weitergeleitet.
2. Der AppController speichert den Task, auf den der Benutzer getippt hat, als einen laufenden Task im AppState.
3. Der AppController erstellt eine TaskTime für den laufenden Task im TaskTime-Modell und speichert die Startzeit des laufenden Tasks in der TaskTime.
4. Der AppController ruft den laufenden Task aus dem AppState ab.
5. Der AppState gibt den laufenden Task zurück.
6. Der AppController ruft die TaskTimes aller ausgewählten Tasks aus dem TaskTime-Modell ab.
7. Das TaskTime-Modell gibt die abgerufenen TasksTimes zurück.
8. Der AppController rendert die Taskliste in „YOUR BOOKING“.
9. Der AppController aktualisiert stetig die aktuelle Arbeitszeit der laufenden Task.
10. Der Benutzer stoppt den laufenden Task, indem er den Task antippt. Das System öffnet das Pop-up „YOUR BOOKING-STOP“. Dabei wird die Benutzeraktion an den AppController weitergeleitet.
11. Der AppController ruft die TaskTime des laufenden Tasks aus dem TaskTime-Modell ab.
12. Der AppController berechnet die zwei gerundeten Arbeitszeiten des laufenden Tasks.
13. Der AppController rendert die zwei gerundeten Arbeitszeiten im Pop-up „YOUR BOOKING-STOP“.
14. Der Benutzer wählt eine gerundete Arbeitszeit aus. Die Benutzeraktion wird an den AppController weitergeleitet.
15. Der AppController entfernt den laufenden Task im AppState.
16. Der AppController berechnet die tatsächliche Arbeitszeit des gestoppten Tasks, speichert sie im TaskTime-Modell und entfernt die Startzeit in der TaskTime des gestoppten Tasks.

5.4.7 UC7: Buchung schicken

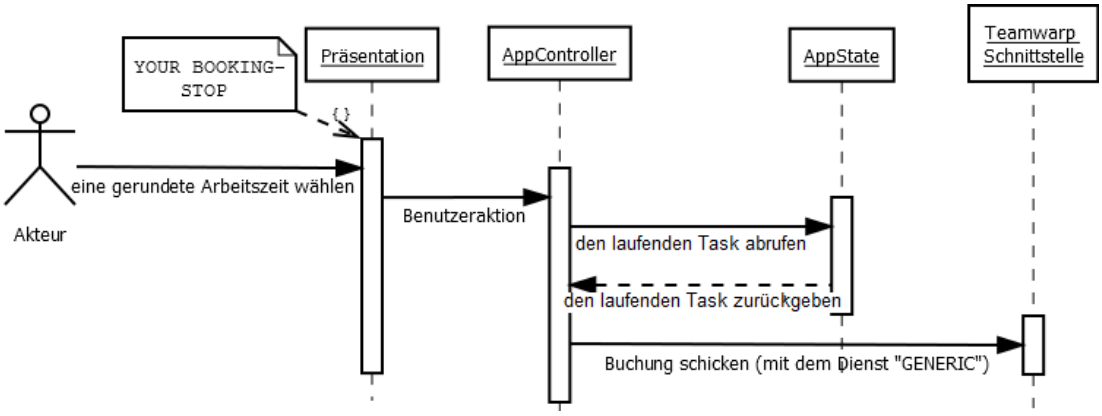

Abbildung 27: Sequenzdiagramm "Buchung schicken"

1. Wenn der Benutzer in „YOUR BOOKING-STOP“ eine gerundete Arbeitszeit auswählt, wird die Benutzeraktion zusammen mit der ausgewählte Arbeitszeit an den AppController weitergeleitet.
2. Der AppController ruft den laufenden Task im AppState ab.
3. Der AppController bereitet eine Anfrage auf Basis des Tasks und der Arbeitszeit vor. Er schickt diese Buchungsanfrage an die Teamwarp-Schnittstelle mit dem Dienst „GENERIC“.

6 Implementierung

In diesem Kapitel wird die Implementierung der einzelnen Bestandteile der Anwendung vorgestellt Wie im Abschnitt 5.1 dargelegt wurde, soll iWarp mit Hilfe der weborientierten Technologien HTML5, CSS und Javascript und den Frameworks jQuery und jQueryMobile entwickelt werden. Der erste Abschnitt 6.1 des Kapitels behandelt die Benutzeroberfläche. Es wird erläutert, wie die Benutzeroberfläche grundsätzlich mit Hilfe von jQueryMobile realisiert wird. Dafür wird der Aufbau der Seite „LOGIN“ als Beispiel beschrieben. Im zweiten Kapitel 6.2 wird die Umsetzung des Modells vorgestellt. Als ein Beispiel wird das Kontextmodell ausführlich beschrieben. Die anderen Komponenten des Modells werden nur grob vorgestellt, indem die Schnittstellen der einzelnen Komponenten ausgeführt werden. Das letzte Kapitel 6.3 behandelt die Steuerung. Im diesem Kapitel wird die Implementierung der drei Komponenten der Steuerung „UserData“, „AppState“ und „AppController“ verdeutlicht. Die zwei Nebenkomponenten „UserData“ und „AppState“ werden vorgestellt, indem ihre Inhalte und Schnittstellen beschrieben werden. Am Ende des Kapitels wird die Hauptkomponente der Steuerung-der „AppController“ vorgestellt, wobei erläutert wird, wie dieser die Anwendung verwaltet. Die Hauptaufgabe vom „AppController“ ist, auf Events zu erwarten und darauf zu reagieren. Wie der „AppController“ diese Aufgaben löst, wird im Kapitel ausführlich beschrieben. Darüber hinaus werden in diesem Abschnitt einige wichtige Probleme der Implementierung und ihre Lösungen vorgestellt.

6.1 Benutzeroberfläche mit jQueryMobile

Basierend auf den in Kapitel 4.4 vorgestellten Mockups wird die Benutzeroberfläche in HTML5 und CSS mit Hilfe von jQueryMobile realisiert.
jQueryMobile stellt eine Bibliothek von Webelementen, die auf HTML5 basieren und spezifisch für die Bedienung auf dem Touchscreen eines mobilen Geräts gestaltet sind, zur Verfügung. Darüber hinaus bietet jQueryMobile die Möglichkeit, Benutzeroberflächen für mobile Geräte aller gängigen Plattformen (iOS, Android, Blackberry, WebOS, Symbian, Window Phone usw.) zu erstellen.

In diesem Kapitel wird beschrieben, wie eine HTML-Seite grundsätzlich mithilfe von jQueryMobile aufgebaut werden kann. Es wird der Code der Seite „LOGIN“ als Beispiel vorgestellt. Um zu verdeutlichen, wie die Benutzeroberfläche der Anwendung mit jQueryMobile aufgebaut ist, wird der Code der Seite „LOGIN“ ausführlich vorgestellt.
Eine HTML-Seite besteht aus einem oder mehreren HTML-Elementen. Die meisten dieser Elemente werden durch ein Tag-Paar markiert, und zwar durch ein Starttag und ein Endtag. Ein Starttag beginnt immer mit einem „<“ Zeichen, es folgt der Elementname und optional eine Liste seiner Attribute. Das Starttag wird mit einem „>“ Zeichen geschlossen. Das Endtag besteht aus dem „</“, dem Elementname und dem abschließende „>“. Ein Element kann andere Elemente in sich beinhalten, diese Subelemente befinden sich innerhalb des Tag-Paars des Oberelements.
Ein jQueryMobile-Element wird durch ein Element mit dem speziellen Attribut „data-role“ definiert. Dieses Attribut legt fest, von welchem Typ das jQueryMobile-Element ist.

```
<div id="login" data-role="page">
</div>
```

Abbildung 28: Quelltext "jQueryMobile Element"

Der Code in Abbildung 28 stellt ein Beispiel für ein jQueryMobile-Element dar und erzeugt ein div-Element mit dem Attribut `data-role="page"`. Dieses Attribut legt fest, dass das div-Element eine Seite (page) einer Webanwendung darstellt.
Im Folgenden wird der ganze Codeblock der Seite „LOGIN“ ausgeführt.

```
<div id="login" data-role="page">
      <div data-role="header" data-theme="d" data-
position="fixed" data-tap-toggle="false">
            <h1>iWarp</h1>
      </div>
      <div data-role="content">
            <div id="login-form" data-role="fieldcontain"
align="center">
                  <input type="text" name="name" id="username"
value="" placeholder="Username"/>
                  <input type="password" name="password"
id="password" value="" placeholder="Password"/>
            </div>
            <a data-role="button" id="login-button" data-
theme="b">LOGIN</a>
      </div>
</div>
```

Abbildung 29: Quelltext "LOGIN" Seite-Teil 1

In Abbildung 29 sind die Hauptcodezeilen der Seite „LOGIN“ dargestellt. Das oberste div-Element hat die ID „login“ und den Typ „page“. Eine Seite kann außer dem

Hauptinhaltsbereich (mit dem Typ „content“) eine Kopfzeile (Typ „header“) und eine Fußzeile (Typ „footer“) haben.
Die Seite „LOGIN“ verfügt über eine Kopfzeile, deren Position mithilfe des Attributs `data-position="fixed"` fixiert wird, auch wenn der Benutzer die Seite weiter nach unten scrollt. Das Attribut `data-tap-toggle="false"` sorgt dafür, dass sich die Kopfzeile nicht ein- und ausblendet wenn der Benutzer auf den Bildschirm tippt. `data-theme="d"` legt die Farbe der Kopfzeile fest. Die Kopfzeile enthält die Überschrift „iWarp“.
Innerhalb des Tag-Paars `<div data-role="content">…</div>` befindet sich mit dem Login-Formular der Hauptinhalt der Seite. Das Formular wird durch das div-Element des Typs „fieldcontain“ definiert, damit der Inhalt des Elements innerhalb eines Containers gekapselt wird. `align="center"` sorgt dafür, dass das Formular in der Mitte der Seite steht. Die zwei input-Elemente erzeugen zwei Eingabefelder für den Benutzername und das Passwort. Das a-Element stellt mit Hilfe des Attributs `data-role="button"` den LOGIN-Knopf dar.
Die Seite „LOGIN“ soll Fehlermeldungen zeigen, wenn der Log-in-Vorgang wegen fehlenden oder falschen Zugangsdaten nicht erfolgreich ist. Diese Fehlermeldungen sollen in der Form eines Pop-ups erscheinen. Wenn ein Pop-up in einer Seite auftreten soll, muss es innerhalb des div-Elements der Seite definiert werden.

```
<div id="login" data-role="page">
     <div id="alert-incorrect-login" data-role="popup"
data-overlay-theme="a">
          <div data-role="header">
               <h1>Attention</h1>
          </div>
          <div data-role="content">
               <h3 align="center">Incorrect username or
password</h3>
               <a data-rel="back" data-role="button"
data-theme="b">CANCEL</a>
          </div>
     </div>
     <div id="alert-no-login" data-role="popup" data-
overlay-theme="a">
          <div data-role="header">
               <h1>Attention</h1>
          </div>
          <div data-role="content">
               <h3 align="center">Invalid username or
password</h3>
               <a data-rel="back" data-role="button"
data-theme="b">CANCEL</a>
          </div>
     </div>
</div>
```

Abbildung 30: Quelltext "LOGIN" Seite-Teil 2

In Abbildung 30 werden die Pop-ups in div-Elementen mit `data-role="popup"` definiert. Das Attribut `data-overlay-theme="a"` bestimmt auf welche Weise ein Pop-up erscheint. Beide Pop-ups haben jeweils eine Kopfzeile „Attention“. Der Hauptinhalt jedes Pop-ups stellt die Fehlermeldung zusammen mit einem „CANCEL“-Knopf dar. Das Attribut `data-rel="back"` bestimmt, dass wenn ein Knopf angetippt wird, das System zum vorherigen Zustand zurückkommen soll. In diesem Fall wird das Pop-up ausgeblendet und das Login-Formular wird wieder verfügbar.
Die komplette Seite „LOGIN“ ist in einem div-Tag-Paar mit dem Attribut `data-role="page"` enthalten. Die anderen Seiten sind in ähnlicher Weise aufgebaut. Alle Seiten der Anwendung sind in einer einzigen HTML-Datei „index.html“ definiert.
Wie in der Bausteinsicht Level 1 in Abbildung 19 vorgestellt, soll die Präsentation weder die Steuerung noch das Datenmodell kennen. Das bedeutet, dass die Datei „index.html“ nur die statischen Teile der Benutzeroberfläche erstellen soll, die vor dem Systemstart schon vorhanden sind und sich nicht auf das Datenmodell beziehen. Die dynamischen Teile der Benutzeroberfläche wie zum Beispiel die Ticketliste oder die Taskliste werden von der Steuerung generiert.
Im Folgenden sind die einzelnen Seiten der Anwendung, welche in „index.html“ definiert sind, dargestellt.

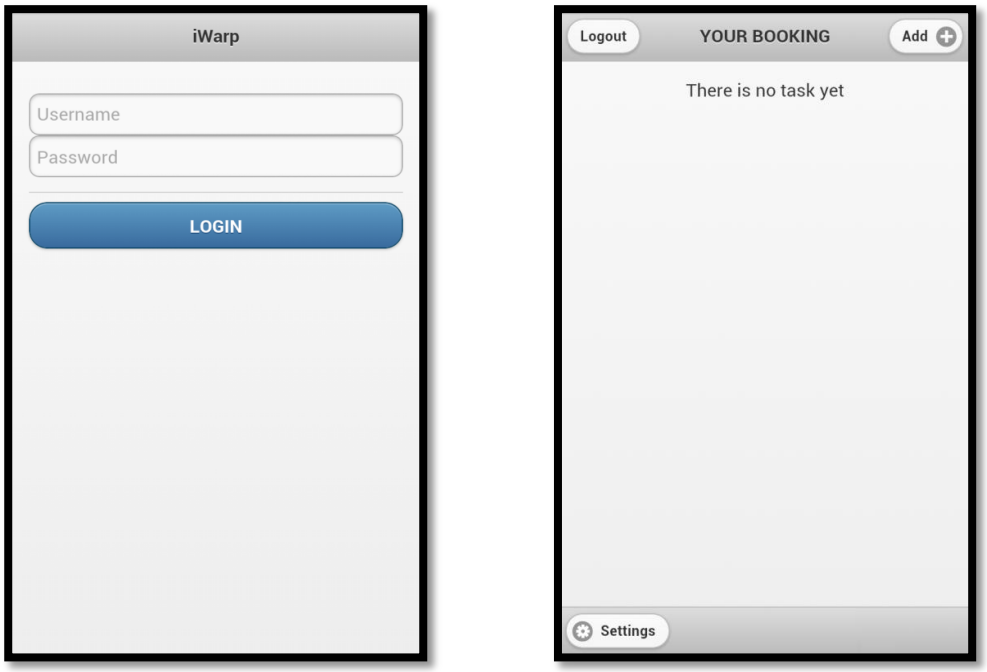

Abbildung 31: Screenshot "LOGIN" (links), Screenshot "YOUR BOOKING" (rechts)

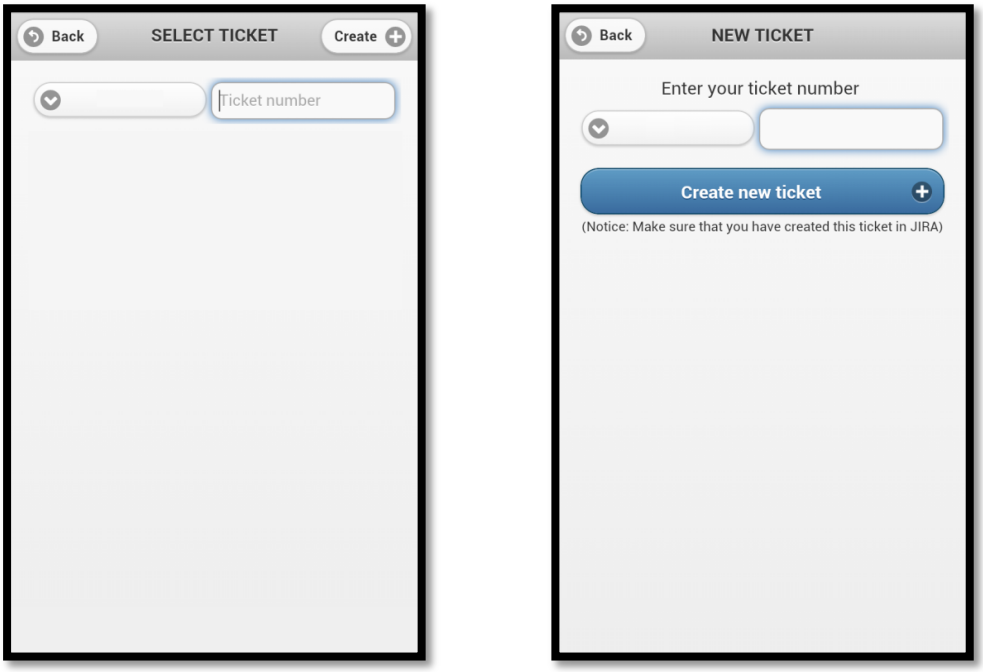

Abbildung 32: Screenshot "SELECT TICKET" (links), Screenshot "CREATE TICKET" (rechts)

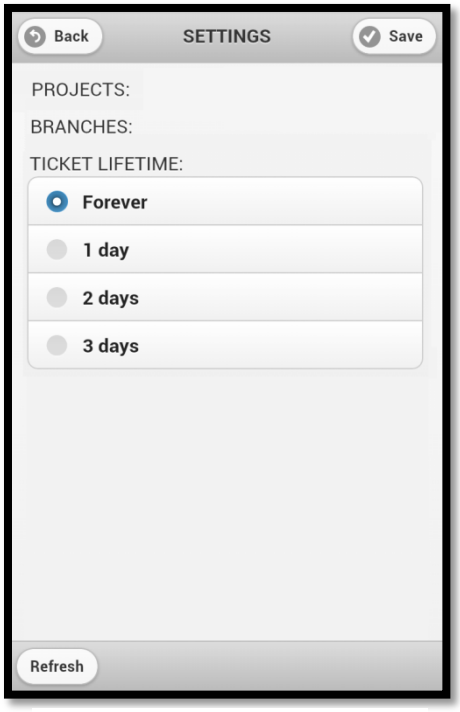

Abbildung 33: Screenshot "SETTINGS"

In Abbildung 31, Abbildung 32 und Abbildung 33 wurden in Reihenfolge die Screenshots von den Seiten „LOGIN“, „YOUR BOOKING“, „SELECT TICKET“, „CREATE TICKET“ und „SETTINGS“ dargestellt. Diese Screenshots wurden auf einem Samsung GALAXY S3 mit dem Browser Google Chrome gemacht.

6.2 Datenmodell

In diesem Kapitel wird vorgestellt, wie das Modell allgemein aufgebaut ist. Es wird zuerst dargelegt, wie das Modell strukturiert wird. Darüber hinaus werden die einzelnen Komponenten des Modells vorgestellt, indem der interne Aufbau sowie die Schnittstelle jeder Komponente erläutert werden.

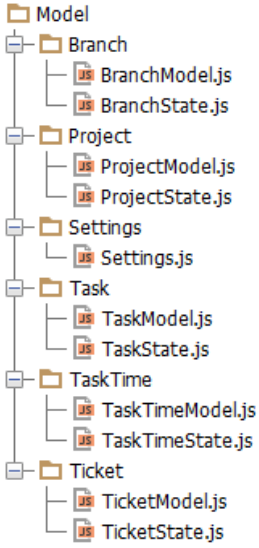

Abbildung 34: Strukturierung des Datenmodells

Nach der Bausteinsicht Level 2 in Abbildung 20 wird das Datenmodell in verschiedene Komponenten aufgeteilt (in Abbildung 34). Jede Komponente (außer „Settings“) verfügt über zwei Teile „-Model“ und „-State“.
Die „-Model“ Dateien haben keinen Zustand und enthalten nur Funktionen zur Erstellung von Objekten. Die „-Model“ Dateien basieren auf dem fachliche Datenmodell und sind in Javascript geschrieben. Im Folgenden wird der Quelltext von „BranchModel.js“ als Beispiel vorgestellt.

```
var Branch = Branch || {};
Branch.BranchModel = function(name, id){
    this.name = name;
    this.id = id;
    this.isMarked = true;
};
```

Abbildung 35: Quelltext "BranchModel.js"

Der Quelltext in Abbildung 35 erstellt eine Klasse „`Branch.BranchModel`“. Es können Objekte dieser Klasse erzeugt werden. Jedes Objekt „`BranchModel`“ verfügt über drei Eigenschaften:

- „`name`“: Der Name des Kontexts.
- „`id`“: Die generische ID des Kontexts, unter dem eine Buchung erfolgen soll (siehe 3.2).
- „`isMarked`“: Bestimmt, ob ein Kontext in Pop-ups der Kontexte verfügbar sein soll (siehe Kapitel 4.4.3, 4.4.6 oder 4.4.8).

Die „-State“ Dateien verwalten den Zustand der Komponenten.

```
var Branch = Branch || {};
Branch.BranchState = (function () {
    var branchesList = [],
            branchesListStorageKeyWithoutPrefix =
'iwarp.local.branchesList',
            branchesListStorageKey;

    var setKeysPrefix = function (prefix) {
        branchesListStorageKey = prefix + '.' +
branchesListStorageKeyWithoutPrefix;
    };
    var unSetKeysPrefix = function () {
        branchesListStorageKey = "";
    };
    var saveBranchToLocalStorage = function () {
        $.jStorage.set(branchesListStorageKey,
branchesList);
    };
    var loadBranchesFromLocalStorage = function () {
        var storedBranches =
$.jStorage.get(branchesListStorageKey);
        if (storedBranches!== null) {
            branchesList = storedBranches;
        }
    };
})();
```

Abbildung 36: Quelltext "BranchState.js"-Teil 1

Der Quellcode in Abbildung 36 stellt den ersten Teil von „`BranchState.js`“ dar. Es wird hier eine Klasse `Branch.BranchState` in Form einer Javascript-Funktion definiert. Innerhalb der Anwendung existiert immer nur eine Instanz dieser Klasse. Diese Instanz enthält eine Liste der Kontexte („`branchesList`“) die dem Benutzer zugewiesen sind.

Außerdem verfügt sie über zwei Schlüssel, einen mit und einen ohne Präfix, um die Kontextliste im Speicher zu speichern. Mit `setKeysPrefix` und `unSetKeysPrefix` kann BranchState ein Präfix an den Anfang der Schlüssel hinzufügen bzw. das Präfix am Anfang der Schlüssel entfernen. Mit `saveBranchToLocalStorage` und `loadBranchesFromLocalStorage` kann BranchState die Kontextliste im Speicher speichern bzw. vom Speicher laden.
Die Klasse Branch.BranchState definiert noch weitere Funktionen, die aber im Detail nicht ausgeführt werden.

```
var Branch = Branch || {};
Branch.BranchState = (function () {
     return {
        addBranch: addBranch,
        getBranchesList: getBranchesList,
        getBranchByName: getBranchByName,
        unMarkABranchByName: unMarkABranchByName,
        markAllBranches: markAllBranches,
        getMarkedBranchesList: getMarkedBranchesList,
        isThereOneMarkedBranch: isThereOneMarkedBranch,
        setKeysPrefix: setKeyPrefix,
        unSetKeysPrefix: unSetKeyPrefix,
        hasNoBranch: hasNoBranch,
        hasThisBranchId: hasThisBranchId
    }
})();
```

Abbildung 37: Quelltext "BranchState.js"-Teil 2

Der Quelltext in Abbildung 37 stellt die Schnittstelle der Kontextkomponente dar. `Branch.BranchState` wird als eine Funktion definiert und am Ende sofort aufgerufen, damit nur eine Instanz von `Branch.BranchState` in der Anwendung vorhanden ist und die angebotenen Funktionen aufgerufen werden können. Außer den vorgestellten stehen folgende Funktionen zur Verfügung:

- `addBranch`: Einen Kontext in die Kontextliste hinzufügen.
- `getBranchesList`: Die Kontextliste abrufen.
- `getBranchByName`: Einen Kontext durch seinen Name abrufen.
- `unMarkABranchByName`: Die Markierung eines Kontexts löschen, der im Pop-ups der Kontexte verfügbar sein soll.
- `markAllBranches`: Alle Kontexte markieren, damit sie im Pop-up der Kontexte verfügbar sind.
- `getMarkedBranchesList`: Die Liste der markierten Kontexte abrufen.
- `isThereOneMarkedBranch`: Abfragen, ob es genau einen markierten Kontext gibt.
- `hasNoBranch`: Abfragen, ob in der Kontextliste kein Kontext vorhanden ist.

- `hasThisBranchId`: Abfragen, ob die Kontextliste den Kontext mit der gegebenen ID enthält.

Die anderen Modellkomponenten sind in ähnlicher Form aufgebaut. Im Folgenden wird der Inhalt sowie die Schnittstellen der einzelnen Komponenten im Allgemeinen vorgestellt.

PROJECT verfügt über eine Projektliste. Die Schnittstelle der Projektkomponente enthält folgende Funktionen:

- `addProject`: Ein Projekt in die Projektliste hinzufügen.
- `getProjectsList`: Die Projektliste abfragen.
- `getProjectByName`: Ein Projekt durch seinen Name abrufen.
- `unMarkAProjectByName`: Die Markierung eines Projekts löschen, das in der Wahlliste der Projekte verfügbar sein soll.
- `markAllProjects`: Alle Projekte markieren, damit sie in der Auswahlliste der Projekte verfügbar sind.
- `getMarkedProjectsList`: Die Liste der markierten Projekte abrufen.
- `hasNoProject`: Abfragen, ob in der Projektliste kein Projekt vorhanden ist.
- `hasThisProject`: Abfragen, ob die Projektliste das Projekt mit der gegebenen ID enthält.

TICKET verfügt über zwei Ticketlisten. Eine Liste enthält die Tickets, die im lokalen Speicher gespeichert sind. Die andere Liste enthält die Tickets, die in der Anwendung darzustellen sind. Die Schnittstelle der Ticketkomponente bietet folgende Funktionen an:

- `addTicket`: Ein Ticket zur Liste der gespeicherter Tickets hinzufügen.
- `getTicketsListFromStorage`: Die Ticketliste vom lokalen Speicher mit dem übergebenen Projekt und der übergebenen Ticketnummer abfragen.
- `generateTicketsList`: Eine Ticketliste aus einer Reihe von Angaben generieren.
- `getTicketById`: Ein Ticket mit der übergebenen ID abfragen.
- `getStoredTicketsList`: Die Liste der gespeicherten Tickets abrufen.
- `createTicket`: Ein neues Ticket mit den übergebenen Angaben erstellen.
- `removeOldTicket`: Tickets im lokalen Speicher löschen, deren Lebensdauer die Ticketlebensdauer in der Anwendung überschritten hat.

TASK verfügt über eine Taskliste. Die Schnittstelle der Taskkomponente enthält folgende Funktionen:

- `getTasksList`: Die Taskliste abfragen.
- `createTask`: Einen Task aus dem übergebenen Ticket und dem übergebenen Kontext erstellen.
- `addTask`: Einen Task in die Taskliste hinzufügen. Wenn mindesten ein Task der Liste den gleichen Kontext hat wie der hinzuzufügende Task, wird der Task an den Anfang der Liste eingefügt. Ansonsten wird der Task ans Ende der Liste eingefügt.

- `removeTaskByIndex`: Einen Task mit dem gegebenen Index aus der Taskliste entfernen.
- `removeTaskByTaskId`: Einen Task mit der gegebenen ID aus der Taskliste entfernen.
- `getTaskByTaskId`: Einen Task mit der gegebenen ID aus der Taskliste abrufen.
- `setTicket`: Einem Task mit dem gegebenen Index ein neues Ticket zuordnen.

TASKTIME verwaltet die Arbeitszeiten der einzelnen Tasks anhand der Task-ID. Tasktime verfügt über eine HashMap „`taskTimeMap`“, die in Form eines Javascript-Objekts erstellt ist. Die Schlüssel der `taskTimeMap` sind die IDs der einzelnen Tasks. Der Wert eines Schlüssels ist ein Objekt der Klasse „`TaskTimeModel`“, welches die Startzeit, die tatsächliche Arbeitszeit und die gebuchte Zeit des entsprechenden Tasks enthält. Ein solches Schüssel-Wert-Paar bildet eine einzelne TaskTime. Folgende ist der Code von „`TaskTimeModel`.

```
var Time = Time || {};
Time.TimeModel = function(start, duration, booked){
    this.start = start;
    this.duration = duration;
    this.booked = booked;
}
```

Abbildung 38: Quelltext "TaskTimeModel.js"

Das `TaskTimeModel`-Objekt besteht wie im Quelltext in Abbildung 38 aus seinem Erstellungszeitpunkt (als `start`-Startzeit) und den Arbeitszeiten (als `duration`-tatsächliche Arbeitszeit und `booked`-gebuchte Arbeitszeit).
Die Schnittstelle der TaskTime-Komponente bietet folgende Funktionen an:

- `setTaskTime`: Ein neues Schlüssel-Wert-Paar aus der gegebenen Task-ID und einem `TaskTimeModel`-Objekts zu `taskTimeMap` hinzufügen.
- `getTaskTimeMap`: Die `taskTimeMap` abrufen.
- `getTaskTimeByTaskId`: Das TaskTimeModel-Objekt der gegebenen Task-ID abrufen.
- `stopTaskTime`: Einen TaskTime stoppen, indem die tatsächliche Arbeitszeit (`duration`) berechnet und die Startzeit (`start`) auf null gesetzt wird.
- `continueTaskTime`: Einen TaskTime mit der gegebenen Task-ID fortsetzen, indem die Startzeit (`start`) des TaskTimeModel-Objekts auf den Zeitpunkt des Funktionsaufrufs gesetzt wird.
- `setBookedTime`: Die gebuchte Zeit von einer TaskTime mit der gegebenen Task-ID auf die übergeben Zeit setzen.
- `addBookedTime`: Die gebuchte Zeit von einer TaskTime mit der gegebenen Task-ID auf die übergebene Zeit addieren.

- `addDurationOfTaskTime`: Die tatsächliche Arbeitszeit von einem TaskTime mit der gegebenen Task-ID mit der gegebenen Zeit addieren.
- `removeTaskTimeByTaskId`: Die TaskTime mit der gegebenen Task-ID aus dem `taskTimeMap` löschen.

SETTINGS hat keine „-Model"-Klasse und damit auch keine Objekte. Die Einstellungskomponente verfügt nur über eine Variable `ticketLifetime`, welche die Ticketlebensdauer festlegt. Die Schnittstelle dieser Komponente enthält folgende Funktionen:

- `setTicketLifetime`: Die Ticketlebensdauer auf den gegebenen Wert setzen.
- `getTicketLifetime`: Die Ticketlebensdauer abrufen.

6.3 Steuerung

In diesem Abschnitt geht es um den Aufbau der Steuerung, dem wichtigsten Bestandteil der Anwendung. In der Bausteinsicht Level 1 in Abbildung 19 ist die Steuerung vom Modell und der Benutzeroberfläche abhängig. Zum einen hat die Steuerung Zugriff auf das Modell um Daten der einzelnen Komponenten des Modells zu benutzen und zu manipulieren. Zum anderen ist die Steuerung in der Lage, die Präsentation je nach Bedarf anzupassen. Außerdem hat die Steuerung Zugriff auf die sogenannten „Beobachter", welche in die Benutzeroberfläche hinzugefügt werden. Die „Beobachter" warten auf Benutzeraktionen und leiten diese an die Steuerung weiter.
Wie in Abbildung 20 Kapitel 5.3 verfügt die Steuerung über eine primäre Komponente (AppController) und zwei sekundäre Komponenten (AppState und UserData). Während UserData die Zugangsdaten des Benutzers speichert, verwaltet AppState die Daten über den aktuellen Zustand der Anwendung. Im Folgenden werden diese zwei Steuerungskomponenten vorgestellt.

Die Komponente UserData ist in der Datei „UserData.js" definiert. UserData verfügt über keine Instanz, sondern wird in Form einer Javascript-Funktion definiert. In UserData werden Benutzername und Passwort des Benutzers zur Kommunikation mit der Schnittstelle von Teamwarp gespeichert. Die Zugangsdaten werden nicht im lokalen Speicher gesichert, sondern im sogenannten „Session Storage". Im Gegensatz zum lokalen Speicher wird der Session-Storage nach jeder Sitzung des Browsers geleert. Das bedeutet, dass die Zugangsdaten immer erneut eingegeben werden müssen, wenn der Browser geschlossen und wieder geöffnet wird. Dadurch kann der Missbrauch der Anwendung durch Fremde vermieden werden. Darüber hinaus wird das Passwort mit rc4 (siehe Kapitel Bibliothek 5.1.3) verschlüsselt, bevor es im Session-Storage gespeichert wird, damit Fremde das Passwort des Benutzers nicht einfach auslesen können.

Die Schnittstelle vom „UserData" stellt folgende Funktionen zur Verfügung:

- `setUsernameAndPassword`: Das gegebene Passwort zur Sicherheit verschlüsseln. Der gegebene Benutzername und das verschlüsselte Passwort im Session-Storage speichern.
- `getUsername`: Den Benutzername abrufen.
- `getPassword`: Das Passwort entschlüsseln und abrufen.
- `removeUsernameAndPassword`: Benutzername und Passwort aus dem Session-Storage entfernen.

Die AppState-Komponente ist in einer Datei „AppState.js“ zusammengefasst. AppState wird ebenfalls in Form einer Javascript-Funktion definiert und hat keine Instanz. In AppState.js werden folgende Daten verwaltet:

- `currentProject`: Das aktuell gewählte Projekt.
- `currentTicket`: Das aktuell gewählte Ticket.
- `runningTaskId`: Die ID des laufenden Tasks.
- `idOfTaskToChange`: Die ID des Tasks, dessen Kontext gewechselt werden soll.
- `currentDate`: Das Datum des aktuellen Tages in der Form jjjjmmtt[16].
- `shouldTasksListBeRendered`: Ein Flag das bestimmt, ob die Taskliste in der Präsentation „YOUR BOOKING“ aktualisiert werden soll.
- `shouldProjectsListOfSelectTicketPageBeRendered`: Ein Flag das bestimmt, ob die Projektliste in „SELECT TICKET“ aktualisiert werden soll.
- `shouldProjectListOfCreateNewTicketPageBeRendered`: Ein Flag das bestimmt, ob die Projektliste in „CREATE TICKET“ aktualisiert werden soll.

Die Schnittstelle der AppState-Komponente bietet folgende Funktionen an:

- `setCurrentProject`: Das aktuell gewählte Projekt auf das übergebene Projekt setzen.
- `getCurrentProject`: Das aktuell gewählte Projekt abrufen.
- `setCurrentTicket`: Das aktuell gewählte Ticket auf das übergebene Ticket setzen.
- `getCurrentTicket`: Das aktuell gewählte Ticket abrufen.
- `setIdOfTaskToChange`: Die `idOfTaskToChange` auf die übergebene ID setzen.
- `getIdOfTaskToChange`: Die `idOfTaskToChange` abrufen.
- `setRunningTaskId`: Die ID des laufenden Tasks auf die übergebene ID setzen.
- `getRunningTaskId`: Die ID des laufenden Tasks abrufen.
- `removeRunningTaskID`: Die ID des laufenden Tasks im AppState löschen.
- `setLocalCurrentDate`: Das gegebene Datum als aktuelles Datum im lokalen Speicher speichern.
- `getLocalCurrentDate`: Das aktuelle Datum aus dem lokalen Speicher abfragen.

[16] Zum Beispiel: 10.11.2013 wird als 20131110 dargestellt.

- `getCurrentDate`: Das tatsächliche aktuelle Datum abfragen.
- `setTasksListFlag`: Das Flag der Taskliste auf 1 setzen.
- `unsetTasksListFlag`: Das Flag der Taskliste auf 0 setzen.
- `getTasksListFlag`: Das Flag der Taskliste abrufen.
- `setProjectsListOfSelectTicketPageFlag`: Das Flag der Projektliste in “SELECT TICKET” auf 1 setzen.
- `unsetProjectsListOfSelectTicketPageFlag`: Das Flag der Projektliste in “SELECT TICKET” auf 0 setzen.
- `getProjectsListOfSelectTicketPageFlag`: Das Flag der Projektliste in “SELECT TICKET” abrufen.
- `setProjectListOfCreateNewTicketPageFlag`: Das Flag der Projektliste in “CREATE TICKET” auf 1 setzen.
- `unsetProjectListOfCreateNewTicketPageFlag`: Das Flag der Projektliste in “CREATE TICKET” auf 0 setzen.
- `getProjectListOfCreateNewTicketPageFlag`: Das Flag der Projektliste in “CREATE TICKET abrufen.

Die Hauptkomponente der Steuerung ist der „AppController“. Diese Komponente besteht aus den zwei Javascript-Dateien „AppController.js“ und „AppControllerHelper.js“. Während die wichtigste Aufgabe von „AppController.js“ ist, auf Events zu reagieren, bietet „AppControllerHelper.js“ Hilfsfunktionen für „AppController.js“ an. Ein Event kann entweder eine Benutzeraktion oder ein Seitenwechsel sein. Im Folgenden wird beschrieben, wie die AppController-Komponente aufgebaut ist. Dabei wird auch verdeutlicht, wann Events auftreten, wie sie in der Steuerung ankommen, und wie der AppController darauf reagiert.
„AppControllerHelper.js“ wird in Form einer Javascript-Funktion definiert. In der Schnittstelle bietet „AppControllerHelper“ mehrere Funktionen an, die in „AppController.js“ aufgerufen werden:

- `renderSelectTicketPage`: Die Auswahlliste der Projekte in der Benutzeroberfläche darstellen, die „Beobachter“ in die Auswahlliste und das Eingabefeld in „SELECT TICKET“ hinzufügen (siehe Abbildung 11: Mockup "SELECT TICKET").
- `renderTicketsListOrCreateButton`: Die Liste der gefundenen Tickets in “SELECT TICKET” in der Benutzeroberfläche in Form der einzelnen Buttons darstellen und die „Beobachter“ zu den einzelnen Buttons hinzufügen. Wurde kein Ticket gefunden, wird der „Create new ticket“-Button gerendert.
- `generateNewTicketPage`: Die Auswahlliste der Projekte in der Benutzeroberfläche darstellen und die „Beobachter“ zur Auswahlliste. dem Eingabefeld und dem „Create new ticket“-Button in „CREATE TICKET“ hinzufügen (siehe Abbildung 14: Mockup "CREATE TICKET").

- `renderBranchPopup`: Das Pop-up zur Kontextauswahl rendern und die „Beobachter“ zu den einzelnen Buttons hinzufügen. In diesem Pop-up werden die Kontexte in Form von einzelnen Buttons in der Benutzeroberfläche dargestellt.
- `renderTasksList`: Die Liste der ausgewählten Tasks in „YOUR BOOKING“ darstellen und die „Beobachter“ hinzufügen (siehe Abbildung 8: Mockup "YOUR BOOKING").
- `renderSettingsPage`: Die Projekt- und Kontextliste in „SETTINGS“ darstellen und die „Beobachter“ hinzufügen (siehe Abbildung 16: Mockup "SETTINGS").
- `login`: Die eingegebenen Zugangsdaten vom Benutzer mit dem Dienst „LIST“ (siehe Kapitel 3.2) nach Teamwarp schicken und überprüfen lassen. Wenn die Zugangsdaten richtig sind, wechselt die Anwendung zu „YOUR BOOKING“. Ansonsten wird eine Fehlermeldung angezeigt.
- `logout`: Die Zugangsdaten vom Benutzer verwerfen. Die Anwendung wechselt dabei zu „LOGIN“.
- `saveSettings`: Die Änderungen der Einstellungen in „SETTINGS“ (siehe Abbildung 16: Mockup "SETTINGS") bezüglich der Projekt- und Kontextliste sowie der Ticketlebensdauer speichern.

```
var iWarp = iWarp || {};
iWarp.controller = (function () {
})();
```

Abbildung 39: Quelltext "AppController.js"-Teil 1

Wie in Abbildung 39 dargestellt enthält „AppController.js“ vor allem eine Javascript-Funktion, die sich selbst aufruft. Die Funktion wird dadurch sofort ausgeführt, nachdem sie erstellt wurde, sodass die Funktionen in ihrer Schnittstelle aufgerufen werden können.
„AppController.js“ bietet eine einzige Funktion in der Schnittstelle `init()` an. Die Funktion `init` von „AppController.js“ enthält die gesamte Logik der Anwendung.

```
$(document).bind('mobileinit', function () {
    iWarp.controller.init();
});
```

Abbildung 40: Quelltext "AppController.js"-Teil 2

Es wird der Event `'mobileinit'` mit der Funktion `iWarp.controller.init()` verbunden. `'mobileinit'` bezeichnet die Initialisierung der Webanwendung, welche immer stattfindet wenn die Webanwendung geladen wird. Der Quelltext in Abbildung 40 befindet sich außerhalb der Funktion `iWarp.controller` und sorgt dafür, dass `init` immer dann ausgeführt wird, wenn die Anwendung gestartet wird.

```
  var init = function() {
     loadData();
     var d = $(document);
     d.on("pagechange", onPageChange);
     d.on("vclick", "#login-button", helper.login);
     d.on('vclick', '#save-settings', helper.saveSettings);
     d.on("vclick", '#logout', helper.logout);
     d.on('vclick', '#refresh-projects-and-branches',
helper.refreshProjectsAndBranches);
  };
```

Abbildung 41: Quelltext "AppController.js"-Teil 3

Der Quelltext in Abbildung 41 stellt der Inhalt der Funktion `init()` dar. `init()` enthält folgende Funktionen:

- `loadData()`: Diese Funktion lädt die Daten aus dem lokalen Speicher in das Modell, falls die Zugangsdaten noch im Session-Storage verfügbar sind. Wurde der Browser neugestartet, können keine Daten des Benutzers geladen werden, weil der Session-Storage geleert wurde.
- `$(document)` ist das gesamte jQuery-Objekt, welches die ganze Webanwendung kapselt.
- `d.on("pagechange", onPageChange)`: Diese Anweisung weist dem allgemeinen Seitenwechsel die Funktion „`onPageChange`" zu, sodass diese Funktion automatisch ausgeführt wird wenn ein Seitenwechsel stattfindet.
- `d.on("vclick", "#login-button", helper.login)`: Diese Anweisung weist der Aktion `"vclick"` am Element mit der ID `"login-button"` die Funktion `helper.login` zu. `"vclick"` ist die Simulation der Aktion: Antippen, wobei die Wartezeit von 200 Millisekunden abgeschafft wird. Bei der normalen Tippen Aktion „click" wartet das System 200 Millisekunden bevor es die zugewiesene Funktion ausführt, um sicherzustellen, dass es kein Doppelklick war. `"#login-button"` ist der Bezeichner für das Element mit der ID „login-button", in diesem Fall bezeichnet er den „LOGIN"-Button (siehe Abbildung 7: Mockup „LOGIN"). Diese Anweisung fügt dem „LOGIN"-Button einen „Beobachter" hinzu, sodass immer wenn der Button angetippt wird, die Anwendung die Funktion „helper.login()" aufruft.

Die weiteren Anweisungen fügen den verschiedenen statischen Buttons wie zum Beispiel „Logout"-(Abbildung 8), „Save settings"- und „Refresh"-Button (Abbildung 16) die verschiedene „Beobachter" hinzu.
Die Funktion „`onPageChange`" spielt eine sehr wichtige Rolle. Sie kapselt die gesamten Reaktionen der Anwendung auf Seitenwechsel-Events. Im Folgenden wird der Code dieser Funktion vorgestellt.

```
var onPageChange = function (data) {
      var toPageId = data.toPage.attr('id');
      (…)
}
```

Abbildung 42: Quelltext "AppController.js"-Teil 4

`onPageChange` verfügt wie in Abbildung 42 über den Parameter `data`. Mit dem Parameter `data` können Attribute von den Ziel- und Quelleseiten des Seitenwechsels abgefragt werden.
`data.toPage.attr('id')` liefert die ID der Zielseite. Mithilfe dieses Attributs kann eine passende Funktion aufgerufen werden, wenn die Anwendung zu einer Seite wechselt.

```
var onPageChange = function (data) {
      (…)
      switch (toPageId) {
            case settingsPageId:
                  helper.renderSettingsPage();
                  break;
            (…)
      }
}
```

Abbildung 43: Quelltext "AppController.js"-Teil 5

In Abbildung 43 ist ein Teil des `switch`-Blocks in der Funktion `onPageChange()`.Wenn `toPageId` die ID der „SETTINGS“-Seite ist, wird die Funktion `helper.renderSettingsPage()` aufgerufen, um die „SETTINGS“-Seite zu rendern.
Die Funktion `onPageChange` verbindet noch weitere Seitenwechsel-Events mit den passenden Funktionen. „`init`“ ermöglicht der Anwendung auf alle Seitenwechselevents zu reagieren. Darüber hinaus verbindet „`init`“ die statischen Buttons der Anwendung mit den entsprechenden Funktionen.
Im Folgenden wird ein Beispiel ausgeführt um zu verdeutlichen, wie sich die dynamischen Buttons verhalten und wie es realisiert wird.
Die Funktion `renderTicketsList()` von „AppControllerHelper.js“ hat die Aufgabe, aus einer übergebenen Ticketliste eine Ticketliste in HTML zu generieren und in der Anwendung zu darzustellen. Jedes Ticket wird als ein Button dargestellt, welcher auf Antippen reagiert. „`renderTicketsList`“ wird wie folgend definiert.

```
var renderTicketsList = function (ticketsList) {
      var ticketSuggestedView = $(ticketSuggestedViewId);
      var ticketsListView = $('<ul id="ticket-list"  data-
role="listview"' + </ul>');
      ticketsListView.appendTo(ticketSuggestedView);
      for (var i = 0; i < ticketsList.length; i++) {
          var ticket = ticketsList[i];
          var ticketView = '<li><a class="ticket">' +
'<span class="ticket-id">' + ticket.id + '</span>' +
'<span class="ticket-status"><sup>' + ticket.status +
'</sup></span>' +
<p class="ticket-title">' + ticket.title + '</p>' +
'</a></li>'
          ticketsListView.append(ticketView);
      }
      ticketsListView.listview();
      (…)
};
```

Abbildung 44: Quelltext "renderTicketsList()"-Teil 1

In Abbildung 44 ist der erste Teil der Funktion `renderTicketsList()` abgebildet. Sie verfügt über einen Parameter der eine Ticketliste enthält.

```
      var ticketsListView = $('<ul id="ticket-list"  data-
role="listview"' + </ul>');
      ticketsListView.appendTo(ticketSuggestedView);
```

Abbildung 45: Quelltext "renderTicketsList()"-Teil 2

Mit den zwei Anweisungen in Abbildung 45 erstellt die Funktion ein „ul“-Element[17] mit der ID „ticket-list“ und der Rolle „listview“ und verknüpft dieses Element dann mit dem Element „ticketSuggestedView“. „ticketSuggestedView“ bezeichnet in der Benutzeroberfläche das Div-Tag, in dem sich die vorgeschlagene Ticketliste befinden soll.
Die for-Schleife in **Error! Reference source not found.** sorgt dafür, dass jedes Ticket der Ticketliste mit Ticket-ID, Status und Titel als ein Button dargestellt wird, welcher das Attribut `class="ticket"` besitzt. Jeder Button ist ein Listeelement und wird an die Ticketliste angehängt.

Im Anschluss folgt der zweite Teil der Funktion `renderTicketsList`.

[17] Unordered list-Ein HTML-Element, das eine Liste darstellt.

```
var renderTicketsList = function (ticketsList) {
    (…)
    $('.ticket').on('vclick', function () {
        //Quelltext…
    });
}
```

Abbildung 46: Quelltext "renderTicketsList()"-Teil 3

Mit Hilfe des Events `'vclick'` wird wie Abbildung 46 jedem Button, welcher über das Attribut `class="ticket"` verfügt, eine Funktion zugewiesen. Sollte ein solcher Button angetippt werden, wird diese Funktion ausgeführt.
Andere dynamische Elemente wie beispielweise Task-, Kontext- und Projektliste werden in ähnlicher Weise gerendert.

Während der Implementierung mussten einige Probleme gelöst werden. Im Folgenden werden sie vorgestellt.

Kommunikationen mit Teamwarp
Die gesamte Kommunikation zwischen der Anwendung und Teamwarp erfolgen mithilfe des Direct-Access-Interfaces. Damit die gesendeten http-Anfragen nicht in Klartext in der URL zu sehen sind, werden die Anfragen in einem Formular zusammengefasst und mit dem Befehl POST verschickt.

```
var xmlhttp = new XMLHttpRequest();
```

Abbildung 47: Quelltext "Kommunikation mit Teamwarp"-Teil 1

Diese Anweisung in Abbildung 47 befindet sich in „AppsControllerHelper.js“, sie erstellt im System eine einzige http-Anfrage, die für die gesamte Kommunikation mit Teamwarp verwendet wird. In „AppControllerHelper.js“ ist eine Funktion `sendHttpRequest()` definiert, welche beim Verschicken der http-Anfragen hilft.

```
  var sendHttpRequest = function (params) {
      xmlhttp.open("POST", 'https://tpdevwww.mgm-
edv.de/teamwarp/direct', false);
      xmlhttp.setRequestHeader("Content-type",
"application/x-www-form-urlencoded");
      xmlhttp.send(params);
      return xmlhttp.responseText;
  };
```

Abbildung 48: Quelltext "Kommunikation mit Teamwarp"-Teil 2

Bei jeder http-Anfrage wird das gleiche Formular wie in Abbildung 48 verwendet. Die verschiedenen http-Anfragen unterscheiden sich nur in den Anfrageparametern. Diese Funktion verfügt über einen Parameter, welcher die Parameter einer Anfrage enthält. `sendHttpRequest()` liefert die Antwort von Teamwarp zurück wenn die Kommunikation erfolgt ist.
Eine http-Anfrage kommt in folgenden Situationen zum Einsatz:

- Wenn sich der Benutzer ins System einloggt, werden Zugangsdaten in Teamwarp überprüft.
- Wenn der Benutzer nach Tickets in Teamwarp sucht, werden Tickets mit der entsprechenden Ticketnummer als Antwort übergeben.
- Wenn der Benutzer einen laufenden Task stoppt, werden die geleistete Arbeitszeit und der bearbeitete Tasks an Teamwarp geschickt.
- Wenn der Benutzer den Kontext eines bearbeiteten Tasks wechselt, wird die geleistete Arbeitszeit des alten Tasks an den neuen Task verschoben.
- Wenn der Benutzer die Projekt- und Kontextliste in „SETTINGS“ (Abbildung 16) aktualisiert, werden die Informationen über Projekte und Kontexte von Teamwarp als Antwort übergeben.

Der zweite Einsatzfall, in dem der Benutzer nach Tickets in Teamwarp sucht, wird im Folgenden als Beispiel vorgestellt.

```
   var params = 'login=' + username + '&pwd=' + password +
'&day=' + appState.getCurrentDate() + '&action=issue&key='
+ projectName + '-' + ticketNumber + '*' + '&json';
   var obj = JSON.parse(sendHttpRequest(params));
   var code = obj.code;
   if (code == 200) {
        var dataArray = obj.data;
        if (dataArray != null) {
          return
Ticket.TicketState.generateTicketsList(dataArray);
        }}(…)
```

Abbildung 49: Quelltext "Kommunikation mit Teamwarp"-Teil 3

Die Parameter werden je nach Einsatzfall gesammelt. In Fall von Abbildung 49 kommt der Dienst „ISSUE“ zum Einsatz. Nachdem die Anfrage erfolgt ist wird die Antwort von Teamwarp direkt interpretiert. Wenn der Antwortcode „200“ ist, werden die Daten der Antwort weiter genutzt, um eine Ticketliste zu generieren. Ansonsten gilt die Anfrage als nicht erfolgreich und es wird eine Fehlermeldung angezeigt.

Datenspeicherung im Speicher

Die Datenspeicherung wird durch das Framework jstorage (siehe 5.1.3) vereinfacht. Während die Zugangsdaten des Benutzers im Session-Storage gespeichert werden, sind die

anderen Daten wie zum Beispiel Projekt-, Kontext- und Ticketliste im lokalen Speicher zu sichern.

```
var saveProjectToLocalStorage = function () {
   $.jStorage.set(projectsListStorageKey, projectsList);
};
```

Abbildung 50: Quelltext "Datenspeicherung"-Teil 1

Mit der in Abbildung 50 dargestellten Funktion kann die Projektliste im lokalen Speicher gespeichert werden. Die anderen Daten werden in der gleichen Weise gespeichert. Um Daten über die Projektliste im lokalen Speicher abzuspeichern, wird ein Speicher-Schlüssel benötigt. Dieser Schlüssel gilt als eine eindeutige ID der Daten einer Projektliste im lokalen Speicher.

```
    var loadProjectsFromLocalStorage = function () {
        return $.jStorage.get(projectsListStorageKey);
    };
```

Abbildung 51: Quelltext "Datenspeicherung"-Teil 2

Diese Funktion in Abbildung 51 lädt mithilfe des Speicher-Schlüssels die Projektliste aus dem lokalen Speicher und gibt sie zurück.
Ein Browser speichert Daten, welche unter einer gleichen URL bearbeitet werden, im gleichen lokalen Speicher. Deshalb ist es problematisch, wenn mehrere Benutzer die Anwendung in einem gleichen Handy mit einem gleichen Browser nutzen, und die Anwendung verwendet für jedes Datentyp wie zum Beispiel das Projekt nur einen einzigen Speicher-Schlüssel. In diesem Fall werden die Daten von den Benutzern in lokalen Speicher unter einen gleichen Schlüssel gespeichert. Um diese Datenvermischung zu vermeiden, benötigt jeder Benutzer für jeden Datentyp einen eigenen Schlüssel. In der Implementierung werden Schlüsselpräfixe zu diesem Zweck verwendet.

```
    projectsListStorageKeyWithoutPrefix =
'iwarp.local.projectsList',
```

Abbildung 52: Quelltext "Datenspeicherung"-Teil 3

Für jeden Datentyp gibt es ein gemeinsamer Schlüssel wie in Abbildung 52. Der eigene Schlüssel jedes Datentyps wird je nach dem Benutzer generiert, indem der Benutzername des Benutzers mit dem gemeinsamen Schlüssel des Datentyps kombiniert wird.

```
    projectsListStorageKey;
    var setKeysPrefix = function (prefix) {
        projectsListStorageKey = prefix + '.' +
projectsListStorageKeyWithoutPrefix;
    }
```

Abbildung 53: Quelltext "Datenspeicherung"-Teil 4

Mit der Funktion `setKeysPrefix(prefix)` Abbildung 53 wird der gemeinsame Schlüssel mit dem gegebenen Präfix zusammenkombiniert, um einen eindeutigen Schlüssel zu generieren. Diese Methode wird für jeden Datentyp angewendet.

```
var setKeysPrefix = function (prefix) {
     Branch.BranchState.setKeysPrefix(prefix);
     Project.ProjectState.setKeysPrefix(prefix);
     Ticket.TicketState.setKeysPrefix(prefix);
     Task.TaskState.setKeysPrefix(prefix);
     TaskTime.TaskTimeState.setKeysPrefix(prefix);
     Settings.SettingsState.setKeysPrefix(prefix);
     appState.setKeysPrefix(prefix);
};
```

Abbildung 54: Quelltext "Datenspeicherung"-Teil 5

Die Funktion in Abbildung 54 wird in „AppControllerHelper.js“ definiert, sie sammelt `setKeysPrefix`-Funktionen aller Datentypen. Die Funktion wird immer aufgerufen wenn sich ein Benutzer ins System einloggt oder wenn der Browser aktualisiert wird. Das sorgt dafür, dass ein Benutzer seine Daten mit seinen eigenen Schlüsseln speichert und lädt, wenn die Anwendung benutzt.
Die Datenspeicherung in Session-Storage funktioniert in ähnlicher Weise wie beim lokalen Speicher.

```
sessionStorage.setItem(userDataUsernameSessionStorageKey,
myUsername);

return
sessionStorage.getItem(userDataUsernameSessionStorageKey);
```

Abbildung 55: Quelltext "Datenspeicherung"-Teil 6

Mittels eines Speicher-Schlüssels werden Daten in Session-Storage gespeichert und aus Session-Storage geladen (wie in Abbildung 55). Die Zugangsdaten müssen nicht mit Schlüsseln, welche ein Präfix haben, bearbeitet werden, da Zugangsdaten nur innerhalb

einer Sitzung des Browsers existieren und benutzerspezifisch sind. Deswegen kommt es nie zur Situation, dass sich die Zugangsdaten von zwei verschiedenen Benutzern gleichzeitig im Session-Storage befinden. Der Benutzername und das Passwort des Benutzers werden nach dem erfolgreichen Einloggen im Session-Storage gespeichert.

Ticketliste von lokalen Speicher und Teamwarp abfragen

Dem Benutzer werden die von ihm zuvor eingegebenen Tickets vorgeschlagen, wenn er versucht ein Ticket auszuwählen

Nach der Funktionalität **F9** im Kapitel 4.2 sollen dem Benutzer in der „SELECT TICKET"-Seite die von ihm zuvor ausgewählten Tickets vorgeschlagen, wenn er versucht ein Ticket auszuwählen. Ein Ticket, das vom Benutzer ausgewählt wird, wird in die Ticketliste des Ticketmodells hinzugefügt und im lokalen Speicher gespeichert. Dieses Ticket wird beim nächsten Mal, wenn der Benutzer ein Ticket suchen will, vorgeschlagen.

In der Funktionalität **F10** im Kapitel 4.2 wurde beschrieben: „Wenn der Benutzer mindesten die ersten zwei Ziffern einer Ticketnummer eingibt, werden ihm alle Tickets angezeigt, welche mit diesen Ziffern beginnen". Um eine solche Ticketliste zu erstellen, muss die Anwendung eine http-Anfrage mit dem Dienst „LIST" an Teamwarp schicken und die Antwort von Teamwarp empfangen und interpretieren.

```
   var getTicketList = function (projectName,
ticketNumber, username, password) {
       if (ticketNumber == "" || +ticketNumber < 10) {
           return
Ticket.TicketState.getTicketsListFromStorage(projectName,
ticketNumber);
       }
       else {
           var params = 'login=' + username + '&pwd=' +
password + '&day=' + appState.getCurrentDate() +
'&action=issue&key=' + projectName + '-' + ticketNumber +
'*' + '&json';
           var obj = JSON.parse(sendHttpRequest(params));
           var code = obj.code;
           if (code == 200) {
               var dataArray = obj.data;
               if (dataArray != null) {
                   return
Ticket.TicketState.generateTicketsList(dataArray);
               }
               else {
                    return [];
               }
           }
       }
   };
```

Abbildung 56: Quelltext "getTicketsList()"

Die Funktion `getTicketList()` in Abbildung 56 realisiert die beiden obengenannten Funktionalitäten **F9** und **F10**. `getTicketList()` verfügt über vier Parameter:

- `projectName`: Das Projekt des gewünschten Tickets.
- `ticketNumber`: Ein Teil oder die ganze Nummer des gewünschten Tickets.
- `username`: Der Benutzername.
- `password`: Das Passwort.

Wenn der Parameter `ticketNumber` keine oder nur eine Ziffer enthält, werden Tickets mit dem entsprechenden Projekt und Nummer von der gespeicherten Ticketliste des Ticketmodells abgefragt und angezeigt.
Wenn der Parameter `ticketNumber` zwei oder mehr Ziffern enthält, werden die Parameter für eine http-Anfrage mit Zugangsdaten, Projekt, Ticketnummer usw. gesammelt. Die Anfrage wird dann mit den Parametern und dem Dienst „LIST“ mittels der Funktion `sendHttpRequest` an Teamwarp geschickt. Die Antwort von Teamwarp wird danach direkt mithilfe der Funktion `Ticket.TicketState.generateTicketsList()` interpretiert, wenn die http-Anfrage erfolgt ist. Diese Funktion nimmt das Daten-Array in der Antwort von Teamwarp an, generiert daraus eine Ticketliste und gibt sie zurückt. Die Anwendung stellt diese Ticketliste als gefundene Tickets dar.

Vermeidung von redundanten Funktionsaufrufen
In der Anwendung sollte einige Funktionen immer aufgerufen werden, nachdem ein bestimmtes Ereignis stattgefunden hat. Die Funktion `renderTasksList()` ist ein Beispiel. Es wird erwartet, dass diese Funktion immer dann ausgeführt wird, wenn die Anwendung zur Seite „YOUR BOOKING“ wechselt. Die Taskliste in „YOUR BOOKING“ wird dadurch aktualisiert wird, falls ein neuer Task hinzugefügt wurde. Um zu vermeiden, dass `renderTasksList()` aufgerufen wird, auch wenn kein neuer Task in die Taskliste hinzugefügt wurde, kommt ein sogenannter Taskliste-Flag zum Einsatz.

```
var shouldTasksListBeRendered = 1;
```

Abbildung 57: Quelltext "Vermeindung von redundenten Funktionsaufrufen"-Teil 1

Dieser Flagg wird in „AppControllerHelper.js“ definiert. Ein Flagg kann nur den Wert „0“ oder „1“ haben. Während „1“ bedeutet, dass `renderTasksList()` aufgerufen soll, wenn sich das System in „YOUR BOOKING“ navigiert, sagt „0“ den Gegenteil aus.

```
switch (toPageId) {
   case yourBookingPageId:
      if (appState.getTasksListFlag() == 1) {
           helper.renderTasksList();
           appState.unsetTasksListFlag();
      }(…)}
```

Abbildung 58: Quelltext "Vermeidung von redundanten Funktionsaufrufen"-Teil 2

Immer wenn `renderTasksList()` an dieser Stelle aufgerufen wird, wird der Taskliste-Flagg auf „0“ gesetzt (Abbildung 58). Dieser Flagg wird wieder auf „1“ gesetzt wenn ein neuer Task in die Taskliste hinzugefügt wird.
In der ähnlichen Weise funktioniert die Funktion zur Aktualisierung der Projektliste.

7 Evaluierung

In diesem Kapitel wird ein Testszenario konzipiert, realisiert und ausgewertet. Da es sich um eine erste qualitative Evaluierung handelt, werden fünf Mitarbeiter von mgm-technologies partners als Testteilnehmer die Anwendung in fünf Arbeitstagen testen. Anschließend werden sie einen Fragebogen zum Nutzungserlebnis der Anwendung ausfüllen und diese Testergebnisse werden ausgewertet. Das Ziel dieser Evaluierung ist es, herauszufinden, ob das Konzept der Anwendung grundsätzlich geeignet ist.

7.1 Testszenario

Dieser Abschnitt beschreibt das Testszenario. Für das Szenario wird eine Testversion von Teamwarp, welche nur im internen Netzwerk von mgm funktioniert, eingerichtet. Diese Testversion bietet die gleichen Funktionen an wie Teamwarp. Außerdem verfügt die Testversion über eine eigene Testdatenbank, welche alle Tickets aus JIRA beinhaltet.
Für den Test verwendet jeder Teilnehmer sein Smartphone mit einem beliebigen Webbrowser. Die fünf Teilnehmer sollen die Anwendung in fünf Tagen zur Erfassung ihrer Arbeitszeiten für generische Tasks benutzen. Da die Anwendung nur auf der Testumgebung läuft, müssen die Teilnehmer ihre Arbeitszeiten noch selbst in Teamwarp nachtragen. Bevor das Testszenario stattfindet, wird den Teilnehmern vorgestellt, wie die Anwendung grundsätzlich funktioniert. Zum Abschluss des Testszenarios werden die fünf Teilnehmer jeweils einen Fragebogen ausfüllen.
Das Ziel dieses Tests ist es, eine erste Rückmeldung der Testteilnehmer über die Benutzeroberfläche und die Funktionalitäten der Anwendung zu erhalten. Es soll getestet werden, ob die Anwendung ihre Aufgaben erfüllt. Das bedeutet nicht nur, dass alle fachlichen Anforderungen umgesetzt wurden, sondern die Benutzer sollen auch mit wenigen Schritten und Aufwänden ihr Ziel erreichen können. Dabei wird festgestellt, ob sich die Mitarbeiter nach der Testphase vorstellen können, die Anwendung produktiv einzusetzen. Außerdem werden Verbesserungsvorschläge von Teilnehmern gesammelt.

Der Fragebogen verfügt über zwei Gruppen: allgemeine Fragen und spezielle Fragen. Die allgemeinen Fragen sollen ermitteln, wie die Teilnehmer mit der Benutzeroberfläche und der Arbeitsweise der Anwendung zufrieden sind und ob sie die Anwendung weiter benutzen möchten. Die speziellen Fragen beziehen sich auf bestimmte Funktionalitäten. Sie sollen herausfinden, ob die umgesetzten Funktionalitäten den Bedarf der Benutzer bereits decken oder ob noch zusätzliche Funktionen zu realisieren sind.

Der Fragebogen kann im Anhang unter dem Punkt A.2 angesehen werden.

7.2 Testergebnisse und Auswertung

In diesem Abschnitt werden die Antworten der Teilnehmer zu den einzelnen Fragen zusammengefasst. Am Ende des Abschnitts werden die Testergebnisse ausgewertet.

1. In welcher Plattform haben Sie die Anwendung benutzt?

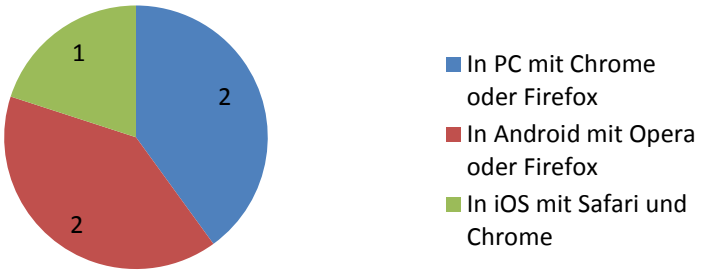

Abbildung 59: Diagramm Antwort zur Frage 1

2. Was ist Ihnen an der Anwendung aufgefallen?

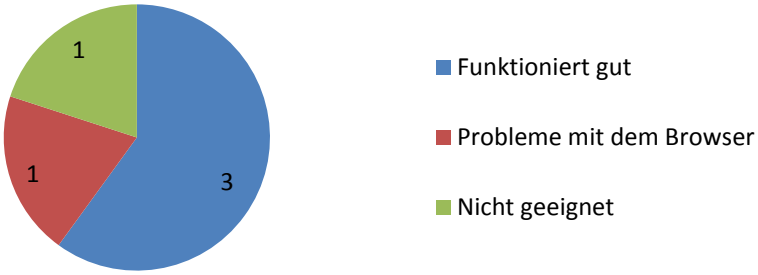

Abbildung 60: Diagramm Antwort zur Frage 2

3. Wie oft haben Sie die Anwendung benutzt?

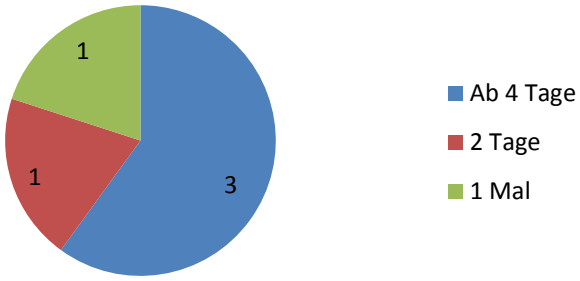

Abbildung 61: Diagramm Antwort zur Frage 3

4. Erfüllt die Anwendung ihre Aufgaben?

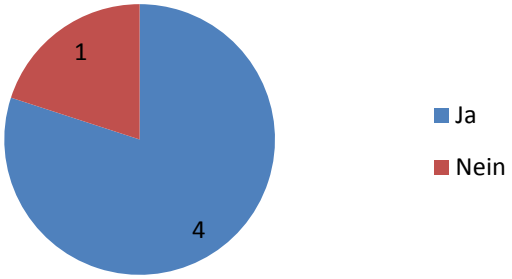

Abbildung 62: Diagramm Antwort zur Frage 4

5. Finden Sie die Anwendung kompliziert oder selbsterklärend?
   Alle Testteilnehmer fanden die Anwendung selbsterklärend.

6. Finden Sie schnell heraus, wie Sie Ihr Ziel erreichen?
   Alle Testteilnehmer gaben an, dass sie schnell herausfinden konnten, wie sie ihr Ziel erreichen.

7. Würden Sie die Anwendung weiter benutzen, wenn sie produktiv gesetzt wird?

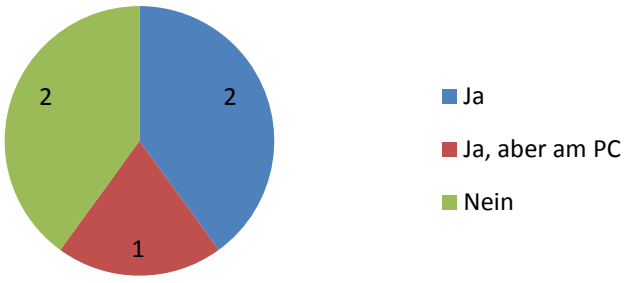

Abbildung 63: Diagramm Antwort zur Frage 7

8. Wie finden Sie die Umsetzung von Tickets? (Wie Sie Tickets suchen und auswählen)
   Alle Testteilnehmer fanden die Umsetzung von Tickets gut.

9. Wie zeitaufwändig war es, ein Ticket zu suchen und einen Task hinzuzufügen?
   Alle Testteilnehmer bewerteten den Zeitaufwand, für die Suche eines Tickets und zum Hinzufügen eines Tasks als gering.

10. Wie finden Sie die Umsetzung von Tasks, dass Sie zuerst ein Projekt auswählen, ein Ticket suchen und wählen und dann einen Kontext selektieren?
    Alle Testteilnehmer bewerteten die Umsetzung von Tasks gut.

11. Wie finden Sie die Rundung der Arbeitszeit? Sollte es automatisch erfolgen oder manuell von Benutzer?
    Alle Testteilnehmer wünschen sich, dass die Rundung der Arbeitszeit automatisch erfolgt.

12. Wie finden Sie, dass die Zeitbuchungen automatisch an Teamwarp geschickt werden?

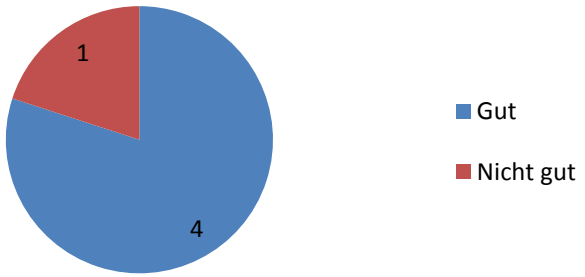

Abbildung 64: Diagramm Antwort zur Frage 12

13. Sollten mehr Optionen für die Ticketlebenszeit zur Verfügung stehen?

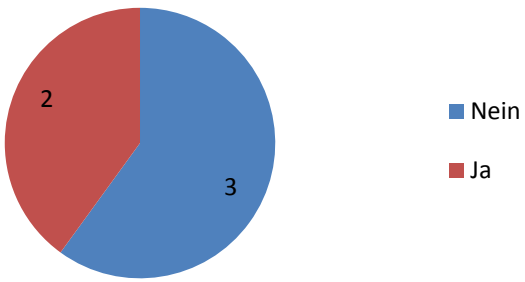

Abbildung 65: Diagramm Antwort zur Frage 13

14. Brauchen Sie überhaupt die Funktion „Kontextwechseln“?

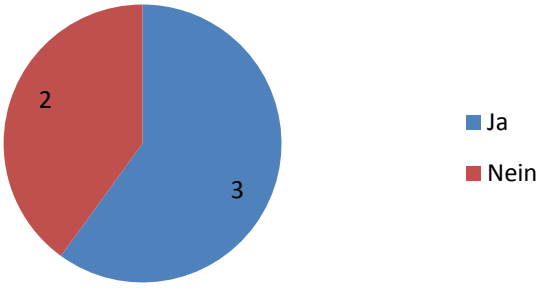

Abbildung 66: Diagramm Antwort zur Frage 14

15. Feedback, Verbesserungsvorschläge.
    -Der Benutzer sollte die geleisteten Arbeitszeiten nachträglich ändern können.
    -Der Benutzer sollte die Zeit für die andere feste Tasks wie zum Beispiel Stand-up Meeting und Seminar erfassen können.
    -Der Benutzer sollte die Start-, die End- und die Pausenzeit eintragen können.
    -Wenn es viele Kontext gibt, sollten sie in Gruppen sortiert werden.
    -Die Anwendung sollte mehr selbsterklärend sein, indem es mehr Hinweise in der Benutzeroberfläche gibt.

Aus den Testergebnissen lassen sich einige Punkte feststellen. Zuerst handelt es sich um die Fachlichkeit. iWarp scheint die fachlichen Anforderungen zu erfüllen. Die zu erfüllenden Aufgaben, Tasks von Teamwarp auszuwählen, Tasks zu bearbeiten und Zeitbuchungen an Teamwarp zu schicken, können in iWarp auf eine unkomplizierte Weise erledigt werden. Da iWarp das klassische Buchungssystem Teamwarp im Moment noch nicht vollständig ersetzen kann, möchten einige Teilnehmer iWarp nicht benutzen. Damit mehr Mitarbeiter die Anwendung benutzen wollen, sollte in der Zukunft die anderen Funktionen von Teamwarp in iWarp umgesetzt werden. Anderseits ergibt sich aus den Testergebnissen, dass iWarp die spezielle Arbeitsweise von einigen Mitarbeitern, wenn sie zum Beispiel gleichzeitig an mehreren Tickets arbeiten, nicht unterstützt. Dieses Problem lässt sich nicht einfach lösen, da für eine solche Arbeitsweise eine andere Art von Anwendungen erforderlich ist. Die technischen Anforderungen werden auch gut erfüllt. iWarp kann in allen gängigen mobilen Plattformen mit fast allen Internet-Browsern benutzt werden. Einige mobile Browser sind zur Anwendung nicht kompatibel beispielweise wie Opera und Firefox auf Android. Der Browser Google Chrome scheint am besten zu iWarp kompatibel zu sein. Alle Teilnehmer waren mit der Benutzeroberfläche der Anwendung zufrieden. Die Testteilnehmer konnten mit wenig Aufwand ihr Ziel erreichen. Jedoch sollte die Anwendung selbsterklärender gestaltet werden, zum Beispiel durch zusätzliche Hinweise

auf der Oberfläche, sodass die Benutzer mit der Anwendung schneller zurechtfinden könnten.

8 Zusammenfassung und Ausblick

Im Rahmen dieser Bachelorarbeit wurde eine mobile Anwendung zur Vereinfachung der Erfassung von Arbeitszeiten entwickelt. Die dafür möglichen Technologien wurden erarbeitet und erläutert. Eine Analyse nach den Standards des Software Engineering trug dazu bei, die fachlichen und die technischen Anforderungen an die zu entwickelnde Anwendung zu identifizieren. Darüber hinaus wurde der Kontext dieser Arbeit, welcher festlegt was im Rahmen der Arbeit relevant ist, abgegrenzt. Aus den Anforderungen wurde die Spezifikation der Anwendung bestimmt. In der folgenden Spezifikation wurde das fachliche Datenmodell des Systems vorgestellt. Die konkreten Funktionalitäten der Anwendung wurden identifiziert und priorisiert. Auf Basis der fachlichen Anforderungen wurden die Anwendungsfälle bestimmt. Die Dialoge wurden mit Hilfe von Mockups auf Basis der Anwendungsfälle gestaltet. Der Ablauf der Benutzerinteraktion auf Basis der Dialoge wurde am Ende der Spezifikation vorgestellt. Daraus folgend wurden die technischen sowie einige Entwurfsentscheidungen getroffen und die Architektur des Systems entwickelt. Die Anwendung wurde mit Hilfe der weborientierten Technologien jQuery und jQueryMobile umgesetzt. Die Architektur besteht dabei aus drei Bausteinsichten, welche den Zusammenhang der Anwendung mit externen Systemen sowie die interne Architektur der Anwendung beschreiben. Zur Verdeutlichung der Anwendungsfälle wurden mehrere Sequenzdiagramme auf Basis der einzelnen Anwendungsfälle erstellt, welche die Abläufe der Anwendung darstellen. Anschließend wurde die Anwendung implementiert. Nach der Implementierung wurde ein Benutzertest durchgeführt. In diesem Test wurde die Anwendung fünf Tagelang von fünf Testpersonen benutzt. Im Anschluss wurde ein Fragebogen zum Nutzungserlebnis ausgefüllt und die Testergebnisse wurde ausgewertet um herauszufinden, ob das Konzept der Anwendung für die Arbeit im Alltag grundsätzlich geeignet ist.
Die Auswertung der Testergebnisse hat verdeutlicht, dass iWarp in einigen Punkten noch zu verbessern ist. Die Anwendung funktioniert zwar gut und erfüllt alle Anforderungen des Auftraggebers. Jedoch müssen noch einige Probleme gelöst werden, damit iWarp produktiv eingesetzt werden kann. Diese Probleme wurden unter dem Antwort auf die letzte Frage

des Fragebogens beschrieben. Die Anwendung kann jetzt den Teil von Teamwarp ersetzen, für den sie vorgesehen war.

Die Anwendung ist jederzeit auf Wünsche von Mitarbeitern erweiterbar, damit sie produktiv gesetzt werden kann. Bereits aus den Verbesserungsvorschlägen der Testteilnehmer sind einige Erweiterungsmöglichkeiten hervorgegangen, die zu einem späteren Zeitpunkt umgesetzt werden könnten:

- Als Alternative zur Zeitbuchung, könnte dem Benutzer eine Ansicht mit einer Zusammenfassung der Zeitbuchungen eines Tages angezeigt werden. In dieser Ansicht könnte der Benutzer die geleisteten Arbeitszeiten nachträglich ändern und manuell abschicken.
- Um Teamwarp ein Schritt weiter ersetzen zu können, wäre es wünschenswert wenn der Benutzer die Start-, Ende- und Pausenzeiten mit iWarp eintragen könnte. Diese Funktion könnte sicherlich in iWarp umgesetzt werden.
- Die Zeiterfassungen für die festgelegten Tätigkeiten wie zum Beispiel Standup Meeting und Seminar könnten auch in iWarp realisiert werden.
- Die Anwendung wäre selbsterklärender, wenn es bei der ersten Nutzung eine kurze Anleitung mit Fotos gäbe.

Es wäre auch möglich, Teamwarp vollständig durch iWarp zu ersetzen, wenn die weiteren Funktionen von Teamwarp wie beispielweise die Monatsübersicht in iWarp umgesetzt werden. Außerdem könnte iWarp als Hybrid App erweitert werden, um die Nutzung komfortabler zu machen.

Anhang

A.1 Inhalt der DVD

•Bachelorarbeit als PDF Dokument

•Projektdateien der Anwendung

A.2 Fragebogen

ALLGEMEINE FRAGEN

1. In welcher Plattform haben Sie die Anwendung benutzt?
2. Was ist Ihnen an der Anwendung aufgefallen? (positiv und negativ)
3. Wie oft haben Sie die Anwendung benutzt?
   Wenn selten oder gar nicht, warum?
4. Erfüllt die Anwendung ihre Aufgaben?
   Welchen Teil, welchen Teil nicht?
5. Finden Sie die Anwendung kompliziert oder selbst erklärend?
6. Finden Sie schnell heraus, wie Sie Ihr Ziel erreichen?
   Wenn nein, warum?
7. Würden Sie die Anwendung weiter benutzen, wenn sie produktiv gesetzt wird?
   Wenn nein, was müsste geändert werden damit Sie sie benutzen möchten?

SPEZIELLE FRAGEN

8. Wie finden Sie die Umsetzung von Tickets? (Wie Sie Tickets suchen und auswählen)
9. Wie zeitaufwändig war es, ein Ticket zu suchen und einen Task hinzuzufügen?

Warum? (wegen Netzwerk, wegen fehlender Einführung, wegen Handy…)

10. Wie finden Sie die Umsetzung von Tasks, dass Sie zuerst ein Projekt auswählen, ein Ticket suchen und wählen und dann einen Kontext selektieren?
11. Wie finden Sie die Rundung der Arbeitszeit? Sollte es automatisch erfolgen oder manuell von Benutzer?
12. Wie finden Sie, dass die Zeitbuchungen so automatisch an Teamwarp geschickt werden?
13. Sollten mehr Optionen für die Ticketlebenszeit zur Verfügung stehen?
14. Brauchen Sie überhaupt die Funktion „Kontextwechseln“?
15. Feedback, Verbesserungsvorschläge.

Glossar

API	Application programming interface oder Schnittstelle zur Anwendungsprogrammierung – ist ein Programmteil, der von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird.
App-Store	Ist die Bezeichnung für eine digitale Vetriebsplattform von Anwendungssoftware. Der Service ermöglicht es Benutzern Software aus einem Anwendungskatalog von Erst- und Drittanbieterentwicklern zu suchen und herunterzuladen.
Beobachter	Engl. Action listener – ist ein Programmteil, der die Aufgabe hat, Benutzeraktionen und Programmereignisse zu empfangen und an den dafür zuständigen Programmteil weiterzuleiten.
Browser-Wiedergabemaschine	Ist nicht der Browser selbst, sondern ein Modul der mobilen Plattform, der die HTML, Javascript Dateien einliest und wiedergibt.
Buchungskontext	Ist ein Kontext, unter dem ein Mitarbeiter arbeitet z.B Entwicklung oder Bugfixing.
CSS	Cascading Style Sheets – ist eine deklarative Sprache für Stilvorlagen von strukturierten Dokumenten.
Framework	Ist ein Programmiergerüst, das in der Softwaretechnik, insbesondere im Rahmen der objektorientierten Softwareentwicklung sowie bei komponentenbasierten Entwicklungsansätzen, verwendet wird.
HTML	Hypertext Markup Language - ist eine textbasierte

	Auszeichnungssprache zur Strukturierung von Inhalten wie Texten, Bildern und Hyperlinks in Dokumenten.
HTTP	Hypertext Transfer Protocol - ist ein Protokoll zur Übertragung von Daten über ein Netzwerk, es wird hauptsächlich eingesetzt, um Webseiten von einem Server in einen Webbrowser zu laden.
ID	Identifikator - ist ein künstlich zugewiesenes Merkmal zur eindeutigen Identifizierung eines Objektes.
JIRA	Ist eine webbasierte Anwendung zur Fehlerverwaltung, Problembehandlung und operativem Projektmanagement. Im Unternehmen mgm-tp wird JIRA eingesetzt.
JIRA-Ticket	Ist eine Tätigkeit oder eine Aufgabe, die in JIRA erstellt wird.
jQuery	Ist eine freie, umfangreiche JavaScript Klassenbibliothek.
jQueryMobile	Ist eine freie, umfangreiche HTML und JavaScript Bibliothek für mobile Webanwendungen.
MVC	Model View Controller oder Modell Präsentation Steuerung – ist ein Muster zur Strukturierung von Software-Entwicklung in die drei Einheiten Datenmodell (engl. model), Präsentation(engl. view) und Programmsteuerung (engl. controller).
Mockup	Wegwerfprototypen der Benutzerschnittstelle einer zu erstellenden Software.
SDKs	Software Development Kits. Ein SDK ist eine Sammlung von Werkzeugen und Anwendungen, um eine Software zu erstellen meist, inklusive Dokumentation.
Session-Storage	Sitzungsspeicher von einem Browser. Dieser Speicher wird geleert wenn der Browser deaktiviert wird.
Task	Ist eine Kombination von einem Ticket und einem Buchungskontext. Ein Task kann in einen Teamwarp-Knoten eingetragen werden.
Teamwarp	Ist eine webbasierte Anwendung zur Zeitbuchung für Mitarbeiter im Unternehmen mgm.

Teamwarp-Knoten	Ist eine Zeile in der Seite der Tagesbuchung in Teamwarp. Unter einem Buchungskontext sind ein oder mehrere Teamwarp-Knoten verfügbar.
URL	Uniform Resource Locators - identifizieren und lokalisieren eine Ressource wie z.B. eine Webseite über die zu verwendende Zugriffsmethode und den Ort der Ressource in Computernetzwerken.

Versicherung über Selbstständigkeit

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

Hamburg, den _____ _____