MASTER THESIS

zur Erlangung des akademischen Grades
„Master of Science in Engineering“
im Studiengang Multimedia und Softwareentwicklung

Design und Implementierung einer Multi-Touch optimierten nativen iOS App für das JVx ERP Applikation Framework

Ausgeführt von: Stefan Fessler, B.A. MCI
Personenkennzeichen: 1210299018

1. BegutachterIn: DI Dr. Markus Schordan
2. BegutachterIn: Roland Hörmann

Wien, 22. Mai 2014

FACHHOCHSCHULE
TECHNIKUM WIEN

Eidesstattliche Erklärung

„Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht. Ich versichere, dass die abgegebene Version jener im Upload Tool entspricht."

Ort, Datum

Unterschrift

Kurzfassung

Für das quelloffene JVx Applikation Framework, welches auf „Java“ basiert und für die Erstellung von ERP-Applikationen konzipiert ist, soll eine native iOS-App mit Multi-Touch optimierter Benutzeroberfläche designt und implementiert werden. Hierzu soll für das auf dem Drei-Schichten-Modell basierende JVx Framework eine Erweiterung der Darstellungsschicht vorgenommen werden und eine universelle mobile API geschaffen werden, um alle (auch zukünftigen) mobilen Plattformen des JVx Frameworks über diese anzubinden. Hierfür wird beim JVx Framework, an einem dafür vorgesehenen Hotspot, ein neuer Web-Service geschaffen, der den Architektur- und Design-Anforderungen einer modernen RESTful API-Schnittstelle entspricht.

Für diese auf dem JSON-Format implementierte Erweiterung des JVx Frameworks wird im zweiten Teil dieser Arbeit eine generische iOS Referenz-App geschaffen, die alle Anforderungen der Beispiel-Anwendung „MyERP“ mobil und den Apple iOS Human Interface Guidelines entsprechend umsetzt. Hierbei werden grundlegende iOS-Spezifika, wie etwa die Programmiersprache „Objective-C“ und deren Design Patterns, erklärt; auch auf die Unterschiede zwischen iPhone und iPad wird eingegangen. Ein besonderes Augenmerk wird hier auf die Anbindung zur JVx API und das JSON Objekt Mapping gelegt, welches durch das ebenfalls quelloffene Framework „RestKit“ umgesetzt wurde. Einen weiteren Kernpunkt stellt der Aufbau der generischen API Controller-Klassen dar, die das Herzstück der erstellten nativen JVx iOS-Referenz-App darstellen. Auf die erst kürzlich erschienene neue iOS7-Version des Betriebssystems mit den neuen Design-Anpassungen sowie die 64-Bit-Erweiterung wird bei der JVx Beispiel-App für die iOS-Plattform eingegangen und die Unterschiede aufgezeigt.

Schlagwörter: JVx, iOS, iPhone, iPad, Multi-Touch, Objective-C, RESTful, API, ERP

Abstract

The purpose of this thesis is the development of a native iOS App with a Multi-Touch optimized interface for the Java-based, open source JVx Application framework for programming ERP Applications. For this task, it is necessary to extend the presentation layer of the three-tier JVx Framework and to create a universal mobile API using the JVx Framework. Therefore, a new Web Service for the JVx Framework which complies with the architecture and design requirements of a modern RESTful API interface has been developed.

This extension, designed in the JSON format, also includes a generic iOS reference application that meets all requirements of the Apple iOS Human Interface Guidelines. Additionally, iOS specifics such as the coding language Objective-C with its design patterns together with the differences between iPhone and iPad are discussed, and special emphasis is on the link between JVx API and JSON as well as Object Mapping, which is implemented via the open source framework “RestKit”. Another essential point is the structure of the generic API controller classes which form the core of the JVx iOS reference application. Due to the recent release of the new iOS7 version with its new design adaptations as well as the 64-bit extension, the thesis explores the differences using the JVx example application for the iOS platform.

Keywords: JVx, iOS, iPhone, iPad, Multi-Touch, Objective-C, RESTful, API, ERP

Danksagung

Ich bedanke mich bei meinen beiden Betreuern Herrn DI Dr. Markus Schordan und Roland Hörmann für die Hilfestellung, Betreuung und ihr konstruktives Feedback zur Arbeit.

Ich bedanke mich beim Team von SIB Visions für die Unterstützung beim Praktischen Teil der Master Thesis, vor allem René Jahn für die fortlaufende Unterstützung bei der Entwicklung des mobilen Clients und der JVx API Erweiterung.

Ich bedanke mich bei Apple Inc. für die gute Entwicklerplattform und die Erfindung der iPhones und iPads.

Inhaltsverzeichnis

1 Einleitung

In dieser Masterarbeit, die dem Themenkreis der Softwareentwicklung zuzuordnen ist, wird das Thema der Framework Erweiterung für das auf dem Drei-Schichten-Modell basierende JVx Framework zur ERP Applikation-Erstellung und der iOS App-Erweiterung beschrieben. Zu Beginn stellt sich an dieser Stelle sicher vielen Personen die Frage nach der Bedeutung dieser fachspezifischen „Fremdwörter“. Hierzu wird in der Einleitung im Bereich „Prämissen und Definitionen“ und im Kapitel 2 „Stand der Literatur“ eine Begriffserklärung sowie deren Beschreibung im Zusammenhang mit dieser Masterarbeit Hilfe geboten.

Der für sehr viele als abstrakt wahrgenommene Begriff „Softwareentwicklung“ impliziert sehr oft den Gedanken an sehr komplexe Vorgänge, und den an Kosten für das Unternehmen. Ein weiterer Grund für den negativ empfundenen Begriff der Softwareentwicklung beruht auf dem Grund, dass sehr oft von technisch unerfahrenen Personen bestimmte, meist noch nicht genau definierte elektronische Hilfsmittel benötigt und spezifiziert werden. Dies führt oft zu einem am Kundenwunsch vorbei entwickeltem Softwareprojekt (in diesem Fall als App bezeichnet), was sehr häufig Kosten- und Zeitüberschreitungen der Projekte zur Folge hat. Die Standish Group hat schon im Jahre 1995 dieses Problem in sehr klare Worte gefasst, die auf einer von Ihnen durchgeführten Umfrage beruhen:

> *"On the success side, the average is only 16.2% for software projects that are completed on time and on-budget. In the larger companies, the news is even worse: only 9% of their projects come in on-time and on-budget." [1]*

Auch Patrick Hamilton [2], S.1f schrieb schon im Jahre 2007, dass die Abschlussquote von erfolgreichen Projekten bei 20% bis 40% liege, und erweiterte die zwei Faktoren "in-Time" und "in-Budget" durch den Faktor "in-Quality". Diese immer größer werdenden Anforderungen an Effizienz und kürzere Entwicklungszeiten, aber auch an bessere Qualität der Software erfordern den Einsatz von Spezifikationen und Frameworks in der Entwicklung. Dadurch können die in der Softwareentwicklung entstehenden Projektkosten und Projektdurchlaufzeiten erheblich gekürzt werden. Dies beruht auf dem Ansatz, dass schon bestehende, funktionierende und getestete Quellcode-Elemente nach dem Baukastensystem wieder verwendet werden können.

In dieser Masterarbeit soll daher für das auf dem Drei-Schichten-Modell basierende JVx Framework eine Erweiterung der Darstellungsschicht vorgenommen werden. Diese soll mittels einer universellen mobilen API geschaffen werden, um alle (auch zukünftigen) mobilen Plattformen des JVx Frameworks über diese anbinden zu können. Um dies zu ermöglichen, soll beim JVx Framework, an einem dafür vorgesehenen Hotspot, ein neuer Web Service geschaffen werden, der den Architektur- und Design-Anforderungen einer mo-

dernen RESTful API-Schnittstelle entspricht. Im nächsten Schritt soll dann diese neue Erweiterung des JVx Backend verwendet werden, um eine native iOS App mit Multi-Touch optimierter Benutzeroberfläche zu designen und implementiere. Dies soll die Erstellung von ERP-Applikationen auf Basis des quelloffenen JVx Frameworks für Entwicklungen attraktiver gestalten, aber auch Qualität und Zeitvorteile bringen. So sollen mit wenig Quellcode in kürzester Zeit mobil taugliche ERP-Applikationen erstellbar werden, was einen wesentlichen Mehrwert bei JVx Anwendungen bringen würden. Der Einsatz des Open Source-Prinzips des JVx Frameworks steigert die Attraktivität des Frameworks und motiviert die Community zu Weiterentwicklungen.

1.1 Prämissen und Definitionen

Um eine bessere Lesbarkeit zu ermöglichen und Begriffsunklarheiten vorzubeugen, werden folgende Fachbegriffe voneinander abgegrenzt bzw. durch deren Synonyme ausgedrückt:

- „iOS“ bezeichnet ein von Apple entwickeltes mobiles Betriebssystem für das iPhone, iPad, iPod touch und den Apple TV der 2. und 3. Generation, welches früher iPhone OS oder iPhone Software genannt wurde. Das Betriebssystem basiert auf einem „Mac OS X“-Kern beziehungsweise Darwin-Betriebssystem, welches wiederum auf einen Unix-Kern zurückgeht. [3]
- Für den Begriff „App“ wird folgende Bezeichnung herangezogen: *„als Kurzform für eine zusätzliche Applikation, die auf bestimmte Mobiltelefone heruntergeladen werden kann.“* [4] Der Begriff wurde von Apple [5] sehr stark forciert und wird daher oft eigenständig verwendet. Auch in der zitierten Literatur wird sehr häufig dieser Kurzbegriff verwendet und daher analog auch in dieser Thesis angewendet.
- Unter dem Begriff “Open Source” wird folgende Definition der Open Source-Initiative angewandt: *“Open source software is software that can be freely used, changed, and shared (in modified or unmodified form) by anyone. Open source software is made by many people, and distributed under "http://opensource.org/licenses" that comply with the Open Source Definition.”* [6]

Die weiteren wichtigen Termini wie „Framework“, „KMU“ und „ERP“ werden umfassend im Kapitel 2 „Stand der Literatur“ erklärt und definiert. Die dort angeführten Begriffserklärungen werden in dieser Masterarbeit dementsprechend verwendet.

1.2 Motivation und Ziele

Die Ergreifung genau dieser Themenstellung begründet sich in der geteilten Aufgabenstellung: Einerseits die Erweiterung eines bestehenden Frameworks und andererseits die komplett freie Neuentwicklung eines mobilen iOS Clients ohne Vorentwicklung oder Vorgaben.

Genau dieses Zusammenspiel aus „bestehend“ und „neu“ waren Motivation und Initialgedanke für den Autor dieser Masterarbeit. Das Ziel war es, einen praxistauglichen Mehrwert mittels einer in der Anwendung funktionierenden und verwendbaren iOS App zu entwickeln, der dann der Open Source-Gemeinschaft zur Verfügung steht und weiter entwickelt werden kann.

1.3 Aufbau der Arbeit

Grundsätzlich gliedert sich die Themenstellung in sechs Kapitel. Das erste Kapitel beinhaltet die Forschungsfrage und Vorgehensweise sowie deren Relevanz und Abgrenzung. Danach folgt die Aufarbeitung der Grundlagen dieser Thesis mittels der Erörterung relevanter Literatur. Hier werden nicht nur die Begriffe ERP, Framework oder RESTful erklärt, sondern auch die Grundpfeiler der iOS App-Entwicklung sowie deren Architektur und Design.

Das dritte Kapitel befasst sich mit der Erweiterung des JVx Frameworks und dem Design der Schnittstelle für mobile Anwendungen. In dieser Erweiterung wird die Grundlage für die im Kapitel 4 beschriebene Entwicklung der generischen iOS App für das JVx Framework beschrieben. Im Praxisteil der Masterarbeit wird die Implementierung der Beispielanwendung „MyERP“ dokumentiert und erörtert.

Die letzten zwei Kapitel befassen sich mit der Reflexion der Forschungsfrage und dem Fazit der erlangten Erkenntnisse aus der Sichtweise des Autors. Es werden auch mögliche Ansätze und Ideen für weitere Entwicklungen der iOS App beziehungsweise des JVx Frameworks geliefert.

1.4 Formulierung der Forschungsfrage

In dieser Masterarbeit soll versucht werden, folgende Forschungsfrage mit Unterfragen zu beantworten:

„Wie kann eine Multi-Touch optimierte native iOS App Erweiterung für das JVx ERP Applikation Framework designt und implementiert werden?“

Ausgehend von der Forschungsfrage stellen sich die Unterfragen wie folgt:

- „Wie und an welcher Stelle kann das JVx ERP Applikation Framework am besten mobil erweitert werden?“
- „Was versteht man unter einer nativen iOS App und wie wird eine solche Applikation und ihr Projekt aufgebaut?“
- „Wie kann das klassische JVx ERP-Userinterface an Multi-Touch angepasst werden?“

Diese Kernpunkte sollen methodisch wie im nachfolgenden Kapitel 1.5 beschrieben in dieser Masterarbeit erörtert werden.

1.5 Methodische Vorgehensweise

Aufgrund der Aufteilung der Masterarbeit in einen Theorie- und einen Praxisteil wird auch in der Vorgehensweise jeweils ein spezifisches Augenmerk gelegt.

Im ersten Teil, in dem die Theorie behandelt wird, wird die Schaffung der Wissensbasis in den Vordergrund gelegt. In diesem Schritt wird ein kurzer Überblick über klassische Paradigmen der Softwarenentwicklung und ihre Theorien gegeben. Das Augenmerk wird dann immer fokussierter in die Erweiterung von JVx und die Erstellung von iOS Apps gelegt, um den nachfolgenden Praxisteil bestmöglich umzusetzen.

Der Praxisteil der Arbeit, welcher als offener Quellcode auf der Plattform „sourceforge“ [7] unter dem Titel „JVx mobile“ zu finden ist, wurde mittels agiler Entwicklungsmethoden umgesetzt, welche von Allan Kelly sehr passend beschrieben wurden:

> *"Agile software development started as a term for a collection of lightweight methodologies, such as eXtreme Programming (XP), Crystal, Scrum, Dynamic Systems Development Method (DSDM) and others. It has evolved into something more than that. Agile embodies a philosophy about software development, a set of common beliefs and some practices. The methodologies themselves are more prescriptive about how development is done and contain specific practices." [8], S.21*

Im Speziellen wurde hier der Grundgedanke des iterativen Vorgehens angewandt. Es wurden daher mehrere Besprechungen für Feedbackrunden mit Mitarbeitern der SIB Visions abgehalten, um ein bestmögliches Ergebnis für das quelloffene Framework zu erlangen.

1.6 Relevanz der Aufgabenstellung

In der gegenwärtigen Zeit und auf dem jetzigen Stand der Technik stellt sich nicht mehr die Frage, ob eine mobile Version einer Software erstellt werden soll, sondern eher die Frage, ob die mobile Version nicht sogar Vorrang hat. Hierzu ein passender Hinweis von John Wiley, der sogar schon im Jahre 2008 folgende Aussage getätigt hat:

> *“WHY MOBILE FIRST? Here’s the elevator pitch: designing for mobile first not only prepares you for the explosive growth and new opportunities on the mobile internet, it forces you to focus and enables you to innovate in ways you previously couldn’t.” [9], S.5*

Der „mobile first“ Gedanke wird weiter sehr stark durch folgende Statistiken unterstrichen:

“PayPal is seeing up to $10 million in mobile payment volume per day” [9], S.9
“eBay’s global mobile sales generated nearly $2 billion in 2010” [9], S.9
“Google’s mobile searches grew 130% in the third quarter of 2010” [9], S.9
“Of Pandora’s total user base, 50% subscribe to the service on mobile” [9], S.9

Daher war es für den Autor dieser Arbeit naheliegend, eine JVx Erweiterung für die mobile Plattform iOS als Aufgabenstellung zu nehmen. Bekräftigt wird dies durch die Aussage von Daniel Jacobson:

“Your customers are quickly moving from a browser-based model to a model of consumption that involves consuming your services through apps on mobile devices.” [10], S.12

1.7 Abgrenzung

Das sehr große Themengebiet der Entwicklung mobiler Apps umfasst neben den sogenannten nativen App-Entwicklungen auch Hybrid- oder Cross-Plattform-Lösungen auf HTML5 Basis, die beispielsweise mit Sencha Touch [11, 12] oder jQuery Mobile [13, 14] erstellt werden. Diese werden dann mit Hilfe von App Containern wie beispielsweise PhoneGap [15] auch zu einer nativ installierbaren App gebündelt, basieren aber auf Webtechnologien. Hier wird vom Autor ausdrücklich eine Abgrenzung auf rein nativen, mittels Objective-C erzeugten Quellcodes und deren App-Erzeugung mittels Xcode gemacht.

Auch die Erweiterung des JVx Frameworks wird nur konzeptionell und in der Anwendung beim iOS Client behandelt. Die Implementierung in Java [16] erfolgte von Herrn Hofer Michael, B. Sc. in seiner parallelen Masterarbeit mit dem Titel „Design und Implementierung einer Multi-Touch optimierten Android Runtime Umgebung (Android App), um JVx ERP- Applikationen auf Smartphone Devices benutzerfreundlich lauffähig zu machen“, welche auch im Juni 2014 am FH-Technikum Wien veröffentlicht werden sollte.

2 Stand der Literatur

Wie der Begriff „Erweiterung“ schon impliziert, wird in dieser Masterarbeit auf eine bestehende Entwicklung zurückgegriffen. Dieser Stand der Softwareentwicklung, der für den theoretischen Teil von großer Bedeutung ist, wurde in der online-Dokumentation von der Firma SIB Visions GmbH [17] sowie der Masterarbeit von M. Sc. Stefan Wurm, mit dem Titel „Umsetzung einer auf Java EE 6 basierenden generischen Persistenzumgebung für das quelloffene Framework JVx“, [18] als Ausgangsbasis genommen. Neben der JVx Basis werden in diesem zweitem Kapitel auch fachliche Begriffe und der Stand der Fachliteratur sowie online-Dokumentationen des „IST“ Standes der iOS App-Entwicklung erörtert und konzeptionell beschrieben. Schon der Titel dieser Arbeit beinhaltet den Begriff „Framework“, mit welchem nun in der Theorie-Recherche begonnen wird.

2.1 Framework

Wird der Begriff „Framework“ in die deutsche Sprache übersetzt (Rahmenwerk), wird sehr schnell klar, dass es sich um einen Überbegriff handelt, der unterschiedliche Bedeutungen und Interpretationen haben kann. Daher ist es notwendig, für diese Masterarbeit eine gültige Formulierung zu finden, die als Basis herangezogen werden kann. Somit wird der Begriff „Framework“ mit der in der Softwareentwicklung etablierten Definition beschrieben:

> *“A framework is a generic design solution to a certain problem or a certain domain. The framework describes the different design elements involved in the solution, as well as their relations“. [19], S.3*

Eine weitere sehr passende Definition wurde auch von Martin Hoffmann wie folgt in seiner Masterarbeit formuliert:

> *“Der Framework ist eine Rahmenanwendung, die eine gewisse Infrastruktur für die Ausführung der einzelnen Module zur Verfügung stellt und die notwendige Schnittstellen liefert, um eine Kommunikation zwischen den Komponenten (Framework, Module, Plugins) zu ermöglichen.“ [20], S.24*

Diese Ansammlung an Funktionen und Modulen und vor allem auch deren Tests, die aktuell schon im JVx Framework integriert sind, ermöglicht es Entwicklern/Entwicklerinnen, schneller und mit weniger Kosten eine qualitativ hochwertige ERP-Lösung auch kleineren Unternehmen zu liefern.

2.2 KMU

In der Literatur wird häufig von „KMU“ als ERP-Anwender geschrieben. Um hier eine Klärung dieser Abkürzung vorzunehmen wird diese nun wie nachstehend gemäß der Wirtschaftskammer Österreich definiert. „KMU“ steht im Deutschsprachigen Wirtschaftsraum für kleine und mittlere Unternehmen, die auch Klein- und Mittelbetriebe genannt werden. [21]

Wie von der Wirtschaftskammer Österreich angegeben, wurden entsprechend der Empfehlung der EU-Kommission für KMU folgende Kriterien als maßgebliche Merkmale betrachtet:

- Anzahl der Mitarbeiter und Mitarbeiterinnen
- Umsatz oder Bilanzsumme
- „Eigenständigkeit“

Hieraus ergeben sich aus der EU-Empfehlung folgende Grenzwerte:

	Mitarbeiter	**Umsatz**	**Bilanz-summe**	**Eigenständigkeit**
Kleinst-unternehmen	bis 9	≤ 2 Mio EUR	≤ 2 Mio EUR	iA Kapitalanteile oder Stimmrechte im Fremdbesitz < 25 Prozent
Klein-unternehmen	10 bis 49	≤ 10 Mio EUR	≤ 10 Mio EUR	
Mittlere Unter-nehmen	50 bis 249	≤ 50 Mio EUR	≤ 43 Mio EUR	
Groß-unternehmen	ab 250	> 50 Mio EUR	> 43 Mio EUR	

Tabelle 1: Grenzwerte der KMU Kriterien [21]

2.3 ERP

Als Enterprise Resource Planning-Systeme (ERP) werden Programme bezeichnet, die zur innerbetrieblichen und externen Materialfluss-Planung eingesetzt werden. Ursprünglich als Stücklisten-Planungssoftware und Produktionsressourcen-Planungshilfsmittel gedacht, sind sie nun das Herzstück der IT-Abteilung jedes Unternehmens. [22]

Eine gute Beschreibung der ERP-Charakterisierung wurde von Norbert Gronau beschrieben, die wie untenstehend als Definition herangezogen wird:

“Wesentliches Merkmal von ERP-Systemen ist die Integration verschiedener Funktionen, Aufgaben und Daten in ein Informationssystem. Als minimaler Integrationsumfang ist eine gemeinsame Datenhaltung anzusehen.“ [22], S.4

2.4 Das quelloffene ERP Framework JVx

Da somit der Begriff eines Frameworks bekannt ist, wird nun das von der SIB Visions GmbH entwickelte JVx Framework näher beschrieben. Hierbei handelt es sich um ein auf Java [16] basierendes plattformunabhängiges ERP-System, das vor allem für KMU konzipiert wurde, die sich mehr Flexibilität wünschen, als dies große ERP-Softwarehersteller bieten können – wie beispielsweise SAP [23].

Wie schon von Stefan Wurm beschrieben handelt es sich bei JVx [17] um ein plattformunabhängiges Full Stack Open Source Framework [24], welches für unterschiedliche Plattformen User-Interfaces anbietet und somit ein Komplettpacket für ERP-Entwickler/Entwicklerinnen darstellt. [18] Die Entwicklung erfolgte von der Firma SIB Visions GmbH, welche das Framework als Open Source zur Verfügung stellt und somit in Zusammenarbeit mit der Gemeinschaft weiter entwickelt.

Der große Vorteil dieses Frameworks ist die Möglichkeit, sehr schnell für viele Plattformen zu entwickeln, da von Seiten des JVx schon folgende Darstellungsmöglichkeiten vor der Erstellung dieser Arbeit gegeben waren [25]:

- Swing (als Applet, Webstart oder Desktop)
- Vaadin (HTML5/Ajax)

Diese ermöglicht es einem Entwickler/einer Entwicklerin, eine ERP-Anwendung sehr schnell [26, 27] zu programmieren und mit diesem Framework diese Applikation dann als Desktop-, Web- oder mobile Applikation zu starten, wie dies von SIB Visions sehr schön beschrieben wurde:

“Sie schreiben den Source Code für die Benutzeroberfläche nur einmal und die Applikation kann als Desktop, Web oder Mobile Applikation gestartet werden.“ [25]

Wie schon erwähnt wurde bei der Entwicklung von JVx, welches ein „Full Stack“ (alle Features) Framework darstellt, stets auf Einfachheit und Modularität geachtet. Dieser pakethafte Aufbau ermöglicht es, JVx in einzelnen Paketen getrennt voneinander zu verwenden oder diese zu kombinieren. [28] Aktuell sind folgende Pakete Bestandteil von JVx, die Out-of-the-Box alle Anforderungen von Enterprise-Applikationen abdecken und sich aber leicht erweitern lassen [28] :

- User Interface (UI)
    - Generic User Interface (Generic UI)
    - Swing
    - andere UI Plattformen ...
- Generic Model
- Remote
- Server
- Persist

Eine dieser möglichen Erweiterungen sind neben der in dieser Masterarbeit beschrieben mobilen Erweiterung beispielsweise die von Herrn Hofer Michael, B. Sc. entwickelte Android-Version, auf die bereits im Kapitel 1. 7. „Abgrenzung“ verwiesen wurde, oder auch die von Herrn Piccardi Christopher, B.A. parallel entstehenden Masterarbeit welche die JavaFX-Erweiterung mit dem Titel „Implementierung des modernen, plattformübergreifenden Rich Internet Application Frameworks JavaFX als generische User Interface Technologie in das Open Source Framework JVx“ behandelt. [29]

2.4.1 Die JVx System-Architektur

Wie schon im vorhergehenden Kapitel beschrieben, ist das JVx Framework auf einer Mehrschichtenarchitektur aufgebaut. Wie in der Abbildung 1: System Architektur von JVx an der vertikalen Unterteilung zu erkennen ist, besteht JVx aus einer Datenhaltungsschicht (Data Tier), einer Anwendungsschicht (Enterprise Tier) und einer Darstellungsschicht (Client Tier) [28]:

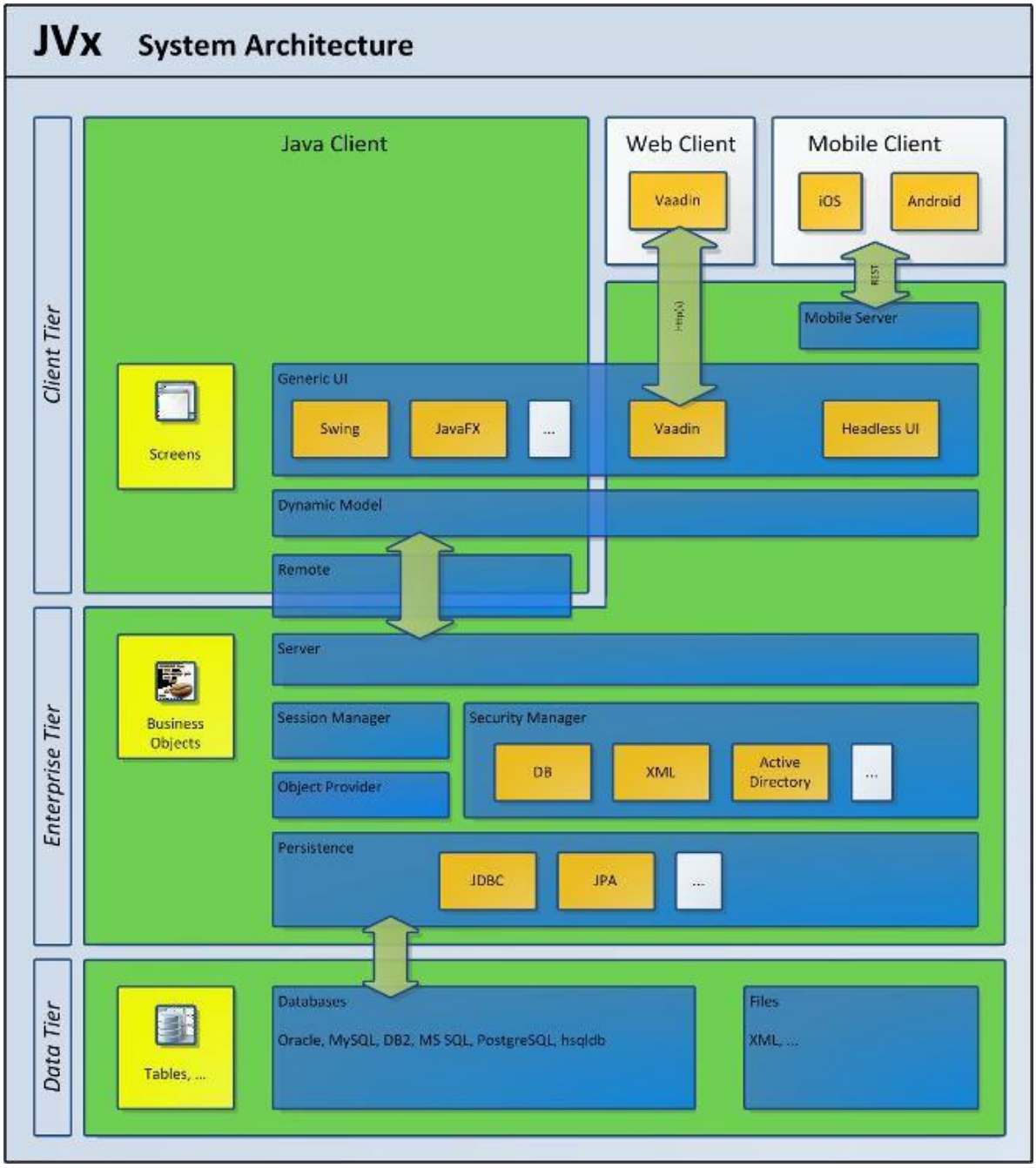


Abbildung 1: System Architektur von JVx [28]

Um einen besseren Überblick über die Tier-Aufteilung von JVx zu bekommen, werden nun die jeweiligen Schichten kurz erklärt:

Datenhaltungsschicht

Für die Datengrundlage, sozusagen als Informationsspeicher, wurde die Datenhaltungsschicht, auch Data Tier genannt, eingeführt, die sowohl relationale Datenbanksysteme wie beispielsweise Oracle, DB2, MS SQL, MySql, PostgreSql und HSQLDB unterstützt als auch spezielle Datenformate, die auf einfachen Datentypen basieren, wie beispielsweise XML und

XLS, verarbeitet. Somit wurde ein universell einsetzbarer Datenhalter dem Entwickler/der Entwicklerin zur Verfügung gestellt. [28]

Anwendungsschicht

Die in der Datenhaltungsschicht gespeicherten Daten werden in der Anwendungsschicht mittels einer API zur weiteren Verarbeitung zur Verfügung gestellt. In diesem Bereich des JVx Frameworks werden, angefangen von der Session-Verwaltung bis hin zur Kommunikationsverwaltung, alle Berechnungen zwischen der Darstellungs- und der Datenhaltungsschicht verarbeitet. Dies geschieht mittels eines Kommunikationsservers, der die Anfragen von der Darstellungsschicht (Client Tier) übernimmt, verarbeitet und dann das Ergebnis, aufbereitet für die Visualisierung, zurück an die Darstellungsschicht weiterreicht. [28]

Darstellungsschicht

Um die Daten eines ERP-Systems auch für reine Anwender/Anwenderinnen „benutzbar“ zu machen, ist eine Darstellungsschicht nötig, um die Daten zu sehen, aber auch um sie zu editieren oder sogar zu erweitern durch Neueingabe. Hierzu werden sogenannte Eingabemasken benötigt, die beispielsweise wie in Abbildung 2 dargestellt aussehen können:

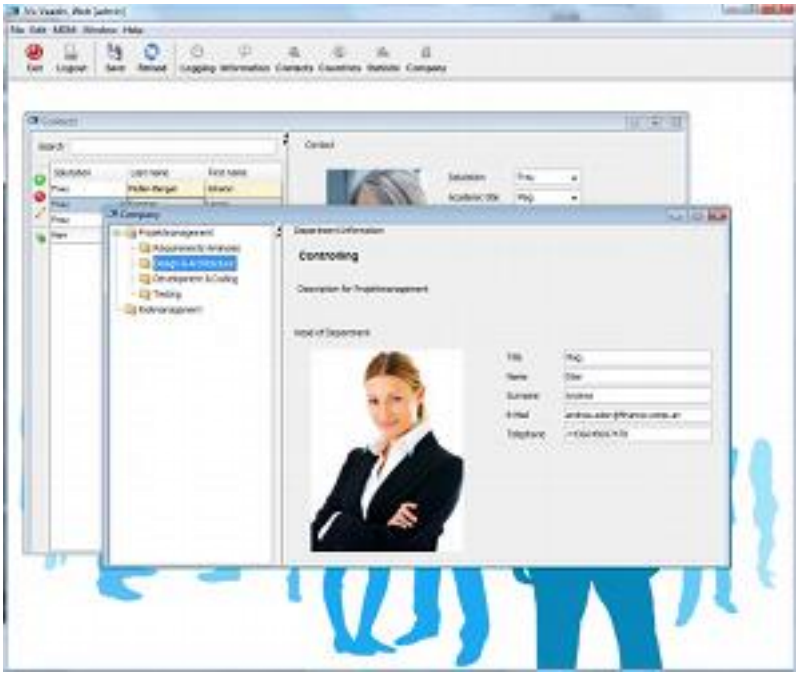


Abbildung 2: JVx Eingabemaske [25]

Die JVx Client Tier erlaubt hier eine breite Anzahl von UI-Technologien, auch Toolkits bezeichnet, für Desktop und Web. Aktuelle Umsetzungen für den Desktop sind hier Java Swing und SWT [16]. Im Web-Bereich werden moderne Javascript-Bibliotheken [16] und -techniken wie beispielsweise Vaadin [30], jQuery [13], oder extJS [11] unterstützt, um eine HTML-Interpretation zu ermöglichen. [28, 24]

2.4.2 Aktuelle mobile Tauglichkeit

Durch den Einsatz von HTML-Darstellungsformen ist es auch möglich, das JVx Framework sowohl auf Desktop-Rechnern als auch auf mobilen Geräten, die allerdings über einen Browser verfügen müssen, zu benutzen.

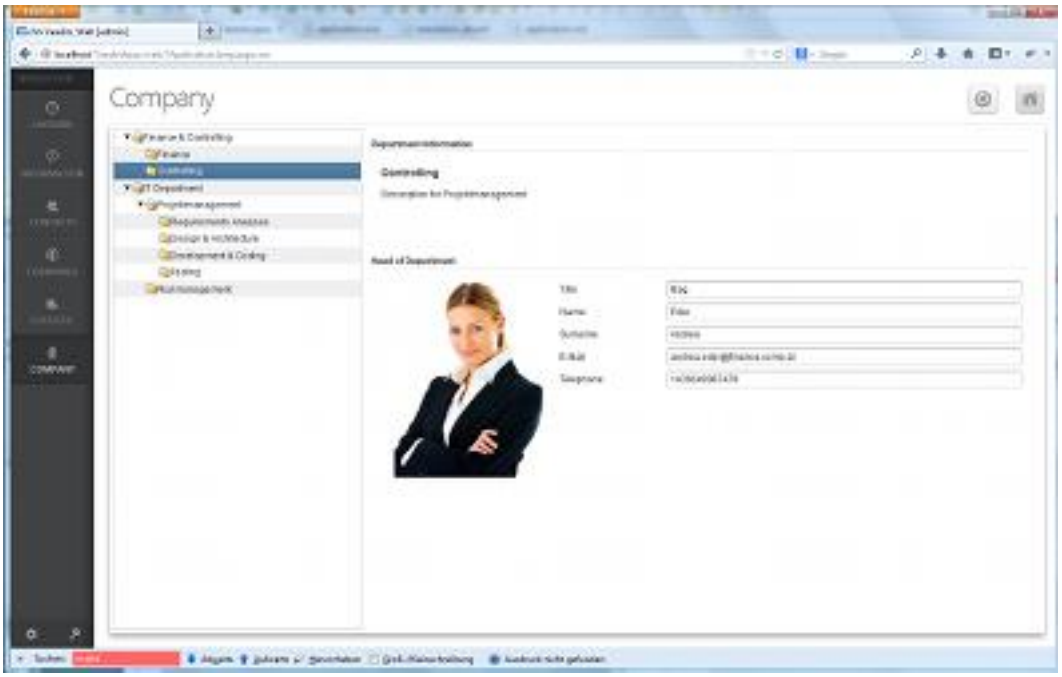


Abbildung 3: JVx Eingabemaske in HTML-Darstellung [25]

Dies ermöglicht einen gewissen Grad an mobiler Tauglichkeit, da auch hier Anpassungen bei der HTML-Interpretation, auflösungsspezifisch wie an Geräten, möglich wären. Doch seitens der SIB Vision wurde hier in der Aufgabenstellung dieser Masterarbeit ein weit generischerer Ansatz gesucht.

2.4.3 Hotspot für mobile Erweiterungen

Wie in Abbildung 1 zu sehen ist, verfügt die JVx Framework-Architektur über ein generisch aufgebautes Screen Layout-Modul (Generic UI), welches als Ausgangspunkt für eine Erweiterung analysiert wurde. Zu diesem Zeitpunkt haben die von der Anwendungsschicht aufbereiteten Daten noch keinen Daten-Overhead wie beispielsweise die HTML-Version, die diesen aber benötigt, um im Browser richtig visualisiert werden zu können. Dies wird in der schematischen Darstellung in Abbildung 1 als „Headless UI“ bezeichnet. Der Ansatz, hier einen „Mobile Server“ zu implementieren, der eine ausschließlich Daten beinhaltende Kommunikationsmöglichkeit mit dem JVx Framework ermöglicht, wurde zusammen mit Herrn Hofer Michael, B. Sc., konzeptioniert. In mehreren Iterationen wurde hier auch das Team von SIB Visions einbezogen, um dem Framework den bestmöglichen Mehrwert zu generieren. Die technische Umsetzung und deren wissenschaftliche Dokumentation entnehmen Sie

bitte der Masterarbeit von Herrn Hofer Michael, B. Sc., die schon im Kapitel 1.7 „Abgrenzung“ erwähnt wurde. Als Grundsatzentscheidung wurde hier eine Erweiterung mittels eines „RESTful API Web Services“ getroffen, welcher folgende entscheidende Vorteile mit sich bringt:

> *“A RESTful system is designed to separate out the UI concerns from the backend. Clients are decoupled from the server, thus allowing both to evolve independently.” [31], S.41*

Dadurch wurde eine Entkopplung von der serverseitigen, auf Java basierenden ERP-Anwendung und den mobilen Anwendungen geschaffen, die nun auch auf anderen Programmiersprachen basieren können.

2.5 RESTful API Web Services

Um eine Kommunikation zwischen einem Sender und einem Empfänger erfolgreich aufzubauen, benötigt es ein gemeinsames Protokoll für die Kommunikation. Hier setzen RESTful API Web Services auf das „HyperText Transfer Protocol“, welches meist in Kurzform als HTTP bekannt ist. Es stellt die Basis aller Web-Anwendungen und wird aus Sichtweise des Autors wie folgt passend beschrieben:

> *“HTTP is the application-level protocol for information systems that powers the Web.” [31], S.8*

Das http-Protokoll wurde ursprünglich von den drei Computer-Spezialisten Tim Berners-Lee, Roy Fielding und Henrik Frystyk Nielsen entwickelt. [31] Es stellt eine einheitliche Schnittstellen-Definition für Server und deren Anwender und Anwenderinnen zur Verfügung, die über ein Netzwerk Daten austauschen. Da dieses Protokoll nur als Grundlage der Kommunikation dient, wird vom Autor dieser Masterarbeit hier nicht die vollständige Funktionsweise erklärt, sondern mit diesem ausführlichen Zitat ein Überblick über das komplexe Protokoll gegeben:

> *“HTTP is designed for dynamically changing systems that can tolerate some degree of latency and some degree of staleness. This design allows intermediaries like proxy servers to intercede in communication, providing various benefits like caching, compression, and routing. These qualities of HTTP make it ideal for the World Wide Web, as it is a massive and dynamically changing and evolving network topology with inherent latency.” [31], S.8*

Dem Anwender/der Anwenderin dieses Protokolls ermöglicht das Erstellen, Auslesen, Aktualisieren oder Löschen von Daten über das Web. Diese Methoden der Datenmanipulation werden in der Fachsprache auch als die „CRUD“-Funktionen (Create, Retrieve, Update und Delete) genannt. Hier liefert uns das HTTP Protokoll folgende Äquivalente:

Data action	HTTP protocol equivalent
CREATE	POST
RETRIEVE	GET
UPDATE	PUT
DELETE	DELETE

Tabelle 2: Gegenüberstellung CRUD mit HTTP Funktionen [29], S.11

Diese vier Möglichkeiten entsprechen nicht einer vollständigen Möglichkeitsliste von http, aber genügen in den meisten Anwendungsfällen. Wie dieses Protokoll nun in der JVx iOS App zur Anwendung kommt, wird in Kapitel 4.1 „Anbindung der iOS App an das JVx Framework“ beschrieben.

Basierend auf dieser Grundlage wurde der für Maschinen verständliche Kommunikationsansatz von „APIs“ (Application Programming Interface) gebildet, der wie folgt zu verstehen ist:

> *“An API is a way for two computer applications to talk to each other over a network (predominantly the Internet) using a common language that they both understand.” [10], S.5*

Wie in der angewendeten Definition richtig beschrieben, findet diese über das Netzwerk stattfindende Kommunikation auf der Ebene der Anwendungen von Maschinen statt. Diese wird jedoch dann auf die Netzwerkebene der zuvor beschriebenen HTTP-Protokolle weitergegeben. Der zuvor sehr generelle Ansatz wurde für den Fall, dass die Kommunikation auch über das Internet geführt werden kann, wie folgt erweitert und verbessert:

> *„A Web API is a programmatic interface to a system that is accessed via standard HTTP methods and headers. A Web API can be accessed by a variety of HTTP clients, including browsers and mobile devices. Web APIs can also benefit from the web infrastructure for concerns like caching and concurrency.” [31], S.23*

Die in diesem Zitat erwähnten „headers“ beziehen sich auf die wie in Abbildung 4: Web API , S.6 abgebildeten Anfragen, die an eine Web-API geschickt werden, und deren Antworten. Für diese Masterarbeit wird dies aber als Basis angenommen und nicht weiter erläutert. Als letzten Punkt gilt es nun noch den Begriff „RESTfull“ zu definieren.

2.5.1 Erklärung RESTful

Der Zusatz „RESTful“ beschreibt, dass die API, die vom Webservice angeboten wird, eine bestimmte Logik hat, und dass auch meist mehrere unterschiedliche Ressourcen, wie beispielsweise Benutzer, Produkte mit deren Preisen usw., miteinander verbunden sind und abgefragt werden können.

> *“Having a REST API makes a web service “RESTful.” A REST API consists of an assembly of in interlinked resources.” [32], S.6*

Eine gut aufgebaute REST-API, welche Schematisch in Abbildung 4 zu sehen ist, kann heutzutage ausschlaggebend für den Erfolg von Webservices sein.

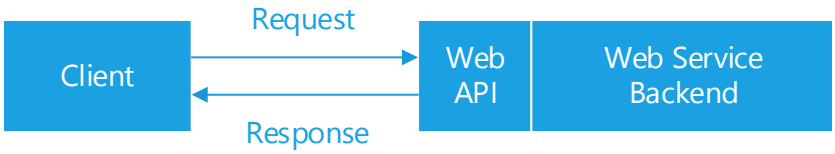


Abbildung 4: Web API [32], S.6, eigene Überarbeitung

Der Konkurrenzkampf um die Vorherrschaft im Netz und dessen Serviceleistungen führt dazu, dass die Qualität der Architektur und des Design von APIs ein weiterer Entscheidungsfaktor neben der eigentlichen Serviceleistung sind. [32]

2.5.2 Architektur & Design von RESTful API Web-Services

Um logische Verknüpfung von Daten auf der Eben einer API anzubieten, ist es notwendig einige Regeln einzuhalten, damit die jeweils angeforderten Daten auch diejenigen sind die, gewünscht und erwartet werden. Hier kommt nun der Grundsatz der Verwendung von „URI“ (Uniform Resource Identifier) zum Tragen. Definiert wurde dies in der Masterarbeit von Jose Sandoval wie folgt:

> *“A Uniform Resource Identifier, or URI, in a RESTful web service is a hyperlink to a resource, and it's the only means for clients and servers to exchange representations.” [33], S.10*

Das URI-Design bestimmt die Komplexität, aber auch die Variationsmöglichkeiten von modernen APIs, die wie in folgenden Beispielen erläutert auch „selbstsprechend“ sein können:
Es können sowohl wie in [32] angegeben
„*http://api.example.restapi.com/france/paris/louvre/leonardo-da-vinci/mona-lisa*“
als auch
„*http://api.example.restapi.com/68dd0-a9d3-11e0-9f1c-0800200c9a66*“
die gleichen Daten liefern.

Es sind auch beide URI-Designs zulässig. [32] Das URI Format kann generisch wie folgt beschrieben werden [34]:

URI = scheme "://" authority "/" path ["?" query] ["#" fragment]

Für einen guten und flexiblen REST API-Aufbau sollte der Einstiegspunkt aller Anfragen am besten eine Subdomain der Servicedomain sein. Ein Beispiel wäre hier „http://api.soccer.restapi.org“. Diese kann, wie von Mark Masse ausführlich in seinem Werk „REST API Design Rulebook“ [32], S.14-16 beschrieben, in folgende Anfragelevels weitergeführt werden, die absteigend immer weiter gefiltert werden und auch unterschiedliche Rückgabewerte liefern können:

http://api.soccer.restapi.org/leagues (Liste an Ligen oder alle Spieler/Liga)
http://api.soccer.restapi.org/leagues/seattle/teams (Liste an Teams oder alle Spieler/Team)
http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet/players (Liste an Spieler)

Dieser Basiseinblick in die Modellierung von RESTfull APIs genügt als Basis für diese Masterarbeit, und daher stellt sich nun die Frage, in welcher Darstellungsform die vom Server erfragten Daten nun an den Client zurückgeschickt werden. Hierzu werden nun die am häufigsten verwendeten Repräsentationsformate erläutert.

2.5.3 Repräsentationsformate von REST APIs

Um eine bessere Verarbeitung der Daten zu ermöglichen, wurden diese standardisiert. Hier haben sich in der Praxis die zwei Formate „XML“ und „JSON“ bewährt, die wie nachfolgend als Beispiele mit Notationsform angeführt sind:

XML	**JSON**
<user> *<username>Stefan</username>* *<password>Passwort</password>* *<link>/users/Stefan</link>* *</user>*	*{"user":* *{ "username":"Stefan",* *"password":" Passwort",* *"link":"/users/Stefan"* *}* *}*

Tabelle 3: Vergleich XML zu JSON Darstellung von Daten

Das ältere der beiden Formate ist das Extensible Markup Language–Format (XML), was übersetzt „erweiterbare Auszeichnungssprache“ bedeutet und noch heute oft Verwendung findet. [32], S.40

Das von Douglas Crockford entwickelte Format „JSON” (JavaScript Object Notation) ist die neuere und auch einfacher interpretierbare Notationsmethode. Sehr viele Programmiersprachen, vor allem JAVA und deren Scriptsprache JavaScript, unterstützen dieses Datencontainerformat, ohne dafür eine externe Erweiterung zu benötigen. JSON wurde daher zu der beliebtesten API Daten-Notation. [10], S.65-66

2.5.4 Sicherheitsanforderungen an moderne APIs

In sehr vielen Fällen werden APIs durch sogenannte API-Schlüssel gesichert. Diese sind meistens auf Entwickler/Entwicklerinnen oder Lizenznehmer/Lizenznehmerinnen einer API einzigartig zugeschnitten und fester Bestandteil der jeweiligen App. Diese einfache Erkennung eines Clients oder einer Gruppe ermöglicht es, diese auch zu sperren, um Missbrauch durch dieselben zu unterbinden. Dies stellt aber nur die Basis einer Sicherheitsstufe dar und sollte durch Benutzername und Password erweitert werden, wenn beispielsweise sensible Kundendaten übertragen werden. Durch diese auf Benutzerebene einzigartige Erkennung eines einzelnen Benutzers/einer Benutzerin ist eine sogenannte „Session-Based Authentication“ möglich. Hierbei wird auf Anwendungsebene durch Server-Logik entschieden, welche URIs ohne Session Key erlaubt sind und welche diesen zwingend erfordern. Ein Beispiel muss hier immer die Authentifizierung sein, da erst nach einer erfolgreichen Anmeldung dem Client der Session Key mitgeteilt wird. Dieser wird dann bei jeder weiteren Anfrage an den Server mitgeschickt. [10], S.77-78

Diese auf der Anwendungsebene eingesetzten Sicherheitsmechanismen werden zusätzlich auf der Ebene der Transport-Schicht erweitert, da sonst die Anfragen selbst geschützt wären, aber die übertragenen Daten nicht. Hier wird meistens mit der Methode “HTTP Over TLS”, die auch als HTTPS bekannt ist, gearbeitet. Dies kann als Basis-Sicherheitsvorkehrung für den Transport der Daten angesehen werden. Hierzu wird das „Transport Layer Security protocol (TLS)“, welches eine Erweiterung des „Secure Socket Layer protocol (SSL)“ ist, angewendet, um die Kommunikation in beide Richtungen zu sichern. [31], S.353-354

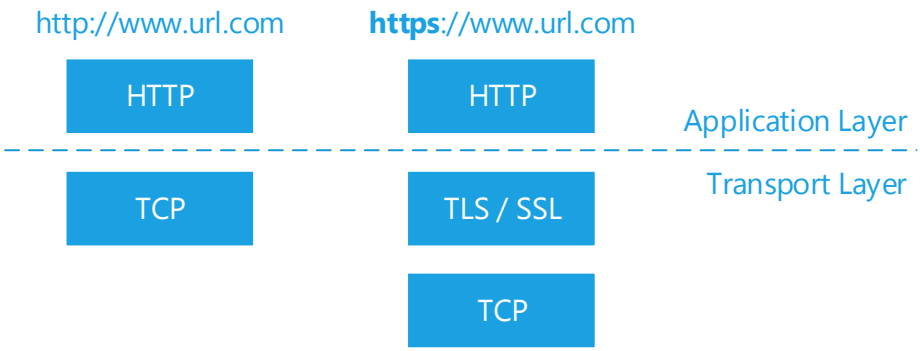


Abbildung 5: Schema der HTTPS Sicherheit [31], S.354, eigen Überarbeitung

Wie in Abbildung 5: Schema der HTTPS Sicherheit , S.354 zu sehen ist, basiert die Übertragung der Daten auf dem PCT-Protokoll. Diese auf der Transport-Ebene eingefügte Sicherheitsstufe wird von den meisten modernen Betriebssystem unterstützt und zur Verfügung gestellt. Hier ist dann für den Programmierer/die Programmiererin einzig das URI-Schema von „http://“ auf „https://“ zu ändern. Des Weiteren müssen dann serverseitig natürlich alle Zertifikate installiert werden, um HTTPS zu unterstützen; dies wird jedoch nicht in dieser Masterarbeit beschrieben, da die serverseitige Implementierung explizit ausgeschlossen wurde.

2.6 Erklärung der Grundlagen der iOS Plattform

In diesem Teil der Masterarbeit wird der Stand der Technik bezüglich der Entwicklung für die iOS-Plattform erörtert. Mittels der Erkenntnisse dieser „Basis“ werden dann aufbauend im Praxisteil Design- und Konzeptionsentscheidungen getroffen und in der Anwendung an dieser Stelle erklärt.

Das von Apple entwickelte mobile iOS-Betriebssystem bildet die Grundlage für die von Entwicklern geschriebenen Applikationen, welche als Apps bezeichnet werden.
Dies ermöglicht es, über den App Store oder über Enterprise-Applikation zertifizierte Versionen der Software für die mobilen Geräten wie das iPhone, das iPad oder den iPod touch zu entwickeln. Weitere Informationen und Beispiele werden auf der sehr großen Entwicklerplattform von Apple unter [35] bereitgestellt. Auch die Anmeldung für das Entwicklerprogramm oder die Verwaltung von Apps, die dann über den App Store von Apple vertrieben werden, sind dort bestens erklärt und werden daher hier nicht weiter behandelt.

2.6.1 Objective-C

Um einen besseren Einblick in den grundlegenden Aufbau von mobilen Applikationen für die iOS-Entwicklungsumgebung zu bekommen, muss zunächst die „Sprache“ erläutert werden,

mit welcher die Software geschrieben wird. Die Grundlage hierfür ist die auf der Programmiersprache „C“ entstandene objektorientierte Version namens „Objective-C“. Deren Entstehung wurde wie folgt in [36] gut beschrieben:

> *“Objective-C is a strict superset of ANSI C. C is a compiled, procedural programming language developed in the early 1970s at AT&T. Objective-C, which was developed by Brad J. Cox in the early 1980s, adds object-oriented features to C. It blends C language constructs with concepts that originated in Smalltalk-80.” [36], S.31*

Die Grundlagen von „C”, kombiniert mit „Smalltalk-80“ und dem Ansatz von objektorientierter Programmierung (OOP), führte zur neuen Programmiersprache und wurde von Apple im Oktober 2007 mit der Einführung von OS X Leopard mit der Version 2.0 veröffentlicht. [36] Die Grundlagen von OOP wie Kapselung, Vererbung und Polymorphie sind mit dieser auf „C“ basierenden Sprache möglich. Aber auch die Einbindung von „altem“, nicht Objekt-orientierten „C“ Quellcode ist weiterhin möglich, welches den Einsatz von bestehenden Quellcodefragmenten, Frameworks oder Bibliotheken ermöglicht. [37]

Zu den wichtigsten Sprachmerkmalen von Objective-C gehören laut [38], S.43 beispielsweise:

- Key-Value-Coding
- Key-Value-Observation
- Protokolle
- Delegates
- Kategorien
- Selektoren
- Blöcke

Viele dieser Merkmale sind in den unterschiedlichsten Programmiersprachen zu finden, werden aber teilweise unterschiedlich bezeichnet oder weichen leicht ab in der Anwendung. Die wichtigsten dieser Features werden im weiteren Verlauf dieser iOS-Übersicht kurz genauer erläutert.

Wie in [38] angegeben hat die Objective-C Syntax einige sehr auffällige und grundlegende Hauptmerkmale, die wie folgt aufgelistet wurden:

- *“Anweisungen können über mehrere Zeilen gehen und werden mit einem Semikolon abgeschlossen.“* [38]*, S.43*
- *“Methodenaufrufe erfolgen in eckigen Klammern.“* [38]*, S.43*
- *“Parameter von Methoden werden mit einem Doppelpunkt angegeben, und mehrere Parameter werden durch ihren Namen sowie jeweils einem weiteren Doppelpunkt voneinander getrennt.“* [38]*, S.43*
- *“Objektverweise beginnen mit einem Stern-Symbol.“* [38]*, S.43*

- *“Zeichenketten (Strings) werden in doppelte Anführungszeichen gesetzt und mit einem vorangestellten @-Zeichen begonnen.“* [38]*, S.43*

Als Beispiel dieser Syntax kann Listing 1: Funktion main in der Main.m oder Listing 16: URI Beispiel JVx herangezogen werden.

2.6.2 Cocoa

Um die Entwicklung von Applikationen basierend auf der Objective-C-Programmiersprache zu erleichtern und auch zu standardisieren, wurde Cocoa sozusagen als Container von Apple entwickelt. Wie in [39] beschrieben wurde ist Cocoa ungefähr seit dem Jahre 1986 verfügbar und wurde Teil der Macintosh-Entwicklungsumgebung, als Apple Inc. das auf dem NeXTStep basierende AppKit von NeXT in die eigene integrierte. Daraus entstand das Benutzeroberflächen- und Benutzerinteraktionsdesign, welches aus dem Desktop-Betriebssystem OS X bekannt ist.

> *“According to Apple, Cocoa is a set of object-oriented frameworks that provide a runtime environment for Mac OS X applications.” [39], S.1*

Es werden wiederkehrende User Interface-Elemente (UI) und Animationen wie Fenster, Textfelder, Buttons, Bilder oder auch Menüs usw. dem Entwickler/der Entwicklerin zur Verfügung gestellt. Dies ermöglicht es, Elemente der Benutzeroberfläche, mit einheitlichem Design und Verhalten, einfach in die zu entwickelnde Anwendung zu integrieren, da schon eine sehr große Anzahl an Elementen als Cocoa-Klassen implementiert wurde. [39]

Basierend auf Cocoa, welches für das OS X-Betriebssystem entwickelt wurde, entstand nun das auf Multi-Touch optimierte Cocoa-Touch Framework für die iOS-Plattform. [40]

Auf diese Masterarbeit bezogen sind die wichtigsten Komponenten dieses Touch Frameworks die nachfolgend angeführten, welche aus [40] entnommen wurden:

Komponenten Name	**Beschreibung**
Multi-touch events	*“These are the events, which are used to determine when a tap, swipe, pinch, double-tap has happened; that is, TouchesMoved, TouchesBegan, and TouchedEnded.”* [40]
Multi-touch controls	*“It is based on the Multi-Touch model, and determines when a user has placed one or more fingers touching the screen before responding to the action accordingly.”* [40]
View hierarchy	*“It deals with the MVC and the objects within the view.”* [40]

Alerts	*"Using the UIAlertView class, these are used to communicate with the user when an error arises, or to request further input."* [40]
Controllers	*"It is based on the MVC for presenting standard system interfaces and to provide much of the logic needed to manage basic application behaviors. For example, managing the reorientation of views in response to device orientation changes."* [40]

Tabelle 4: Cocoa-Touch Komponenten [40]

Diese Cocoa-Touch-Komponenten sind in einem eigenen Framework untergebracht, welches Cocoa-Framework genannt wird. Dieses „Wrapper"-Framework umfasst aber wiederum weitere Sub-Frameworks. Wie in [39] beschrieben, werden in iOS Frameworks von Apple wie folgt angewendet:

> *"Apple has grouped code and supporting files together in special folders (or bundles) called frameworks. Frameworks are like the libraries used on most platforms, but they are more flexible because they are folders rather than flat files. Frameworks can contain images, sounds, and movies. They can even contain other frameworks." [39], S.27*

Das für diese Masterarbeit wichtigste Framework mit dem Namen „Foundation" wird in 2.6.3 kurz vorgestellt.

2.6.3 Das Foundation Framework

Als Basis-Framework des Cocoa und Cocoa-Touch-Frameworks ist das „Foundation" Framework zu betrachten, auf welchem das Kern-Betriebssystem iOS und Mac OS X aufbauen.

> *"The Foundation framework is aptly named. It holds the objects that pretty much everything else is built upon. The Foundation framework is shared between Cocoa and Cocoa Touch." [39], S.28*

Dieses Grundgerüst an Basiklassen wie NSString, NSArray oder NSDictionary stammt noch von der schon erwähnten Integration von Nextstep. Die Namensgebung, welche mit „NS" beginnt, verweist hier noch auf die Historie. Als Basisklasse dient hier meistens die Klasse NSObject, auf welcher die meisten Model-Klassen aufbauen. Die genauere Bedeutung einer „Model"-Klasse wird im Zusammenhang mit dem Design-Konzept Model View Controller nun weiter erläutert.

2.6.4 Model View Controller

Objective-C und die dazugehörigen Cocoa-Frameworks beruhen auf dem Design-Ansatz, dass der Quellcode in drei Bereiche aufzuteilen ist. Cocoa und Cocoa-Touch wurden von

einem Konzept namens Model-View-Controller (MVC) abgeleitet, das eine logische Aufteilung des Quellodes in drei Komponenten mit sich bringt, die wie folgt in [39] beschrieben werden:

- *"Model: The classes that hold your application's data."* [39], *S.29*
- *"View: The windows, controls, and other elements that the user can see and interact with."* [39], *S.29*
- *"Controller: The part that binds the model and view together and contains the application logic that determines how to handle the user's inputs."* [39], *S.29*

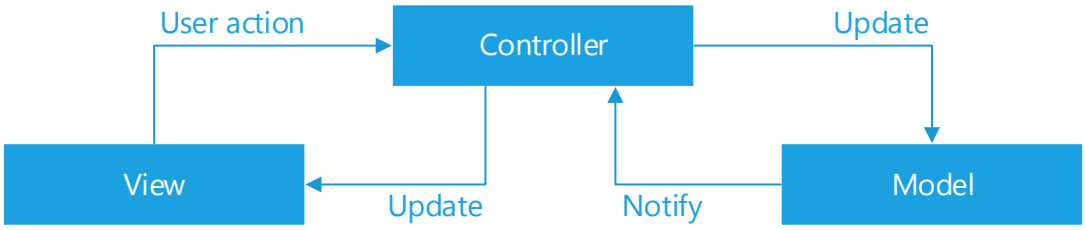


Abbildung 6: Model View Controller (MVC) iOS [41], eigene Überarbeitung

Wie in Abbildung 6: Model View Controller (MVC) iOS zu sehen ist, werden die Daten der Applikation streng von der Logik und den Userinterface-Elementen getrennt. Die ganz klar einzuhaltende Trennung wurde wie folgt gekonnt formuliert:

> "*The goal of MVC is to make the objects that implement these three types of code as distinct from one another as possible. Any object you write should be readily identifiable as belonging to one of the three categories, with little or no functionality within it that could be classified within either of the other two." [39], S.29*

Als Beispiel sollte kein Button (View) selbst Logik zum Ändern von Daten (Model) besitzen, sondern bei einem Klick (Touch) eine Funktion in einem Controller aufrufen, der dann wiederum die Daten ändern kann. Dieses Konzept des Model-View-Controller-Patterns trägt sehr stark dazu bei, dass wiederverwendbarer Quellcode entsteht, der dem Grundgedanken der objektorientierten Programmierung zugrunde liegt. [39]

2.6.5 Xcode

Mit der Einführung von OS X 10.3 wurde die integrierte Entwicklungsumgebung (Abkürzung IDE, von englisch „integrated development environment") Xcode von Apple Inc. eingeführt, welche Tools zur Entwicklung zur Verfügung stellt wie beispielsweise das „build system", den „code editor" und einen „debugger" siehe [42], S.37.

Wie nun in Abbildung 7: Xcode Entwicklungsumgebung" zu erkennen ist, umfasst die IDE mehrere Unterbereiche, welche in dieser Masterarbeit aber nicht im Detail erklärt werden

können. Xcode, mit welchem OS X und iOS Applikationen erstellt werden können, ist aktuell in der Version 5.1.1 gratis im Apple App Store verfügbar.

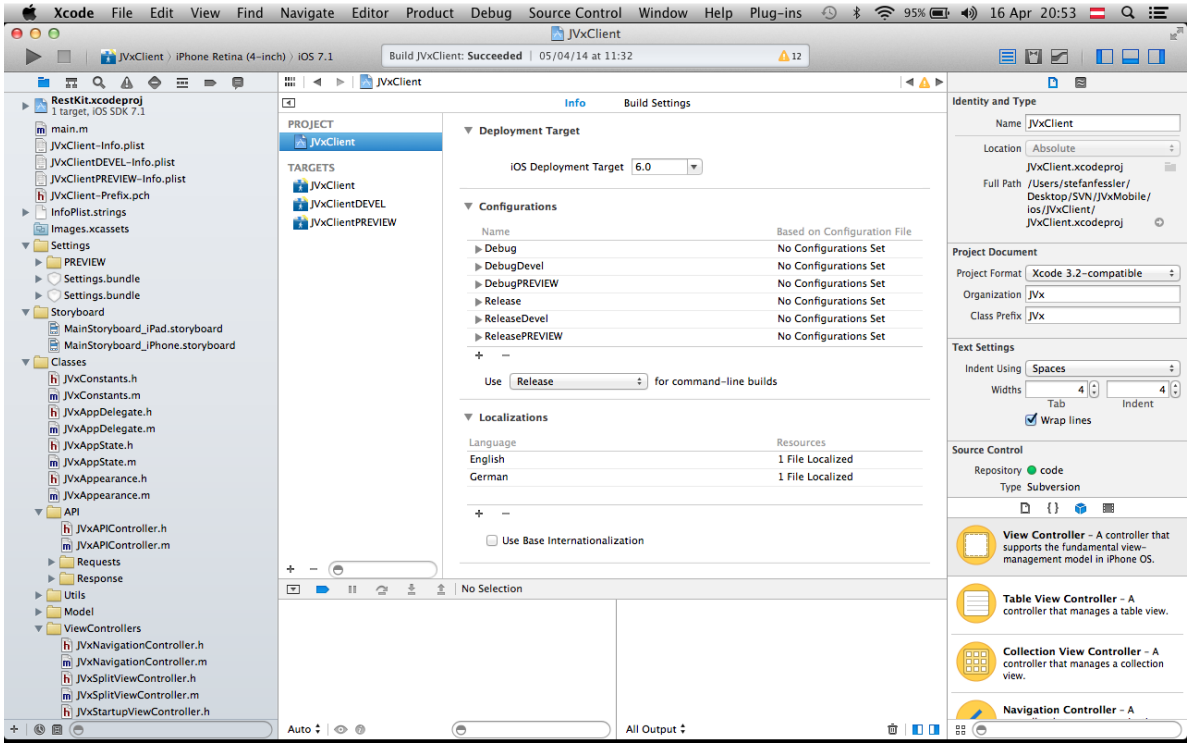


Abbildung 7: Xcode Entwicklungsumgebung

2.6.6 Architektur & Design von iOS Apps

Beim Anlegen eines neuen Projekts mit Xcode werden schon zu Beginn einige Einstellungen abgefragt. Eines der Kriterien ist die Projektvorlage, welche sehr vom Userinterface (UI) und dessen Navigationskonzept abhängt. Da dies sehr grundlegende Entscheidungskriterien einer iOS App sind, sollten durch sogenanntes „wireframing“ Mochups der zu erstellenden Anwendung gemacht werden.

> “*Wireframing is, of course, the sketching out of ideas for a product or web interface in skeletal form, depicting at a high level what it will do, how it might look, and how it will function.” [43], S.8*

Nachdem das Navigationskonzept gedanklich erstellt wurde, kann nun die passendste Projektvorlage ausgewählt werden, da uns Xcode hier einige Auswahlmöglichkeiten bietet.

2.6.7 Apple Projekt-Vorlagen

Als Vorlagen liefert uns Apple Inc. für die iOS Plattform unter anderen folgende Möglichkeiten [38]

- Empty Application
- Page-Based Application
- Single View Application
- Master-Detail Application
- Tabbed Application

Bis auf die Auswahlmöglichkeit „Master-Detail Application", welche nur für das iPad zur Verfügung steht, können alle Vorlagen, die in Abbildung 8: Xcode Standard Projektvorlagen" ersichtlich sind, sowohl für iPhone, iPad oder für beide Gerätegruppen ausgewählt werden, und werden dann als „universal" bezeichnet.

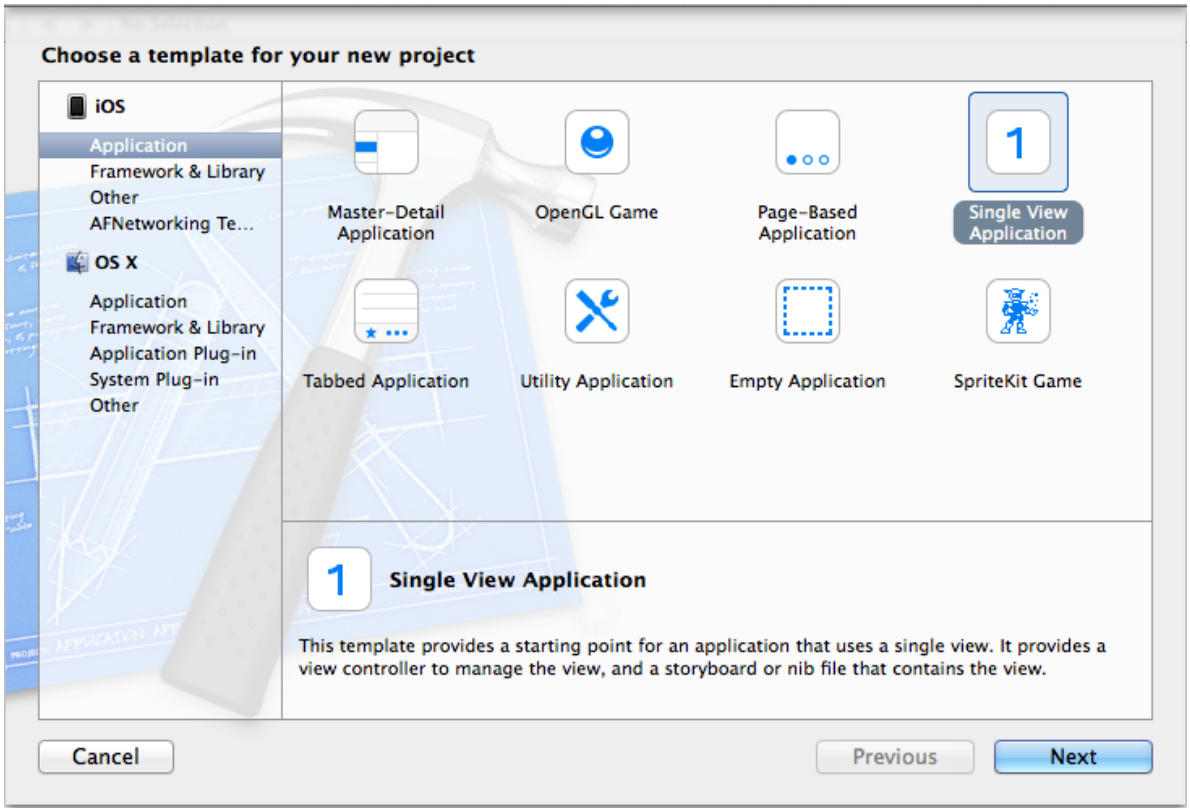


Abbildung 8: Xcode Standard Projektvorlagen

Wird beispielsweise die „Single View Application" ausgewählt, wird von Xcode aus der Projektvorlage das Projekt mit einem AppDelegate-, Viewcontroller- und zwei Storybaord-Files für iPhone und iPad erstellt. Auch Projekt-Einstellungen wie Versionsnummer, minimale kompatible iOS Versionsnummer, Statusbar-Farbe und so weiter werden, wie in Abbildung 9: Xcode Projektvorlage „Single View Applikation"" zu sehen ist, standardmäßig automatisch ausgefüllt.

2.6.8 Information Property List

Die beim Erstellen des Projektes angegebenen Einstellungen und weitere mit Standardwerten ausgefüllten Einstellungen werden unter anderem im „info.plist“-File abgespeichert.

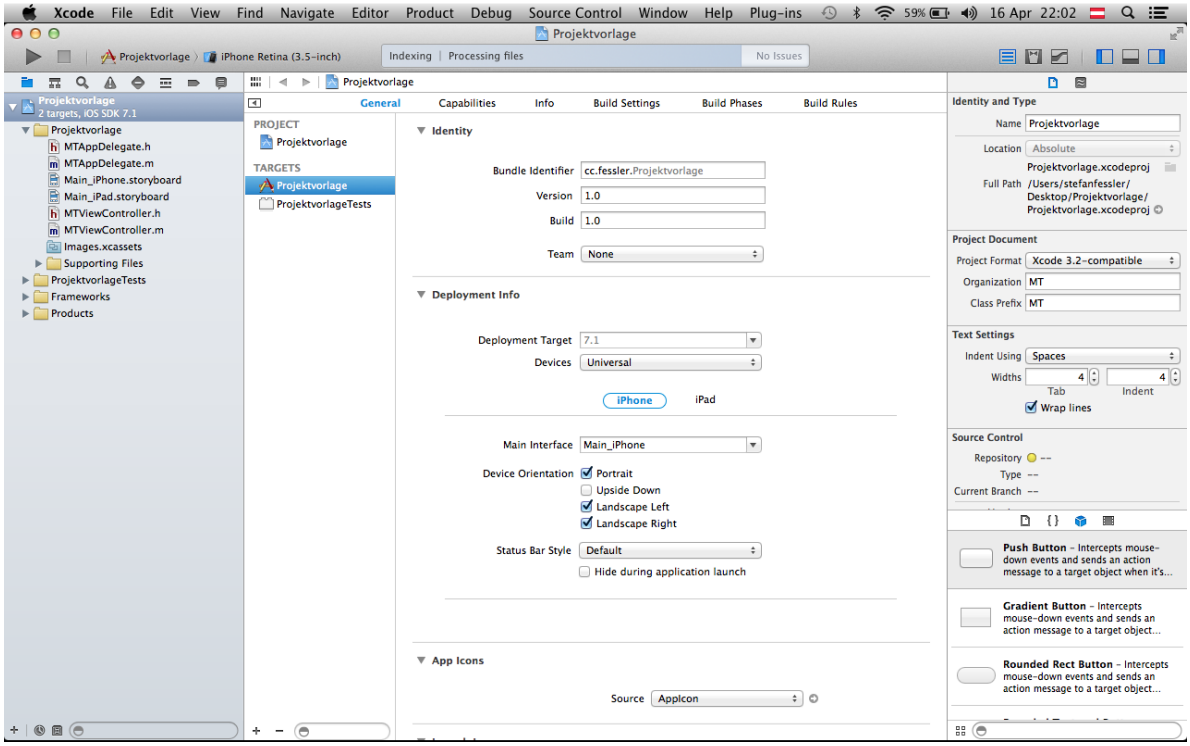
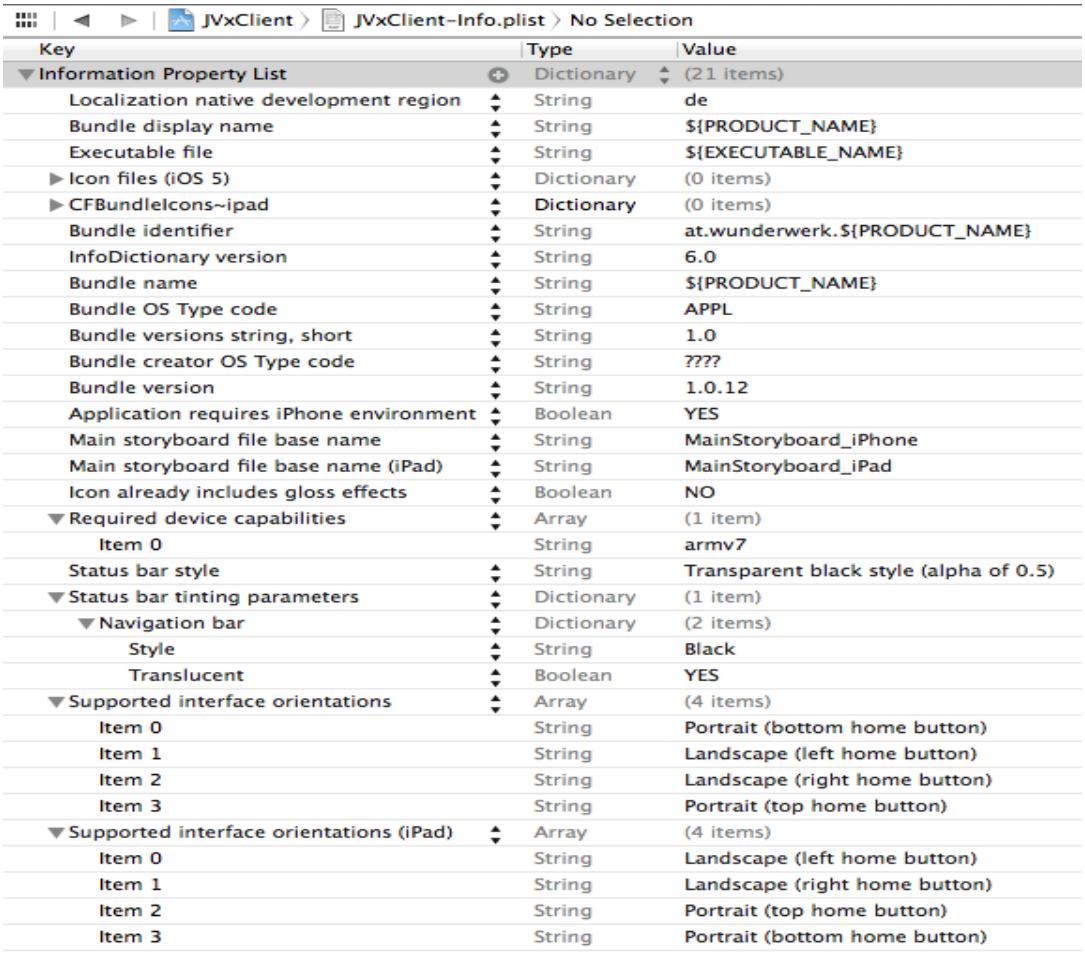


Abbildung 9: Xcode Projektvorlage „Single View Applikation“

Deses von Apple Inc. verwendete Fileformat wird „Property lists“ (plists) genannt und dient dem persistenten Ablegen von Einstellungen, und ist dem XML-Format sehr ähnlich. Die Erfindung von „Plist“ geht auf die Zeiten von OPENSTEP von NeXT zurück, in welcher im einfachen ASCII-Format Informationen persistiert wurden. [42], S.111

Wie in Abbildung 10: JVxClient-Info.plist Date“ zu sehen ist, werden an dieser Stelle neben der unterstützten Geräte-Orientierung, dem App-Produktnamen, dem Statusbar Style oder der zu unterstützenden Sprachregion sehr viele weitere Settings gespeichert. Diese können direkt in dem plist File oder in einigen Fällen auch über Xcode-Eingabemasken editiert werden, wie in Abbildung 8: Xcode Standard Projektvorlagen“ zu sehen ist. Als Beispiel dienen hier die Geräte-Orientierung-Optionen.

Die plist-Einstellungen können auch bei erweiterten Funktionen durch von Apple Inc. vorgegebene „Keys“ und deren dazugehörigen Werte erweitert werden. Hierzu dient das mit „+“ gekennzeichnete Symbol vor der Spalte „Type“.



Key	Type	Value
Information Property List	Dictionary	(21 items)
Localization native development region	String	de
Bundle display name	String	${PRODUCT_NAME}
Executable file	String	${EXECUTABLE_NAME}
Icon files (iOS 5)	Dictionary	(0 items)
CFBundleIcons~ipad	Dictionary	(0 items)
Bundle identifier	String	at.wunderwerk.${PRODUCT_NAME}
InfoDictionary version	String	6.0
Bundle name	String	${PRODUCT_NAME}
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle creator OS Type code	String	????
Bundle version	String	1.0.12
Application requires iPhone environment	Boolean	YES
Main storyboard file base name	String	MainStoryboard_iPhone
Main storyboard file base name (iPad)	String	MainStoryboard_iPad
Icon already includes gloss effects	Boolean	NO
Required device capabilities	Array	(1 item)
Item 0	String	armv7
Status bar style	String	Transparent black style (alpha of 0.5)
Status bar tinting parameters	Dictionary	(1 item)
Navigation bar	Dictionary	(2 items)
Style	String	Black
Translucent	Boolean	YES
Supported interface orientations	Array	(4 items)
Item 0	String	Portrait (bottom home button)
Item 1	String	Landscape (left home button)
Item 2	String	Landscape (right home button)
Item 3	String	Portrait (top home button)
Supported interface orientations (iPad)	Array	(4 items)
Item 0	String	Landscape (left home button)
Item 1	String	Landscape (right home button)
Item 2	String	Portrait (top home button)
Item 3	String	Portrait (bottom home button)

Abbildung 10: JVxClient-Info.plist

Weitere Informationen und Einstellungsmöglichkeiten sind auf der Apple Inc. Entwicklerseite unter [35] zu finden.

2.6.9 Startpunkt einer iOS App

Bei allen iOS Applikationen ist noch der historische Einstiegspunkt aller „C“ Applikationen zu sehen. Es wird wie bei dieser Programmiersprachen-Familie üblich die Funktion „main“ in der zuerst ausgeführten File „Main.m“ aufgerufen.

```
#import <UIKit/UIKit.h>
#import "JVxAppDelegate.h"

int main(int argc, char *argv[])
{  @autoreleasepool
  {
    return UIApplicationMain(argc, argv, nil, NSStringFromClass([JVxAppDelegate class]));
  }
}
```

Listing 1: Funktion main in der Main.m

Wie in Listing 1 zu sehen ist, wird beim Einstiegspunkt der Name der AppDelegate-Datei als eine der Rückgabeparameter-Komponenten zurückgegeben und somit aufgerufen. Dies startet den iOS App-Lebenszyklus. Da zu diesem Zeitpunkt des App-Starts erstmals der Objective-C-Quellcode ausgeführt wird, muss dieser bei ARC-Speicher-gesteuerten Apps darauf geachtet werden, dass dem Compiler durch einen „Autoreleasepool“ die Anweisung zur Speicheroptimierung gegeben wird.

2.6.10 Der App-Lebenszyklus

Der vereinfachte schematische Lebenszyklus (welcher in Abbildung 11: iOS App-Lebenszyklus [35] zu sehen ist) einer iOS App startet mit dem Zustand „nicht laufend“. Durch das Starten der App wechselt diese dann in den „Vordergrund“ und in den Zustand „aktiv“. Wird die App durch eine andere Anwendung „verdrängt“, wechselt diese in den „Hintergrund“, in welchen die Anwendung dann auch wieder bei Speichermangel oder manuellem Beenden in den „Suspended“ Zustand wechselt.

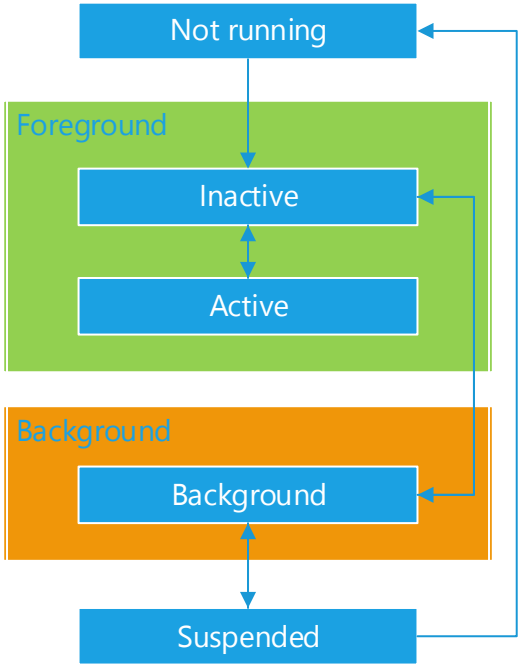


Abbildung 11: iOS App-Lebenszyklus [35], eigene Überarbeitung

Im sogenanntem „AppDelegate“ stehen dem Entwickler/der Entwicklerin nun folgende Methoden zur Verfügung, um auf die unterschiedlichen Zustände reagieren zu können:

```
#import "AppDelegate.h"
@implementation AppDelegate
```

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{  // Override point for customization after application launch.
  return YES;
}
- (void)applicationWillResignActive:(UIApplication *)application
{  // Sent when the application is about to move from active to inactive state. This can occur for certain types of temporary interruptions (such as an incoming phone call or SMS message) or when the user quits the application and it begins the transition to the background state.
  // Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES frame rates. Games should use this method to pause the game.
}
- (void)applicationDidEnterBackground:(UIApplication *)application
{ // Use this method to release shared resources, save user data, invalidate timers, and store enough application state information to restore your application to its current state in case it is terminated later.
  // If your application supports background execution, this method is called instead of applicationWillTerminate: when the user quits.
}
- (void)applicationWillEnterForeground:(UIApplication *)application
{  // Called as part of the transition from the background to the inactive state; here you can undo many of the changes made on entering the background.
}
- (void)applicationDidBecomeActive:(UIApplication *)application
{  // Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously in the background, optionally refresh the user interface.
}
- (void)applicationWillTerminate:(UIApplication *)application
{ // Called when the application is about to terminate. Save data if appropriate. See also applicationDidEnterBackground:.
}
```

Listing 2: AppDelegate Methoden

Nach dem Start einer App wird in der Methode “didFinishLaunchingWithOptions” die Möglichkeit eingeräumt, Startanpassungen vorzunehmen und erst danach den Rückgabewert „YES“ zu schicken, welches den erfolgreichen Start einer iOS Anwendung signalisiert.

Die weiteren Methoden wie

- „applicationWillResignActive“
- „applicationDidEnterBackground“
- „applicationWillEnterForeground“
- „applicationDidBecomeActive“
- „applicationWillTerminate“

ermöglichen es, auf die unterschiedlichen Zustände zu reagieren und erlauben es auch, beispielsweise Daten vor dem Beenden zu speichern oder die iOS App zu pausieren, wenn diese in den Hintergrund wechselt. [44], S.15

2.6.11 Storyboard

Wie schon in Kapitel 2.6.5, der Einleitung, beschrieben, beinhaltet Xcode mehrere Werkzeuge zur Erstellung von iOS-Anwendungen. Seit der Version 4.2 wurde der „Interface Builder“ (IB) in die IDE integriert, welcher zur Erstellung von Userinterfaces (UI) in einzelnen Files (.nib/.xib), aber nun eben auch mittels „Storyboarding“ im „storyboard“-Fileformat ermöglicht. „Storyboarding“ ermöglicht es, Benutzernavigation und die dazugehörigen einzelnen Views in einem „Container” darzustellen. Dieses Hilfsmittel erlaubt es ohne viel Quellcode, sondern mittels „Drag und Drop“, Standardelemente wie Buttons, Textfelder usw. in die dazugehörigen Views zu platzieren. Auch die Verknüpfung zum Quellcode und den entsprechenden Controller-Klassen ist mittels Mausklick möglich. [45], S.28

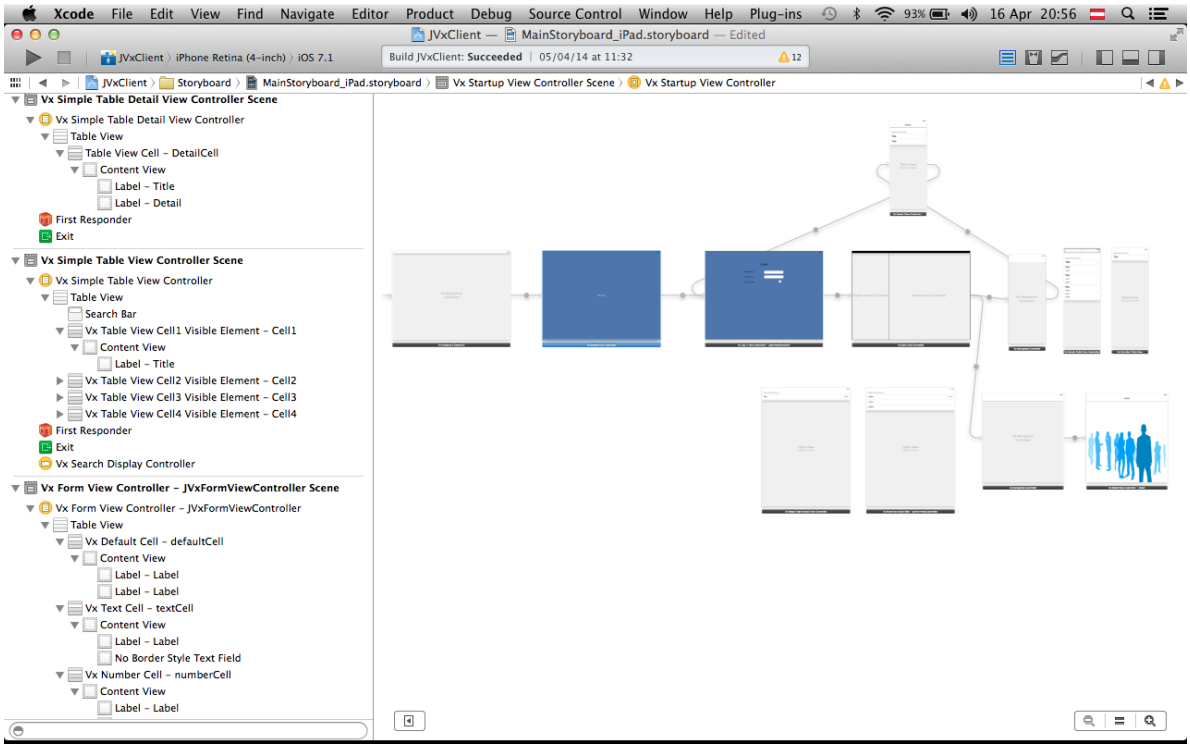


Abbildung 12: Storyboard in Xcode

Wie in Abbildung 12: Storyboard in Xcode“ im linken Bereich zu sehen ist, werden die Elemente der einzelnen Views eines ViewControllers hierarchisch dargestellt. Im rechten Bereich wird die graphische Repräsentation der Views visualisiert. Die möglichen Übergänge zwischen den Views werden mittels Verbindungslinien, sogenannten „Segues“, visualisiert. Diese werden noch genauer im Menüpunkt „Benutzerdefinierte Storyboard-Übergänge“ erläutert. [45]

Diese im „Interface Builder“ erzeugten Benutzerinteraktionselemente in den Views werden in den jeweiligen Storybard Container-Dateien meist für iPhone und iPad separiert gespeichert. Die Speicherung der Informationen erfolgt nicht – wie zu vermuten wäre – im zuvor

erwähnten plist-Format, sondern in der XML-Syntax, zu welcher, wie in Abbildung 13: „Storyboard XML„ zu erkennen ist, auch gewechselt werden kann.

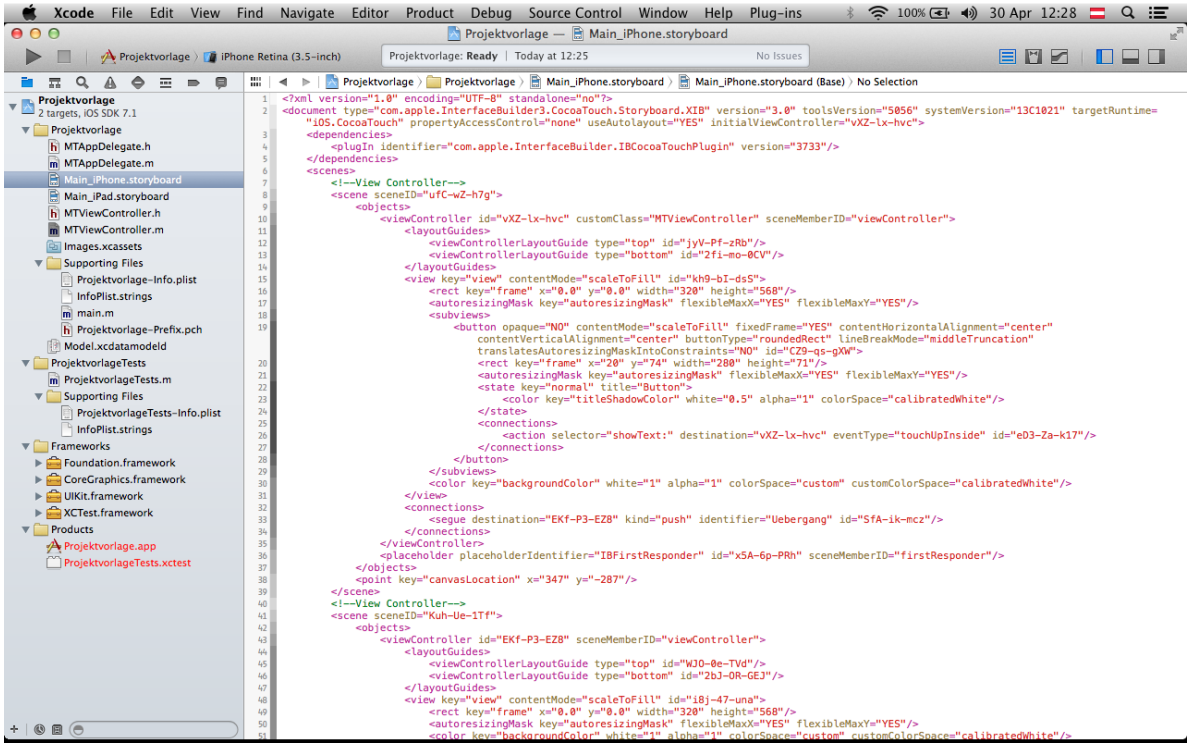


Abbildung 13: Storyboard XML

Xcode gibt dem Entwickler/der Entwicklerin die Möglichkeit einer Visualisierung, die wie folgt passend als sehr vorteilhaft beschrieben wird:

> *“Builder nicely renders it for you in the form of graphical elements that are much more pleasant to interact with than raw XML” [45], S.28*

Weitere Informationen können beispielsweise in der unter [45] angegebenen Literatur nachgelesen werden.

In der Storyboard-Datei wird auch der „initial ViewController“ festgelegt, welcher als allererster ViewController geladen und angezeigt wird. Durch diese Konfiguration kann sehr schnell und ohne viele Anpassungen im Quellcode das Startverhalten einer App verändert werden.

2.6.12 Benutzerdefinierte Storyboard-Übergänge

Xcode Storyboard bietet die Möglichkeit, sehr leicht und elegant Übergange zwischen den einzelnen Views zu erstellen und somit eine Navigationsstruktur aufzubauen. Standardmäßig bietet hier Xcode in der Version 5.1.1 drei Möglichkeiten der Segues.

In Abbildung 14: Segues in Xcode Storyboard“ ist die dritte Auswahl die Verwendung einer eigenen angepassten Navigation. Dies wird für komplexere Übergänge von einzelnen Views zu aufgeteilten UISplitViewController, wie dieser in Abbildung 24: Einstellungen am iPad mittels UISplitViewController“ zu sehen ist, verwendet.

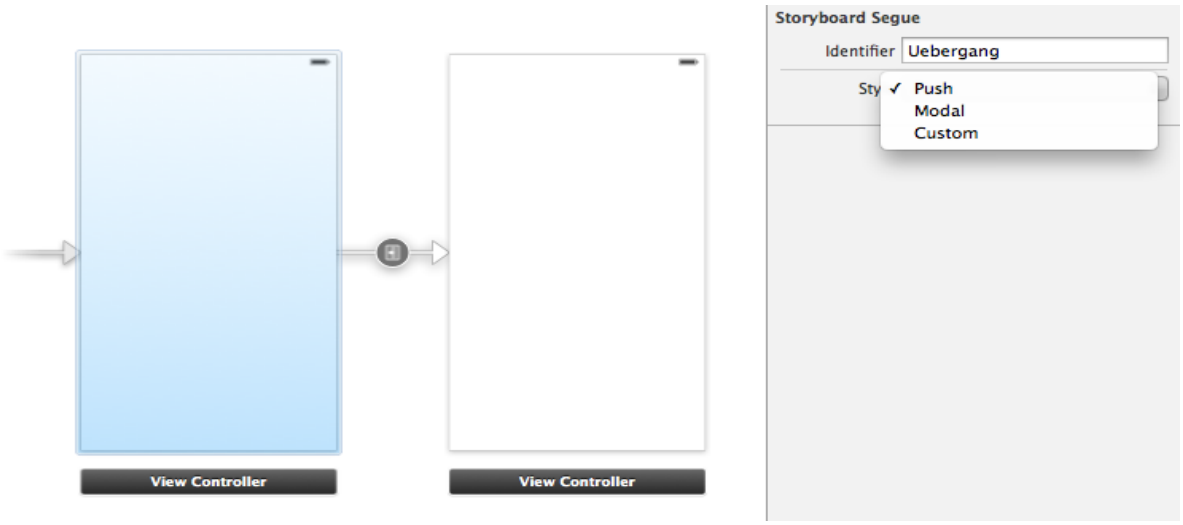


Abbildung 14: Segues in Xcode Storyboard

2.6.13 Angewandte Technologien und Design Patterns

Um einen besseren Überblick über die programmiertechnischen Designentscheidungen der entwickelten JVx iOS App zu bekommen, werden nun die in dieser Masterarbeit verwendeten Design Patterns kurz erklärt und im späteren Verlauf bei der Anwendung dieser nur mehr referenziert, um eine bessere Lesbarkeit zu gewährleisten.

Singleton

Das Singleton Pattern ist in mehreren Programmiersprachen bekannt und ermöglicht es, Daten einzigartig und von mehreren Stellen zugreifbar zu speichern. Dies wird ermöglicht beziehungsweise sichergestellt durch die Prüfung, ob eine Instanz einer Klasse schon einmal erzeigt wurde oder noch nicht. Wurde diese noch nie erzeugt, wird eine neue Instanz erzeugt, ansonsten wird immer die schon initial erzeugte Instanz zurückgegeben. Verständlich formuliert wurde dies von Carlo Chung wie folgt:

> *“A singleton class in an object-oriented application always returns the same instance of itself. It provides a global access point for the resources provided by the object of the class.” [46], S.97*

Kategorien

in der Programmiersprache Objective-C werden Erweiterungen einer Klasse als „Kategorie“ (Categorie in Englsicher Sprache) bezeichnet, diese ermöglichen es, auch bestehende Klassen nachträglich um zusätzliche Methoden zu erweitern. Dies ist sogar dann möglich, wenn diese zu erweiternde Klasse in einer Bibliothek als kompilierte Version ohne Source Code vorliegt. Durch Vererbung wirkt sich diese Erweiterung natürlich auch auf alle Kind-Klassen aus. Dies wird sehr oft für Hilfsmethoden genutzt um beispielsweise UIColor, welche nur eine Bestimmte Plätte an vordefinierten Farben beinhaltet, zu erweitern. Dadurch können nachträglich die für das Projekt benötigten Farbcodes der Klasse UIColor hinzugefügt. [47], S.286

Protokoll

Wie der Name „Protokoll“ schon sprachlich beinhaltet, sorgt dieses Design Pattern für die Einhaltung einer Konvention. In Objective-C werden durch ein Protokoll bestimmte Methoden optional oder verpflichtend einer zu implementierenden Klasse vorgeschrieben. Dies wird auch Methodenbündel genannt. [47] Als sehr einfaches Beispiel kann die „Example“-Klasse in Listing 3: Protokolle in Objective-C“ betrachtet werden.

```
@interface ExampleClass : NSObject <ExampleProtocol>
{
  // some properties
}
@end

@protocol ExampleProtocol
@required
- (void)methodA;
@optional
- (void)methodB;
@end
```

Listing 3: Protokolle in Objective-C

Vor allem die Verwendung von „required“ in Kombination mit einer Anzahl von Methoden ermöglicht es, auch Typen unbekannter Klassen, die „id“ genannt werden, auf deren Protokolleigenschaften zu prüfen.

> “*Eine solche Festlegung nennt man ein »formelles Protokoll«. Man kann damit so etwas Ähnliches wie Typisierung mittels Klassen erreichen, auch dann, wenn die eigentliche »Klasse« id ist“ [47], S.290*

Durch die Erweiterung der „Example“-Mutter-Klasse mittels „<ExampleProtocol>“ wird die Klasse mit diesem Protokoll (ExampleProtocol) konform. Der Compiler überprüft nun, ob die Einhaltung des Protokolls gegeben ist. Es werden nur die im Protokoll angegebenen und somit erforderlichen Methoden geprüft, sollten sie nicht implementiert sein, werden sie als Fehler markiert; die optionalen Methoden dagegen werden bei nicht-Implementierung nur mit Warnungen versehen, da diese erst zur Laufzeit auf eine Implementierung überprüft werden. [47]

Key-Value Coding

In Objective-C werden Variable, die von der Klasse NSString sind, wie in den folgenden zwei Varianten mit neuen Werten befüllt:

```
[myObject setValue1:@”one”];
myObject.value1 = @”one”;
```

Listing 4: Setzen von Attributs Werten ohne KVC

Diese Variante, Werte eines Attributes zu setzen, ist meist ausreichend. Doch für eine generische Implementierung ist es oftmals nötig, die Attribute einer Instanz-Variable beispielsweise in einer Schlaufe zu iterieren. Hierzu wird nun das „Key-Value Coding“ (KVC) zur Anwendung gebracht, welches nach folgendem Grundsatz funktioniert:

> *“The idea behind KVC is to allow us to refer to an object’s attributes by using strings that match attribute names or that match the names of some getter and setter methods.” [39], S.149*

Somit kann das oben angeführte Beispiel mittels KVC wie folgt generischer aufgebaut werden:

```
[myObject setValue:@"one" forKey:@"value1"];
NSString *valueOne = [myObject valueForKey:@"value1"];
```

Listing 5: Setzen und lesen von Attributs Werten mit KVC

Nun kann durch einfache Änderung der Key Variable ein anderes Attribut gesetzt werden. Dies ermöglich beispielsweise eine Iteration von 0 – x in einer Schlaufe. Diese sehr generische Art und Weise, Werte oder ganze Objekte mit „objectForKey“, zu setzen, ist schon in der NSObject Basis-Klasse implementiert und steht so allen von dieser erbenden Klasse zur Verfügung. Nur Klassen, die Listen beinhalten, wie NSArray oder NSSet, bilden hier oft eine Ausnahme. [39], S.149

Notifications

Da durch das MVC Pattern immer nur ein View mit einem Controller in Verbindung steht, wurde nach einer Möglichkeit gesucht, Controller bezüglich Änderungen von Daten durch einen anderen Controller informieren zu können. Hier bietet das Cocoa Touch Framework sogenannte „Notifications“ als Möglichkeit, die wie folgt zu verstehen sind:

> *“The Cocoa Touch framework implements the one-to-many, publish-subscribe model with NSNotificationCenter and NSNotification objects. They allow the subject and its observers to communicate in a loosely coupled manner. The communication can take place between them without either needing to know much about the other.” [42], S.168*

Diese Kommunikation zweier sonst nicht in Verbindung stehender Instanzen wird mittels dem NSNotificationCenter ermöglicht. Hierzu genügt es, wenn die Klasse A, die die Änderung an den Daten vorgenommen hat, eine NSNotification mit einem Namen, der als eindeutiger Key dient, schickt.

```
NSNotification *notification = [NSNotification notificationWithName:@"showLoginScreenToLockScreen" object:self];
NSNotificationCenter *notificationCenter = [NSNotificationCenter defaultCenter];
[notificationCenter postNotification:notification];
```

Listing 6: Senden einer NSNotification

Die Klasse B, die die nun informiert werden soll, kann sich dann für diese NSNotification mit dem gleichen Key als Observer registrieren.

```
NSNotificationCenter *notificationCenter = [NSNotificationCenter defaultCenter];
[notificationCenter addObserver:self selector:@selector(doShowLoginScreenToLockScreen:)
name:@"showLoginScreenToLockScreen" object:subject];
```

Listing 7: Empfangen einer NSNotification

Wird nun eine NSSotification verschickt und dadurch die Methode „doShowLoginScreenToLockScreen“ durch den Aufruf der @selector Funktion ausgeführt, bekommt diese als Input das Objekt „subject“ von der Klasse NSNotification. [42]

Dies ermöglicht es auch, an mehrere Observer gleichzeitig die gleiche NSNotification zu schicken, um beispielsweise Daten über mehrere Views in Echtzeit zu synchronisieren. Wichtig ist hier noch, dass sich die Klasse, die sich als Observer registriert, auch wieder entregistriert, wenn diese vom Lebenszyklus nicht mehr gebraucht und beispielsweise vom Speicher wieder freigegeben wird.

Actions

Wie in Kapitel 2.6.4 „Model View Controller“ erklärt wurde, werden Benutzerinteraktionen von Elementen, die im View vorkommen, wie beispielsweise UIButtons, nicht direkt mit Quellcode zur Änderung von Daten implementiert, sondern geben diese an den Controller weiter. Hier werden sogenannte „IBAction“ verwendet. Dies sind grundsätzlich normale „void“-Methoden, die keinen Rückgabewert, aber „IBAction“ als offiziellen Rückgabewert haben, so dass diese Methoden mit dem Storyboard verknüpft werden können. Hier steht wiederum „IB“ für Interface Builder.

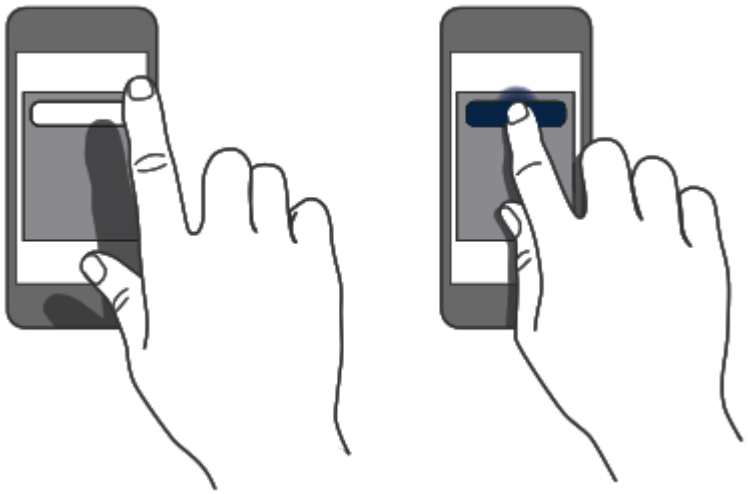


Abbildung 15: Benutzerinteraktionen durch Berührung [35], eigene Überarbeitung

Wird nun durch den Benutzer/die Benutzerin, wie in Abbildung 15 dargestellt, auf einem Touch Screen ein Button gedrückt, wird wie in [38] beschrieben die mit dem UIButton verknüpfte Methode “doSomeAction” ausgeführt.

```
- (IBAction)doSomeAction:(UIButton *)sender
{
  // do something
}
```

Listing 8: IBAction Methode

In der “doSomeAction“ Methode, welche nun in der Controller-Klasse ist, kann nun vom View getrennt Programmlogik ausgeführt werden, die entweder Änderungen im Daten Model oder im View bewirken. Dies wurde wie folgt treffend beschrieben:

> *“Actions are created in exactly the same way as other Objective-C methods, except that they must be declared using a special return type: IBAction. Actions must take a single*

argument (typically declared as type id). This argument is used to tell the method which interface item is calling it." [39], S.30

Die Verknüpfung des View-Elementes und der IBAction-Methode im Controller wird wie folgt in Xcode per „Drag und Drop" erstellt, wie dies in Abbildung 16: Erstellen einer Storyboard IBAction" zu sehen ist:

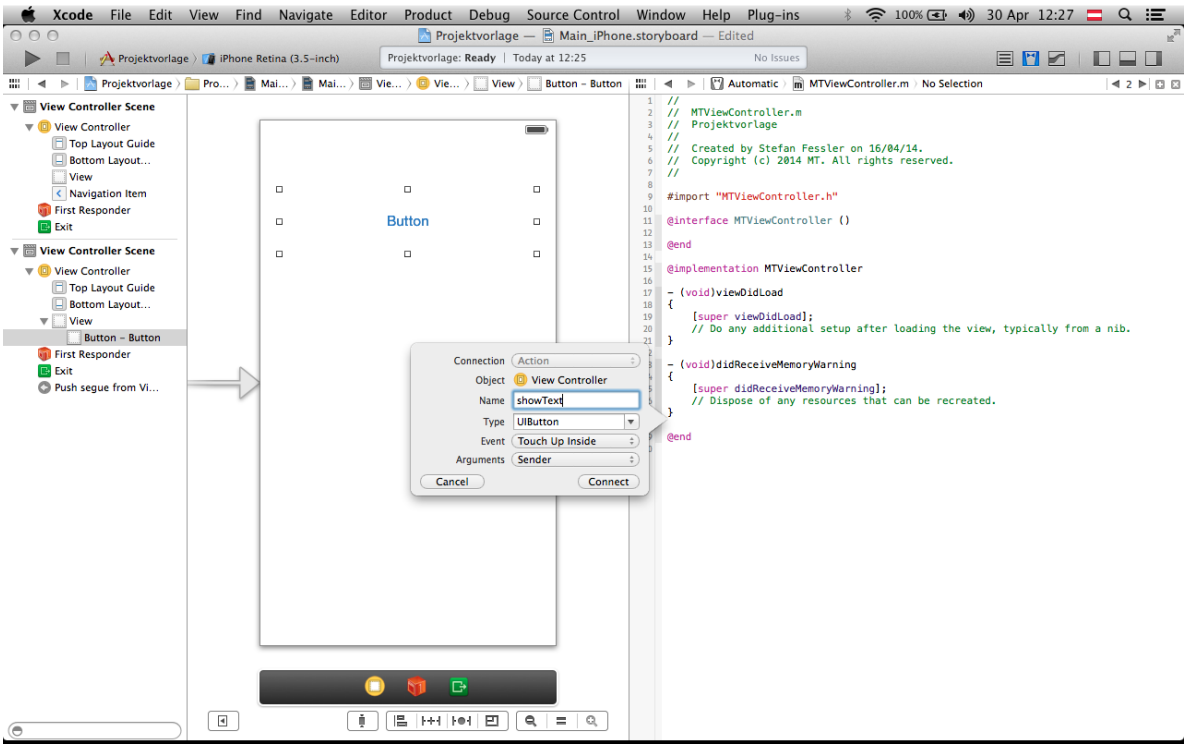


Abbildung 16: Erstellen einer Storyboard IBAction

Solche „IB"-Methoden können aber auch von anderen View-Elementen wie beispielsweise UISlider genutzt werden.

2.6.14 Speicherverwaltung

Da Objective-C im Gegensatz zu anderen objektorientierten Programmiersprachen keinen „Garbage Collector" besitzt, wurde durch „Manual Reference Counting" der Speicher für die benötigten Daten verwaltet. Dies führte sehr häufig zu App-Abstürzen oder zu sehr hohem Speicherverbrauch der mobilen Anwendungen. Seit der Einführung von „Automatic Reference Counting" (ARC) wurde diese bei Entwicklern sehr unbeliebte und mühsame Speicherverwaltung mit ihrer Komplexität dem Compiler überlassen. [48], S.52

3 RESTful JVx Erweiterung

Nachdem die Grundlagen für die Erstellung des JVx iOS Clients beschrieben wurde, wird nun im Zweiten Teil der Masterarbeit der Praxisteil, der die Anbindung an die JVx-Plattform beschreibt, erläutert. Im ersten Abschnitt der RESTful JVx-Erweiterung werden die in Zusammenarbeit mit Herrn Hofer Michael, B. Sc. und der SIB Visions erarbeiteten Grundsatzentscheidungen der API erklärt.

3.1 Designentscheidungen der Schnittstelle

Wie in 2.4.3 Hotspot für mobile Erweiterungen beschrieben, wurde der Ansatz, einen „Mobile Server“ zu implementieren, umgesetzt. Diese neue Kommunikationsmöglichkeit mit dem JVx Framework ermöglicht nun eine ausschließlich Daten beinhaltende Kommunikation von mobilen Clients durch diese neue API. Diese schon erwähnte Entkopplung von der serverseitigen, auf Java basierenden ERP-Anwendung und den mobilen Anwendungen wird nun etwas genauer beschrieben.

In mehreren Besprechungen wurde iterativ wie in 2.5 beschrieben ein RESTful API Web Services designt, der nun von den mobilen Anwendungen Anfragen mittels POST-Parametern entgegennimmt, verarbeitet und auf diese auch antwortet.

Schon nach den ersten Besprechungen wurde sehr schnell erkannt, dass die neue API eine großen Anzahl an unterschiedlichen Anfragen mit unterschiedlichen Parametern beinhalten wird, die außerdem flexibel erweiterbar bleiben sollen. Aus diesem generischen Ansatz entstammend wurden die neu geschaffenen URIs wie in 2.5.2 Architektur & Design von RESTful API Web-Services beschrieben designt. Zu Beginn einer jeden Client Session wird der sogenannte „Startup“-Request an den JVx Server geschickt. Eine vollständige Liste aller API URI-Typen sind unter 4.7.3 API Controller zu finden.

Schon bei dieser ersten Kommunikation zwischen mobiler App und dem JVx Server werden die ersten Parameter entsprechend der Beschreibung 2.5.3 Repräsentationsformate von REST APIs im JSON-Format an den Server geschickt:

```
NSMutableDictionary *jsonDictionary = [[NSMutableDictionary alloc] initWithObjectsAndKeys:
    [JVxUtils applicationName], @"applicationName",
    [JVxUtils osName],          @"osName",
    [JVxUtils osVersion],       @"osVersion",
    [JVxUtils appVersion],      @"appVersion",
    [JVxUtils screenWidth],     @"screenWidth",
    [JVxUtils screenHeight],    @"screenHeight",
    [JVxUtils langCode],        @"langCode",
    [JVxUtils deviceLocale],    @"deviceLocale",
```

```
    [JVxUtils deviceType],      @"deviceType",
    [JVxUtils deviceModel],     @"deviceModel",
    [JVxUtils openUDID],        @"deviceUUID",
    nil];

// preview app mode settings
if ([JVxUtils isPreviewMode])
{
  [jsonDictionary setValue:@"PREVIEW" forKey:@"appMode"];
}
```

Listing 9: Startup Parameter

In Listing 9: Startup Parameter ist zu erkennen, dass der Applikationsname der JVx-Anwendung am Server mitgeschickt wird, da am Server mehrere Instanzen simultan laufen können. Auch die Sprache und Gerätetypen werden hier dem JVx Framework mitgeteilt, so dass dem User beispielsweise der Login Screen in der richtigen Sprache oder später auch die sprachlich angepasste Menüstruktur ausgeliefert werden kann.

Aber auch beispielsweise der sicherheitstechnische Parameter „deviceUUID“, der eine eindeutige Geräteidentifikation ermöglicht, wurde berücksichtigt, um diesen gegebenenfalls sperren zu können. Dies wurde allerdings serverseitig noch nicht implementiert.
Als weiterer Aspekt für die Sicherheit wird, wie in 2.5.4 Sicherheitsanforderungen an moderne APIs beschrieben, für alle Kommunikationen HTTPS unterstützt.

Da der Server oftmals schneller mit Updates oder Erweiterungen ausgestattet wird und mobile Kunden und Kundinnen nicht immer App-Updates durchführen, wurde auch ein „osVersion“-Parameter vorgesehen, so dass der Server auch weiterhin mit älteren Client-Versionen kommunizieren könnte. Dieser Kompatibilitätsmodus wurde aber ebenfalls serverseitig noch nicht umgesetzt.

Ist die JVx-App in der „PREVIEW“-Konfiguration, wird dieser Wert auch als Parameter „appMode” gegebenenfalls mitgeschickt. Konfiguration werden unter 4.5 Targets & Schemes eines Xcode-Projekts genauer beschrieben.

3.2 Vorteile der RESTful Integration in das JVx Framework

Durch die Integration von REST in das JVx Framework wurde es ermöglicht, dass auch Plattformen, die nicht auf der Programmiersprache JAVA basieren, die Vorteile des JVx Frameworks nutzen können.

Der praktische Teil der Masterarbeit, welcher die iOS-Plattformerweiterung umfasst, wird nun nachfolgend erörtern, welchen Mehrwert der neue RESTful-Service des JVx Frameworks in der Praxisanwendung mit sich bringt.

4 Entwicklung der generischen iOS App

Basierend auf den Grundlagen, welche in den bisherigen Kapiteln erörtert wurden, hat die Entwicklung einer eigenständigen generischen iOS Applikation stattgefunden, wie nun folgend in mehreren Schritten erklärt wird.

4.1 Anbindung der iOS App an das JVx Framework

Beginnend mit der Anbindung, welche schon im Kapitel 3 „RESTful JVx Erweiterung" theoretisch ausgelegt wurde, wird nun iOS- beziehungsweise Objective-C-spezifisch die Verarbeitung und Anbindung der auf JSON basierenden Daten der neuen JVx Framework API erläutert.

4.1.1 JSON Support der iOS Plattform

Mit der fünften iOS Version wurde die NSJSONSerialization Klasse eingeführt welche eine sehr einfache und elegante Umwandlung der rohen JSON-Daten in ein NSDictionary oder NSArray. Dieser Vorgang wird „Serialization" genannt und kann auch umgekehrt angewendet werden, was dann als „Deserialization" genannt wird. [49]

```
NSError *error = nil;

// NSData to NSDictionary or NSArray
id json = [NSJSONSerialization JSONObjectWithData:self.data options:kNilOptions error:&error];

// NSDictionary or NSArray to NSData
NSData *data = [NSJSONSerialization dataWithJSONObject:json options:NSJSONWritingPrettyPrinted error:&error];
```

Listing 10: iOS JSON Serilization und Deserilization

Diese sehr einfach wirkende, aber immens wichtige JSON-Methode wurden von Kyle Richter und Joe Keeley wie folgt beschrieben:

> *"NSJSONSerialization's method JSONObjectWithData:options:error: expects as parameters the data to be serialized, any desired options (for example, returning a mutable array instead of a regular array), and a reference to an NSError in case there are any parsing errors." [49], S.150f*

Dieses einfache Umwandeln von Rohdaten (NSData) in iOS Objekte (NSDictionary oder NSArray) beinhaltet aber noch keine Umwandlung in die gewünschten Model Daten-Objekte.

4.1.2 Natives Objekt Mapping

Diese nun zuvor von NSData in NSArray oder NSDictionary umgewandelten Informationen werden nun weiter in die Objekt-Klassen umgewandelt. Bei diesen nun umgewandelten Daten in die iOS Basisklassen kann dann wie in [49] beschrieben mittels Key-Value Coding auf die Werte eines NSDictionary zugegriffen werden, um den eigentlichen Text des NSString zu bekommen.

```
NSString *stringValue = [dict objectForKey : @"key" ];
customClass.attribute = stringValue;
```

Listing 11: iOS Natives Objekt Mapping

Während der Masterarbeit wurden die Keys, aber auch die Datentypen mehrfach iterativ verändert und optimiert. Dies führte zu großem Aufwand beim Nachkorrigieren des Objekt Mapping, daher wurde hier nach einer eleganteren Methode gesucht.

4.1.3 RestKit

Während der iterativen Konzeption der Anbindung des iOS Clients an die JVx API wurde nach mehrfachem Testen mit unterschiedlichen externen Bibliotheken RestKit gefunden:

> *“RestKit is an Objective-C framework for iOS that aims to make interacting with RESTful web services simple, fast and fun. It combines a clean, simple HTTP request/response API with a powerful object mapping system that reduces the amount of code you need to write to get stuff done.“ [50]*

Eines der größten Vorteile von RestKit ist, dass sich der Entwickler/die Entwicklerin nicht mehr um das parsen der einzelnen Attribute und deren Umwandlung in den Entsprechenden Datentyp kümmern muss. Die Implementierung von RestKit beruht sehr auf Key-Value Coding und erlaubt eine schnelle und typkonforme Umwandlung von JSON-Objekten. Die Implementierung der Anwendung von RestKit ist in der Klasse JVxJSONUtils zu finden, welche das Mapping und retour-Mapping für die einzelnen Daten Model-Klassen in einer „Utility“ zusammenfasst. [50]

4.2 Anforderungen an Touch optimiertes mobiles Design

Folgenden gravierenden Unterschied zwischen einem Desktop-PC und einem mobilen, meist auf Touchscreen basierenden Gerät, den es zu beachten gilt, beschreibt Adrian Mendoza:

> "*Without the pinpoint accuracy of a mouse, forms and complicated selections need to be completely redesigned. As a result, it has focused us on making smaller pages, cleaner information layouts, and larger user interface elements. We will have to optimize user experiences for mobiles, making them better for our users. Goodbye mouse, hello touch.*" *[51], S.15*

Dieser grundlegende Einfluss auf das Benutzerinteraktionsdesign (UI Design) führt weiter zu einer komplett anderen minimalen Größe der Interaktionselemente. Hierzu werden wesentlich größere Bereiche für einzelne Elemente, beispielsweise einer Eingabemaske oder eines ganzen Navigationskonzeptes, benötigt, welche folgender Grundlage entsprechen sollte:

> *"Since arguably all modern UIs should be designed for touch, you should consider designing desktop UI for touch as well. iOS recommends using a 44 × 44 point layout grid for comfortable touch support, whereas Android recommends 48 × 48 points." [52], S.213*

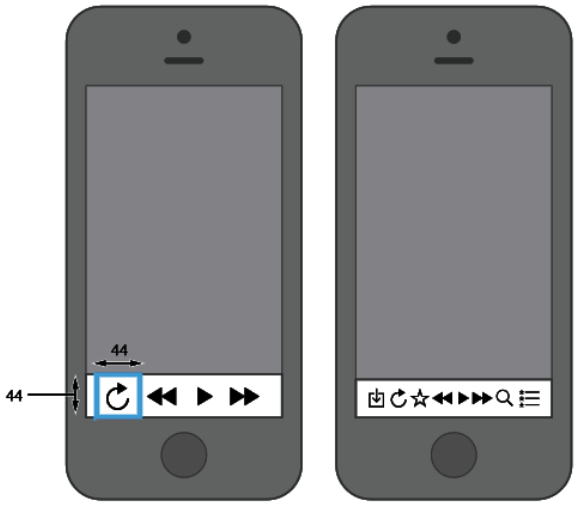


Abbildung 17: Layout von UI Elementen iOS [35], eigene Überarbeitung

Wie in Abbildung 17 dargestellt, sollen Benutzer Eingabe-Elemente nicht kleiner als 44 x 44 Punkte sein, so dass diese mittels einer Berührung, meist durch einen Finger, gut getroffen werden und nicht verfehlt werden können. [35]

4.2.1 Was bedeutet Multi-Touch-Tauglichkeit?

Neben den diskreten Benutzereingaben wie dem Drücken – und Loslassen – eines UIButtons (und wieder Loslassen) können Multi-Touch-Displays, ihrem Namen entsprechend, auch mehrere und kontinuierliche Berührungen entdecken und diese als Gesten verarbeiten [53]. Wie von Patrick Alessi beschrieben, wurde dies von Apple Inc. für das iPhone und iPad eingeführt:

> "*To assist developers in implementing more complex behaviors such as recognizing swipe or pinch gestures, Apple introduced the concept of Gesture Recognizers. You can use gesture recognizers on both the iPhone and the iPad." [53], S.109*

Apple Inc. hat hier die abstrakte Basis Klasse UIGestureRecognizer, die selbst in eigenen Klassenimplementierungen als Elternklasse verwendet werden kann, aber auch bereits vorgefertigte Implementierungen zur Verfügung gestellt, wie in [53] aufgelistet:

- UIPinchGestureRecognizer
- UIPanGestureRecognizer
- UITapGestureRecognizer
- UIRotationGestureRecognizer
- UILongPressGestureRecognizer
- UISwipeGestureRecogsnizer

Für ein besseres Verständnis wird in Abbildung 18: Multi-Touch Beispiele iOS [35] schematisch dargestellt, wie solche Benutzerinteraktionen auf den Geräten von Benutzern/Benutzerinnen angewendet werden können.

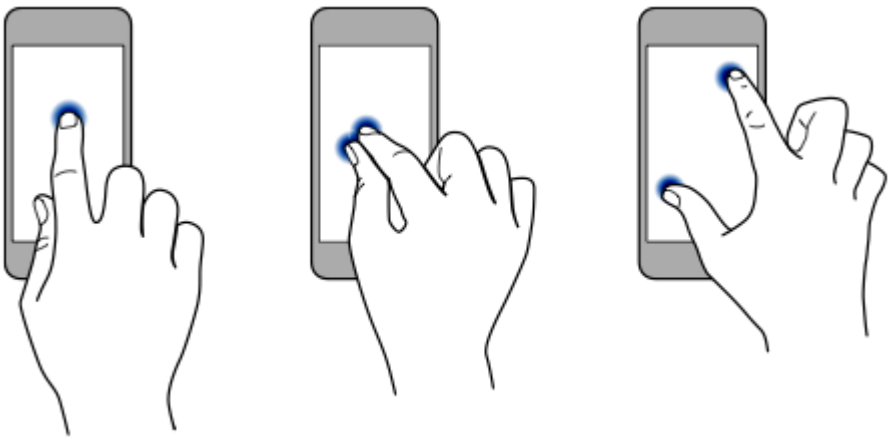


Abbildung 18: Multi-Touch Beispiele iOS [35], eigene Überarbeitung

Bei der Implementierung der JVx iOS Applikation wurden die UISwipeGestureRecognizer-Klasse zweimal verwendet, die wie untenstehend in der Klasse JVxFormViewController zur Anwendung kommt und welche die „Wischbewegung" von nach links oder rechts erkennt und die zugeordnete Methode aufruft.

```
UISwipeGestureRecognizer *swipeLeftGesture = [[UISwipeGestureRecognizer alloc] initWithTarget:self
                    action:@selector(handleSwipeLeft:)];
[swipeLeftGesture setDirection: UISwipeGestureRecognizerDirectionLeft];
[self.tableView addGestureRecognizer:swipeLeftGesture];

UISwipeGestureRecognizer *swipeRightGesture = [[UISwipeGestureRecognizer alloc] initWithTarget:self
                    action:@selector(handleSwipeRight:)];
[swipeRightGesture setDirection: UISwipeGestureRecognizerDirectionRight];
[self.tableView addGestureRecognizer:swipeRightGesture];
```

Listing 12: JVx UISwipeGestureRecognizer

4.2.2 iOS Human Interface Guidelines

Wie in Kapitel 4.2 „Anforderungen an Touch optimiertes mobiles Design“ schon angedeutet, wird auch von Apple vorgegeben, nach welchen Kriterien iOS Apps „designt“ werden müssen, um in den Apple App Store gestellt werden zu können. Diese Vorgaben werden in den iOS Human Interface Guidelines allen iOS Entwicklern und Entwicklerinnen unter [54] zur Verfügung gestellt. Um eine Zulassung für den Apple Store für die jeweilige App – oder deren Update – zu bekommen, wird diese durch Apple nach diesen Kriterien geprüft. Diese Vorgaben werden auch in der JVx iOS Erweiterung als Grundlagenkriterien angesehen.

4.2.3 Mobile Displays und deren Besonderheiten

Um einen bessren Überblick über die Besonderheiten von mobilen Displays zu bekommen wird nun versucht, mit den untenstehenden Aussagen von Steven Hoober und Ericxy Berkman zwei grundlegende Problematiken und die daraus entstehenden Herausforderungen aufzuzeigen:

> “*Mobile device displays can range in size, resolution, and pixel density (ppi). As a mobile designer and developer, it’s helpful to become familiar with these differences so that you can make appropriate decisions throughout the design process. Depending on the requirements of the project, you may be designing for one particular device or for multiple devices with varying displays.” [55], S.420*

> “*Mobile devices are small and portable, and unlike desktop computers they can be manipulated and viewed in any manner. Naturally, users will rapidly face the screen in the correct direction, but after this they should be allowed to choose their preferred viewing Orientation. Content must be presented in a useful format in whichever orientation is chosen, modified to fit the screen, but without changing context or modifying existing user entry.” [55], S.434*

Vor allem die immense Reduktion der Größe eines mobilen Displays mit der oft sehr schnell wechselnden Orientierung erfordert vom Entwickler/von der Entwicklerin ein Umdenken von den klassischen Desktop ERP-Anwendungen mit deren meist sehr großen Anzahl an Menü-

Elementen und Icons. Als einfaches Beispiel kann die Darstellung in Abbildung 19: MyERP Menu mit "Projects" Eingabemaske - Desktop” herangezogen werden.

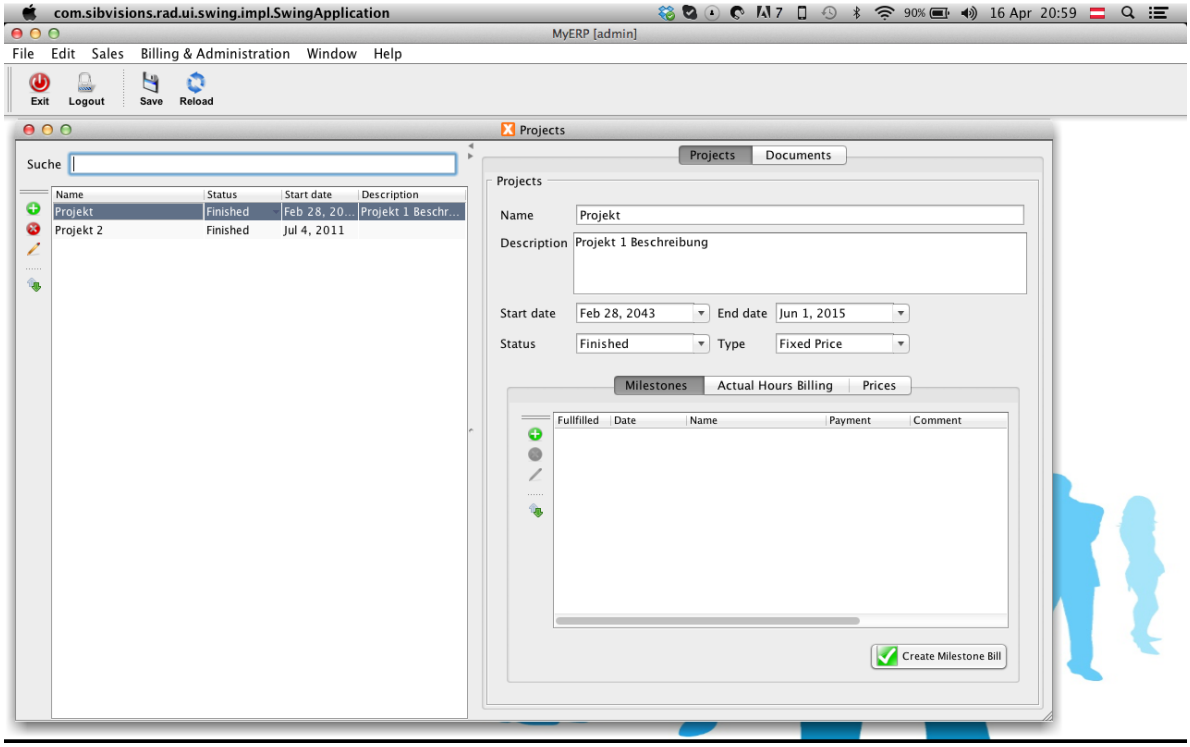


Abbildung 19: MyERP Menu mit "Projects" Eingabemaske - Desktop

4.3 Simplifizierung von komplexen Daten

Wenn nun eine Eingabemaske wie in Abbildung 19: MyERP Menu mit "Projects" Eingabemaske - Desktop” eins zu eins auf ein mobiles Gerät wie das iPhone umgelegt würde, käme folgendes Problem zum Tragen:

> “*A common misconception with mobile devices is that the screen size limits the user’s ability to see the screen.” [55], S.420*

Daher bestand für die Umsetzung eines mobilen Client der dringende Bedarf, die einzelnen Bereiche und deren Eingabemasken in einzelne Screens aufzuteilen und auch auf die Userinterface-Elemente der mobilen iOS Plattform anzupassen. Diese Trennung der Views und die mobil-taugleiche Anordnung der Interaktionselemente wird in den weiterführenden Unterkapiteln zunächst konzeptionell und dann im Kapitel 4.7 „Beispielsanwendung MyERP” in der Praxisanwendung durch die Umsetzung auf der iOS Plattform für iPhone und iPad erklärt.

4.3.1 Grunddesign des Benutzerinteraktionskonzeptes

Zunächst muss aber nochmals der Fokus auf die größte Umstellung von Desktop-Eingabemasken zu mobilen Geräten, wie dem iPhone oder iPad, gelegt werden, der wie folgt zum Ausdruck gebracht werden kann:

> "*A growing number of devices today are using gestural interactive controls as the primary input method." [55], S.318*

Die Umstellung von der Eingabe mit einem Mauszeiger und einer externen Tastatur auf ein Eingabekonzept, bei dem sogar die Tastatur ein Teil des Anzeigebereiches sein kann, erfordert eine Evaluierung jedes einzelnen Elementes zur Interaktion auf Notwendigkeit und Priorität.

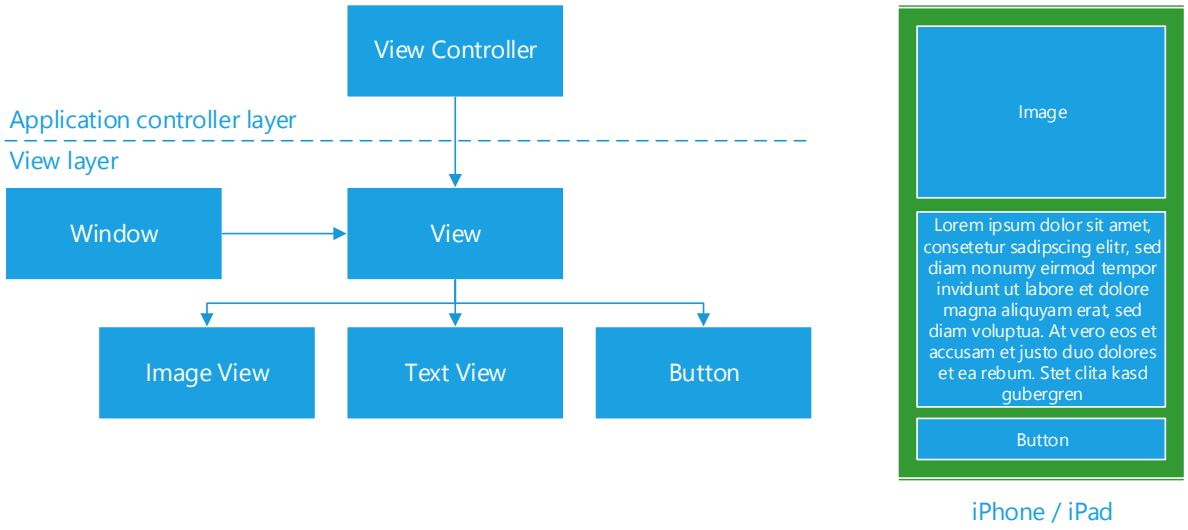


Abbildung 20: Benutzerinteraktion iOS [35], eigene Überarbeitung

Wie in Abbildung 20: Benutzerinteraktion iOS [35] schematisch angedeutet, ist auch die Anordnung und Reihenfolge von Bildern, Texten und Interaktionselementen von großer Bedeutung, da die Screen-Größe vor allem auf dem iPhone sehr stark begrenzt ist.

4.3.2 Reduktion und Separation von Informationen

Da ERP-Systeme sehr oft tabellarisch dargestellte Daten haben, müssen kreative Lösungen vom Entwickler/von der Entwicklerin gefunden werden, wie dies von Mary Treseler richtig formuliert wurde:

> "*Displaying tabular data in mobile presents a challenge and an opportunity. The challenge is finding usable ways to display the data on small screens. The opportunity is that we can take a more critical view of the data to determine what the user really needs to take away*

from it, and then develop creative solutions to display the data so that it will be even more useful." [56], S.99

Vor allem die Reduktion und Trennung von Informationen auf einzelne Eingabemasken ist hier eines der Kernkonzepte, welches in Abbildung 21: Navigation iOS [35] von Apple Inc. in deren iOS Human Interface Guidelines sehr gut dargestellt worden.

Abbildung 21: Navigation iOS [35], eigene Überarbeitung

Die Navigation von der Übersicht bis hin zur Detail-Seite, auf welcher dann gegebenenfalls die Daten verändert werden können, ist in sehr vielen iOS Applikationen der Schlüssel zur Reduktion der Informationen des einzelnen Views und somit zur Separation.

"Persistent navigation encompasses simple menu structures like the List Menu and Tab Menu. As soon as you open an app with persistent navigation, it is immediately clear what the primary navigation options are." [56], S.2

Wie von Theresa Neil richtigerweise erkannt wurde, ist es wichtig, dass sich der Benutzer/die Benutzerin auf eine einfache und intuitive Navigation einstellen kann. Die nun bereits erwähnten Listen als Menü-Optionen werden nun genauer erläutert.

4.3.3 Tabellen als Grundlage der Visualisierung

Am iOS Betriebssystem selbst kommt oft die Listendarstellung als Menü zur Anwendung. Anhand der Abbildung 22: Einstellungen Liste iOS" ist zu erkennen, dass Apple Inc. selbst den sogenannten UITableView bei der Kontaktliste oder den Einstellungen verwendet.

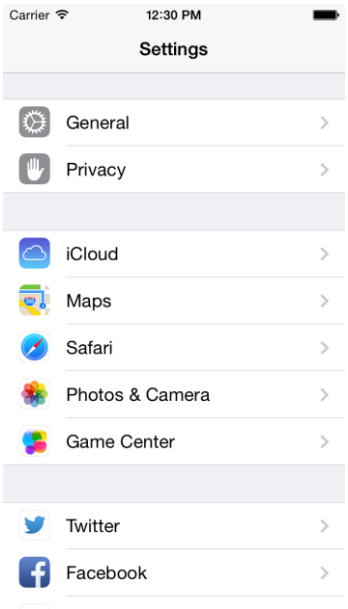


Abbildung 22: Einstellungen Liste iOS

Wie in [57] beschrieben, benötigt jeder UITableView eine Datenquelle (datasource). Diese kann ein NSArray oder ein NSDictionary sein. Zusätzlich muss ein Controller, der einen UITableView beinhaltet, mit dem UITableViewDelegate und dem UITableViewDataSource Protokoll konform sein. Die verpflichtend zu implementierenden Delegate-Methoden sind:

- tableview:numberOfRowsInSection:
- numberOfSectionsInTableView:
- tableview:cellForRowAtIndexPath:

Die Methoden numberOfRowsInSection und numberOfSectionsInTableView bilden zusammen die schon erwähnte Datenquelle. In der Funktion cellForRowAtIndexPath wird bestimmt, wie die Visualisierung des jeweiligen Tabelleneintrages ausschauen soll. Hierauf wird noch in 4.3.4 „Datenbasierte Visualisierung“ genauer Bezug genommen.

Wird nun eine Zelle, welche einem Tabelleneintrag entspricht, angeklickt, wird wie bei einem UIButton ein Event ausgelöst. Hierzu wird die im UITableViewDelegate Protocol definierte Methode didSelectRowAtIndexPath aufgerufen. [48], S.278

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
  NSLog(@"%@", [NSString stringWithFormat:@"Cell %ld in Section %ld is selected",
  (long)indexPath.row, (long)indexPath.section]);
}
```

Listing 13: UITableView didSelectRowAtIndexPath

Wie in Listing 13: UITableView didSelectRowAtIndexPath“ zu erkennen ist, wird hier der Index der selektierten Zelle (indexPath) als Übergabeparameter übergeben. Diese Information, welche wiederum aus Sektion und Reihe besteht, kann je nach Selektion nun unterschiedlich auf jeden Index reagieren. In oben angeführten Beispiel wird aber bei jeder Selektion der jeweilige Index in der Konsole ausgegeben.

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    [self toggleEdit];
}
```

Listing 14: JVxSimpleTableDetailViewController didSelectRowAtIndexPath

In der JVx iOS Applikation wird diese Delegate-Methode beispielsweise in der Klasse JVxSimpleTableDetailViewController wie in Listing 14: JVxSimpleTableDetailViewController didSelectRowAtIndexPath“ beschrieben verwendet, um den Editiermodus zu aktivieren.

4.3.4 Datenbasierte Visualisierung

Aufbauend auf dem Konzept der einspaltigen Tabellen als Grundlage der Navigationsstruktur wird nun in der Methode cellForRowAtIndexPath die Visualisierung der jeweiligen Zelle implementiert. Hierzu werden mittels der UITableViewCell Klasse, oder eigener Kind-Klassen--Implementierung, die gewünschten Informationen dargestellt.

> “*You can allocate instances of the UITableViewCell class and return them to the table view. There are, of course, properties that can be set for each cell, including the title, subtitle, and color of each cell, among other properties.“ [48], S.286*

Die Visualisierung wird im Storyboard mit sogenannten Prototype Cells vorgenommen. Hier bietet Apple Inc. schon vorgefertigte Zelllayouts, mit nur einem Titel oder beispielsweise auch mit Titel und Untertitel. Wie in Abbildung 23: Unterschiedliche datenbasierte Darstellung von Informationen“ zu sehen ist, können aber auch eigene Zellen mit einem oder mehreren Labels und auch Ein/Aus-Swiches erstellt werden. Um für die JVx iOS Applikation ein gewisses Maß an generischem Aufbau der Daten zu bekommen, wurden die Zellen JVxTableViewCell1VisibleEment bis JVxTableViewCell4VisibleEment mit den jeweiligen unterschiedlichen Darstellungsmöglichkeiten geschaffen.

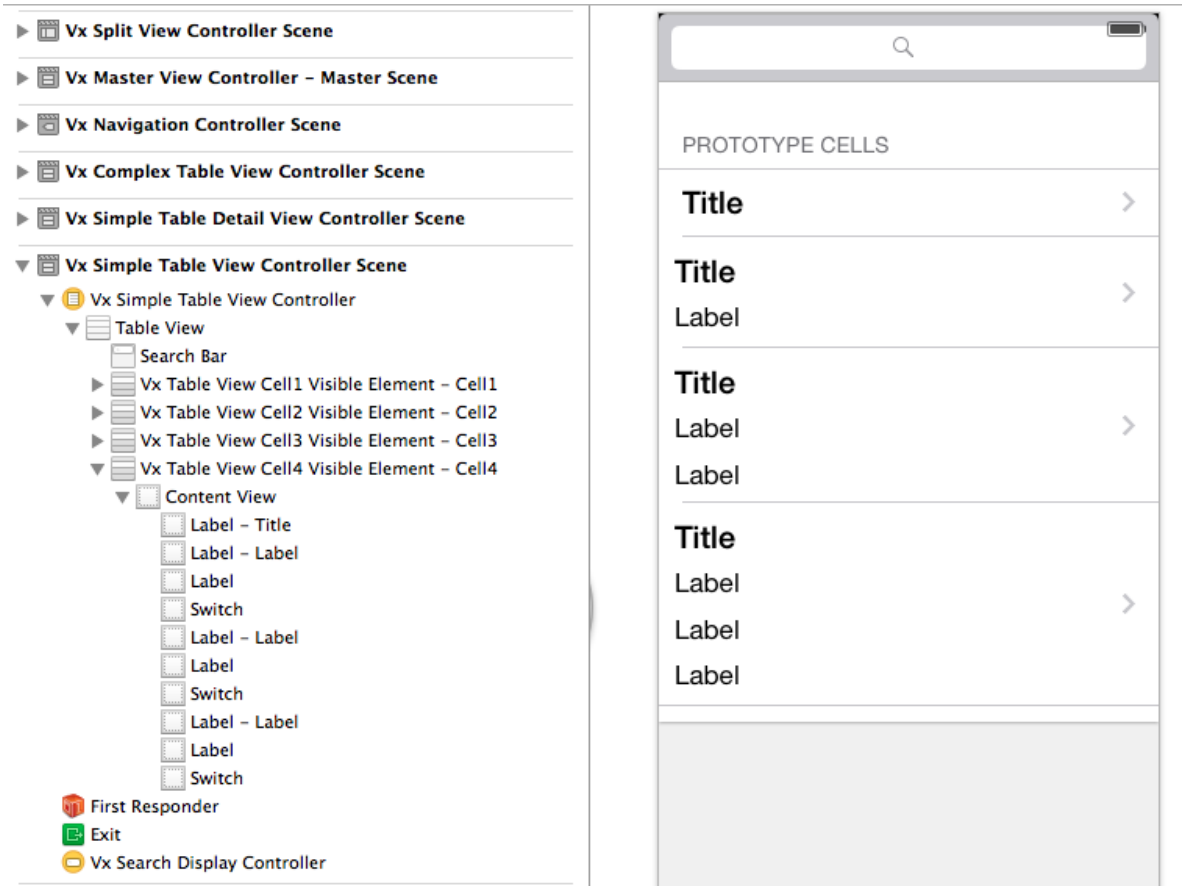


Abbildung 23: Unterschiedliche datenbasierte Darstellung von Informationen

4.3.5 Unterschiede zwischen iPhone und iPad

Neben dem Unterschied, dass das iPad keine Telefonie-Funktion besitzt, ist die Größe des Displays einer der Hauptunterschiede dieser zwei Gerätegruppen der iOS Plattform. Für die JVx-Anwendung ist der Unterschied in der Anordnung der hierarchischen Navigation vom Übergeordneten zum Detail wie in 4.3 „Simplifizierung von komplexen Daten“ erörtert wurde. Hier wird von Apple Inc. für das iPad das Konzept der „Master-Detail“ Anzeige forciert und auch in der Settings App umgesetzt, wie in Abbildung 24: Einstellungen am iPad mittels UISplitViewController“ zu sehen ist. Im Vergleich zur iPhone-Visualisierung war dies in Abbildung 22: Einstellungen Liste iOS“ einzeln, ohne Detail-Seite, dargestellt. Hierzu bietet die iOS Plattform im Cocoa Touch Framework den sogenannten UISplitViewController, der zwei UIViewController in der „Master-Detail“ Visualisierung ermöglicht. [54]

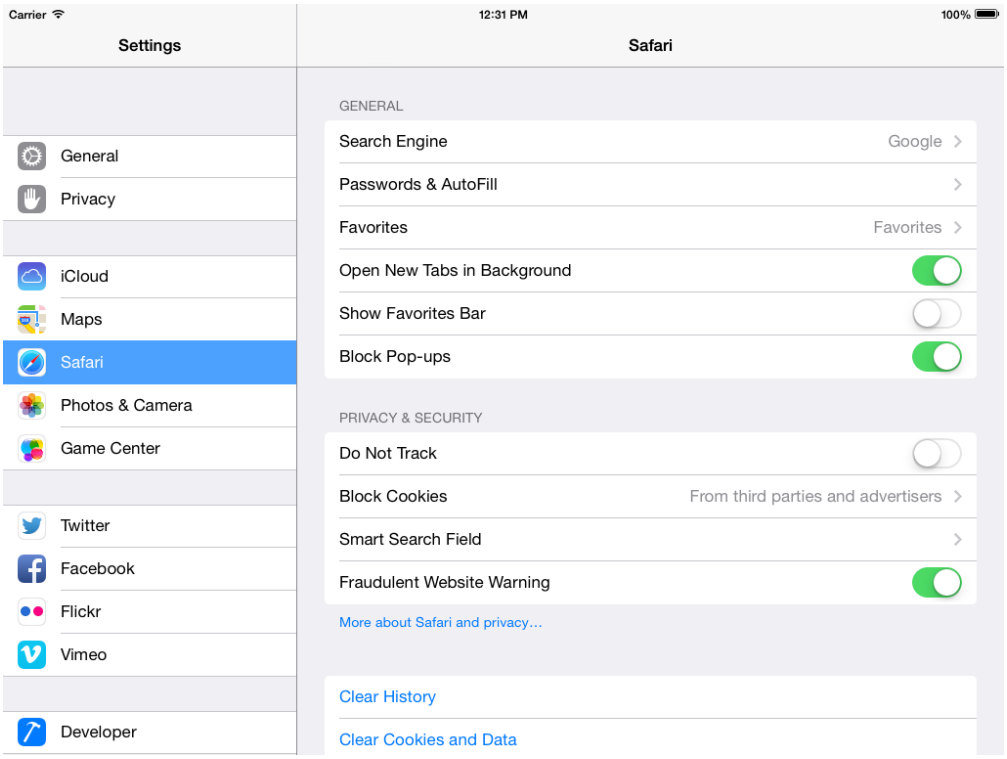


Abbildung 24: Einstellungen am iPad mittels UISplitViewController

Wie in Abbildung 34: Popover iPad Splitscreen“ im späteren Verlauf zu sehen ist, wurde der UISplitViewController in der JVx iOS App für das iPad verwendet, da dieser dem Entwickler/der Entwicklerin auch eine Funktion zum Ausblenden der „Master“-Seite bei vertikaler Orientierung des iPad zur Verfügung stellt. Dies ermöglicht es, den Fokus des Users auf den Detailbereich zu halten und nur bei Notwendigkeit das Menü durch eine „Wischbewegung“ einzublenden, wie dies in 4.2.1 „Was bedeutet Multi-Touch-Tauglichkeit?“ beschrieben wurde.

4.4 Sicherheitsanforderungen an den Client

Neben der schon in 2.5.4 „Sicherheitsanforderungen an moderne APIs“ beschriebenen Sicherheitsvorkehrungen an die API wurde auch Client-seitig ein Augenmerk auf die Sicherheit der User- und Anwendungsdaten gelegt.

4.4.1 Sandbox

Als eines der grundlegendsten Sicherheitsmerkmale der iOS Plattform ist die sogenannte „Sandbox“, die wie folgt gut beschrieben wurde:

> *“OS X and iOS implement a number of features to improve the overall level of security for the user. One of these features is the application sandbox, a tool that restricts what an application is allowed to do. The application exists inside the sandbox, and may not try to*

access any system resources (hardware, user data, and so on) that is outside the sandbox." [58], S.71

Dieses „getrennt-Halten" der einzelnen Applikationen stellt sicher, dass die vom JVx Server heruntergeladenen Daten am iOS Client von keiner anderen Applikation ausgelesen werden können. Die Verzeichnisstruktur einer jeder iOS Sandbox schaut wie in Abbildung 25: JVx App "Sandbox"""" dargestellt aus: [58]

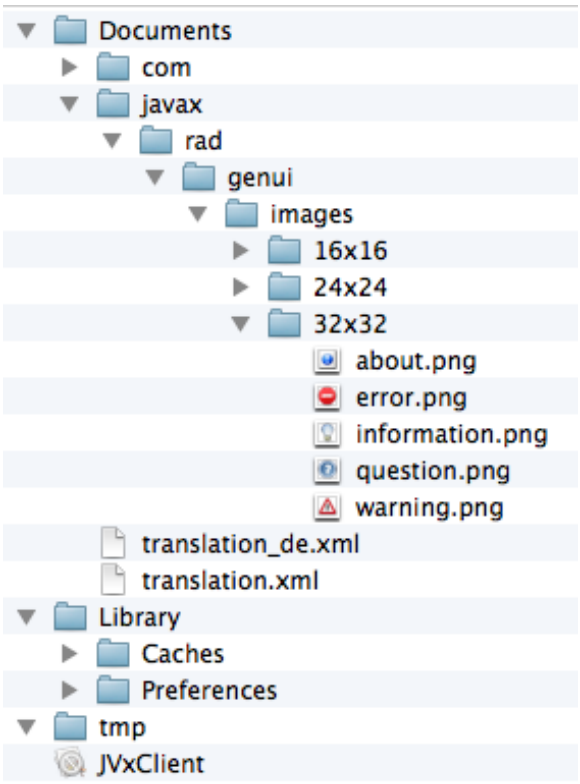


Abbildung 25: JVx App "Sandbox"

Da jede iOS Applikation begrenzt ist in ihren Möglichkeiten, Dateien persistent in der Verzeichnisstruktur abzulegen, wird wie in [58] beschrieben ein Überblick der einzelnen Verzeichnisse samt deren Verwendungszwecken in Tabelle 5: iOS Sandbox Verzeichnisstruktur" gegeben:

Titel	**Beschreibung**
Documents	"*Stores all documents belonging to the application*" [58], S.72
Library	*"Stores all settings and configuration info"* [58], S.72

Caches	"*Contains data that is useful to have on disk, but could be re-generated; items in this folder are deleted by the system if it needs to free some space*" [58], S.72
Preferences	"*Stores settings and preferences*" [58], S.72
tmp	"*Stores files temporarily; items in this folder are periodically deleted by the system*" [58], S.72
JVxClient.app	"*The application package*" [58], S.72

Tabelle 5: iOS Sandbox Verzeichnisstruktur

Die JVx iOS Applikation speichert die persistenten Daten wie Bilder- oder Übersetzungs-Feiles im dafür vorgesehenen Dokumenten Verzeichnis.

4.4.2 Schlüsselbund

Eine der Anforderungen an die JVx iOS Applikation war es, dass auch sicherheitsrelevante Daten, wie beispielsweise der Benutzername und das Passwort, am Client gespeichert bleiben. Diese sensiblen Daten sollten wie von Apple Inc. vorgegeben im sogenannten „Keychain" (Schlüsselbund) gespeichert werden. Hierzu wird von Apple Inc. auf deren Entwicklerportal [35] eine KeychainItemWrapper-Klasse dem Entwickler/der Entwicklerin zur Verfügung gestellt.

> *"Storing the Password in a Keychain Apple has developed a keychain wrapper class that makes it easy for you to work with the keychain." [57], S.20*

Die Verwendung benötigt, wie in [57] beschrieben, den Import des Security Frameworks und muss dem File ein „–fno –objc –arc" Compiler Kennzeichnung gesetzt werden, da diese ohne ARC entwickelt wurde, welches in 2.6.14 der Speicherverwaltung beschrieben wurde.

4.5 Targets & Schemes eines Xcode-Projekts

Im Laufe des iterativen Entwickelns der iOS Applikation wurde zusammen mit dem Team der SIB Visions festgestellt, dass auch unterschiedliche Konfigurationen des Clients notwendig werden. Um dies zu erreichen und vor allem ohne großen Aufwand Änderungen im Quellcode zu ermöglichen wurde das Xcode-Projekt durch unterschiedliche Schemes und Targets erweitert. Von Richard Wentk wurde in [59] über das Build System von Xcode folgendes geschrieben:

“Although you can use the build system in a simple one-click way, the underlying technology is powerful, but complex. The default one-click option deliberately hides the complexity, but you must be familiar with its key elements before you can begin customizing builds.” [59], S.271

Um nun diese “one-click”-Einstellungen für die JVx iOS App nutzen zu können, werden die Begriffe „Scheme“ und „Target“ genauer erklärt und in der Anwendung des JVx iOS Clients beschrieben.

4.5.1 Schemes

Schemes werden genutzt, um unterschiedliche Build-Konfigurationen zu definieren, welche dann wiederum einem Target zugeordnet werden können, um dieses auch aus dem Xcode „Build and Run“-Direktmenü heraus starten zu können, ohne die Command Line im Terminal nutzen zu müssen.

“*You can use schemes to customize build actions; for example, the Run action in one scheme builds one target, but in another scheme, it builds every target in the workspace.“ [59], S.273*

Wie in Abbildung 26: JVx Schemes“ zu sehen ist, hat das JVx iOS Projekt nun neben der Grundkonfiguration eine zusätzliche Konfiguration für „Development“ und „Vorschau“:

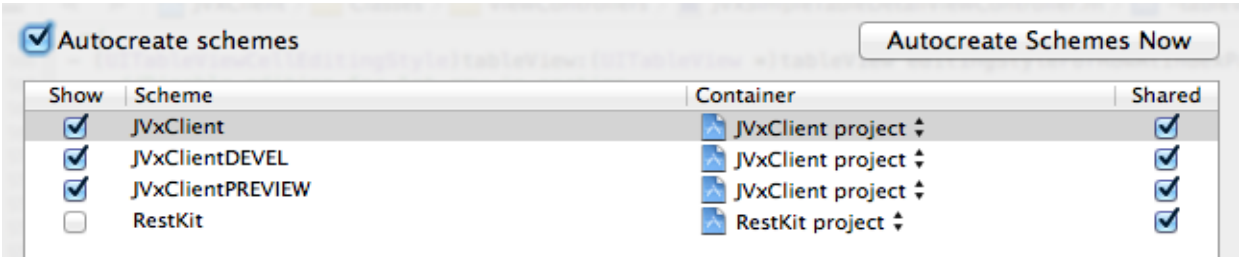


Abbildung 26: JVx Schemes

4.5.2 Targets

Diese nun erstellten Konfigurationen für „Development“ und „Vorschau“ wurden auch bei den Targets wie in Abbildung 27: JVx Targets“ zu sehen ist, als Schnellauswahl hinzugefügt. Es könnten hier aber auch, wie ebenfalls von Richardin Wentk beschrieben, auch eigene Framework Builds erstellt werden:

“A target is a recipe for building the files in a project, and it defines its product— for example, an app or a framework. In Xcode 5, many projects include a default target and an optional target for unit testing” [59], S.273

Abbildung 27: JVx Targets

4.6 Erreichbarkeit

Um die Komplexität und die Anforderungen an die JVx iOS Applikation in dieser Masterarbeit dem Aufwand einer Masterarbeit angemessen zu gestalten wurde von einem reinen online-Betrieb ausgegangen, der zu einem späteren Zeitpunkt durch einen offline-Betrieb erweiterbar sein sollte.

Dennoch wurde, um Fehlermeldungen angemessen abfangen zu können, zumindest der Status der Erreichbarkeit vor jedem Request geprüft. Hierzu wurde die von Apple Inc. veröffentlichte Reachability-Klasse verwendet, welche auf dem iOS Entwicklerportal zu finden ist und den Import des SystemConfiguration Frameworks erfordert [35]. Durch den Einsatz dieser Reachability-Klasse können, wie in [60], S.188 angegeben, folgende drei Netzwerkzustände erkannt werden:

- NotReachable
- ReachableViaWiFi
- ReachableViaWWAN

Mit diesem Status wird somit die Erreichbarkeit des Servers sichergestellt oder dem User ein offline-Status mitgeteilt, wie dies in 4.7.10 „Fehlerbehandlung“ genauer erklärt wird.

4.7 Beispielsanwendung MyERP

Das sehr umfangreiche JVx Framework, welches über Jahre von der SIB Visions weiterentwickelt wurde, umfasst neben den üblichen Eingabemasken eines ERP-Systems auch sehr viele Sonderfälle und die entsprechenden Sonderkonfigurationen. Um den Umfang dieser Masterarbeit etwas eingrenzen zu können, wurde eine Beispielanwendung mit dem Namen „MyERP“ von der SIB Visions geschaffen, welches für die Umsetzung des mobilen iOS Clients als Basis diente.

4.7.1 Limitierung der Komplexität

Wie in Abbildung 28: MyERP Pipeline Eingabemaske“ zu sehen ist, umfassen die jeweiligen Eingabemasken der MyERP-Anwendung meist eine Liste an Einträgen und dazu passend auch gleich eine Editiermaske, in der die Werte des selektierten Eintrages editiert werden können. Auch die weiterführenden Verknüpfungen der Daten, beispielsweise zum Kontakte- oder Aktivitäten-Screen, werden in einer Ebene visualisiert.

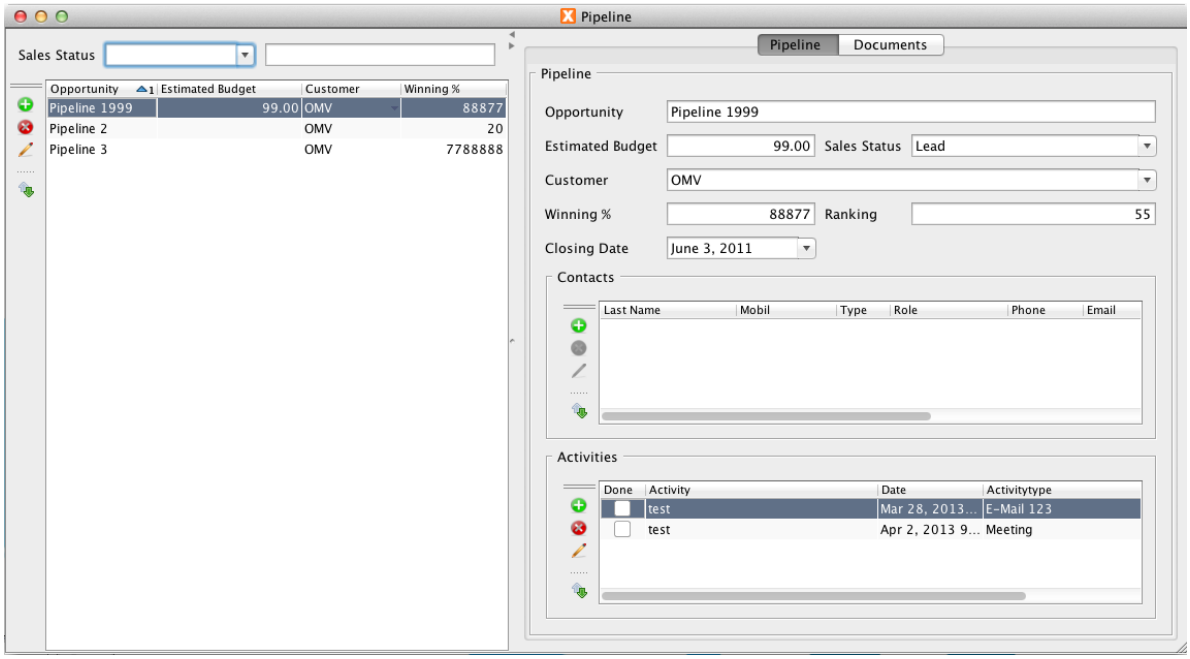


Abbildung 28: MyERP Pipeline Eingabemaske

Aufgrund dieser Komplexität und Fülle an Informationen wurde, wie in 4.3 „Simplifizierung von komplexen Daten“ schon theoretisch behandelt, nun die Aufteilung der JVx Eingabemasken Daten in einzelne Views vorgenommen, wie in Abbildung 34: Popover iPad Splitscreen“ als Resultat zu erkennen ist. Weitere Eingabemasken und Screens des Menüs oder des Login-Fensters sind im Anhang von Abbildung 37: MyERP Menu mit "Holidays" Eingabemaske - Desktop“ bis Abbildung 44: MyERP Login – Desktop“ zwecks vollständiger Nachvollziehbarkeit angeführt.

4.7.2 Model-Klassen

Dank der Vorteile der Objektorientierung, die durch Objective-C unterstützt wird, wurden die Daten der API wie in 4.1 „Anbindung der iOS App an das JVx Framework“ angegeben wurde in logische Modelle (Klassen) umgewandelt. Die wichtigsten Klassen für die visuelle Darstellung der JVx-Eingabemasken sind wie folgt mit kurzer Beschreibung angeführt:

Komponenten Name	Beschreibung
JVxMenuView	Dies ist die Repräsentation der Menü-Struktur, welche die Navigation zu dem jeweiligen View ermöglicht. Dieser kann entweder ein JVxSimpleTable oder ein JVxComplexTable sein.
JVxTable	Da der Unterschied zwischen einem JVxSimpleTable und einem JVxComplexTable nur die Anzahl (1 oder n) der darzustellenden Screens (Tabels) ist, wurden diese Daten im JVxTable Daten Container zusammengefasst.
JVxSimpleTable	Daten des Screens mit nur einem JVxTable
JVxComplexTable	Daten des Screens mit mehreren JVxTables
JVxForm	Daten des Editiermodus eines JVxSimpleTable

Tabelle 6: Wichtigsten JVx Model-Klassen

Als Einstiegspunkt nach einer erfolgreichen Anmeldung einer JVx-Anwendung ist der JVxMenuView zu sehen, welcher das klassische Menü einer Desktopapplikation ersetzt.

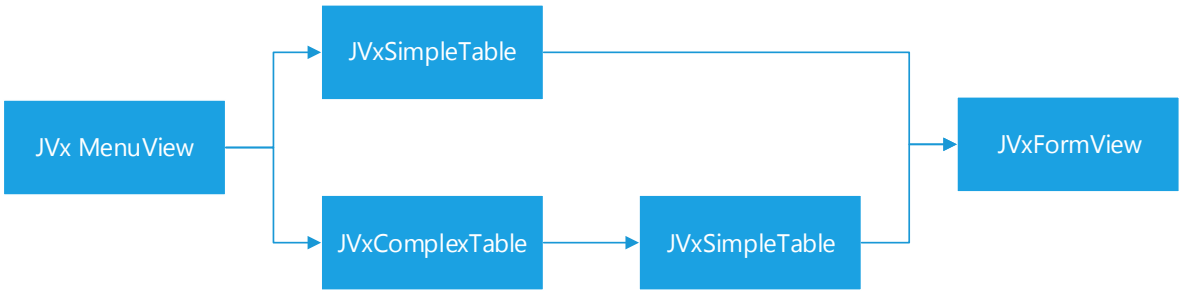


Abbildung 29: JVx Model-Klassen

Wie in Abbildung 29: JVx “ zu sehen ist, können die Menüpunkte entweder ein einfacher JVxSimpleTable oder ein JVxComplexTable sein. Bei einem JVxComplexTable wird als Navigation zuerst eine Liste dieser JVxTables dargestellt und dann bei einer Selektion dieser zu einem JVxSimpleTable umgewandelt und dargestellt. Wird nun ein JVxSimpleTable in den Editiermodus versetzt, wechselt die Anzeige in den JVxFormView, der die visuelle Repräsentation der JVxForm Klasse darstellt.

Wie nun genau die API-Anbindung des JVx iOS Clients umgesetzt wurde, ist im nachfolgenden Kapitel 4.7.3 „API Controller“ zusammengefasst.

4.7.3 API Controller

Als Kernstück der Kommunikation zwischen dem Server und der JVx iOS Applikation wurde die Klasse JVxAPIController aufgebaut, die einen abstrakten Aufbau der Anfragen und Antworten ermöglicht und als Container für die unterschiedlichen „JobTypen“ fungiert. Dieser objektorientierte Ansatz ermöglicht einen generischen Aufbau und eine leichte Erweiterbarkeit für neue „JobTypen“ der Kommunikation.

Abstrakter Aufbau

Wie in 2.6.13 „Angewandte Technologien und Design Patterns“ erklärt wurde, werden Protokolle zur Bündelung und Sicherstellung ihrer Implementierung verwendet. Dieser Ansatz kommt nun in der JVxAPIController-Klasse zur Anwendung, und es wurden für eine erfolgreiche Kommunikation zum Server die Methode finishedSuccessfullyWithResponse und bei einem Fehler die alternative failedWithError im Protokoll als erforderlich implementiert, wie dies in Listing 15: APIControllerDelegate-Methoden“ zu sehen ist.

```
@protocol JVxAPIControllerDelegate
@required
- (void)apiController:(JVxAPIController *)apiController finishedSuccessfullyWithResponse:( JVxAbstrac-
tResponse *)response;

- (void)apiController:(JVxAPIController *)apiController failedWithError:(NSError *)error;
@end
```

Listing 15: APIControllerDelegate-Methoden

Die jeweiligen Controller, welche die API-Anfragen mit deren Rückmeldungen initialisieren, müssen nun diese Methoden dem Protokoll entsprechend implementieren.

Typen von API Anfragen/Antworten

Für die Umsetzung der MyERP-Anwendung wurden bisher folgende „JobTypen“ implementiert:

- Unknown
- Startup
- LogIn
- LogOut
- OpenScreen
- BeforeInsert
- BeforeUpdate
- Insert
- Update
- Delete
- GetTable
- OpenDetailTable
- InsertDetail
- UpdateDetail
- DeleteDetail
- GetDetailTable
- DownloadImages
- DownloadLanguageResource
- ButtonPressed

Für den jeweiligen Typ von Anfrage/Antwort ist wie im Kapitel 2.5.2 „Architektur & Design von RESTful API Web-Services“ beschrieben eine jeweilige URI erforderlich, die wie in Listing 16: URI Beispiel JVx“ beispielhaft für „Startup“ und „LogIn“ angeführt wurde.

```
+ (NSURL *)urlForStartup {
  NSString *url = [[JVxUtils getServerBaseURL] stringByAppendingPathComponent:@"api/startup"];
  return [NSURL URLWithString:url];
}
+ (NSURL *)urlForLogIn {
  NSString *url = [[JVxUtils getServerBaseURL] stringByAppendingPathComponent:@"api/login"];
  return [NSURL URLWithString:url];
}
```

Listing 16: URI Beispiel JVx

Die vollständige Liste ist in der Klasse JVxUtils angeführt und muss natürlich bei einem Erweitern der „JobTypen“ entsprechend ebenfalls erweitert werden.

Anfrage

Die in der JVxAPIController-Klasse verwendeten Anfragen und Antworten wurden nun ebenfalls in einem JVxAbstractRequest und einen JVxAbstractResponse gekapselt und somit flexibel für Erweiterungen aufgebaut.

```
@implementation JVxAbstractRequest
@synthesize urlRequest;
@synthesize jobType;

- (NSMutableURLRequest *)createURLRequestWithURL:(NSURL *)url withJSONDictionary:(NSDictionary
*)dict
{
  NSMutableURLRequest *theRequest = [NSMutableURLRequest requestWithURL:url
                cachePolicy:NSURLRequestUseProtocolCachePolicy
                timeoutInterval:REQUEST_TIMEOUT_TIME];
  NSError *error;
  NSData *jsonData = [NSJSONSerialization dataWithJSONObject:dict
                options:NSJSONWritingPrettyPrinted
                error:&error];
  [theRequest setHTTPMethod:@"POST"];
  [theRequest addValue:@"application/json" forHTTPHeaderField:@"Content-Type"];
  [theRequest setHTTPBody:jsonData];
  return theRequest;
}
@end
```

Listing 17: JVxAbstractRequest

Wie in Listing 17: JVxAbstractRequest“ zu sehen ist, stellt die Klasse JVxAbstractRequest allen Kind-Klassen die Methode „createURLRequestWithURL:withJSONDictionary“ zur Verfügung, mit welcher die jeweilige „POST“-Anfrage erstellt wird. Mittels des Parameters „dict“ können beispielsweise die Parameter für den Startup-Vorgang übergeben werden, welche in 3.1 „Designentscheidungen der Schnittstelle“ beschrieben wurden. Es ist zu sehen, dass hier, wie in den vorgehenden Kapiteln beschrieben, theoretische Konzepte sowie global definierte Konstanten wie „REQUEST_TIMEOUT_TIME“ oder auch die „NSJSONSerialization“ zur Anwendung kommen.

Antwort

Auf dem gleichen Prinzip, welches schon für die Anfragen in der Klasse JVxAbstractRequest erklärt wurde, wurde auch für die Antworten eine abstrakte Basisklasse mit dem Namen JVxAbstractResponse erstellt. Neben den Attributen response, data, error und jobType wurde die von den Kind-Klassen zu implementierende Methode createResponse leer vorimplementiert.

```
@implementation JVxAbstractResponse
@synthesize response;
@synthesize data;
@synthesize error;
@synthesize jobType;

- (id)initWithUrlReponse:(NSURLResponse *)aResponse withJobType:(JVxAPIController_JobType)aJob-
Type data:(NSData *)aData error:(NSError *)anError
{
  if ( self = [super init])
  {
    response = aResponse;
    data = aData;
    jobType = aJobType;
    error = anError;
  }
  return self;
}
- (void)createResponse
{ //DLog(@"This Method has to be implemented in inherited class because Objective-C doesn't have a real
   abstract class!");
}
@end
```

Listing 18: JVxAbstractResponse

Für den entsprechenden „JobTypen“ muss nun die Methode createResponse den jeweiligen Anforderungen angepasst werden, so dass die erhaltenen Daten wieder entserialisiert werden und entsprechend dem dynamischen Inhalt verarbeitet und in die dafür vorgesehenen Klassen umgewandelt werden.

```
@implementation JVxStartupResponse
- (void)createResponse
{
  [JVxAppState instance].responseViews.MenuView.items = nil;
  NSError *error = self.error;
  id json = [NSJSONSerialization JSONObjectWithData:self.data options:kNilOptions error:&error];
  self.error = error;

  if ([json isKindOfClass: [NSArray class]])
  {
    for (NSDictionary *dict in json)
    {
      NSString *name = [dict valueForKey:@"name"];

      if ([name isEqualToString:@"applicationMetaData"])
      {
        [JVxAppState instance].responseViews.ApplicationView =
                    [JVxJSONUtils getApplicationViewWithData:i];
      }
      else if ([name isEqualToString:@"login"])
      {
        [JVxAppState instance].responseViews.LogInView = [JVxJSONUtils getLogInViewWithData:i];
      }
      else if ([name isEqualToString:@"menu"])
      {
        [JVxAppState instance].responseViews.MenuView.items = [JVxJSONUtils
                     getMenuViewItemsWithData:i];
      }
      else
      {
        [JVxErrorViewHelper setErrorViewsWithData:i];
      }
    }
  }
}
@end
```

Listing 19: JVxStartupResponse

Wie in Listing 19: JVxStartupResponse“ dargestellt, kann die Antwort mittels des Schlüsselwortes „name“ identifiziert werden. Anhand des Beispiels JVxStartupResponse kann dieser entweder die „applicationMetaData“ und die „login“ Daten liefern oder bei einem „Autologin“

des Servers direkt das Hauptnavigationsmenü selbst, welches mittels „menu“ identifiziert wird. Diese Daten werden dann mittels der JVxJSONUtils-Klasse in die jeweiligen Objekte umgewandelt. Sollte keine dieser zu erwartenden Daten in der Antwort enthalten sein, wird mittels der JVxErrorViewHelper-Klasse die Fehlerbehandlung vorgenommen.

Um nun die Anfrage- und Antwort-Klassen für den Startup anzuwenden, wird die im JVxAPI-Controller implementierte Methode „startStartupAPICallWithDelegate“ vom entsprechenden ViewController aufgerufen, die eine NSURLConnection mit einer asynchronen Anfrage an den JVx-Server beginnt. Entsprechend dem JVxAPIControllerDelagate-Protokol lautet die Antwort an die Delegate-Klasse " apiControler:finishedSuccessfullyWithResponse" oder bei einem Fehler "apiController:failedWithError" - also wird eine Fehlermeldung an die Delegate-Klasse zurückgegeben, welche diese Anfrage an das JVx Backend gestellt hat. Hierzu wird auch der http-Statuscode überprüft, um bei einem Fehler diesen ebenfalls auswerten zu können.

```
- (void)startStartupAPICallWithDelegate:(NSObject <JVxAPIControllerDelagate>*) delegate;
{
  [[JVxAppState instance] showActivityIndicatorWithText:[JVxTranslationUtils
 getTranslationforString:@"Loading..."] withMode:MBProgressHUDModeIndeterminate withImageName:nil];
  JVxStartupRequest *theRequest = [[JVxStartupRequest alloc] init];
  [theRequest createRequest];
  [NSURLConnection sendAsynchronousRequest:theRequest.urlRequest queue:[NSOperationQueue
mainQueue] completionHandler:^(NSURLResponse *response, NSData *data, NSError *error)
  {
    if (error) {
      if (delegate != nil) {
        [delegate performSelector:@selector(apiController:failedWithError:) withObject:self
              withObject:error];
        return;
      }
    }
    else {
      JVxStartupResponse *theResponse = [[JVxStartupResponse alloc] initWithUrlReponse:response with-
JobType:JVxAPIController_JobType_Startup data:data error:error];

      // check the HTTP status code
      [JVxErrorViewHelper checkHTTPStatusCodeWithResponse:response];
      [theResponse createResponse];

      if ([JVxAppState instance].responseViews.ApplicationView.clientId != nil)
      { [delegate performSelector:@selector(apiController:finishedSuccessfullyWithResponse:)
            withObject:self withObject:theResponse];
      }
      else
      {
```

```
        NSString *domain = @"com.sibvisions.JVxClient.ErrorDomain";
        NSString *desc = NSLocalizedString(@"clientId was not in the response json from the startup", @"");
        NSDictionary *userInfo = @{ NSLocalizedDescriptionKey : desc };
        NSError *errorClientID = [NSError errorWithDomain:domain code:-101 userInfo:userInfo];

        [delegate performSelector:@selector(apiController:failedWithError:) withObject:self
              withObject:errorClientID];
        return;
      }
    }
  }];
}
```

Listing 20: startStartupAPICallWithDelegate

Basierend auf dieser generisch und einfach erweiterbaren Server Client-Architektur werden nun in den entsprechenden Views die jeweiligen Daten mobil-tauglich und für Touch-basierte Benutzereingaben/Benutzerinneneingaben optimiert dargestellt.

4.7.4 JVxViewController Klassen

Wie in Kapitel 2.6.4 „Model View Controller“ beschrieben, ist der Controller für die „Logik“ eines Views zuständig. Die in Unterkapitel 4.7.2 „Model-Klassen“ beschriebenen Daten werden nun von den verschiedenen Controllern an bestimmten Events durch Benutzereingaben/Benutzerinneneingabe oder durch eine Programmabfolge, beispielsweise beim Start, mittels des zuvor erwähnten JVxAPIControllers vom Server angefordert und verarbeitet.

In der Umsetzung der „MyERP“-Beispielanwendung wurden die für den Anwender/die Anwenderin sichtbaren ViewController

- JVxStartupViewController
- JVxLoginViewController
- JVxMasterViewController
- JVxDetailViewController
- JVxSimpleTableViewController
- JVxSimpleTableDetailViewController
- JVxComplexTableViewController
- JVxFormViewController

für die Umsetzung erstellt, die mit den für die Navigation zuständigen Hilfs-Controller-Klassen JVxNavigationController und JVxSplitViewController, welcher in Abbildung 34: Popover iPad Splitscreen“ zu sehen ist, in der JVx iOS Applikation verwendet. Der JVxSplitViewController, welcher nur bei der iPad Variante zum Einsatz kommt ermöglicht es den größeren Anzeigebereich durch eine Aufteilung in zwei Bereiche besser nutzen zu können.

Durch die Verwendung des JVxNavigationController wurde es möglich, dass immer die Orientierungen des aktuell sichtbaren ViewController auf jeweilige Drehungen des Gerätes reagieren kann und den View gegebenenfalls anpasst.

Nachdem, wie im Kapitel 2.6.11 „Storyboard“ beschrieben, der initiale ViewController der Klasse JVxStartupViewControler geladen wurde, beginnt der wie in Abbildung 30: JVx iOS Konzept“ beschriebene konzeptionell mögliche Workflow der JVx iOS Applikation.

Da die Daten, wie in 4.7.2 „Model-Klassen“ beschrieben, vom Server gesteuert und dadurch unterschiedlich sind, können an mehreren Stellen die Navigation der JVx iOS App in unterschiedliche Entscheidungszweige navigieren.

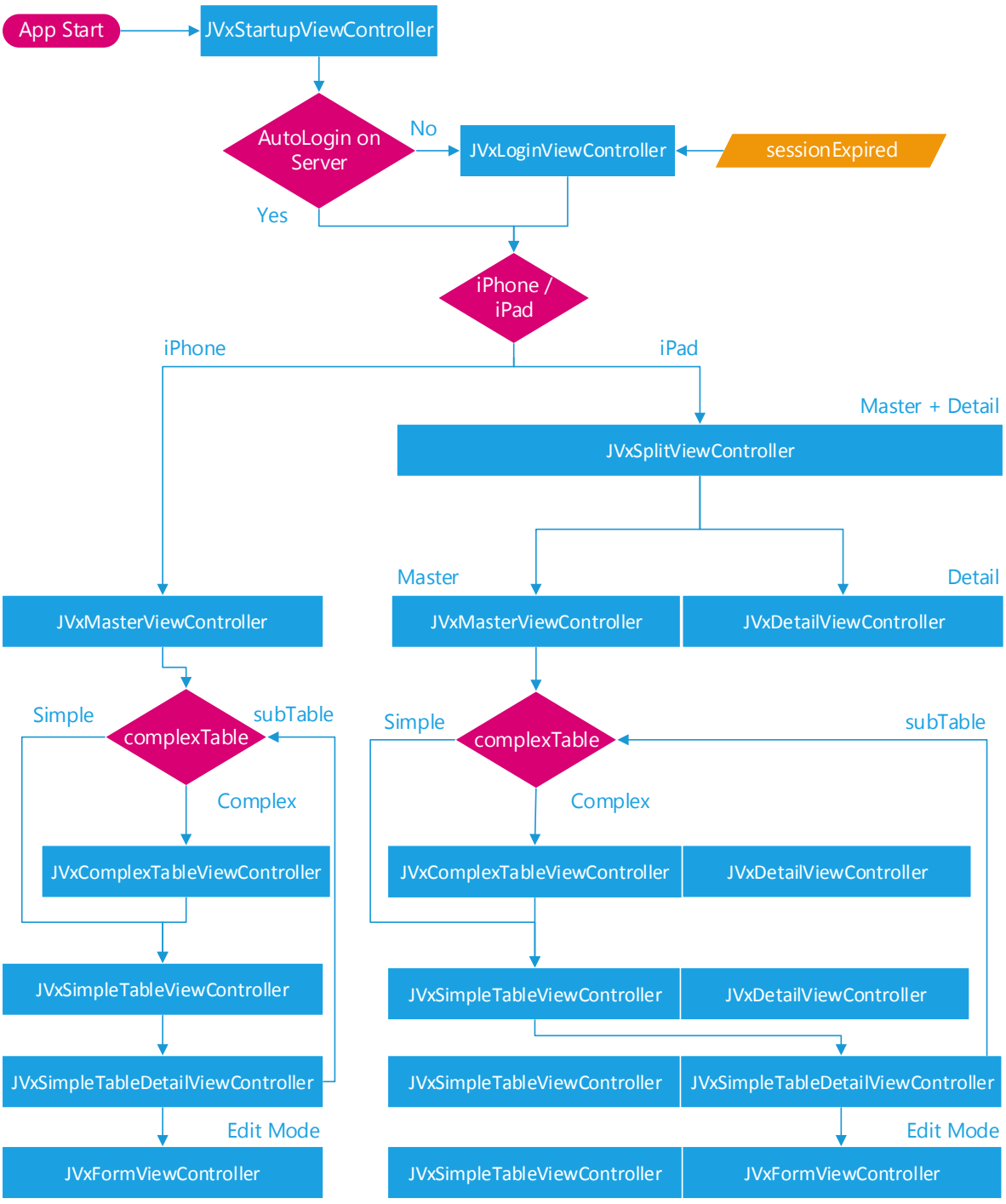


Abbildung 30: JVx iOS Konzept

Als eines der möglichen Beispiele ist es nach dem Startup Request möglich, dass der Server entweder eine „Login“-Maske für eine Anmeldung zurückgibt, oder eben auch bei einem „AutoLogin“ direkt eine Liste an Menüeinträgen mit den jeweiligen Anzeigenamen. Auch diese Menü-Listeneinträge können dann wiederum entweder einfache JVxSimpleTable-

Klassen oder JVxComplexTable Klassen sein, wie in Kapitel 4.7.2 „Model-Klassen“ beschrieben und in Abbildung 29: JVx “ visuell dargestellt wurde. Auch die jeweiligen JVxSimpleTable können wiederum Verweise über Detail-Tabellen beinhalten, welche auf der JVxSimpleTableDetailViewController-Visualisierung wiederum aufgerufen werden kann. Dadurch ist die JVx-Anwendung Server-gesteuert und in der Anwendung sehr generisch, da sehr viele unterschiedliche Navigationsstrukturen den Daten der API entsprechend zur Anzeige gebracht werden können. Neben den Unterschieden in der Navigation wird aber auch grundsätzlich zwischen den Geräteklassen iPhone und iPad unterscheiden. Am kleineren Anzeigebereich des iPhone wird der JVxNavigationController verwendet, um nach dem Login zur Menü-Anzeige zu navigieren und diese zu präsentieren. Im Gegensatz dazu wird am iPad, wie schon kurz beschrieben, der JVxSplitViewController verwendet, welcher zwei Bereiche gleichzeitig abdecken kann. Diese Aufteilung wird durch eine im Kapitel 2.6.12 „Benutzerdefinierte Storyboard-Übergänge“ beschriebene „Segue“ ermöglicht.

Um einen besseren Einblick in die Funktionen und Anwendungen der jeweiligen ViewController zu bekommen, werden diese nun wie folgt kurz einzeln erklärt:

JVxStartupViewController

In diesem initialen ViewController wird der in Listing 20: startStartupAPICallWithDelegate“ als Beispiel angeführte erste API-Aufruf vorgenommen. Durch die Übermittlung der in Listing 9: Startup Parameter“ angegebenen Client-Informationen wird die Auswertung der Antwortdaten vorgenommen und entsprechend die weitere Navigation zum JVxLoginViewController oder gleich zum JVxMasterViewController gestartet.

JVxLoginViewController

Um den in Kapitel 2.5.4 „Sicherheitsanforderungen an moderne APIs“ angegebenen Anforderungen zu entsprechen, wurde auch die Implementierung einer Authentifizierung, wie sie in Abbildung 31: JVxLoginViewController“ zu sehen ist, umgesetzt. Diese vom Benutzer/der Benutzerin angegebenen Anmeldedaten werden wie von Apple Inc. im dafür vorgesehenen Schlüsselbund wie in Kapitel 4.4.2 beschrieben mittels einer Hilfsklasse gespeichert. Der JVxLoginViewController schickt diese dann an den JVx-Server und verarbeitet wiederum die daraufhin empfangenen Daten. Diese können von einem Fehler, einem erfolgreichen Menü mit entsprechender Liste an Elementen bis hin zu einem „sessionExpired“ sehr unterschiedlich sein. Der Client informiert hier den Benutzer/die Benutzerin aber bei einem Fehler, wie im noch folgenden Kapitel 4.7.10 „Helfer-Klassen“ beschriebenen Unterkapitel „Fehlerbehandlung“ noch genauer erläutert wird. Wie in Abbildung 30: JVx iOS Konzept“ zu sehen ist, kann der JVxLoginViewController von jedem anderen ViewController aus nach einer Server-Anfrage angezeigt werden, wenn die Verbindung abgelaufen ist, der Server eine neue Authentifizierung benötigt und „sessionExpired“ als Rückgabewert liefert.

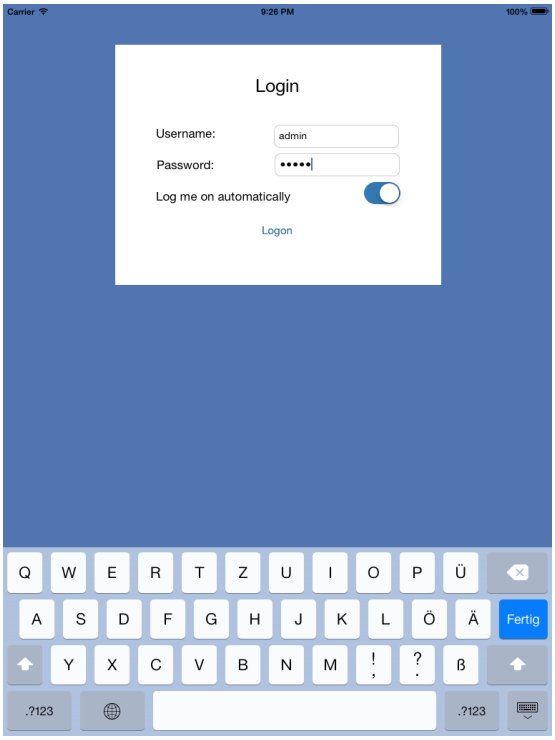


Abbildung 31: JVxLoginViewController

JVxMasterViewController

Für die Anzeige der Menüpunkte ist der JVxMasterViewController zuständig, der wie in Abbildung 36: iOS 6.x.x / 7.x.x Design Anpassungen“ am iPad auf der linken Seite oder am iPhone als eigenständiger View angezeigt wird. Neben der Menü-Liste, mit der der Benutzer oder die Benutzerin zu den Unterpunkten navigieren kann, bietet der ViewController auch die Möglichkeit, sich aktiv aus der aktuellen Server-Client-Verbindung abzumelden, um beispielsweise vertrauliche Daten besser schützen zu können. Bei einer Auswahl eines Menüpunktes wird, wie in Kapitel 4.7.3 „API Controller“ angegeben, dem Server die „OpenDetailTable“-Anfrage gestellt, welche wie zuvor schon beschrieben von der Klasse JVxComplexTable oder JVxSimpleTable sein kann. Je nach Rückmeldung wird entsprechend der JVxComplexTableViewController oder der JVxSimpleTableViewController angezeigt.

JVxDetailViewController

Wie auch in Abbildung 36: iOS 6.x.x / 7.x.x Design Anpassungen“ zu sehen ist, hat der JVxDetailViewController im JVxSplitViewController bei der Anzeige des Menüs eine Platzhalter-Funktion, die aktuell nur ein Bild anzeigt. Dieser Controller besitzt ansonsten in der aktuellen Implementierung keinerlei weitere Funktionen; könnte aber beispielsweise mit einem UIWebView zu einem firmeninternen Blog verlinken und diesen in der App anzeigen, um so den Benutzer/die Benutzerin über Neuigkeiten oder Updates informieren.

JVxComplexTableViewController

Wie in Abbildung 29: „JVx Model-Klassen“ visuell dargestellt, wird bei einem JVxComplexTable diese nach einer Auswahl des jeweiligen Eintrages, visualisiert in einer Liste, zu einem JVxSimpleTable umgewandelt und als JVxSimpleTableDetailViewController angezeigt.

JVxSimpleTableViewController

Nach der Auswahl eines einzelnen JVxSimpleTable werden diese Daten durch einen JVxSimpleTableViewController visualisiert, wie dies in Abbildung 23: Unterschiedliche datenbasierte Darstellung von Informationen“ im Storyboard als Vorlage und dann, mit Daten befüllt, in Abbildung 34: Popover iPad Splitscreen“ zu sehen ist.
Die Anzahl an anzuzeigenden Elementen wurde wie in Kapitel 4.7.6 „Konstanten“ angegeben als globale Variable „MAX_NUMBER_OF_ELEMENTS_IN_ROW“ auf vier Elemente begrenzt. Diese in Tabellenform dargestellten Einträge können mittels Suchfeld auch auf dessen Werte durchsucht beziehungsweise gefiltert werden. Hierzu wurde die UISearchBar verwendet, welche ebenfalls in der Abbildung 34: Popover iPad Splitscreen“ zu sehen ist.

Im oberen Bereich ist der Titel der jeweiligen JVxSimpleTable zu sehen und es wird, je nach Eigenschaften der JVxSimpleTable, auch die Option eines neuen Eintrages in der Navigationsbar als „+“ Icon angezeigt. Dieses Navigationsicon ist natürlich optional und wird vom JVx-Server gesteuert. Wird ein neuer Eintrag gewählt, wird ein mit Vorgabewerten befüllter JVxSimpleTableViewController angezeigt, welcher ausgefüllt und gespeichert – oder wieder verworfen – werden kann.

Im Falle der Verwendung eines iPad wird bei Erscheinen der Tabelle auch automatisch der erste Eintrag im Detailbereich auf der rechten Seite mittels eines JVxSimpleTableDetailViewController selektiert und somit angezeigt.

JVxSimpleTableDetailViewController

Die im zuvor angegebenen JVxSimpleTableViewController begrenzt angezeigten Daten werden nach einer Selektion im JVxSimpleTableDetailViewController mit allen Attributen des jeweiligen Eintrages dargestellt. Hier wurde darauf geachtet, dass die erste Zeile Serverseitig gefiltert wird und keinen BOOL Variable anzeigt, da diese bei iOS Geräten wie in 4.7.5 angegeben als UISwich angezeigt werden soll und somit keine richtige Bezeichnung darstellen kann. In Abbildung 34: Popover iPad Splitscreen“ ist auf der rechten Bildschirmseite gut zu erkennen, wie die einzelnen Einträge, mit Titel linksbündig und der dazugehörige Wert rechtsbündig, dargestellt werden.

Je nach den Eigenschaften der JVxSimpleTable werden am iPad Untertabellen als Icon der Navigationsbar angezeigt, wobei für das iPhone aus Platzgründen UIButtons unterhalb der

Wertetabelle angezeigt werden. Diese starten beim JVx-Server die „GetDetailTable“-Anfrage, welche wie zuvor schon beschrieben ebenfalls wieder von der Klasse JVxComplexTable oder JVxSimpleTable sein kann.

Da die JVxSimpleTable aber auch JVxTableActions besitzen können, werden diese ebenfalls oben in der Navigationsbar als „UIBarButtonSystemItemAction“ dargestellt, sofern der Table eine oder mehrere JVxTableActions besitzt.

Wenn der angezeigte Eintrag editierbar ist, wird dieser wie in 4.2.1 „Was bedeutet Multi-Touch-Tauglichkeit?“ durch einen UISwipeGestureRecogsnizer mittels einer „Wischbewegung“ in den Editierbaren Zustand versetzt. Es kann auch alternativ der „Edit“-Knopf in der Navigationsbar gedrückt werden, um an den JVx-Server eine Anfrage „BeforeUpdate“ zu stellen, die eine JVxForm retourniert bekommt, welche dann im JVxFormViewController zum Editieren bereitsteht.

JVxFormViewController

Der JVxFormViewController ist die zentrale Editiermaske für die JVx-Daten, welche nach dem Editieren immer noch verworfen oder mittels “UpdateDetail”, oder bei einem neuen Eintrag durch “InsertDetail”, an den Server übertragen werden können. Da die einzugebenden Werte von unterschiedlichen Typen sein können wird hier, wie auch am JVx Backend, zwischen den Klassen JVxTextCellEditor, JVxNumberCellEditor und JVxChoiceCellEditor unterschieden. Die jeweiligen Interaktionselemente werden nachfolgend im Unterkapitel 4.7.5 unterschiedlich beschrieben.

Detailliertere Informationen und zusätzliche Hinweise zu den ViewController-Klassen sind den jeweiligen Klassen, im Source Code in den Methodenbeschreibungen und auch in den Kommentaren zu entnehmen, da dies sonst den Umfang dieser Masterarbeit sprengen würde und redundant angeführt wäre.

4.7.5 Interaktionselemente

Die für mobile Geräte optimierten Benutzereingabemöglichkeiten unterscheiden sich, wie schon im vorhergehenden Kapitel 4.2.3 „Mobile Displays und deren Besonderheiten“ durch die Literatur theoretisch erörtert, sehr stark von Desktop-Eingabeoptionen. In den nachstehenden Erklärungen werden die für die Masterarbeit wesentlichen Interaktionselemente, welche für das iPhone und das iPad teilweise unterschiedlich sind, nach den Vorgaben von Apple Inc. wie in [54] beschrieben und in der Anwendung in der „MyERP“ App gezeigt.

Buttons

Für einfache Eingaben wie die Bestätigung oder die Revidierung von Eingaben werden, wie in Abbildung 32: Textfelder iPhone“ zu sehen ist, UIButtons verwendet. Werden diese in der

Navigationsbar verwendet, werden diese im Verhalten gleichwertigen Interaktionselemente durch die abgeleitete Klasse UIBarButtonItem ersetzt. [54]

Textfelder

Für die Eingabe von Werten der JVxTextCellEditor - oder auch der JVxNumberCellEditor Klasse ist wie in Abbildung 32: Textfelder iPhone“ zu sehen, wird ein auf Touch optimiertes Eingabetextfeld verwendet. Dieses blendet nach der ersten Interaktion die virtuelle Systemtastatur ein. Auch eine einfache Möglichkeit, den schon eingegebenen Wert zu löschen, ist durch den grau angezeigten „x“-Button zu sehen.

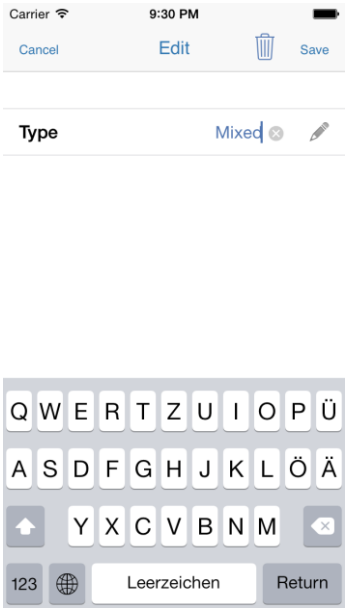


Abbildung 32: Textfelder iPhone

Alert Views

Bei Fehlern oder auch bei temporär auftretenden Anzeigeelementen wird, wie in Abbildung 33: Meldungen mittels UIAlertViews“ dargestellt, der Benutzer/die Benutzerin durch den iOS-üblichen UIAlertView benachrichtigt und kann gegebenenfalls darauf reagieren oder diese Meldung nur quittieren.

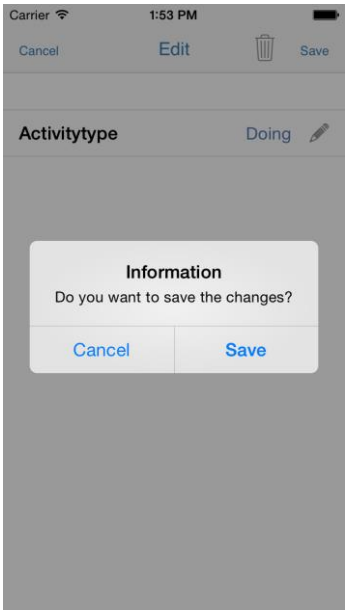


Abbildung 33: Meldungen mittels UIAlertViews

Ein UIAlertView hat als Option einen Titel, einen Beschreibungstext und mindestens einen Button, optional aber auch weitere Auswahlmöglichkeiten durch weitere Buttons. Für diese kann ausschließlich der Text gesetzt werden kann. [54]

Swich

Die in Abbildung 31: JVxLoginViewController“ zu sehen ist, wird wie für die iOS Plattform üblich für BOOL Werte der von Apple Inc. vorgeschlagene UISwich verwendet. [54]

Eingabe Picker

Für mehrfache Auswahlmöglichkeiten soll laut Apple Inc. [54], wie in Abbildung 35: Actionsheet iPhone“ oder Abbildung 34: Popover iPad Splitscreen“ dargestellt, der auf Touch optimierte UIPicker verwendet werden. Für die Umsetzung der JVx iOS Applikation wurden drei verschiedene Varianten implementiert:
Für die Anzeige von Text-basierten Auswahlmöglichkeiten wurde der „Default Picker“ mit einer Spalte implementiert; für die Datumsanzeige der „Date Picker“, welcher auf dem UIDatePicker basiert und drei Segmente für Monat, Tag und Jahr besitzt; für die vom JVx Framework bildbasierten Auswahloptionen des JVxChoiceCellEditors wurde ein „Image Picker“ umgesetzt, welcher wie der „Default Picker“ nur eine Spalte zur Auswahl hat.

Popovers

Die zuvor genannten Picker werden am iPad innerhalb von Popover verwendet. [54]

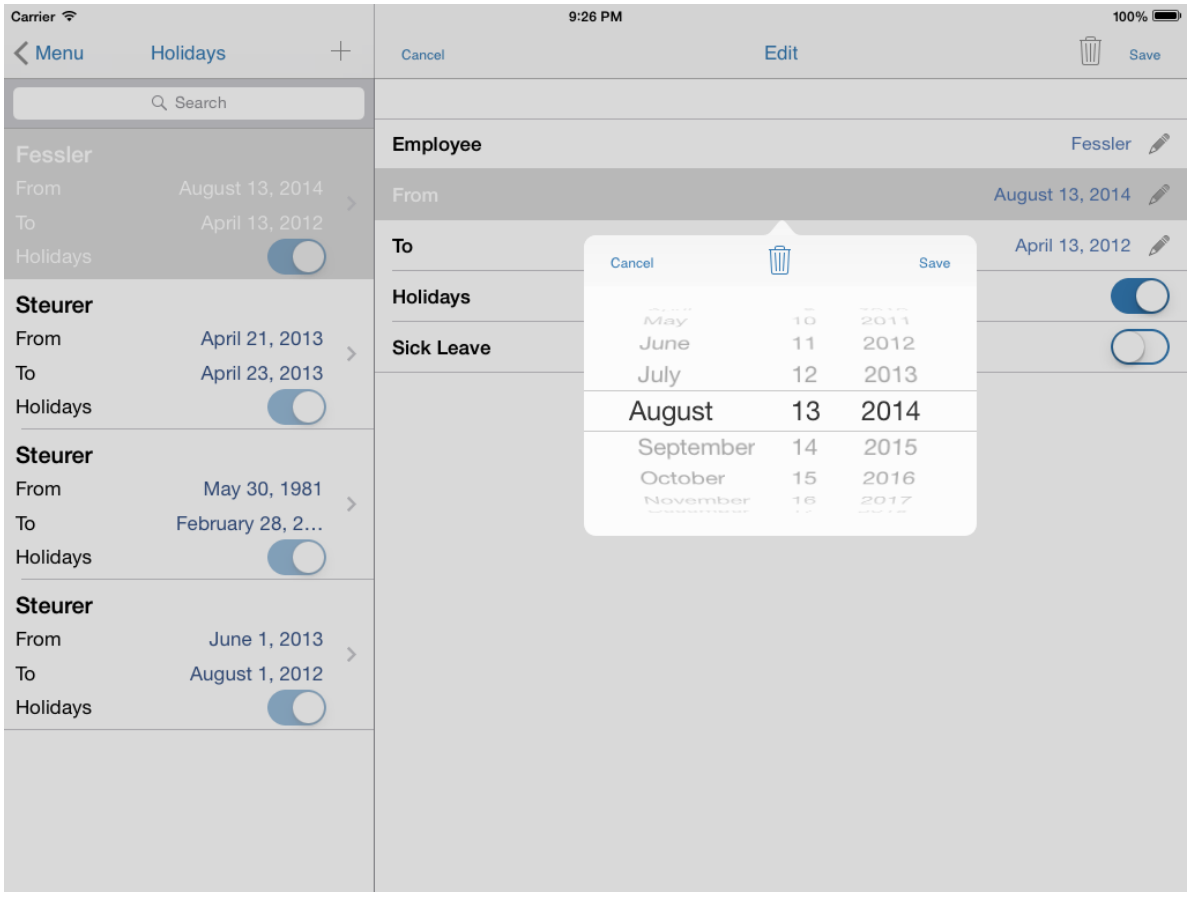


Abbildung 34: Popover iPad Splitscreen

Ein Date Picker ist in Abbildung 34: Popover iPad Splitscreen“ zu sehen; er ermöglicht die Datumseingabe, aber auch das Löschen oder Verwerfen bei einer Fehleingabe, oder das Speichern. [54]

ActionSheets

Da für die freistehende Anzeige eines Popover mit eingebettetem Picker am iPhone kein Platz vorhanden ist, wird von Apple Inc. wie in [54] beschrieben ein sogenanntes „Actions-heet“ verwendet, welches an der unteren Bildschirmseite eingeblendet wird.

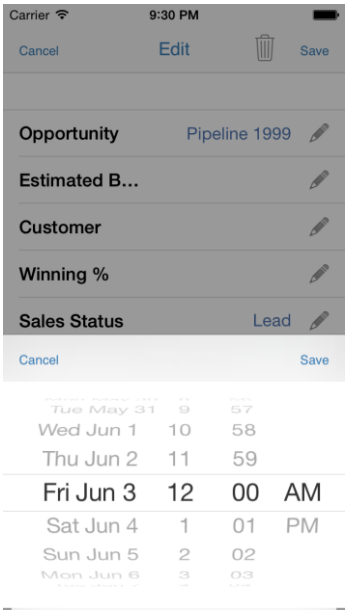


Abbildung 35: Actionsheet iPhone

Als Vergleich zum iPad ist in Abbildung 35: Actionsheet iPhone“ zu sehen dass der Picker bis an den Rand geführt wird und an die iPhone Größe angepasst ist.

4.7.6 Konstanten

Damit mehrfach verwendete Werte oder auch Basiseinstellungen wie der Applikationsname, die Serveranwendung oder der Pfad des Servers leichter zu finden sind, wurden diese zentral in der Klasse JVxConstants ausgelagert, die in Listing 21: JVx-Konstanten“ angeführt wurde. Somit kann beispielsweise sehr schnell für die gesamte JVx iOS Anwendung die Request Timeout-Zeit verändert werden.

```
NSString *const APPLICATION_NAME                    =  @"MyERP";
NSTimeInterval const REQUEST_TIMEOUT_TIME           =  60.0f;
NSString *const SERVER_BASE_URL             =  @"http://localhost:8080/JVx.mobile/services/mobile";
NSInteger const MAX_NUMBER_OF_ELEMENTS_IN_ROW  =  4;
NSString *const NUMBER_CELLEDITOR_CHARACTERS    =  @"1234567890.";
NSString *const NUMBER_FORMAT                       =  @"#.##";
NSString *const DATE_FORMAT                         =  @"yyyy-MM-dd hh:mm:ss";
NSInteger const NEW_ENTRY_INDEX                     =  -1;
```

Listing 21: JVx-Konstanten

Dieser Ansatz ermöglicht eine schnelle Anpassung an Wünsche von Kunden und Kundinnen oder auch an höhere Sicherheitsanforderungen, da hier sehr schnell die Server Base URL von http auf https umgestellt werden kann. Dieser Sicherheitsmechanismus wurde in 2.5.4 „Sicherheitsanforderungen an moderne APIs“ schon theoretisch erklärt.

4.7.7 App State

Die JVxAppState-Klasse wurde als Singleton umgesetzt, um die JVx iOS App-Daten und die Zustände der Einstellungen jederzeit und von jeder Stelle aus zugänglich zu machen, wie dies in Kapitel 2.6.13 “Angewandte Technologien und Design Patterns“ beschrieben wurde.

Alle von der API empfangenen Daten der Views, wie beispielsweise für das Menü oder den Login werden im „Container“-Attribut „responeViews“, welches von der Klasse JVxResponseContainer ist, einzigartig, aber nicht persistent, gespeichert.Aber auch einfache Zustände wie der BOOL-Zustand „hasInternet“, welcher im Kapitel 4.6 „Erreichbarkeit“ evaluiert wurde, werden in dieser „Datenhalter“-Klasse gesammelt und nicht persistent gespeichert.

4.7.8 User Defaults

Um der JVx iOS App die Möglichkeit zu geben, Informationen auch persistent zu speichern, werden die NSUserDefaults verwendet, die von David Chisnall wie folgt beschrieben werden:

> *„One of the problems on any system is how to store preferences for an application. Numerous solutions have been suggested for this problem, from individual configuration files to a centralized registry. OS X picks a path somewhere in the middle. Each application has a property list file containing a dictionary associated with it. This is accessed via the NSUserDefaults class, which also handles notifying parts of an application of changes to individual keys.” [42], S.115*

Diese nicht-flüchtigen Informationen, die auch nach einem App-Neustart zur Verfügung stehen. Bei einem manuellen Log Out wird beispielsweise das Merkmal “autoLogin” auf „Nein“ gesetzt, welches die Sicherheit der Client-Anwendung erhöht.

```
+ (void)setLocalAutoLoginStateTo:(BOOL)value
{
  [[NSUserDefaults standardUserDefaults] synchronize];
  NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
  [defaults setBool:value forKey:@"autoLogin"];
  [defaults synchronize];
}
+ (BOOL)getLocalAutoLoginState
{
  [[NSUserDefaults standardUserDefaults] synchronize];
  NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
  return [defaults boolForKey:@"autoLogin"];
}
```

Listing 22: JVx NSUserDefaults

Wie in Listing 22: JVx NSUserDefaults“ zu sehen ist, werden die globalen „standardUserDefaults“ synchronisiert, dann mit dem neuen Wert überschrieben und zur persistenten Speicherung neu synchronisiert.

4.7.9 App Settings

Wie in Abbildung 24: Einstellungen am iPad mittels UISplitViewController“ gezeigt wird, haben System-Applikationen in den Einstellungen eigene Optionsmenüs. Hier kann auch eine vom Entwickler/der Entwicklerin erstellte App Einstellungen anlegen, welche in der Setting.bundle einstellbar sind.

```
NSString *mainBundlePath = [[NSBundle mainBundle] bundlePath];
NSString *settingsPath = [mainBundlePath stringByAppendingPathComponent:@"Settings.bundle/Root.plist"];
NSMutableDictionary *settings = [NSMutableDictionary dictionaryWithContentsOfFile:settingsPath];
```

Listing 23: JVx Settings.bundle

Zum Auslesen der Einstellungen im Settings-Menü wird wie in Listing 23: JVx Settings.bundle“ das gesamte File als NSDictionary übergeben. Danach kann auf die gewünschten Benutzereinstellungen mittels KVO zugegriffen und diese somit ausgelesen werden.

4.7.10 Helfer-Klassen

Einige Helfer-Klassen, welche auch als Utilities bezeichnet werden, wurden schon beschrieben, wie beispielsweise Benutzer-/Passwort-Einstellungen oder die JSON-Helfer-Klassen.

Weitere Utilities wie die Fortschrittsanzeige, welche eine „Laden…“ Anzeige bei einer Serverkommunikation visualisiert, oder auch Funktionen zur Geräte-Identifikation wurden in einzelne Klassen aufgegliedert, um einen besseren Überblick, aber auch eine gute Wiederverwend- und Erweiterbarkeit zu garantieren.

Die Zip-Helfer-Klassen ermöglichen das Entpacken von Zip-Dateien, welche für den effektiven und gesammelten Download von Bildern oder auch Übersetzungen im XML-Format benötigt werden. Wie in Abbildung 25: JVx App "Sandbox"“ zu sehen ist, werden diese in das Dokumenten-Verzeichnis entpackt und stehen der Anwendung dann zur Verfügung.

Besonders zu erwähnende Hilfsklassen sind die Klassen für die Fehlermeldung und die Übersetzungen, welche nun nachfolgend beschrieben werden.

Übersetzungen

Aktuell werden vom JVx Framework die Sprachen Deutsch und Englisch unterstützt, doch um die iOS Anwendung mehrsprachig zu gestalten wurde die JVxTranslationUtils erstellt, welche mittels der Funktion getTranslationForString den zu übersetzenden Text in dem schon zuvor erwähnten Übersetzungs-XML sucht und übersetzt retourniert.

```
NSString *translatedText = [JVxTranslationUtils getTranslationForString:@"Ok"];
```

Listing 24: JVx Übersetzungen

Wie in Listing 24: JVx Übersetzungen“ zu sehen ist, werden auch Menü-Eingaben wie ein einfaches „OK“ immer mittels dieser Hilfsklasse gesetzt. Dies ermöglicht eine Mehrsprachigkeit der iOS App, sobald das JVx Framework dies unterstützt, ohne Änderung am iOS Client. Das iOS Betriebssystem unterscheidet hier auch noch unter Lokalisierung (Datumsformate, Währungen) und der Sprache. Hier wurde in der aktuellen Version der JVx iOS Applikation nur die Sprache als Basis für die Formatierung der API-Daten verwendet, da hier die fehlende Lokalisierung des JVx Frameworks keine Kompatibilität ergeben hätte.

Fehlerbehandlung

Da eine Server-Client-Kommunikation auch Fehler haben kann, beispielsweise durch eine unterbrochene Internetverbindung oder einen Timeout, wurde in der Klasse JVxErrorViewHelper ein leicht erweiterbares Fehlerhandling geschaffen.

Bei fehlgeschlagenen Verbindungen wird entweder ein APIErrorWithError oder ein handleNoInternetErrorView bei fehlender Konnektivität angezeigt. Hierzu wird bei einem dieser API-Fehler der jeweilige übersetzte Fehler verarbeitet und dem Benutzer/der Benutzerin angezeigt.

Weiter werden von der API auch Meldungen an den Benutzer/die Benutzerin gegeben, wenn serverseitig Fehler auftreten oder der Benutzer/die Benutzerin über besondere Vorkommnisse informiert werden muss. Die aktuell unterstützten Typen von Fehlmeldungen sind wie folgt:

Komponenten Name	**Beschreibung**
message	Ist die Basisklasse, von der alle Meldungen abgeleitet werden, da diese dann allen abgeleiteten Klassen die JVx-Elementen Identifikationsparameter als Attribut zur Verfügung stellt.

message.error	Wie der Titel „Error“ schon erahnen lässt, wird hiermit ein Fehler mit Fehlertext vom Server an den Client geschickt, um diesen zu informieren.
message.information	Sollte die Information an den Client nur als Zusatzinformation für den Benutzer/die Benutzerin genutzt werden, kann diese mit dem Schlüssel „message.information“ übertragen werden.
message.sessionexpired	Sollte der Client längere Zeit keine Kommunikation mit dem Server haben, wird bei einem erneuten Request zur Sicherheit ein „message.sessionexpired“ zurückgeschickt, und der Client kann dann auf diesen Serverzustand reagieren und gegebenenfalls ein erneutes Einloggen vom Benutzer/der Benutzerin verlangen.

Tabelle 7: JVx Fehlertypen

Um die in Tabelle 7: JVx Fehlertypen“ angegebenen Fehler an jeder Stelle der JVx iOS Applikation anwenden zu können, wird nun in der handleError-Methode, der Priorität nach geordnet, die Fehlerüberprüfung der zuvor von der API erhaltenen Daten vorgenommen. Hierzu wird auf die in 4.7.7 „App State“ beschriebene zentrale Datenhaltung der App zugegriffen. Es wird, wie in Listing 25: JVxErrorViewHelper“ abgebildet, zunächst immer auf den jeweiligen Fehler geprüft, gegebenenfalls angezeigt und dieser dann wieder gelöscht, da er behandelt wurde. Nur beim „message.sessionexpired“ wird wie in 2.6.13 „Notifications“ beschrieben eine globale Nachricht verschickt, so dass der jeweilige aktive Controller darauf reagieren kann. Bei diesem Sonderfall wird auch erst bei der Fehlerbehandlung im Controller selbst der Status zurückgesetzt. Damit kann sichergestellt werden, dass dem Benutzer/der Benutzerin ein Login-Controller angezeigt wird.

```
@implementation JVxErrorViewHelper
+ (void) handleError
{
  [[JVxAppState instance] hideActivityIndicator];
  if ([JVxAppState instance].responseViews.ErrorView != nil )
  {
    UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:[JVxAppState
        instance].responseViews.ErrorView.title message:[JVxAppState
        instance].responseViews.ErrorView.message delegate:nil cancelButtonTitle:[JVxTranslationUtils
        getTranslationforString:@"Ok"] otherButtonTitles:nil];
    [alertView show];
    [JVxAppState instance].responseViews.ErrorView = nil;
  }
```

```
  if ([JVxAppState instance].responseViews.InformationView != nil )
  {
    UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:[JVxAppState
        instance].responseViews.InformationView.title message:[JVxAppState
        instance].responseViews.InformationView.message delegate:nil
        cancelButtonTitle:[JVxTranslationUtils getTranslationforString:@"Ok"]
        otherButtonTitles:nil];
    [alertView show];
    [JVxAppState instance].responseViews.InformationView = nil;
  }
  if ([JVxAppState instance].responseViews.SessionexpiredView != nil )
  {
    [[NSNotificationCenter defaultCenter] postNotificationName:@"sessionexpired" object:nil];
  }
}
```

Listing 25: JVxErrorViewHelper

Dieser Aufbau ermöglicht es, leicht neue Fehlertypen zu integrieren und alle schon implementierten Fehler-Prüfungsstellen ebenfalls ohne Änderungen zu erweitern.

4.8 Anpassungen an iOS7

Zum Zeitpunkt des Entwicklungsbeginnes der JVx iOS Anwendung war die aktuelle Hauptversion des iOS Betriebssystems bei 6.x.x. Im Verlauf der Arbeiten wurde von Apple Inc. die neue Versionsnummer 7.x.x veröffentlicht. Diese neue Version des mobilen Betriebssystems hat die in die Jahre gekommene Benutzeroberfläche sehr stark verändert. Um für die Entwickler/die Entwicklerinnen einen einfachen Umstieg auf die neue iOS Version zu ermöglichen, hat Apple Inc. in [61] einen „iOS 7 UI Transition Guide“ zur Verfügung gestellt. In diesem wird zu Beginn auch gleich auf die Änderungen, beispielsweise bei den UIButton, die nun keine „Ränder“ mehr haben, wie folgt hingewiesen:

> “*iOS 7 introduces many UI changes, such as borderless buttons, translucent bars, and full-screen layout for view controllers. Using Xcode 5, you can build a project for iOS 7 and run it in iOS 7 Simulator to get a first glimpse of the way the app looks with iOS 7 UI.“ [61]*

Entsprechend den „best practice“-Anweisungen von Apple Inc. wurde auch der JVx Client an iOS 7.x.x angepasst, aber dennoch nicht auf die Abwärtskompatibilität von iOS 6.x.x vergessen.

4.8.1 Design-Anpassungen

Um eine einfache und global wirksame Konfiguration der Benutzerelemente in der JVx iOS Applikation zu ermöglichen, wurden alle Erscheinungsanpassungen in die JVxAppearance-

Klasse ausgelagert, welche, wie in 2.6.10 „Der App-Lebenszyklus” beschrieben, in der Methode „didFinishLaunchingWithOptions” aufgerufen wird.

Abbildung 36: iOS 6.x.x / 7.x.x Design Anpassungen

Wie in Abbildung 36: iOS 6.x.x / 7.x.x Design Anpassungen” links zu sehen ist, wurde bei iOS 6.x.x beispielsweise die Navigationsbar in einem Blauton eingefärbt, und auch der Titel mit der Systemschrift „Arial-BoldMT“ in Weiß dargestellt. Im Gegensatz dazu ist rechts iOS 7.x.x zu sehen, bei dem wie in [62] beschrieben sehr viele Anpassungen von Apple Inc. bezüglich einem „leichterem“ Design gemacht wurden. Es wurde auch die Möglichkeit der neuen globalen Farbgebung „TintColor“ eingeführt. Diese Design-Umstellung führte, wie von Apple Inc. selbst in [62] beschrieben, zu Layout- und Farbanpassungen. Auch Anpassungen an die neue Systemschrift „Helvetica Neue“ wurden, wie in Listing 26: Design Anpassungen iOS6/7“ zu sehen ist, vorgenommen. Es wird bei den UI Anpassungen nun zwischen „iOS 7 und neuer“ und allen „älteren“ Versionen unterschieden. Durch diesen Aufbau könnten sehr leicht Anpassungen speziell für neuere, kommende Versionen realisiert werden.

```
if ([UIDevice isIOS7AndAbove])
{  // iOS 7
  // UIBarButtonItem
  [[UIBarButtonItem appearance] setTitleTextAttributes: @{
              NSForegroundColorAttributeName: JVxTextAttributeTextColor,
              NSFontAttributeName: [UIFont fontWithName:@"Helvetica Neue" size:0.0f]
  } forState:UIControlStateNormal];
  [[UIBarButtonItem appearance] setTintColor:[UIColor DEFAULTCOLOR]];
}
else
{  // iOS 6
```

```
  // UIBarButtonItem
  [[UIBarButtonItem appearance] setTitleTextAttributes: @{
        UITextAttributeTextColor: [UIColor whiteColor],
        UITextAttributeTextShadowColor: JVxTextAttributeTextShadowColor,
        UITextAttributeTextShadowOffset: [NSValue valueWithUIOffset:UIOffsetMake(0.0f, 0.1f)],
        UITextAttributeFont: [UIFont fontWithName:@"Arial-BoldMT" size:0.0f]
  } forState:UIControlStateNormal];
}
```

Listing 26: Design Anpassungen iOS6/7

Sehr gut beschrieben wurden die generellen UI-Änderungen von Apple Inc. wie folgt:

> *“iOS 7 brings several changes to how you lay out and customize the appearance of your UI. The changes in view-controller layout, tint color, and font affect all the UIKit objects in your app.” [62]*

In Listing 26: Design Anpassungen iOS6/7“ ist die „Appearance“-Anpassung am Beispiel der UIBarButtonItem-Klasse zu sehen, welche in Abbildung 36: iOS 6.x.x / 7.x.x Design AnpassungenAbbildung links unten mit dem Titel „Logout“ zu sehen ist.

4.8.2 64 Bit-Architektur

Wie in [63] beschrieben, hat Apple Inc. neben den großen Design-Anpassungen auch eine große Architektur-Erweiterung auf 64-bit Prozessoren vorgenommen. Hierzu wurde von Apple Inc. ebenfalls eine Anleitung mit dem Titel „64-bit Transition Guide for Cocoa Touch“ [64] zur Verfügung gestellt. Der mit dem iPhone 5S neu vorgestellte A7 ARM Prozessor verfügen als erster mobiler Smartphone -/Tablet Prozessor über einen 64-bit Befehlssatz, der aber auch zu bestehenden 32-bit iOS Anwendungen kompatibel bleibt. Neben der doppelten Anzahl an Integer- und floating-point-Registern des 64-bit Designs der CPU wurde auch der LLVM Compiler optimiert, wie dies von Apple Inc. selbst angegeben wird:

> “*The 64-bit architecture supports a new and streamlined instruction set that supports twice as many integer and floating-point registers. Apple’s LLVM compiler has been optimized to take full advantage of this new architecture. As a result, 64-bit apps can work with more data at once for improved performance.” [64]*

Die Änderungen am Quellcode sind bei der JVx iOS Applikation nicht groß ausgefallen, es mussten nur ein paar externe Frameworks wie beispielsweise RestKit aktualisiert werden. Auch die Umstellung der „Build Setting“ musste wie folgt umgestellt werden:

> *“If you’re updating an older app for iOS 7, and if you want it to run natively on a 64-bit device, change the Architectures build setting for your app target to “Standard architectures (including 64-bit).”” [63], S.200*

Dies waren aber keine großen Änderungen, da von Beginn an kein iOS 5.x.x Support der JVx iOS App erforderlich war. Diese Abwärtskompatibilität hätte mit iOS 7.x.x aufgegeben werden müssen.

5 Schlussfolgerungen und Ausblick

In diesem letzten Kapitel wird nun nochmals die Forschungsfrage reflektiert und die Kriterien einer erfolgreichen iOS Erweiterung des JVx Frameworks erörtert. Als weiteren Ausblick werden noch mögliche und erforderliche Erweiterungen der neuen iOS Applikation erörtert und eine Zusammenfassung über den aktuellen Stand der Entwicklung gegeben.

5.1 Hauptkriterium einer erfolgreichen iOS Erweiterung

Als eines der wichtigsten Kriterien einer iOS App kann die Überprüfung durch Apple Inc. selbst gesehen werden, wenn die App in den App Store eingereicht wird. Hier wird jede App, und auch jedes Update einer App, genauestens nach den iOS Human Interface Guidelines geprüft – und sehr oft beim ersten Versuch abgelehnt. Hier werden auch Netzwerkfunktionalität und Sicherheitskriterien, wie die Verwendung der Schlüsselbund-Funktion für Passwörter, begutachtet. Die Kriterien einer erfolgreichen Einreichung in den Apple App Store sind leider nicht öffentlich einsehbar, doch werden diese von den Entwicklern/Entwicklerinnen als sehr konsequent erachtet und erfordern oft hohen Aufwand. Die neu entwickelte JVx iOS App wurde zusammen mit der SIB Visions als „Preview"-Client, der mit Hilfe der Targets & Schemes eines Xcode-Projekts eingeschränkte Möglichkeiten bietet, erfolgreich eingereicht.

Der JVx Client (Preview) kann unter folgender Adresse vom Apple App Store geladen werden: https://itunes.apple.com/at/app/jvx-client-preview/id829822359?mt=8

5.2 Reflexion der Forschungsfrage

Wird nun auf die im Kapitel 1.4 „Formulierung der Forschungsfrage" formulierte Fragestellung:

> *„Wie kann eine Multi-Touch optimierte native iOS App Erweiterung für das JVx ERP Applikation Framework designt und implementiert werden?"*

zurück geblickt, kann dies unter Berücksichtigung der Erklärungen im Kapitel 2 „Stand der Literatur" und der beschriebenen Umsetzung der MyERP-Applikation für die jeweilige Unterfrage wie folgt beantwortet werden:

> *„Wie und an welcher Stelle kann das JVx ERP Applikation Framework am besten mobil erweitert werden?"*

Für die mobile Erweiterung von JVx wurde die Entwicklung eines „Mobile Server", wie in 2.4.3 „Hotspot für mobile Erweiterungen" beschrieben, als flexibler und gut erweiterbarer Ansatz eruiert. Dieser Hotspot wurde im Kapitel 3 „RESTful JVx Erweiterung" mittels den

„Designentscheidungen der Schnittstelle“ in der Umsetzung beschrieben. Im Unterkapitel „Vorteile der RESTful Integration in das JVx Framework“ wurde die Entkopplung der Programmiersprache JAVA durch den Einsatz einer API beschrieben.

> *„Was versteht man unter einer nativen iOS App und wie wird eine solche Applikation und ihr Projekt aufgebaut?“*

Diese theoretische Grundlagenfrage wurde anhand der Erörterung in Kapitel 2.6 „Erklärung der Grundlagen der iOS Plattform“ hinreichend beschrieben, sodass, darauf basierend, im Gliederungspunkt 4 „Entwicklung der generischen iOS App“ die praktische Entwicklung der JVx iOS Anwendung darauf konzeptionell aufgebaut werden konnte.

> *„Wie kann das klassische JVx ERP Userinterface an Multi-Touch angepasst werden?“*

Inhaltlich stellt dies einen der Kernpunkte dieser Masterarbeit dar, und daher wird dieser Fragestellung in Kapitel 4.2 mit dem Titel „Anforderungen an Touch optimiertes mobiles Design“ auch eine Reihe von untergliederten Erklärungen gewidmet. Es werden Detailfragen und –aspekte wie

- Was bedeutet Multi-Touch-Tauglichkeit?
- iOS Human Interface Guidelines
- Mobile Displays und deren Besonderheiten

recherchiert und erklärt. Aber auch die in 4.3 beschriebene Simplifizierung von komplexen Daten leistet einen wesentlichen Beitrag zur Beantwortung dieser Frage. In der Anwendung kommen diese der Literatur entnommenen Erkenntnisse in der Multi-Touch-angepassten JVx-Variante für iOS im Kapitel 4.7 der Beispielsanwendung MyERP mit deren JVxViewController Klassen und Interaktionselemente als eine mögliche Lösung, zur praxistaugleichen Umsetzung.

Durch die Zusammenfassung und nochmalige Reflexion der Forschungsfrage sollte sichergestellt werden, dass inhaltlich schnell zu den jeweiligen Antworten gefunden wird und auch nochmals unterstrichen werden, dass eine vollständige Beantwortung dieser gegeben ist.

5.3 Mögliche Erweiterungen

Wie im ersten Kapitel der Einleitung beschrieben, wurde die Erweiterung von JVx durch den iOS Client auf den Funktionsumfang der Beispielsanwendung „MyERP“ begrenzt, um dem Aufwand und der Komplexität einer Masterarbeit zu entsprechen.

Daher wurde schon während der Entwicklung der JVx iOS Anwendung festgestellt, dass Erweiterungen wie die Unterstützung von Dateien (Bilder, PDF-Dateien,…) dem Benutzer/der Benutzerin sehr viele Vorteile bringen würde und den mobilen Client der Desktop-Variante funktional sehr annähern würde. Hierbei müsste für jeden Datentyp eine entsprechende Darstellung überlegt werden. Der Upload und Download von großen Dateien müsste hier auch in Zusammenhang mit der „offline“-Fähigkeit konzeptioniert werden.

Einfachere Erweiterungen wären der Support von mehrzeiligen Textfeldern und Texteingabefeldern sowie servergesteuerte Eingabelimitierungen. Auch der Support von getrennten Einstellungen der Sprache und der Lokalisierung, wie in 4.7.10 „Übersetzungen“ beschrieben, würde zu einer besseren internationalen Nutzung von JVx beitragen.

Für einen professionellen Einsatz für Unternehmen ist einer der wichtigsten Aspekte die Einführung von einem speziellem Test Target, der wie folgt beschrieben wird:

> *“You can also create targets that process code in other ways—for example, to run a selection of optional test macros to check that important features work correctly.” [59], S.273*

Somit könnten automatisierte UI-Tests, aber auch funktionale Tests eingeführt werden, die dann bei jedem „Build“ automatisiert die Qualität sicherstellen könnten.

Von Seiten des SIB Visions Teams wurde noch die Möglichkeit formuliert, die iOS App serverseitig stylen zu können, sowie die des Supports von Diagrammen bei der Darstellung von zahlenbasierten Anzeigen. Auch eine „Password ändern” oder „Password vergessen” Funktion wurde schon angedacht.

Da JVx Open Source ist und somit von jedem Benutzer/jeder Benutzerin verändert, aber auch erweitert werden kann, steht neuen Anforderungen an den JVx iOS Client nichts im Wege.

6 Diskussion und Fazit

Als Abschluss dieser Masterarbeit wird nun noch aus der Sichtweise des Autors ein persönliches Fazit zur thematischen und allgemeinen Erfahrung mit der Masterarbeit und dem Thema „Design und Implementierung einer Multi-Touch optimierten nativen iOS App für das JVx ERP Applikation Framework“ gegeben.

6.1 Themenspezifisch

Zum Zeitpunkt der Themenauswahl der Masterarbeit waren die iOS Programmierkenntnisse des Autors noch auf einem „Beginner“-Niveau, da das Vorstudium und die Berufspraxis mehr im Wirtschaftsinformatik-Bereich des Produktmanagements für mobile Anwendungen angesiedelt waren. Die Aufgabe war aber sehr reizvoll, da dadurch die Objective-C und iOS Erfahrungen ausgebaut werden konnten. Vor allem der selbständige Aufbau der JVx iOS Applikation in Kombination mit der Server-Client-Verbindung machte das Thema sehr reizvoll. Diese anfängliche Ungewissheit wurde sehr schnell durch Literaturrecherche und der Erörterung von Beispielen beseitigt, bis hin zur Fähigkeit, eine funktionierende und immer weiter wachsende iOS Applikation zu kreieren. Auch die Unterschiede zur JAVA-basierten Server-, aber auch Android-Implementierung bei den Abstimmungen mit der SIB Visions und/oder mit Herrn Hofer Michael, B. Sc. waren sehr interessant und brachten fachlich sehr viele neue Kenntnisse.

6.2 Allgemein zur Arbeit

Durch die Erfahrungen, die während der Masterarbeit gemacht wurden, ist es dem Autor nun sicher möglich, bessere Projekte von mobilen Apps zu leiten und auch bessere Zeit- und Kostenabschätzungen zu erstellen. Auch die Definition einer API-Spezifikation oder eines mobilen Navigationskonzeptes mit einem iOS 6.x.x oder 7.x.x Design können nun wesentlich besser erfasst und umgesetzt werden. Das strukturierte und systematische Herangehen an eine neue mobile Plattform für ein bestehendes Projekt, wie es das JVx Framework schon war, war eine sehr gute Erfahrung und daher sehr lehrreich und wichtig für die weitere persönliche Entwicklung des Autors.

Literaturverzeichnis

[1] T. S. Group, 1995. [Online]. Available: http://www.projectsmart.co.uk/docs/chaos-report.pdf. [Zugriff am 8 April 2014].

[2] P. Hamilton, Dynaxity. Management von Dynamik und Komplexität im Softwarebau, Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2007.

[3] Apple Inc., „iOS App Programming Guide: The iOS Environment," Apple Inc., 23 Oktober 2013. [Online]. Available: https://developer.apple.com/library/ios/documentation/iphone/conceptual/iphoneosprogrammingguide/TheiOSEnvironment/TheiOSEnvironment.html. [Zugriff am 13 April 2014].

[4] Bibliographisches Institut GmbH, „Duden | App," [Online]. Available: http://www.duden.de/rechtschreibung/App. [Zugriff am 13 April 2014].

[5] Apple Inc., „Apple," Apple Inc., [Online]. Available: http://www.apple.com/. [Zugriff am 13 April 2014].

[6] O. S. Initiative, „The Open Source Initiative," [Online]. Available: http://opensource.org/. [Zugriff am 13 April 2014].

[7] SIB Visions GmbH, „JVx mobile," SIB Visions GmbH, [Online]. Available: http://sourceforge.net/projects/jvxmobile/. [Zugriff am 13 April 2014].

[8] A. Kelly, Changing software development. learning to be agile, Chichester, England ;Hoboken, NJ: John Wiley, 2008.

[9] L. Wroblewski, Mobile first., New York, NY: A Book Apart, 2011.

[10] D. Jacobson, G. Brail und D. Woods, APIs A Strategy Guide, Sebastopol: O'Reilly Media, Inc, 2011.

[11] Sencha Inc., „HTML5 App Development for Desktop and Mobile. JavaScript Frameworks and Dev Tools from Sencha. | Home," [Online]. Available: http://www.sencha.com/. [Zugriff am 10 April 2014].

[12] Sencha, „JavaScript Framework for Building Rich Desktop Web Applications | Sencha Ext JS | Products," Sencha, [Online]. Available: http://www.sencha.com/products/extjs/. [Zugriff am 15 April 2014].

[13] jQuery Foundation, „jQuery Mobile," jQuery Foundation, [Online]. Available: http://jquerymobile.com/. [Zugriff am 10 April 2014].

[14] The jQuery Foundation, „jQuery,“ The jQuery Foundation, [Online]. Available: http://jquery.com/. [Zugriff am 15 April 2014].

[15] Adobe, „PhoneGap,“ http://phonegap.com/, [Online]. Available: http://phonegap.com/. [Zugriff am 10 April 2014].

[16] Oracle, „java.com: Java + Sie,“ 26 Februar 2014. [Online]. Available: http://www.java.com/de/. [Zugriff am 11 April 2014].

[17] SIB Visions GmbH, „SIB Visions,“ SIB Visions GmbH, [Online]. Available: http://www.sibvisions.com/. [Zugriff am 11 April 2014].

[18] W. Stefan, Umsetzung einer auf Java EE 6 basierenden generischen Persistenzumgebung für das quelloffene Framework JVx, Wien: FH Technikum Wien, 2012.

[19] C. Schröpfer, SOA-Management-Framework. ein ganzheitliches, integriertes Konzept für die Governance Serviceorientierter Architekturen, Berlin: Gito-Verl., 2010.

[20] M. Hoffmann, Dokumentation der Entwicklung und des Konzepts eines Frameworks., 1. Aufl. Hrsg., s.l.]: Diplomica, 2004.

[21] Wirtschaftskammer Österreich, „Klein- und Mittelbetriebe in Österreich,“ Wirtschaftskammer Österreich, [Online]. Available: https://www.wko.at/Content.Node/Interessenvertretung/ZahlenDatenFakten/KMU_Definition.html. [Zugriff am 14 April 2014].

[22] N. Gronau, Enterprise resource planning und supply chain management. Architektur und Funktionen, München [u.a.]: Oldenbourg, 2004.

[23] SAP, „SAP Software & Solutions | Business Applications & IT,“ SAP, [Online]. Available: http://www.sap.com/index.html. [Zugriff am 14 April 2014].

[24] SIB Visions GmbH, „Paketübersicht,“ SIB Visions GmbH, [Online]. Available: http://www.sibvisions.com/de/jvxmdocumentation/88-jvxpackages. [Zugriff am 14 April 2014].

[25] SIB Visions GmbH, „Features,“ SIB Visions GmbH, [Online]. Available: http://www.sibvisions.com/de/jvxmfeatures. [Zugriff am 14 April 2014].

[26] SIB Visions GmbH, „Hello World,“ SIB Visions GmbH, [Online]. Available: http://www.sibvisions.com/de/jvxmdocumentation/94-jvxhelloworld. [Zugriff am 14 April 2014].

[27] SIB Visions GmbH, „Erste JVx Applikation," SIB Visions GmbH, [Online]. Available: http://www.sibvisions.com/de/jvxmdocumentation/86-jvxfirstappl. [Zugriff am 14 April 2014].

[28] SIB Visions GmbH, „Systemarchitektur," SIB Visions GmbH, [Online]. Available: http://www.sibvisions.com/de/jvxmdocumentation/85-jvxsysarch. [Zugriff am 11 April 2014].

[29] Oracle, „1 JavaFX Overview (Release 8)," Oracle, [Online]. Available: http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784. [Zugriff am 14 April 2014].

[30] Vaadin, Inc., „Vaadin - thinking of U and I," Vaadin, Inc., [Online]. Available: https://vaadin.com. [Zugriff am 15 April 2014].

[31] G. Block, P. Cibraro, P. Felix, H. Dierking und D. Miller, Designing Evolvable Web APIs with ASP.NET., 1. ed. Hrsg., Sebastopol, CA: O'Reilly & Associates, 2013.

[32] M. Masse, REST API Design Rulebook., Sebastopol: O'Reilly Media, Inc, 2011.

[33] J. Sandoval, RESTful Java web services. Master core REST concepts and create RESTful web services in Java, Birmingham [u.a.]: Packt Publ., 2009.

[34] T. Berners-Lee, Adobe Systems, [Online]. Available: http://www.rfc-editor.org/rfc/rfc3986.txt. [Zugriff am 16 April 2014].

[35] Apple Inc., „Apple Developer," Apple Inc., [Online]. Available: https://developer.apple.com/. [Zugriff am 30 April 2014].

[36] E. S. Maurice Sharp, Learning iOS - A Hands-on Guide to the Fundamentals of iOS Programming, Pearson Education, Inc., 2014.

[37] E. Gamma, Design patterns. elements of reusable object-oriented software, 37th print Hrsg., Boston [u.a.]: Addison-Wesley, 2009.

[38] J. Tittel und J. Baumann, Apps für iOS entwickeln. am Beispiel einer realen App ; [kompakter Schnelleinstieg für IT-Profis], München: Hanser, 2013.

[39] J. Nutting und P. Clark, Learn Cocoa on the Mac., Second Edition Hrsg., Berkeley, CA ;s.l: Apress, 2013.

[40] S. F. Daniel, iPad Enterprise Application Development BluePrints., Birmingham: Packt Publishing, 2012.

[41] Apple Inc., „Cocoa Core Competencies: Model-View-Controller," Apple Inc., 17 September 2013. [Online]. Available: https://developer.apple.com/library/mac/documentation/general/conceptual/devpedia-cocoacore/MVC.html. [Zugriff am 21 April 2014].

[42] D. Chisnall, Cocoa programming developer's handbook., Upper Saddle River, NJ [u.a.]: Addison-Wesley, 2010.

[43] S. Faranello, Balsamiq Wireframes Quickstart Guide., Birmingham: Packt Publishing, 2012.

[44] M. E. Jakob Iversen, Learning Mobile App Development - A Hands-on Guide to Building Apps with iOS and Android, Pearson Education, Inc., 2013.

[45] R. Lewis, Y. McCarthy und S. M. Moraco, Beginning iOS storyboarding with Xcode. easily design and develop your app, from concept and vision to code, New York, NY: Apress / Springer, 2012.

[46] C. Chung, Pro Objective-C Design Patterns for iOS., Berkeley, CA: Apress, 2011.

[47] A. Negm-Awad, Objective-C und Cocoa., 3. Aufl. Aufl. Hrsg., Bd. 1, Baar: Smart Books, 2008.

[48] V. Nahavandipoor, iOS 6 programming cookbook., Beijing ;Sebastopol, California: O'Reilly Media, 2012.

[49] J. K. Kyle Richter, iOS Components and Frameworks: Understanding the Advanced Features of iOS SDK, Addison Wesley, 2013.

[50] The RestKit Project, „RestKit," The RestKit Project, [Online]. Available: http://restkit.org/. [Zugriff am 1 Mai 2014].

[51] A. Mendoza, Mobile user experience. patterns to make sense of it all, Amsterdam, [Netherlands]: Morgan Kaufmann, an imprint of Elsevier, [2014].

[52] E. N. McKay, UI is Communication. How to Design Intuitive, User Centered Interfaces by Focusing on Effective Communication, Burlington: Elsevier Science, 2013.

[53] P. Alessi, Professional iOS Database Application Programming., 2nd ed Hrsg., New York: Wiley, 2013.

[54] Apple Inc., „iOS Human Interface Guidelines: Designing for iOS 7," Apple Inc., 10 März 2014. [Online]. Available: https://developer.apple.com/library/ios/documentation/userexperience/conceptual/MobileHIG/index.html. [Zugriff am 21 April 2014].

[55] S. Hoober und E. Berkman, Designing mobile interfaces, Beijing [u.a.]: O'Reilly, 2012.

[56] T. Neil, Mobile Design Pattern Gallery. UI Patterns for Smartphone Apps, 2. Aufl Hrsg., Sebastopol, CA: O'Reilly & Associates, 2014.

[57] P. v. d. Put, Professional iOS Programming., 1. Aufl. Hrsg., s.l.]: Wrox, 2013.

[58] P. B.-A. T. N. Jon Manning, Learning Cocoa with Objective-C, 4 Hrsg., United States of America: O'Reilly Media, Inc., 2014.

[59] R. Wentk, Xcode 5 Developer Reference., 1. Aufl. Hrsg., s.l.]: Wiley, 2014.

[60] A. Allan, Learning iOS programming., Third edition Hrsg., Beijing ;Sebastopol, California: O'Reilly Media, 2013.

[61] Apple Inc., „iOS 7 UI Transition Guide: Before You Start," Apple Inc., 22 Oktober 2013. [Online]. Available: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/TransitionGuide/index.html#//apple_ref/doc/uid/TP40013174. [Zugriff am 8 Mai 2014].

[62] Apple Inc., „iOS 7 UI Transition Guide: Appearance and Behavior," Apple Inc., 22 Oktober 2013. [Online]. Available: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/TransitionGuide/AppearanceCustomization.html#//apple_ref/doc/uid/TP40013174-CH15-SW1. [Zugriff am 8 Mai 2014].

[63] M. Neuburg, iOS 7 programming fundamentals. objective-C, Xcode, and Cocoa basics, Sebastopol, California: O'Reilly, 2013.

[64] Apple Inc., „64-Bit Transition Guide for Cocoa Touch: About 64-Bit Cocoa Touch Apps," Apple Inc., 11 Februar 2014. [Online]. Available: https://developer.apple.com/library/ios/documentation/General/Conceptual/CocoaTouch64BitGuide/Introduction/Introduction.html. [Zugriff am 7 Mai 2014].

Abbildungsverzeichnis

Quellcodeverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
ERP	Enterprise Resource Planning
JVx	Java Application Extention
KMU	Klein und Mittelunternehmen
JSON	JavaScript Object Notation
UI	User Interface
MVC	Model-View-Controller
OOP	Objektorientierter Programmierung
ARC	Automatic Reference Counting

Anhang A: Eingabemasken von MyERP in Java

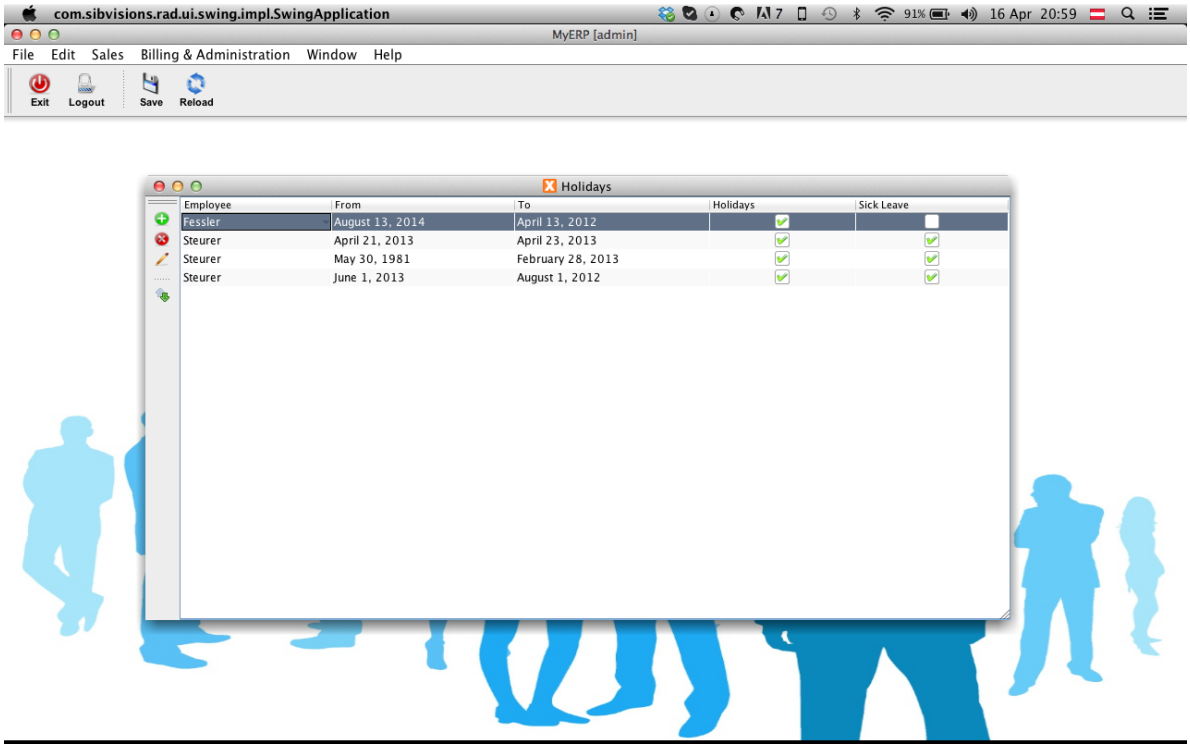


Abbildung 37: MyERP Menu mit "Holidays" Eingabemaske - Desktop

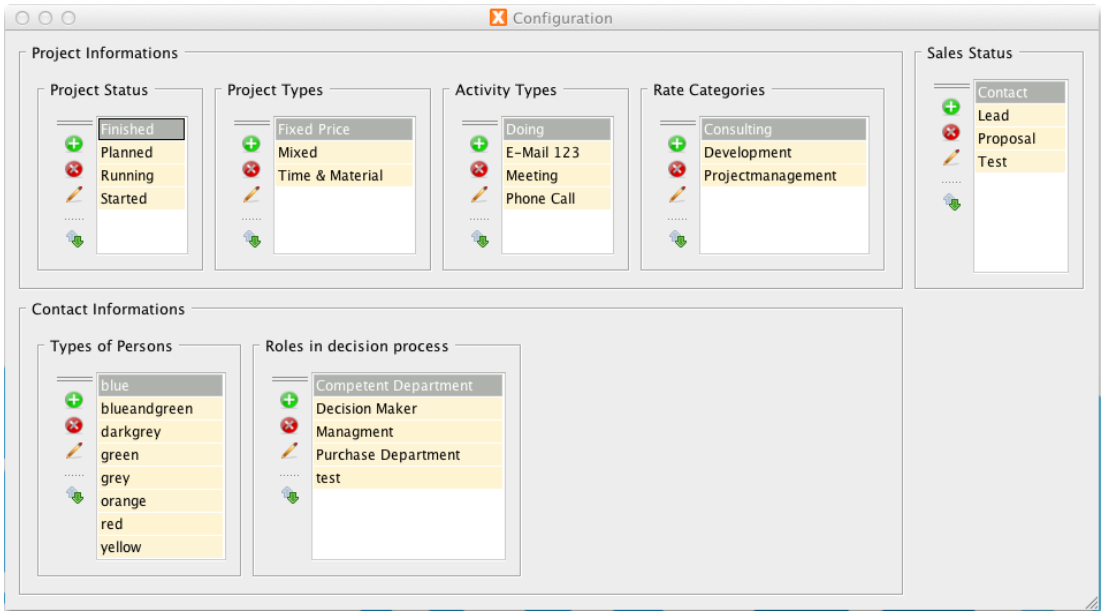


Abbildung 38: MyERP "Configuration" Eingabemaske - Desktop

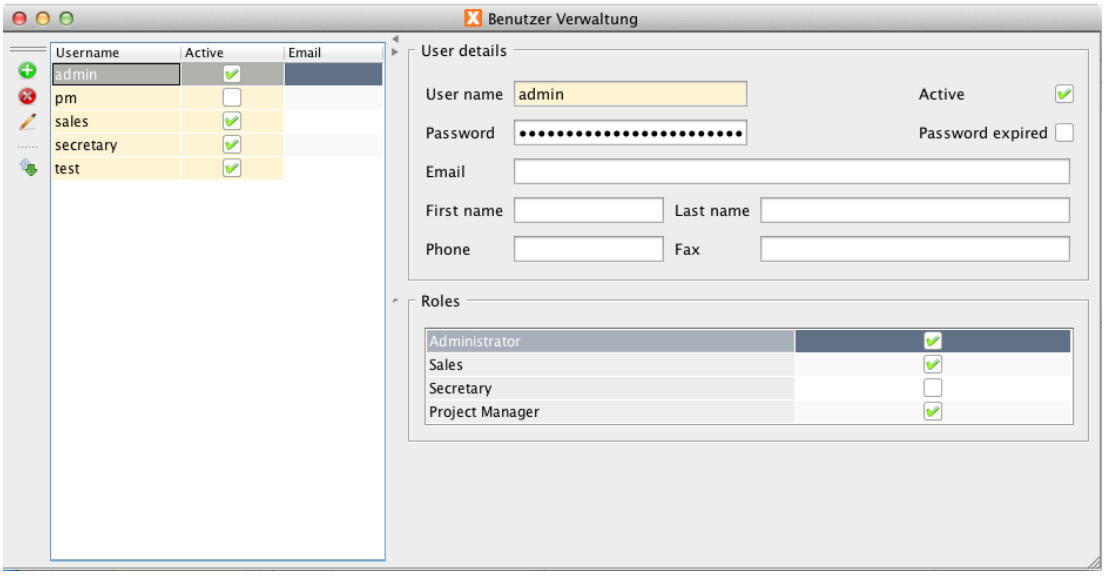


Abbildung 39: MyERP "Benutzerverwaltung" Eingabemaske - Desktop

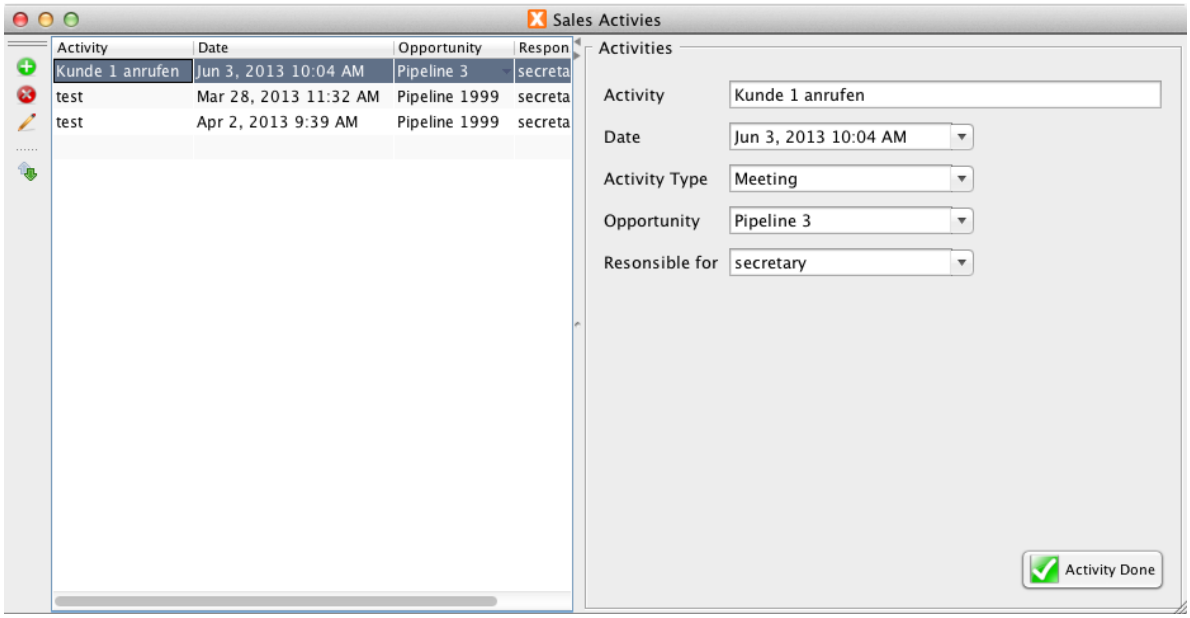


Abbildung 40: MyERP "Sales Activities" Eingabemaske - Desktop

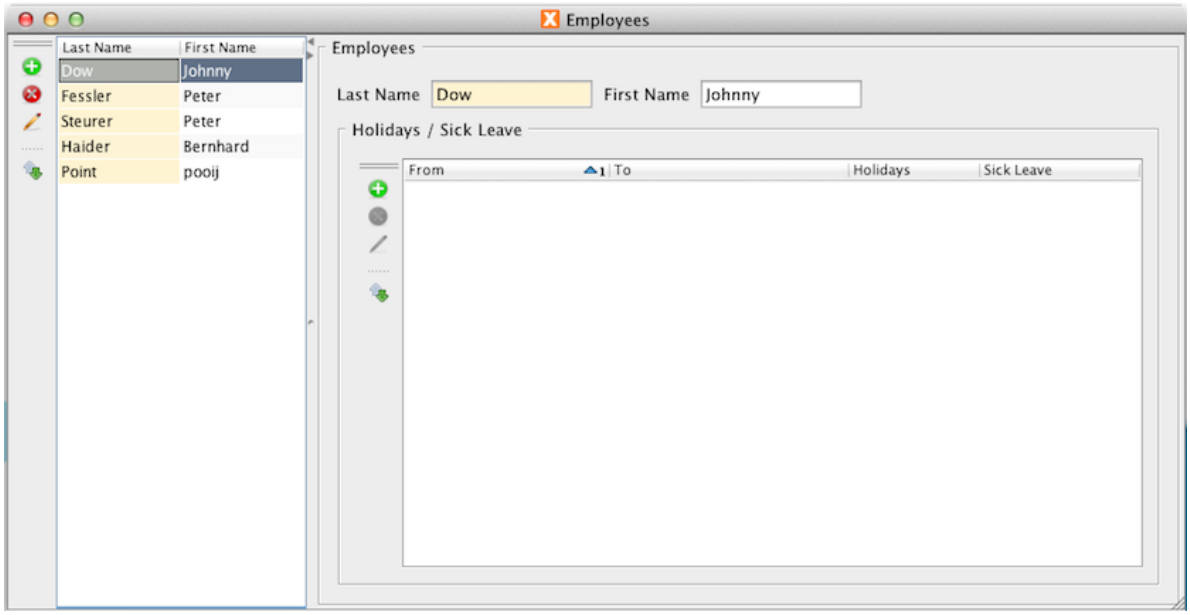


Abbildung 41: MyERP "Employees" Eingabemaske - Desktop

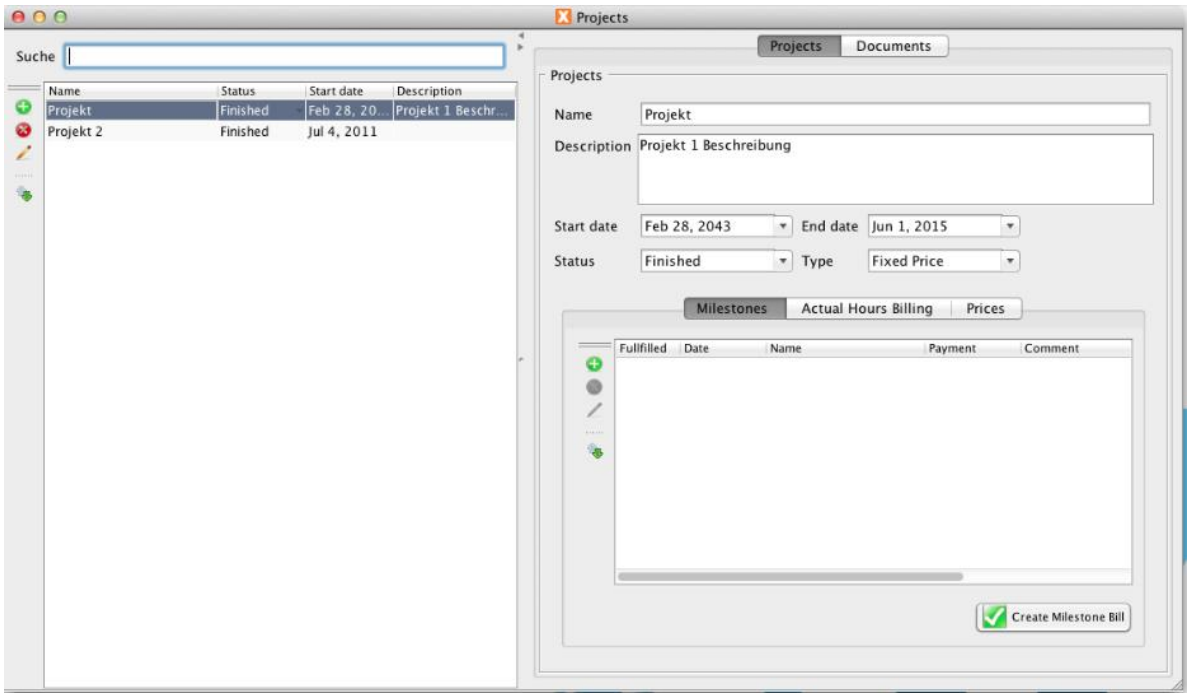


Abbildung 42: MyERP "Projects" Eingabemaske - Desktop

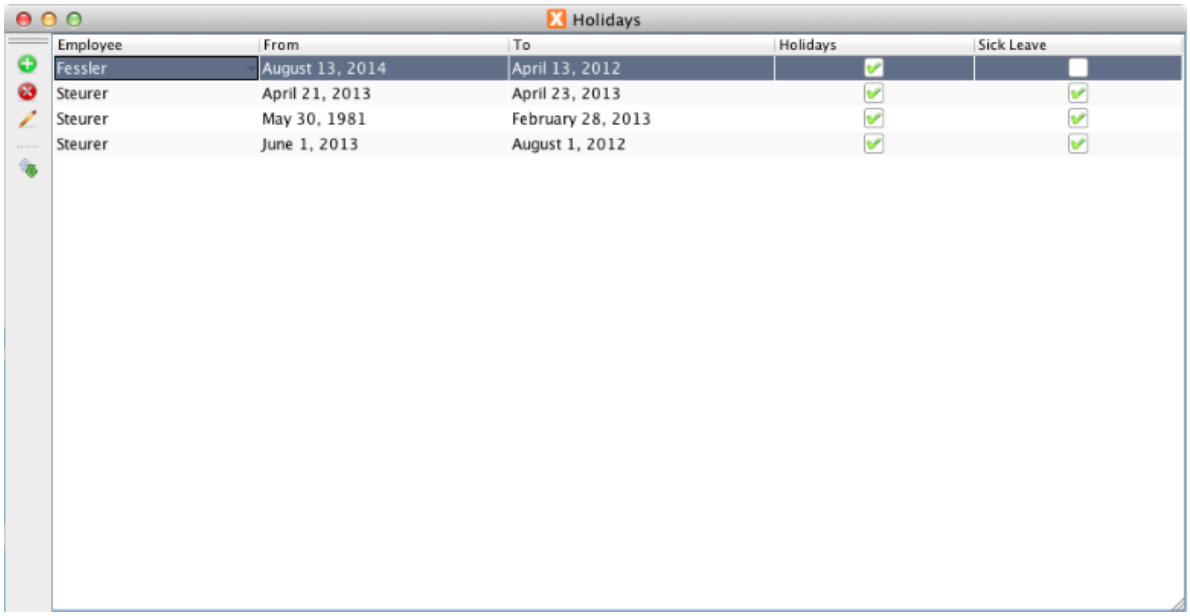


Abbildung 43: MyERP "Holidays" Eingabemaske - Desktop

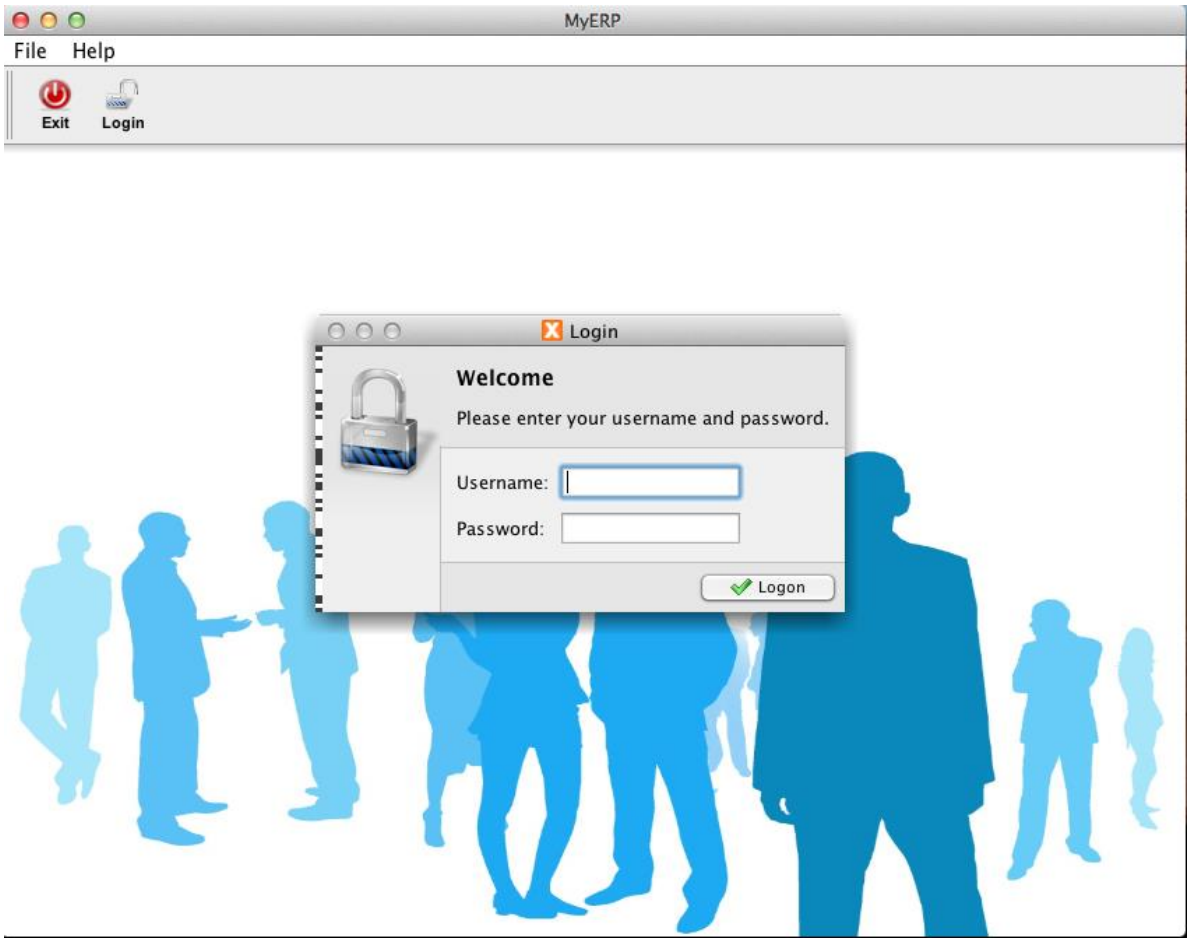


Abbildung 44: MyERP Login – Desktop