



Die „KiBa“-App

Projektbericht

von

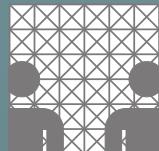
Alexander Droste
Markus Fasselt
Marco F. Jendryczko
Konstantin S. M. Möllers
Michael Schaarschmidt
Julius Wulk

Veranstalter

Dr. Guido Gryczan, Dr. Martin Christof Kindsmüller und Christian Zoller



Universität Hamburg



Fachbereich Informatik

Inhaltsverzeichnis

1	Einleitung und Projektkontext	4
2	App-Idee	5
2.1	Problematik	5
2.2	Vision	6
2.3	Name	7
2.4	Plattform	7
2.5	Funktionen	7
2.5.1	Finder	7
2.5.2	Authentifizierung	7
2.5.3	Self-Service	8
2.5.4	Individueller Finanzierungsrechner	9
3	Kontexterkundung	10
3.1	Fragebogen	10
3.2	Ergebnisse	10
4	User-Interface & User-Experience	13
4.1	Mockups	13
4.2	Prototyp	13
4.3	Logo & Farbharmonie	15
4.4	UI-Elemete	16
4.5	UX Inspektion & Test	16
5	Vorgehen	18
5.1	Orientierungsphase	18
5.2	Umsetzungsphase	20
5.3	Vorgehensweise nach Scrum	22
6	Ergebnis	25
6.1	Features	26
6.1.1	Dashboard	26
6.1.2	Filialfinder	26
6.1.3	Sortenanfrage	26

6.1.4	Authentifizierung	26
6.1.5	Überweisung	27
6.1.6	Self-Service-Station	27
6.1.7	Bescheinigungen	28
6.1.8	Kontoauszüge	28
6.1.9	Umbuchungen	28
6.1.10	Finanzierungsrechner	30
6.1.11	Mein Bereich	30
6.2	Die Design-Highlights	31
6.3	Herausforderungen	32
7	Architektur	34
7.1	Umsetzung des MVC-Ansatzes	34
7.2	Datenmodell	35
7.3	Sicherheitsaspekte	36
7.4	Dependency Injection	37
8	Toolchain	40
8.1	Versionsverwaltung und Informationsaustausch	40
8.2	Issue-Tracking-System	41
8.3	Konzept- und Designwerkzeuge	42
9	Qualitätssicherung	44
9.1	Entwicklungsprozess	44
9.2	Qualität des Designs	44
9.3	Umgang mit Feedback	45
9.4	Ausblick	45
10	Fazit	47
Abkürzungsverzeichnis		49
Programmausdrucke		49

1 Einleitung und Projektkontext von Michael Schaarschmidt

Dieser Bericht ist eine Zusammenfassung unserer Ergebnisse im "Projekt Objektorientierte Softwareentwicklung", bei dem über ein Semester hinweg in Kooperation mit der T-Systems MMS eine iOS-Anwendung entwickelt wurde. Die Herausforderung war dabei, universitäre Erfordernisse mit kommerziellen Anforderungen in Einklang zu bringen. Ziel war es, anhand einer gegebenen Aufgabenstellung in Gruppen mit universitär heterogenem Hintergrund einen Prototypen zu entwickeln.

Dieser sollte in Architektur und Gestaltung das Verständnis softwaretechnischer und interaktiver Konzepte deutlich machen, gleichzeitig aber geeignet sein, in einer konkreten Produktdemonstration im Projektabschluss praktischen Nutzen unter Beweis zu stellen. Zusätzlich wurden wir im Rahmen einer in das Projekt integrierten, einführenden Zertifizierung mit der Scrum-Methodik vertraut gemacht, welche dann als Grundlage für die Projektarbeit diente.

Die unser Gruppe übertragene Aufgabe bestand darin, eine App zu entwickeln, welche die Bindung zwischen einer fiktiven Filialbank und ihren Kunden erhöht. In den ersten Überlegungen wurde deutlich, dass nahezu jede Dienstleistung einer Filiale auf die eine oder andere Weise von Direktbanken abgebildet werden kann. Nicht umsonst ist das Filialgeschäft im Abschwung¹. Insofern bestand unsere besondere Herausforderung darin, einen Ausweg aus dieser strukturell schwierigen Lage zu finden.

Im Bericht sollen nun zunächst in Kapitel 2 und 3 die Ideen und Diskussionen während der Konzeptphase und der Kontexterkundung deutlich gemacht werden. Mit Blick auf das Design werden dann in Abschnitt 4 unsere Mockups und UI-Elemente erläutert. In Teil 5 wird der Projektzeitraum noch einmal mit Blick auf die Vorgehensweisen, insbesondere Scrum, analysiert. Schließlich wird in Teil 6 und 7 ausführlich das Ergebnis in Aussehen und Funktion sowie auch im Bezug auf die Softwarearchitektur vorgestellt. Abschnitt 8 bespricht die verwendeten Software-Tools, also etwa die Versionsverwaltung und Designwerkzeuge. Der Bericht schließt mit einer Übersicht über die von uns eingesetzten Maßnahmen zur Qualitätssicherung in Kapitel 9 und einem kurzen Gesamtfazit in Kapitel 10.

¹<http://www.welt.de/print/wams/wirtschaft/article124228415/Das-ist-ein-Sterben-auf-Raten.html>,

Abruf am 23.02.2014 um 16:00 Uhr.

2 App-Idee von Michael Schaarschmidt

2.1 Problematik

Mit Rücksicht auf oben genannte Aspekte haben wir uns in der Ideenfindungsphase auf Lokalität als entscheidendes Schlagwort konzentriert. Apps wie etwa Yelp², welche Bezug auf lokale Unternehmen nehmen und Benutzern die Möglichkeit geben, sich darüber auszutauschen, sind beispielhaft für eine erfolgreiche Umsetzung dieses Konzepts.

Für das Bankgeschäft ergab sich jedoch das Problem, dass die eigenen finanziellen Verhältnisse in den innersten persönlichen Lebensbereich fallen und in der Regel darüber zumindest weniger Austausch stattfindet als über Restaurants, Nachtclubs oder Freizeitaktivitäten.

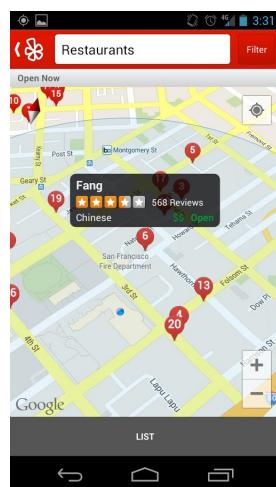


Abbildung 1: Screenshot der Yelp-App

Erste Überlegungen zielten in Richtung interaktiver Beratungsdienstleistungen, welche jederzeit über Videotelefonie Kontakt zum individuell vertrauten Berater einer bestimmten Filiale herstellen sollten. Der individuelle Berater ist aber kein Alleinstellungsmerkmal der Filialbank mehr, eine virtuelle Filiale ist ohne weiteres denkbar.

Allerdings, und hier haben wir die Chance unserer Anwendung gesehen, bedeutet die technische Möglichkeit eines Vertriebsweges nicht gleich, dass dieser auch angenommen wird. Eine von uns im Sinne einer Markterkundung durchgeführte Umfrage ergab ein gewisses Misstrauen

²<http://www.yelp.de/>, Abruf am 23.02.2014 um 16:15 Uhr, siehe auch Abbildung 1

gegenüber der Idee, Finanztransaktionen per App durchzuführen oder überhaupt eine Bank-App zu benutzen.

Gründe hierfür könnten vielfältig sein; beispielsweise könnten die befragten Studenten am Informatikcampus eine besondere Sensibilität bezüglich Aspekten des Datenschutzes haben. Eine von der Unternehmensberatung „Bain and Company“ durchgeführte Studie zur Frage, wie digitalisierte Vertriebswege das Privatkundengeschäft der Banken zukünftig prägen könnten, ergab jedoch einen ähnlichen Tenor³. Insbesondere ergab die Studie, dass für Geldanlagen dem persönlichen Gespräch der Vorzug vor digitaler Beratung gegeben würde. Somit könnte es trotz aller technischen Möglichkeiten auch langfristig noch einen Markt für persönliche Beratung von Angesicht zu Angesicht geben.

2.2 Vision

Schließlich entstand die Vision einer Anwendung, die nicht versucht, aus einer Filialbank eine Direktbank mit Filialen als Zusatzangebot zu machen, sondern die die bestehenden Produkte des Filialgeschäfts dem Kunden näher bringt und ihm Grund gibt, in eine Filiale zu gehen.

Als Kernprodukte und Kompetenzen haben wir das klassische Sparbuch, die Anlageberatung sowie die Finanzierung ausgemacht. Das Sparbuch bietet scheinbar keine Funktionalität, die einen konkreten Vorteil gegenüber beispielsweise dem Tagesgeldkonto einer Direktbank hätte. Wie aber einführend erläutert, ist dies nicht ausschlaggebend. Denn ungeachtet seiner Nachteile ist das Sparbuch populär; vielleicht auch, weil in den Verwerfungen der europäischen Finanzkrise hier besondere Sicherheit vermutet wird. Diesem Sicherheitsbedürfnis der Kunden wollten wir dabei konsequent nicht nur Architektur, sondern auch mit Rücksicht auf die vom Kunden wahrgenommene Sicherheit gerecht werden – etwa durch visuelle Elemente, die mit Sicherheit assoziiert werden.

In den nächsten Abschnitten wird nun erläutert werden, welche Funktionen wir uns in Konsequenz überlegt haben und mit welchen Use-Cases diese korrespondieren.

³http://www.bain.de/Images/Retail_Banking_II_Digitalisierung_ES.pdf, Abruf am 23.02.2014 um 16:30 Uhr.

2.3 Name

Der Name der App, „KiBa“, steht – in Anlehnung an eine bekannte Direktbank – für eine kundeninteressierte Bank, die mit der Anwendung mehr über ihre Kunden erfahren möchte und mit ihnen in Austausch treten will, um spezifischere Angebote und Beratungen anbieten zu können.

2.4 Plattform

Für die Wahl einer geeigneten Plattform – iPad oder iPhone – haben wir uns zunächst überlegt, in welchen Situationen wir die Interaktion mit einer Banking-App für wahrscheinlich hielten. Es erschien uns, als wäre der typische Anwendungsfall für Bankgeschäfte eher stationär, etwa auf dem Sofa oder im Büro, jedenfalls aber in Ruhe.

Somit überwogen in der Gruppe eindeutig die Argumente für eine iPad-Anwendung. Zudem war es Aufgabe, eine Vision für die Banking-App der Zukunft zu entwickeln. Der Trend geht unserer Meinung nach zum ubiquitären WLAN. Insofern darf davon ausgegangen werden, dass die mobile Konnektivität zukünftig auch beim iPad unproblematisch ist.

2.5 Funktionen

2.5.1 Finder

Ein intuitiv klarer Anwendungsfall ist der Wunsch eines Nutzers, die nächstliegende Filiale aus der App heraus auf einer Karte zu finden und dorthin navigieren zu können. Um diese Funktionalität noch gegenüber einem bekannten Kartendienst abzuheben, soll schon an dieser Stelle zusätzliche Interaktion mit einer Filiale möglich sein. Über eine Sortenanfrage können Devisen bestellt und aus der Filialseite, welche aus der Karte geöffnet wird, kann ein Termin vereinbart werden.

2.5.2 Authentifizierung

Der Authentifizierungsmechanismus trennt die Funktionen in sicherheitsrelevante und unkritische. Ohne Authentifizierung steht dem Benutzer nur passive Funktionalität zur Verfügung, also

etwa der Filialfinder oder die Umsatzanzeige. Um vom vollen Funktionsumfang der App profitieren zu können, muss der Benutzer in eine KiBa-Filiale gehen und von einem Berater einen Sicherheitscode eingeben lassen. Dieser garantiert dann in Kombination mit der App-ID der Anwendung eine eindeutige Zuordnung eines Benutzers an sein Gerät. Ähnlich einer Kreditkarte soll dann bei Verlust des Geräts auch die App selbst jederzeit gesperrt werden können. Insbesondere dient die Aktivierung aber auch dazu, den Kunden in einem Beratungsgespräch besser kennenzulernen und in einer Datenbank einen persönlichen Ansprechpartner festzuhalten.

2.5.3 Self-Service

Im Zuge der Plenumsdiskussionen und anschließenden internen Debatten haben wir den zeitlichen Aspekt als zentrale Komponente identifiziert. Lokalität bedeutet, Dinge direkt zur Verfügung gestellt bekommen zu können. Viele Bescheinigungen und Unterlagen im täglichen Leben werden auch heute noch konkret ausgedruckt benötigt. Der typische Ablauf einer Direktbank sieht so aus, eine Anfrage – etwa nach Wertpapierzweitschriften oder Bonität – per Kontaktformular abzuschicken und dann einige Tage auf die entsprechenden Ausdrucke zu warten. Unsere Idee besteht in einer Self-Service-Station innerhalb der Bank, die das Konzept bestehender Automaten erweitert.

Ein Kunde kann sein iPad auf eine Ablage legen und sich mit der Station verbinden, welche eine Art Multifunktionsdrucker beinhaltet, vergleiche Abbildung 2. Über die App können dann verschiedenste Bescheinigungen ausgedruckt und Konten eröffnet werden. Das entscheidende dabei ist, dass der Kunde durch die Authentifizierung in einer Filiale seinem Gerät zugeordnet wurde. Somit können an derartigen Stationen auch Interaktionen vorgenommen werden, die normalerweise die Identifikation per Lichtbildausweis am Schalter erfordern würden. Somit entsteht hier Mehrwert für alle Stakeholder: Kunden müssen weniger anstehen für standardisierte Abläufe, die Bank spart unter Umständen Personalkosten und kann Mitarbeiter für das Wesentliche, die Beratung, einsetzen.

Umbuchungen am Sparbuch können üblicherweise nur in einer Filiale vorgenommen werden, überwiesen werden kann nur auf das Sparkonto. Die Greifbarkeit des Sparbuchs vermittelt konservativen, besorgten Sparern ein Gefühl von Sicherheit. Gleichzeitig kann es aber durch diese funktionale Einschränkung auch zu unerwünschten Situationen kommen: ist etwa durch

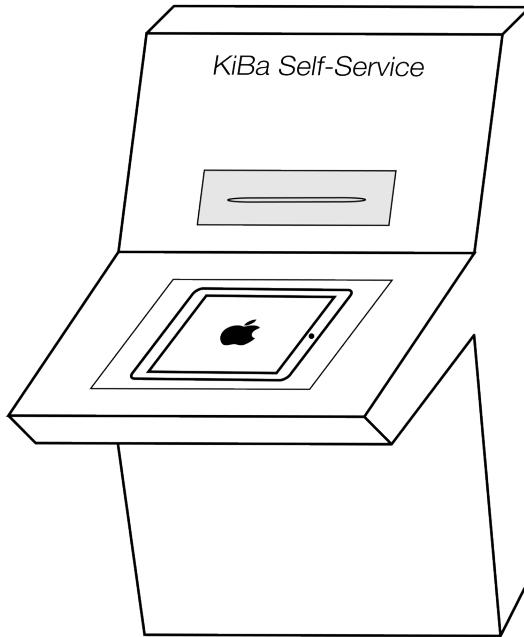


Abbildung 2: Piktogramm unserer Self-Service-Station

eine Fehlkalkulation an einem Samstagabend kein Geld mehr auf dem Girokonto, muss bis zur Öffnung einer Filiale am Montag gewartet werden. Um dem vorzubeugen, soll über die Self-Service-Station auch Geld umgebucht werden können. Da die Station im Vorraum der Filiale steht, ist sie immer zugänglich.

2.5.4 Individueller Finanzierungsrechner

Wie oben erläutert, besteht eine wesentliche Dienstleistung von Filialbanken in der Finanzierung, etwa für Eigenheime. Im Zentrum unserer Überlegungen stand dann auch die Frage, wie eine App dabei helfen kann, diese Dienstleistung für den Endkunden zu verbessern.

Im Ergebnis möchten wir einen individualisierten Kreditrechner anbieten, der den App-Benutzer in die Lage versetzt, mittels individuell berechneter Profildaten für sich selbst Finanzierungsrechnungen durchzuführen. Die Idee dahinter ist, dass ein Kunde zunächst für sich selbst einige Finanzierungsvarianten durchspielen kann. Hat er sich für eine Variante entschieden, kann ein Termin mit dem persönlichen Berater vereinbart werden. Wichtig dabei ist, dass die angebotenen Finanzierungsdaten mit Rücksicht auf die der Bank zur Verfügung stehenden Informationen so konservativ gewählt sind, dass sie in jedem Falle von der Bank eingehalten werden können.

3 Kontexterkundung von Julius Wulk

Im Rahmen der oben beschriebenen Überlegungen wurde bereits angesprochen, dass wir eine Markterkundung durchgeführt haben. Deren Umsetzung soll hier noch einmal im Detail erläutert werden, um daraus Spezifizierungen für die Mockups zu identifizieren.

3.1 Fragebogen

Bei der Datenerhebung haben wir uns, wie in Abbildung 3 zu sehen, für eine Umfrage mittels Fragebogen entschieden, da die App in vielen verschiedenen Kontexten eingesetzt werden soll und wir möglichst viele Meinungen einfließen lassen wollten.

Für unser Konzept wollten wir wissen, welche Funktionen besonders beliebt sind und welche Bedenken die Teilnehmer bei der Benutzung von Banking-Apps haben. Außerdem war es uns ein Anliegen, nicht nur quantitative, sondern auch qualitative Daten zu erheben. Daher gab es Freitextfelder, in denen die Teilnehmer Ideen, Kritik und allgemeine Bedenken in Bezug auf Banking-Apps äußern konnten. Dieses half uns, die potentiellen Nutzer besser zu verstehen und Dinge aufzudecken, an die wir nicht gedacht hätten. An der Umfrage am Informatikum haben 92 Studenten teilgenommen. Hier folgen nun die wichtigsten Ergebnisse.

3.2 Ergebnisse

Die drei nützlichsten Features sind der Bankautomatenfinder, Filialenfinder und die Kartensperre. Der Bankautmatenfinder wurde von 93% der Befragten als nützlich bewertet. Das sehr ähnliche Feature Filialenfinder fanden 81% nützlich und die Kartensperre wurde von 61% als nützlich angesehen. Diese Ergebnisse zeigen, dass wir die Kernfeatures einer Banking-App nicht aus den Augen verlieren dürfen und dass ein optimierter Finder Potential hat.

48% der befragten Studenten hielten die Features „Depot-Verwaltung für Aktien“ und „Personalisierte Angebote der Bank“ für nicht nützlich. Eine mögliche Erklärung dafür ist, dass die Studenten bei diesen Themen einfach noch keine eigene Relevanz sehen.

Ein überraschendes Ergebnis der Umfrage war, dass nur 17% der befragten Studenten eine

Umfragebogen zu einer Banking-App					
1. Welchen Studiengang belegst du?					
Informatik <input type="checkbox"/> Wirtschaftsinformatik <input type="checkbox"/> Mensch-Computer-Interaktion <input type="checkbox"/> Andere, und zwar: _____					
2. Findest du folgende Funktionen für eine Banking-App nützlich?					
	sehr nützlich	nicht nützlich			
A	<input type="checkbox"/>				
B	<input type="checkbox"/>				
C	<input type="checkbox"/>				
D	<input type="checkbox"/>				
E	<input type="checkbox"/>				
F	<input type="checkbox"/>				
G	<input type="checkbox"/>				
H	<input type="checkbox"/>				
I	<input type="checkbox"/>				
J	<input type="checkbox"/>				
K	<input type="checkbox"/>				
L	<input type="checkbox"/>				
M	<input type="checkbox"/>				
3. Welche weiteren Features findest du nützlich? (Bitte angeben)					
_____ _____ _____					
4. Wie oft kontrollierst du deinen Kontostand?					
	stetig	_____			
	2-3 Mal die Woche	<input type="checkbox"/>			
	1 Mal die Woche	<input type="checkbox"/>			
	alle 2 Wochen	<input type="checkbox"/>			
	sel tener	<input type="checkbox"/>			
5. Benutzt du derzeit eine Banking-App ?					
Ja <input type="checkbox"/> Nein, und zwar aus folgendem Grund: _____					
5A. Wenn ja, was gefällt dir an der App?					
_____ _____					
5B. ... und was gefällt dir nicht?					
_____ _____					
6. Hast du Bedenken bezüglich folgender Punkte bei der Bedienung einer mobilen App?					
	keine Bedenken	größere Bedenken			
A	<input type="checkbox"/>				
B	<input type="checkbox"/>				
C	<input type="checkbox"/>				
D	<input type="checkbox"/>				
E	<input type="checkbox"/>				
F	<input type="checkbox"/>				
G	<input type="checkbox"/>				
6H. Sichtest du etwas besonders kritisch?					
_____ _____					
7. Das wollte ich noch hervorheben ...					
_____ _____					

Herzlichen Dank für deine Unterstützung!

Abbildung 3: Endfassung des Umfragebogens

Banking App nutzen. Für uns ist es natürlich interessant, aus welchen Gründen Banking-Apps gemieden werden. Studenten gaben an, dass „Apps zu unsicher“ wären, „Apple, Microsoft und Co. Zugriff auf meine Daten“ nähmen, „Unsicherheit der Datenübertragung“, „Bankgeschäfte sollte man in Ruhe bearbeiten und nicht hektisch to go“.

In allen abgefragten Kategorien hatte mindestens über die Hälfte der Studenten Bedenken. Fast 80% der befragten Studenten gaben an, Angst vor Missbrauch der Daten zu haben. Das zeigt, dass das Thema Sicherheit der eigenen Daten beim Banking eine Schlüsselrolle spielt und wir beim Konzept diesem Aspekt eine besondere Bedeutung zukommen lassen sollten.

Abschließend wollen wir die Ergebnisse in Form eines Diagramms (Abbildung 4) präsentieren.

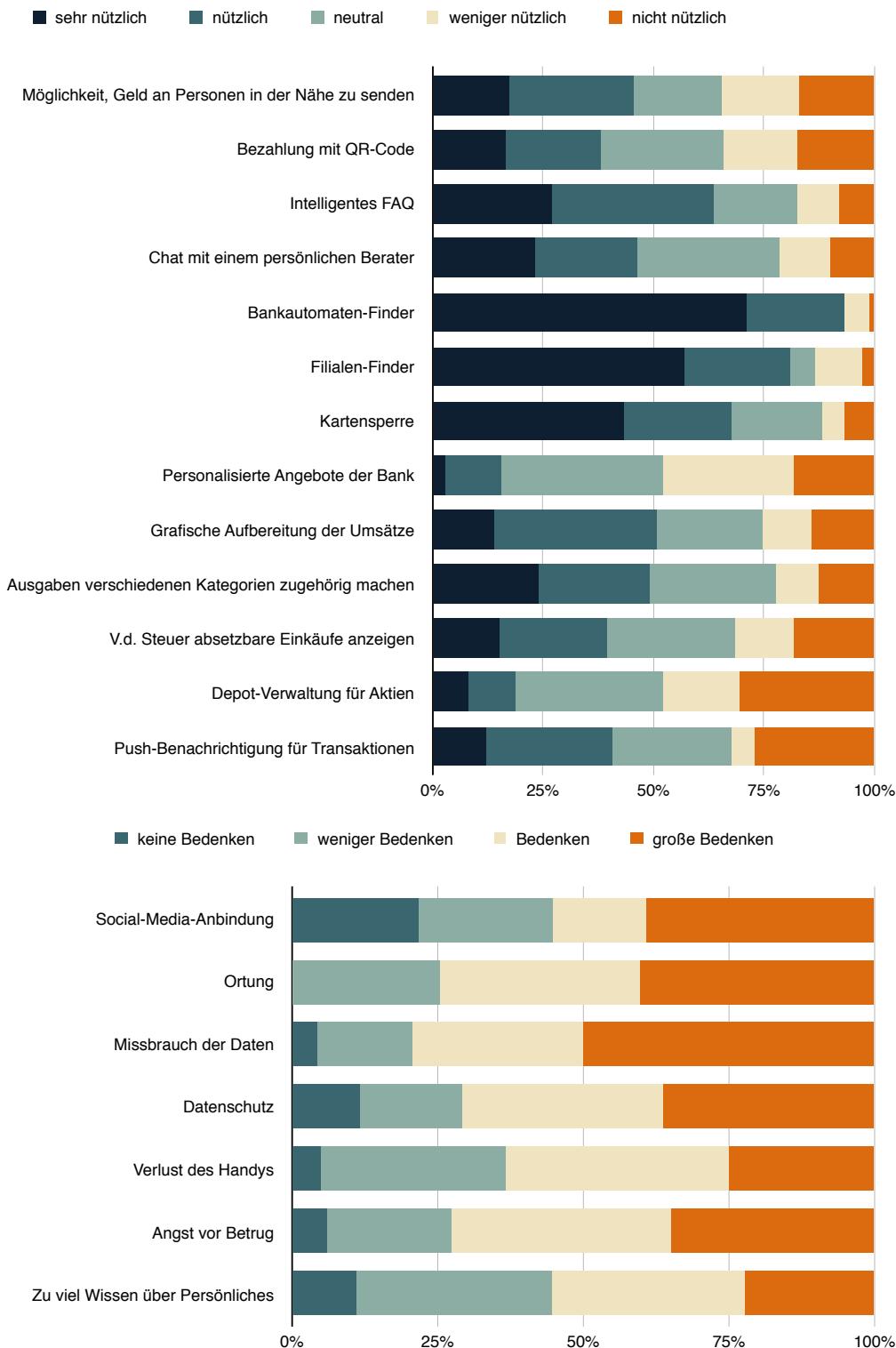


Abbildung 4: Ergebnisse der Kontexterkundung

4 User-Interface & User-Experience von Julius Wulk

4.1 Mockups

Vor Beginn der Gestaltung der Mockups haben wir uns auf dem Markt der Banking-Apps umgeschaut, um die verschiedenen UI-Patterns, die in diesem Zusammenhang genutzt werden, zu vergleichen. Mit Balsamiq entwarfen wir anschließend erste Mockups. Bei der Anordnung der Elemente haben wir uns die Prinzipien der Gestaltungsgesetze zunutze gemacht. Insbesondere halfen uns das Gesetz der Ähnlichkeit und das Gesetz der Nähe, Ordnung in die UI zu bringen.

Unser Ziel war es, ein möglichst gebrauchstaugliches Design zu erreichen. Dazu haben wir uns an den Grundsätzen der Dialoggestaltung nach der DIN EN ISO 9241 110 orientiert. Bei der Gestaltung haben wir nur eine Schriftart verwendet und uns nach festgelegten Farbsemantiken gerichtet. Dadurch geht der Fokus auf den Inhalt nicht verloren und der Nutzer kann die App erwartungskonform nutzen. Bei der Navigation haben wir uns für eine Master-Detail-View entschieden. Dadurch haben wir eine flache Navigationshierarchie, die es uns ermöglicht, alle wichtigen Funktionen von der Hauptseite direkt zu erreichen. Auf diese Weise haben wir unnötige Interaktionen minimiert und der Nutzer kann effizient und aufgabenangemessen mit der App interagieren.

Auf jeder Detail-View befindet sich unten rechts ein Informationbutton. Mit diesem Button wollen wir die Lernförderlichkeit der App verbessern und die Nutzer bei Problemen unterstützen. Allerdings haben wir diese Funktion nur rudimentär umgesetzt.

Die Elemente in der Detail-View sind asymmetrisch angeordnet mit einer Gewichtung oben links und unten rechts. So erreichen wir eine diagonale Balance, die optimal für den visuellen Fluss ist.

4.2 Prototyp

Allgemein haben wir uns beim Design des User-Interface an den Guidelines von IOS7 orientiert. In diesem Zusammenhang haben wir uns mit den wichtigsten Grundsätzen laut Apple „Defence“, „Clarity“ und „Depth“ beschäftigt, um diese auf unser User-Interface zu übertragen.



Abbildung 5: Mockup des Dashboards, der Filial-Seite und der Überweisung

„Deference“ haben wir bewirkt, indem wir den Fokus auf die Wahrnehmung von Inhalten gelegt haben und versucht haben, UI-Elemente, die in Konflikt mit dem Inhalt stehen, wegzulassen. Uns war es wichtig, das User-Interface möglichst funktional zu gestalten, was auch im Sinne von „Clarity“ ist. Daher haben wir eine möglichst selbsterklärende Ikonographie gewählt und auf eine unnötige Farbenvielfalt verzichtet. Den Guidelines entsprechend haben wir die Icons mindestens in der Größe 44×44 pts dimensioniert. Andernfalls wird es schwierig, ein Icon mit dem Finger zu treffen. Außerdem gibt es zu jedem Icon 2 Versionen, damit auf einem Retina iPad die hohe Auflösung ausgenutzt werden kann. Mit iOS7 setzt Apple auf Flat-Design. Ganz im Sinne des Flat-Designs haben wir mit kräftigen Kontrasten statt mit Schlagschatten oder Gradienten gearbeitet. Darüber hinaus wurde eine dünne, seriflose Schriftart gewählt.

Aufgrund von leichten Konzeptänderungen während des Projekts haben wir manche Details im Prototypen anders als in den Mockups gestaltet.

Beispielsweise gab uns Markus Foos den Tipp, dass Nutzer mehr interessiert sind an den Umsätzen als an dem aktuellen Kontostand. Das ist besonders der Fall bei denjenigen Leuten, deren Kontostand niedrig ist oder die sogar im Minus liegen.



Abbildung 6: App-Icon mit Farbskala

4.3 Logo & Farbharmonie

Unsere ausgewählte Farbharmonie spiegelt die Werte einer modernen Bank wider. Mit den Farben wollen wir Sicherheit, Nachhaltigkeit und Seriosität vermitteln. Allerdings waren wir bei der Logogestaltung in dieser Hinsicht nicht ganz konsequent. Wir hatten an dieser Stelle die Diskussion, ob wir uns nicht für ein anderes Logo entscheiden sollten. Mit dem KiBa-Glas im Logo haben wir uns dann darauf geeinigt, als Studenten ein wenig Humor zuzulassen. Im Zusammenspiel mit dem recht nüchternen Design wirkt das Logo auflockernd.

4.4 UI-Elemente

Grundsätzlich haben wir mit den Standard-View-Elementen von iOS7 gearbeitet. Allerdings haben wir in manchen Fällen auf Custom-Elemente zurückgegriffen und diese dann möglichst konsequent eingesetzt. Dabei haben wir darauf geachtet, dass die Eigenlösungen zu dem iOS7 Design passen. Bei dem Standard-Text-Field störte uns, dass der Placeholder-Text bei der Eingabe verschwand und so die Eingabe gelöscht werden musste, um diesen wieder zu sehen. Man stelle sich vor, man möchte ein Überweisungsformular ausfüllen und während der Eingabe eines Feldes ist man sich nicht mehr sicher, wofür dieses steht. Daher entschieden wir uns für eine Eigenlösung, die den Vorteil hat, dass der Placeholder-Text bei erfolgter Eingabe nach oben rückt und auf diese Weise sichtbar bleibt.(JVFloatLabeledTextField von Jared Verdi) Der Verbindungsauflauf mit der KiBa-Station und die Authentifikation sind wichtige Elemente in unserem Konzept. Besonders an dieser Stelle war es uns wichtig Sicherheit bei der Datenübertragung zu vermitteln. Mit dem flexiblen und individuell gestaltbaren SVProgressHUD von Sam Vermette konnten wir nach unseren Vorstellungen diese Vorgänge abbilden. Bei der Standard-Alert-View hatten wir das Problem, dass wir die Button-Farbe nicht anpassen konnten. Allerdings hatten wir bereits eine bestimmte Farbe für Buttons festgelegt. Um Erwartungskonformität zu gewährleisten, griffen wir auf das CustomIOS7AlertView von Richard Dancsi zurück. Mit dieser Alert-View konnten wir u.a. die Button-Farbe auf unser Design anpassen. Auf der Überweisungsseite hatte das Standard-Element des Datepickers zu viel Gewicht in der UI bekommen. Um dieser Unausgeglichenheit entgegenzuwirken haben wir den Picker durch den FlatDatePicker von Steven Shen ersetzt. Dieser Picker hat den Vorteil, dass er im Design und Gebrauch dem Standard-Datepicker sehr ähnlich ist, dafür aber weniger Raum in der UI einnimmt.

4.5 UX Inspektion & Test

Wöchentlich hatten wir im Plenum die Möglichkeit, unseren Stand der App vorzustellen. Hier konnten in einer anschließenden Diskussion Unklarheiten im Konzept und Design angesprochen werden. Da sich im Plenum unter anderem Dr. Kindsmüller befand, konnten wir von dem Wissen eines erfahrenen UX-Experten profitieren. Beispielsweise deckte er auf, dass wir eine Geste nicht erwartungskonform verwendeten. Bei der Scheck-Überweisung wollten wir anfangs

mit der Wischgeste nach unten die Überweisung abschicken. Allerdings wird diese Wischgeste normalerweise als „Pull-to-refresh“-Geste verwendet. Daher war hier die Affordanz (Angebotscharakter) nicht gegeben. Um Verwirrung beim potentiellen Nutzer zu vermeiden, haben wir die Geste durch einen einfachen Button zum Abschicken der Überweisung ersetzt und die Animation entsprechend angepasst.

Zwischendurch haben wir auch Kommilitonen die KiBa-App testen lassen. Wir stellten bei diesen kleinen Usability-Tests fest, dass das Konzept und der Sinn hinter der Authentifizierung nicht ganz verständlich war. Daher entschieden wir uns einen Comic zu erstellen, der die Kernschritte und Vorteile explizit erklärt. Auch ging aus der App nicht hervor, bei welchen Funktionen die Authentifizierung eine Rolle spielt. Diesen Mangel behoben wir, indem wir die Funktionen, welche erst nach der Authentifizierung verwendbar sind, mit einem Schloss-Icon markierten.

Im Schlusssprint hatten wir die Möglichkeit, mit Herrn Kindsmüller jede View zu inspizieren, um auch die kleinen Interaktionsmängel aufzudecken. Beim Comic, der das Authentifizierungsverfahren erklärt, war beispielsweise der „Vorteil genießen“-Button für Nutzer nicht auffordernd genug. Dieses Problem der Salienz lösten wir durch eine leichte Animation. Des Weiteren wurde bei dem Kreditrechner nicht klar, dass individuelle Konditionen beim Berechnen eine Rolle spielen. Hier wurde uns vorgeschlagen, persönliche Assoziationen zu verwenden, wie „Ihr Kreditrechner“ oder „Johns Kreditrechner“.

All diese Erfahrungen zeigten uns, wie wichtig es ist, Kritik von außen einzuholen. Da wir in der Gruppe viel Vorwissen hatten, erschienen uns Dinge eindeutig, die für spätere Nutzer nicht unbedingt eindeutig sind. Sicherlich wäre besser gewesen, früher mit dem Testen anzufangen.

Dadurch das wir mit High-Fidelity-Prototypen getestet haben, gab es hauptsächlich Feedback zu kleineren Details und das gesamte Konzept wurde nicht in Frage gestellt. Trotzdem haben wir viel hilfreiche Kritik bekommen und konnten auf diese Weise mögliche Abbruchstellen in der UI beseitigen.

5 Vorgehen von Alexander Droste

5.1 Orientierungsphase

Zu Beginn des Vorgehens standen Ideenfindung, Konzeption, Lernprozesse und das Finden eines gemeinsamen Arbeitsrhythmus im Vordergrund. Eine erste grobe Orientierung, welche Aufgabenteile die einzelnen Beteiligten der Gruppe im Verlauf annehmen würden, ergab sich nach der „Kick-Off“-Veranstaltung bei T-Systems. Die Kompetenzen des Teams wurden anschließend auf die Positionen des Entwicklers, Konzepters, Projektleiters und Beraters aufgeteilt.

Zum Zeitpunkt der Ideenfindung wurde das Vorgehen noch weniger zentral organisiert, als es später durch den Projektleiter realisiert werden sollte. Es galt zunächst herauszufinden, mit welchen konkreten Aufgaben man sich im weiteren Verlauf konfrontieren konnte. Um eine grundlegende Kommunikation zwischen den Teammitgliedern zu etablieren, wurde eine Facebook-Gruppe gegründet, die dazu diente Ideen festzuhalten, zu besprechen und Termine oder Aufgaben für kommende Treffen zu vereinbaren. Um die Ideen darüber hinaus geordnet aufzuschreiben und für alle Beteiligten abrufbar zu machen, wurde zusammen mit dem Coderepository ein Wiki bei GitHub angelegt, siehe Abbildung 7. Das Wiki fand im Speziellen in dieser frühen Phase Verwendung.

Neben der Formfindung von Konzepten diente das Wiki ebenfalls zum Festhalten von Code-Style Konventionen (NYTimes Objective-C-Style Guide) oder Workflows, wie etwa dem toolunterstützten Erstellen von Doxygen-kompatiblen Methodenkommentaren mit dem Xcodeplugin VVDocumenter⁴. Auf Basis des Letzterem ließe sich bei Bedarf aus den Kommentaren eine Dokumentation im html-Format generieren.

Einen der ersten Schritte im Arbeitsprozess nach Erstellung des Exposés konnten wir mit Hilfe einer Umfrage bezüglich der Kundenbedürfnisse, -erwartungen und -bedenken respektive des Genres der Applikation, sowie unserer spezifischen Vorstellung, sprich der möglichen Features des Endprodukts, erreichen. Im Rahmen der ersten internen Treffens des Teams und der Plenumsveranstaltungen konnten sich die Vorstellungen langsam konkretisieren. Deutlich wurde dabei, wie diese zu priorisieren sind und welche Einschränken (wie bspw. Sicherheitsaspekte,

⁴<https://github.com/onevcat/VVDocumenter-Xcode>

Konzept	Zielgruppe
<h2>Home</h2> <p>Willkommen zum Gruppe 1 Banking-Projekt wiki!</p> <p>Weiterführend:</p> <ul style="list-style-type: none">• Features• Exposé• Team-Mitglieder• Konzept• Toolchain• Planung• Xcode Workflow	<p>• junge Leute • Berufstätige • Studenten</p> <p>Begründung:</p> <ul style="list-style-type: none">• Junge Leute für den Markt offen• Rentner nicht iPad- oder iPhone-affin• kein Vertrauen in Direktbanken => keine Anwerbung <p>Weitere Informationen im Umfragebogen.</p>
<h2>„To Hire a Product“</h2> <ul style="list-style-type: none">• Persönlicher Ansprechpartner• Visitenkarte, direkte Durchwahl• Klappt sowohl auf dem Land als auch in der Stadt• Menschliche Komponente• Komplexere Finanzprodukte, da pers. Anspr.• Problem: Gebühren für Konten etc.• Hilfe mit Steuererklärung bzgl. Zinsen• Grobe Kategorisierung des Kunden	

Abbildung 7: Wiki bei GitHub

Wertigkeit des Features) an eine mögliche Umsetzung geknüpft sind. In diesem Prozess wurde das Konzept stetig auf die Rückmeldungen der Plena angepasst. Begleitet wurde das Vorgehen durch das Erstellen von Mockups, die bereits in diesem Stadium helfen konnten, Inhalte nicht zuletzt visuell zu vermitteln und zu besprechen. Die Verwendung von Mockups fand im weiteren durchgängig Anwendung, um Skizzen für eine mögliche Visualisierung zu erstellen, bevor diese final umgesetzt wurden.

Besonderes Augenmerk lag in der frühen Phase bezogen auf das Konzept im Herausarbeiten eines Alleinstellungsmerkmals, durch das sich mit Hilfe der Applikation die Filialbank von der Direktbank positiv abheben kann. Dieser Punkt markierte gewissermaßen bei unserem Team die Hürde, um mit der eigentlichen Umsetzung zu beginnen. Die Featureideen sortierten wir fortlaufend neu nach geschätzter Priorität. Im Wiki unterschieden wir dabei grundsätzlich zwischen „Major“ und „Minor“.

Mit den Major-Features sollten die Kernfunktionalitäten der Applikation beschrieben werden. Hierzu zählten unter anderem der Filialfinder, der individuell angepasste Kreditrechner, sowie später primär fokussiert der Self-Service. Features, welche die App in der Gesamtheit aufwerten sollten aber nicht zur Kernfunktionalität gehören, bildeten die Minor-Kategorie. Ein Beispiel hierfür ist die Überweisungsfunktion für Girokonten. Sie wird in diesem Kontext erwartet, stellt

aber keine Möglichkeit zur Abgrenzung gegenüber ähnlichen Anwendungen dar. Sie trägt außerdem nicht direkt dazu bei, die übergeordnete Fragestellung den Kunden wieder mehr an die Filialbank zu binden, zu erfüllen. Im Rahmen der Priorisierung verdeutlichte sich des weiteren, welche Features später nicht umgesetzt würden. Ein Feature das aufgrund niedrig eingestufter Priorität nicht den Weg bis in das Endprodukts geschafft hat, ist bspw. der SEPA-Umrechner, der das zuvor bestehende Format in die neue Kodierung umrechnen sollte.

Ebenfalls gab ausgedehnte Überlegungen, ob die Software für iPad, iPhone oder sogar dual für beide Geräte entwickelt werden sollte. Wie bei den Features war es hier hilfreich, wenn auch nicht einfach, demokratisch über eine Entscheidung abzustimmen. Dies zog nach sich, dass das Kollektiv den Einzelnen teils entgegen seiner eigenen Überzeugung zu einer gemeinsamen Lösung stimmte.

Da die Kernfrage der Kundenbindung an die Filialbank schwer zu beantworten war, kam die Fragestellung des Zielgeräts und der damit verbundenen Auswahl an Features nach einer zuvor vermeintlich finalen Entscheidung mehrmals auf. Schlussendlich fiel die Entscheidung auf das iPad, weil wir die Kernfeatures durch dieses besser abdeckt sahen und sich nach der Umfrage der Eindruck einstellte, dass Nutzer sicherheitskritische Anwendungen bevorzugt in Ruhe Zuhause statt unterwegs verwenden wollen. Der Entschluss, sich bei der Entwicklung auf ein Gerät zu fokussieren, wurde teamintern unter den Gesichtspunkten mangelnden Gewinns einer dualen Entwicklung und dem damit verbundenen zusätzlichen Aufwand entschieden.

5.2 Umsetzungsphase

Nachdem das grundlegende Konzept ausgearbeitet war, konnten wir mit der konkreten Umsetzung beginnen. Infolgedessen musste sich das Team auf einen Satz von Tools einigen, mit denen die Entwicklungsaufgaben verwaltet, zugeteilt und festgehalten wurden. Initial fiel die Entscheidung auf eine Kombination von Google-Docs und GitHub-Issues/Milestones. In GitHub lassen sich sog. Issues definieren. Diese Issues stellen Aufgaben dar, die Mitgliedern des Teams zugewiesen werden. Einzelne Issues werden wiederum Zwischenzielen zugewiesen, den Milestones. Sind alle Issues bezüglich eines Milestones abgearbeitet, ist dieses erfüllt. Die Verwaltung und Verteilung der Aufgaben wurde von diesem Zeitpunkt an durch den Projektleiter zentral durchgeführt. Alle neu entstehenden Aufgaben wurden also immer in Absprache mit diesem in das

Issuesystem eingefügt. Dies war für die Übersicht der kommenden Aufgaben, verbunden mit der Gewissheit, dass alle Mitglieder über den Fortschritt im Ganzen wie im Einzelnen den gleichen Informationsstand haben, eine echte Hilfe. Die in GitHub eingetragenen Issues wurden neben der Zuweisung an Personen mit Labels versehen, die zum einen die Priorität zum anderen, das Überthema der Aufgabe beschreiben. Hochprior wurde u. a. das Erstellen einer Basisarchitektur auf der im weiteren softwaretechnisch aufgebaut werden sollte, sowie das Beschreiben der vorkommenden Entitäten und der damit verbundenen Relationen eingestuft.

Während wir GitHub zum Zuteilen und Festhalten der Aufgaben einsetzten, konnte mit Google-Docs der zeitliche Aufwand dafür festgehalten werden. Pro Aufgabe wurde eine Zeile mit entsprechendem Titel angelegt. Falls vorhanden, ist in der Tabelle das korrespondierende GitHub-Issue mit Verlinkung eingetragen. Die den Aufgaben zugewiesenen Teammitglieder sind mit Initialen vermerkt. Aus der tatsächlich aufgewandten Zeit multipliziert mit der Anzahl der zugeordneten Personen resultiert der kumulierte Aufwand. Die Vorstellung der zu erledigenden Arbeiten war zu diesem Zeitpunkt allerdings meist unvollständig konkretisiert. Davon abgesehen gab es keine Vorerfahrungen bezüglich des iOS-Frameworks. Entsprechend mussten wir davon ausgehen, dass Aufwandsschätzungen zu Beginn nur bedingt hilfreich sein konnten. Trotzdem war dies als Vorlauf evtl. wichtig, um im weiteren Verlauf ein gutes gemeinsames Verständnis für die Dokumentation und Verwaltung des Arbeitsaufwands zu entwickeln und zu präzisieren.

Xcode Workflow beschreiben				
	A	B	C	E
	Issue	Link	Bearbeiter	#Personen
1	Feature			
2	Zelterfassung eingerichtet		MF, KSM	2
3	Tickerboard erstellt und erste Tickets anlegen		MF	1
4	Aufenden Projektmanagement (bisher)		MF	1
5	Laufenden Projektbericht erstellen			
6	Datamodell Architektur			
7	MasterView einrichten	61655916 Pivotal	MS	1
8	Dependency Injector			
9	Xcode Workflow beschreiben	27,26 Github	KSM, MS, AD	4
10	Prototypen implementieren	19 Github	MF, AD	2
11	Mockup			
12	LoginSystem			
13	Dashboard Beta			
14	Navigation View-Basis-Struktur	30 Github	KSM	1
15	Überweisung durchführen	27 Github	MF, KSM	2
16	Authentifizierung	24 Github	MF, J, MS	2
17	Navigation gefixt	33 Github	MS, AD	2
		35 Github	MF	1

F	G	H	I	J
Erwarteter Aufwand	Geschlossen am	Tatsächlicher kumulierter Aufwand	(h)	Kommentar
01:00:00	17.11.2013	00:30:00	01:00:00	
nicht soviel	18.11.2013	03:00:00	03:00:00	
		02:00:00	02:00:00	
30:00:00		05:00:00	05:00:00	
15:00:00	26.11.2013	15:00:00	60:00:00	erster Entwurf am 26.11. fertiggestellt, evtl. weitere Iteration notwendig
03:00:00		02:00:00	06:00:00	
02:00:00	19.11.2013	02:00:00	04:00:00	
00:30:00	19.11.2013	00:30:00	00:30:00	
06:00:00		05:00:00	05:30:00	
08:00:00		04:00:00	04:00:00	
		03:00:00	03:00:00	(noch nicht ins Haupt-Projekt integriert)
05:00:00	26.11.2013	03:00:00	03:00:00	HTML + DummyPieCharts
02:00:00	26.11.2013	03:00:00	03:00:00	löst alten DetailViewController ab
05:00:00		05:00:00	10:00:00	
03:00:00		02:00:00	04:00:00	
01:00:00		02:00:00	02:00:00	funktioniert jetzt für jeden Controller

Abbildung 8: Google Doc

Während der Anfangsphase etablierte sich im Team ein Arbeitsrhythmus mit zwei Terminen

pro Woche, in denen jeweils das Arbeitspensum eines Werktages oder darüber hinaus absolviert wurde. Die zeitlichen Gegebenheiten mussten auf unsere Gruppe von Studenten mit je unterschiedlichen Stundenplänen abgestimmt werden. In voller Gruppenstärke war es, abgesehen von Ausnahmen, folglich kaum möglich, außerhalb der festen Projekttermine zusammenzukommen. Um in unseren Arbeitsabläufen dennoch Kontinuität herzustellen, haben wir die vereinbarten Termine entsprechend organisiert, dass mindestens die Hälfte der Gruppe anwesend sein konnte. Folglich wurden die Termine alternierend in unterschiedlichen Besetzungen durchgeführt.

Die frühen Teamtreffen waren neben den eigentlichen Aufgabenstellungen stark davon geprägt, sich mit Objective-C, dem iOS-Framework sowie der Entwicklungsumgebung vertraut zu machen. Vielleicht wurde auch infolge dessen zunächst primär eine technische Umsetzung unter Vernachlässigung visueller Aspekte versiert. Nach der Rückmeldung in einem der frühen Plenumstermine, dass visuelle Gestaltung der technischen zeitlich nicht nachsteht und parallel dazu entwickelt werden sollte, wurde die Arbeitsweise dahingehend umgestellt. Technische und visuelle Gestaltung wurden von diesem Zeitpunkt an gleichzeitig entwickelt.

5.3 Vorgehensweise nach Scrum

Im Anschluss an die Scrum-Zertifizierung wurden Arbeitsabläufe und Positionierung der Mitglieder im Team erneut reflektiert, um einen Scrum-Prozess mit den entsprechenden Rollen zu realisieren. Korrespondierend zu den zuvor zugewiesenen Positionen des Entwicklers, Projektleiters, Beraters und Konzepters wurde nach Äquivalenten in der Scrum-Terminologie gesucht. Entwickler und Konzepter wurden der Gruppe des Development Teams zugewiesen. Während beim Berater eine Analogie zum Scrum Master gebildet wurde, lag es nahe, den Projektleiter dem Product Owner zuzuordnen.

Bereits vor der Zertifizierung hatte sich durch den Scrum Master die Praxis eines Daily Scrums eingestellt. Das Daily Scrum Meeting wurde immer zu Beginn jedes Treffens umgesetzt. Dies brachte den positiven Effekt mit sich, dass trotz alternierender Besetzungen sich alle Mitglieder zu Arbeitsbeginn über den aktuellsten Stand ausgetauscht hatten. Der Product Owner zuvor Projektleiter verwaltete weiterhin wie zuvor beschrieben zentral das Aufgabenmanagement. Problematisch erwies sich die zuvor getroffene Toolauswahl zur Organisation der Aufgaben unter den Gesichtspunkten von Scrum. Primär fehlte es den Tools an Möglichkeiten der Idee des iterativen

Sprints gerecht zu werden. Infolgedessen stellte das Team die gesamte Toolchain zur Verteilung der Aufgaben und deren Aufwandsmessung um. Die Wahl fiel auf das Tracking-Tool PivotalTracker, in welchem die Terminologie Scrums mit Sprint-Backlog, Sprint-Velocity, „Done“- Status, usw. vertreten war. Die zuvor erstellten Issues und Aufwandseinschätzungen der noch offenen Aufgaben wurden komplett in die neue Umgebung übertragen.

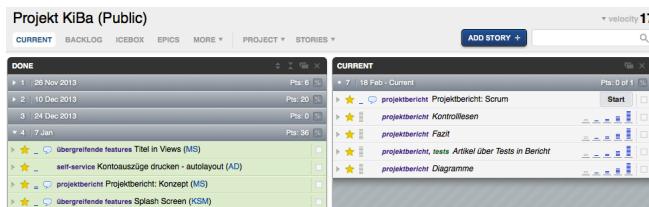


Abbildung 9: Stories bei PivotalTracker

Als Iterationsdauer legten wir einen Zeitrahmen von zwei Wochen fest. Diese Dauer erschien uns als sinnvoll; einerseits lang genug um, kleine oder mittelgroße Aufgaben innerhalb einer Iteration zu bewerkstelligen, anderseits nicht zu lang, um dynamisch auf geänderte Anforderungen reagieren zu können, ohne den Sprint-Backlog innerhalb eines Sprints modifizieren zu müssen.

Der Prozess des dynamischen Reagierens auf Kritik und sich damit ändernden Anforderungen vollzog sich über die gesamte Zeitspanne des Projekts. Das stetige Überdenken und Anpassen voriger Überlegungen und Ausarbeitungen stellte sich als eine der größten, wenn nicht als die größte Herausforderung des Projekts dar. Eine ständige Korrektur führt zum Verwerfen voriger Arbeit, woraus die Anforderung entsteht, sich stetig für neue Richtungswechsel zu motivieren.

Nach der Anfangsphase wurden die Aufwandsschätzungen präziser. Weiterhin war es zwar schwierig, genaue Schätzungen für einzelne Aufgaben vorzunehmen, insgesamt konnten wir aber trotz alledem auf Basis der sich ergebenden Sprint-Velocity ab dem letzten Drittel des Semesters eine sinnvolle Einschätzung über den verbleibenden Arbeitsaufwand und die dafür benötigte Zeit abgeben. Wie sich herausstellte, konnten wir diese Einschätzung einhalten.

Bezüglich der strengen Richtlinien von Scrum – entweder es wird auf voller Linie praktiziert oder aber das Vorgehensmodell ist nicht eingehalten und somit anders zu benennen – ist erwähnenswert, dass diese eigentlich vorsehen, dass das Development Team zu Beginn seiner Arbeit bereits alle notwendigen technischen Kompetenzen zur Umsetzung eines Projekts besitzt. Wir waren allerdings damit konfrontiert, selbstständig fortlaufend Wissen zu erarbeiten, das zur Be-

wältigung der Aufgaben notwendig war.

Neben den erwähnten Vorteilen des Tracking-Tools, respektive Struktur und Terminologie, ermöglichte uns die neue Umgebung das Verhältnis von bestehenden und noch zu erledigenden Aufgaben in Balkendiagrammen und Burndown-Charts zu visualisieren. Der Burndown-Chart bzw. Progress-Report über die gesamte zweite Hälfte der Projektdauer ergibt sich wie folgt:

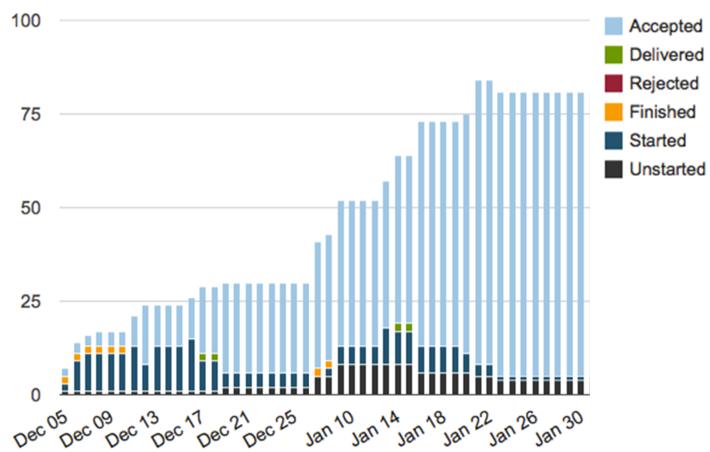


Abbildung 10: Von PivotalTracker generiertes Burndown-Chart

Aus der Tendenz ist ersichtlich, dass die Sprint-Velocity unter Vernachlässigung der Ferienzeit stetig bis zur Deadline zunahm.

6 Ergebnis von Marco F. Jendryczko

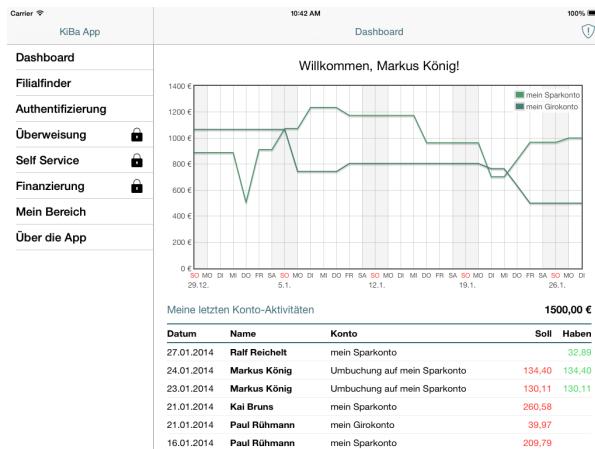


Abbildung 11: Screenshot des Dashboards

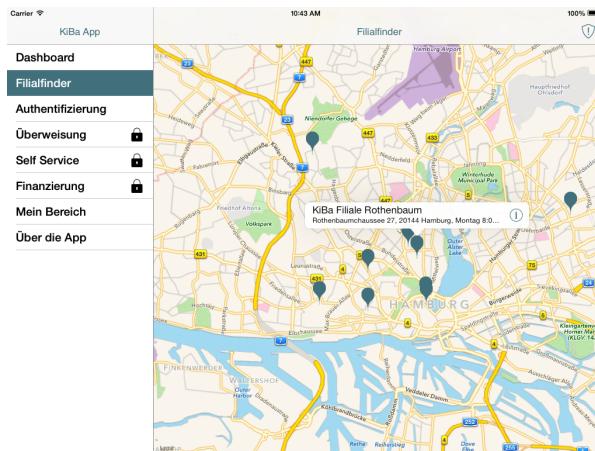


Abbildung 12: Screenshot des Filialfinders

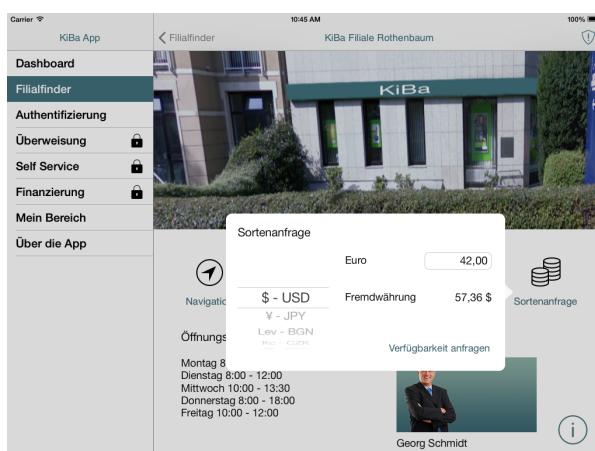


Abbildung 13: Screenshot der Filialseite mit Sortenanfrage Pop-Over

6.1 Features

6.1.1 Dashboard

Das Dashboard ist die zentrale Anlaufstelle der Applikation; dort haben wir einen Überblick über den Verlauf der Kontostände aller unserer Konten. In der Tabelle darunter haben wir die Möglichkeit, alle Kontobewegungen nachzuverfolgen.

6.1.2 Filialfinder

Beim Filialfinder findet man eine Karte von Apple Maps mit Markierungen, welche die Standorte der einzelnen Filialen markieren. Mit einem Klick auf das Infosymbol gelangt man zu einer Filialseite, auf der man die Öffnungszeiten nachschauen, sowie eine Terminanfrage oder Sortenanfrage stellen kann.

6.1.3 Sortenanfrage

Mit einem Klick auf Sortenanfrage bekommt der Nutzer ein Pop-Over zu sehen, in welchem er seine Wunschwährung auswählen kann. Die entsprechenden Umrechnungskurse werden intern direkt von der Europäischen Zentralbank bezogen. Hat man die Summe, welche umgetauscht werden soll eingegeben, so kann man die Anfrage stellen und bekommt eine Nachricht, ob die gewünschte Währung in der entsprechenden Höhe bei der Filiale verfügbar ist.

6.1.4 Authentifizierung



Abbildung 14: Die Authentifizierungsstati

Im oberen rechten Rand der Applikation ist ein kleines Symbol in Form eines Schildes zu sehen; dieses gibt den Authentifizierungsstatus zurück. Ein Schild mit einem Ausrufezeichen symbolisiert eine fehlende Authentifizierung. Ein Haken signalisiert eine gültige Authentifizierung.

Solange das Gerät nicht authentifiziert ist, kann man einige Funktionen nicht benutzen, diese sind durch das Symbol eines ungeöffneten Schlosses gekennzeichnet.

Unter dem Menüpunkt Authentifizierung findet man ein kleines Comic vor. Dieses erläutert, wie der Nutzer sein Gerät bei seiner Filiale authentifizieren kann. Außerdem wird der Nutzer auf seine Vorteile aufmerksam gemacht.

6.1.5 Überweisung

Unter dem Menüpunkt Überweisung findet man ein Überweisungsformular in digitaler Form vor. Nachdem das Formular ausgefüllt ist, muss die Überweisung abschließend mit einer TAN bestätigt werden. Kontonummer und Bankleitzahl werden im Zuge dessen auf Korrektheit ihres Formats geprüft.

6.1.6 Self-Service-Station

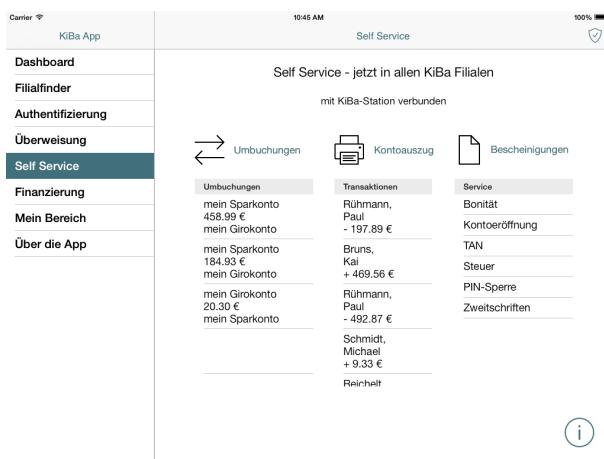


Abbildung 15: Self-Service Startseite. Die App ist verbunden.

Bei der Self-Service-Station haben wir die Möglichkeit, sofern wir mit dem Stationsgerät verbunden sind, drei verschiedene Aktionen eigenständig durchzuführen. Eine Umbuchung, Kontoauszüge anschauen und drucken, sowie Bescheinigungen ansehen, ausfüllen und ausdrucken

gegebenfalls direkt abschicken.

Jede Auswahlmöglichkeit hat eine Tabelle mit Inhalten, die den User erwarten, wenn er auf den entsprechenden Button drückt.

6.1.7 Bescheinigungen

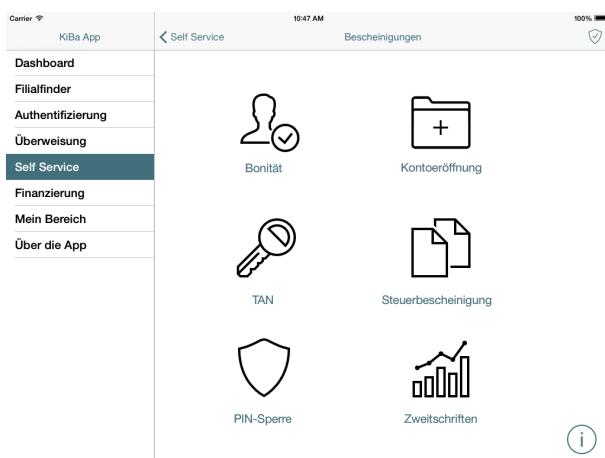


Abbildung 16: Die verschiedenen Dokumenttypen

Bei den Bescheinigungen sehen wir verschiedene von der Bank zur Verfügung gestellte Dokumente, die der Nutzer auswählen kann. Insbesondere beispielsweise sind Aktionen wie eine Kontoeröffnung möglich, für die man normalerweise am Schalter anstehen müsste.

6.1.8 Kontoauszüge

Im Kontoauszugsbildschirm kann man Kontoaktivitäten für einen bestimmten Zeitraum drucken lassen. Somit integriert die Station den althergebrachten Kontoauszugsdrucker und gibt ihm eine zeitgemässere Oberfläche.

6.1.9 Umbuchungen

Im Umbuchungsbildschirm kann der Nutzer sowohl Ziel- als auch Quellkonto auswählen, zwischen denen der angegebene Betrag transferiert werden soll. Das Ganze wird mit einem Button bestätigt. Insbesondere wird hier das klassische Sparbuch um eine komfortable Umbuchungs-

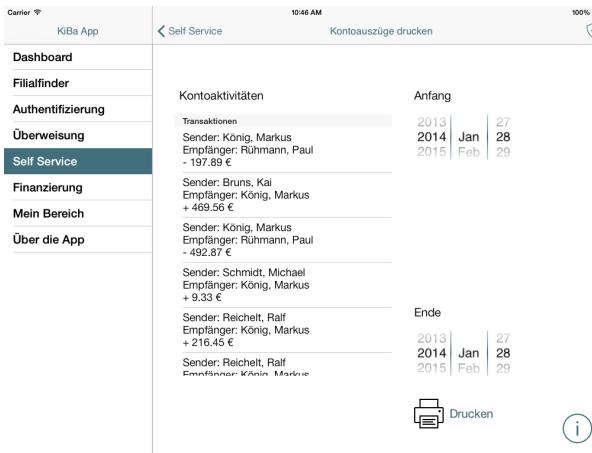


Abbildung 17: Übersicht der Kontoauszüge

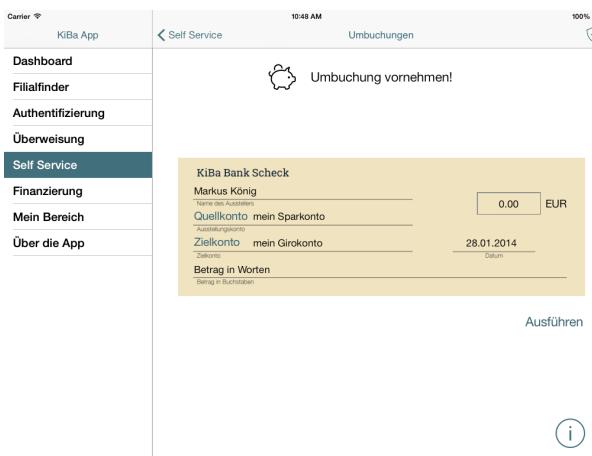


Abbildung 18: Eine Scheckgrafik als Formular für die Umbuchung

funktionalität erweitert. Umbuchungen werden weiterhin in den Räumlichkeiten der Bank vorgenommen, weshalb der filialgebundene Charakter erhalten bleibt.

6.1.10 Finanzierungsrechner

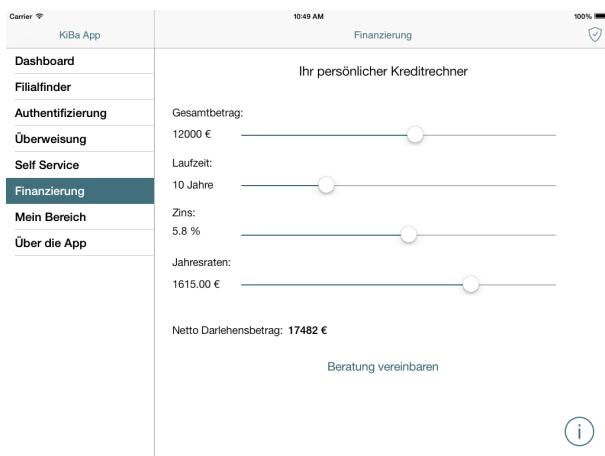


Abbildung 19: Finanzierungsrechner mit individuellen Konditionen

Mit dem Finanzierungsrechner kann man sich Kreditkonditionen zusammenstellen. Die Werte der einzelnen Schieberegler werden aus dem Profil des Kundens von der Bank vorgegeben.

Anschließend kann man einen Beratungstermin vereinbaren, um die Konditionen mit seinem Berater im Detail durchsprechen zu können. Im Rahmen des Gesprächs bietet sich die Möglichkeit zu prüfen, inwiefern die ausgewählten Konditionen empfehlenswert für den Kunden sind oder ob die Bank vielleicht noch bessere Konditionen anbieten kann.

6.1.11 Mein Bereich

Der Nutzer kann in seinem persönlichen Bereich Nachrichten der Bank empfangen und somit beispielsweise eine Bestätigung für seine Terminanfrage erhalten. Mit einem Klick auf die Zelle einer Nachricht bekommt man die entsprechende Nachricht in einer neuen Ansicht mit dem kompletten Nachrichtentext.

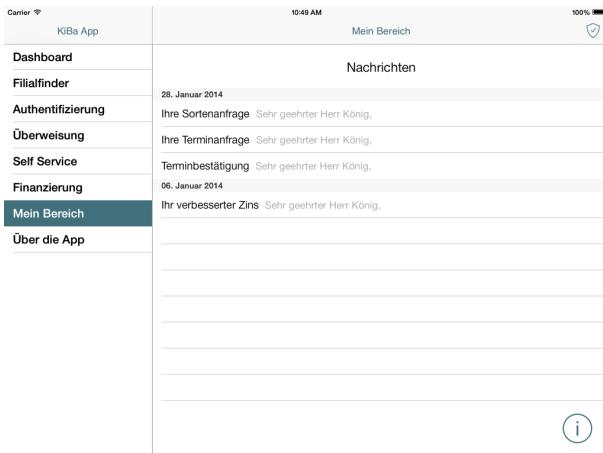


Abbildung 20: Ein minimalistischer Posteingang des Nutzers

6.2 Die Design-Highlights

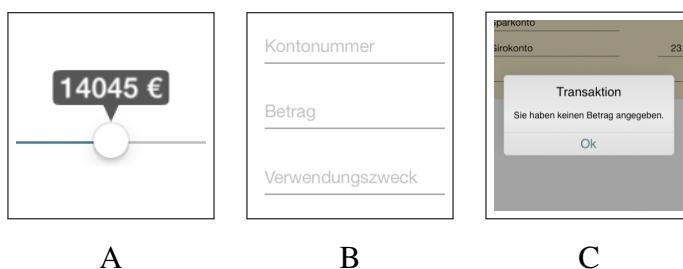


Abbildung 21: Eigenkreationen für das Design

Um unsere Applikation in mancher Hinsicht besser auf unsere Bedürfnisse anzupassen, haben wir bei einigen Designelementen die Standardimplementierungen von Apple ergänzt. Dennoch blieben die iOS 7 Richtlinien ein Qualitätsmaßstab für uns.

Wie in (A) zu sehen ist, bekam unser Schieberegler als kleine Erweiterung eine schwarze Sprechblase, welche den aktuellen Wert anzeigt. Dies ist im Speziellen unter dem Gesichtspunkt der Bedienbarkeit implementiert worden, damit man beim Verschieben des Reglers nicht immer auf die linke Seite schauen muss, um den aktuellen Wert abzulesen.

Die mitgegebenen Textfelder von Apple haben in der Regel immer einen Text, welcher dem Nutzer einen Hinweis gibt, was für ein Inhalt erwartet wird. Diese Umsetzung sieht oftmals unschön aus und nimmt viel Platz ein. Mit unserer Implementierung (B) verliert das Textfeld keine Funktionalität, ist aber schlanker.

Die Standard Pop-Over lassen sich nur bedingt anpassen. So konnten wir die Farbe des Buttons nicht in unserem Farbton einfärben. Daher mussten wir auf eine Eigenimplementation (C) zurückgreifen um die Konsistenz zu wahren.

6.3 Herausforderungen

Mit zu den größten Herausforderungen war das Design der einzelnen Bildschirme. Das „Flat“-Design braucht ein gutes Händchen und kleine Details können schon zu Unstimmigkeiten führen. Gleichzeitig war uns bewusst, dass wir eine Bank repräsentieren und somit ein gewisses Maß an Seriosität erforderlich war. Gleichzeitig mussten wir die Usability für den Endnutzer gewährleisten. Wir mussten ein App-Design erschaffen, welches drei Stakeholder gleichzeitig zufriedenstellt.

Mit einer neu zu erlernenden Programmiersprache und deren Entwicklungsumgebung geht auch immer eine Herausforderung einher. Die ersten Wochen hatten wir somit erwartungsgemäß etwas geringere Produktivität, weil wir uns auf diese Gegebenheiten erst einstellen mussten. Je weiter jedoch das Projekt voranschritt, desto sicherer fühlten wir uns in Objective-C und im Umgang mit Xcode.

Während der Entwicklung stießen wir immer wieder auf Einschränkungen seitens Apple bezüglich der Standard GUI-Elemente. So konnten wir beispielsweise die Tint-Color eines Buttons im Pop-Over nicht ändern. Wie erwähnt griffen wir, falls nötig, in diesen Fällen auf Eigenimplementierungen zurück.

Darüber hinaus wollten wir gewährleisten, dass die GUI-Elemente unserer Applikation, sowohl in der Portrait-Ansicht, als auch in der Landscape-Ansicht gut gesetzt sind. Dafür haben wir in den meisten Fällen die Auto-Layout-Technik verwendet. Die Verhältnisse und Größen der Elemente werden dabei mit Hilfe von Constraints beschrieben. Ändert sich das Ansichtsformat, passen sich diese dynamisch an die neuen Gegebenheiten an.

Bei einigen Features überdeckte die hereinfahrende Tastatur das vom Nutzer ausgewählte Eingabefeld. Anstatt die Anordnung der Elemente der einzelnen Bildschirme zu verändern, haben wir dynamische Scrollviews in das Projekt integriert. Diese reagieren stets so, dass sich das vom Nutzer fokussierte Element möglichst zentral im freien Sichtbereich befindet.

Zusammenfassend betrachtet konnten wir mit Unterstützung der Betreuer, alle größeren und

kleineren Herausforderungen bewältigen und hatten nie das Gefühl, vor einem unüberwindbaren Problem zu stehen.

7 Architektur von Markus Fasselt

Nachdem zunächst in diesem Bericht bereits Konzepte und Ergebnisse erläutert wurden, gehen wir an dieser Stelle nun auf die Implementierung und technische Umsetzung der App ein. Im Fokus dabei steht die Architektur, die maßgeblich zum Ablauf der Entwicklung und zur Organisation der Kernkomponenten beiträgt.

Unsere App soll den Kontakt zwischen einer Filialbank und seinen Kunden stärken. Da es sich bei KiBa lediglich um eine fiktive Bank handelt und die App auch anderen interessierten Banken vorgestellt werden soll, bietet es sich an, einen „Click-Dummy“ zu entwickeln. Dieser soll sich bereits wie eine vollwertige Banking-App bedienen lassen, die jedoch an keine reale Bank, respektive deren Datenbank, angeschlossen ist. Aufgrund dieser Rahmenbedingungen haben wir uns für die Architektur entschieden, die im Folgenden vorgestellt wird.

7.1 Umsetzung des MVC-Ansatzes

Die Architektur muss uns dabei auf die Entwicklung der Kernfeatures fokussieren. Wenn nicht klar ist, wo welche Teile der Logik, der GUI oder anderer Komponenten zu platzieren sind, verzögert dies den Entwicklungsprozess und macht das Entwicklungsteam weniger produktiv.

Daher fiel unsere Entscheidung auf den MVC-Ansatz; das heißt, wir versuchten unseren Code in Datenmodellierung, Visualisierung und Kontextualisierung zu gliedern.

Auf oberster Ebene befinden sich deswegen nun die Ordner „Entities“ für die Modellierungsanitäten, „Services“ für Dienstleisterklassen, „Views“ für View-Controller und die die View repräsentierenden xib-Dateien und zuletzt ein „Library“-Ordner, der für alle Bereiche unterstützende Helferklassen sammelt.

Innerhalb dieser Ordner haben wir darüber hinaus eine funktionale Substruktur erschaffen, die nach unseren Feature-Komponenten aufgebaut ist. Das hatte auch den positiven Nebeneffekt, dass kollaboratives Arbeiten am Projekt erleichtert wurde. Denn auf diese Weise war es nicht zwangsläufig für jede Einheit erforderlich, eine eigene Git-Branch zu eröffnen, da in den Ordnern gearbeitet werden konnte.

7.2 Datenmodell

Zur Modellbildung der App fiel unsere Entscheidung auf ein klassisches Entitäten-Beziehungs- system. Unser Weg zur Abstraktion einer Bank führt daher über Klassen, die Modelle zu Objekten der realen Welt darstellen. Wie wir dabei vorgegangen sind, ist im Entitäten-Relationen-Diagramm unseres Datenmodells in Abbildung 22 zu sehen.

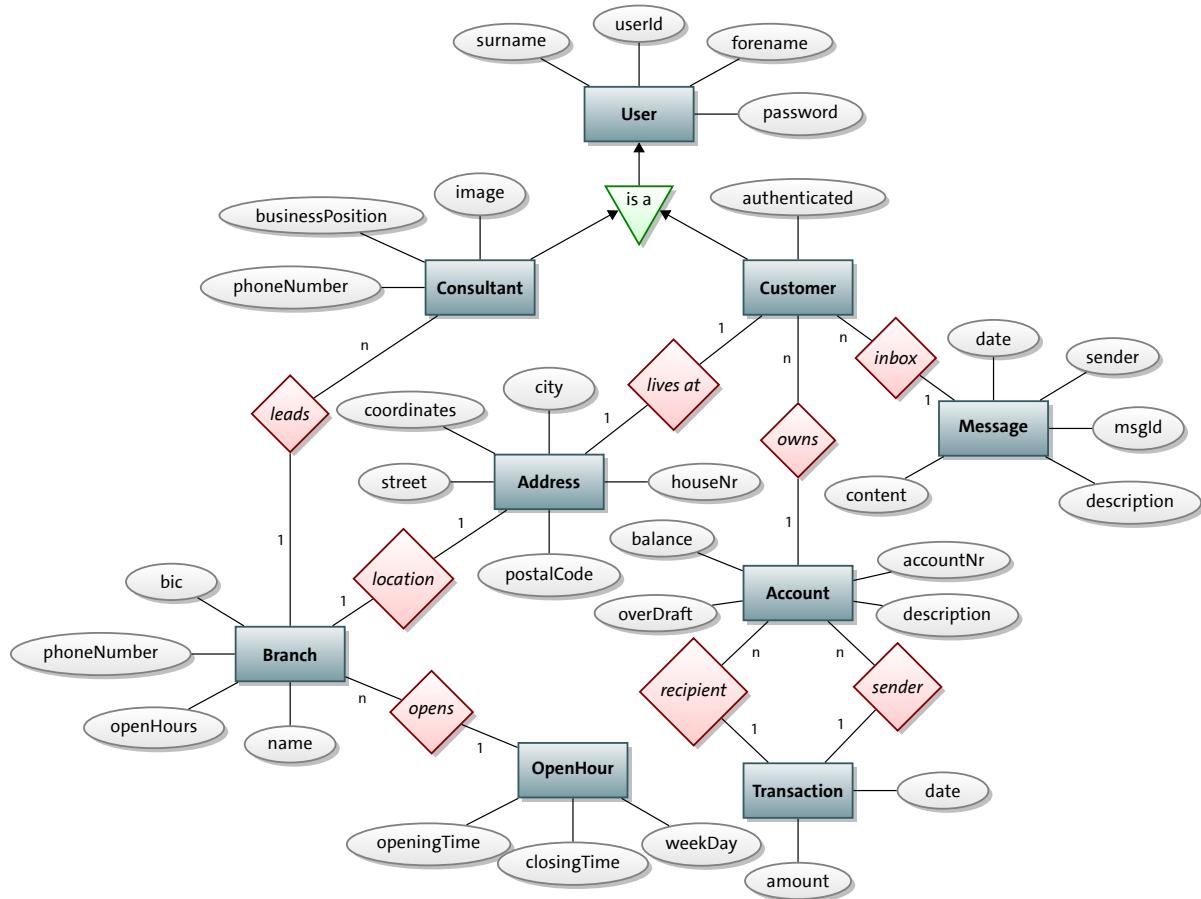


Abbildung 22: Entitäten-Relationen-Diagramm

Da gibt es zum einen die mit der Bank agierenden Benutzer, nämlich den Berater und den Kunden, die von einer gemeinsamen Klasse `User` erben. Kunden, sprich `Customer`, besitzen mehrere Konten (`Account`) und erhalten mehrere Nachrichten (`Message`). Des weiteren verfügen sie, ebenso wie die Filialen, über eine Adresse. Diese Filialen (`Branch`) wiederum werden von Beratern geleitet und sind zu verschiedenen Öffnungszeiten (`OpenHour`) besuchbar. Darüber hinaus finden Transaktionen (`Transaction`) zwischen Konten statt.

7.3 Sicherheitsaspekte

Ein wichtiger Aspekt, insbesondere im Hinblick auf den Umgang mit sensiblen Finanzdaten, ist die Sicherheit der App. Dabei ist es auch eine Aufgabe der Architektur, diese gewährleisten zu können. Im Folgenden erläutern wir die Schritte, die wir dementsprechend für die realistische Umsetzung zogen.

Eine Fragestellung von essentieller Bedeutung für uns ist, was mit der App im Falle eines Diebstahls passieren würde. Da wir sowohl der Bank als auch dem Kunden gewährleisten müssen, dass ihre Daten sicher sind, haben wir das Risiko zu groß eingestuft, die Daten auf dem Gerät zu speichern. Das ist der Grund, weswegen alle Informationen nur im Zwischenspeicher des iPads liegen. Das hat für uns den Vorteil, dass beim Beenden der App alle Informationen verloren gehen, wenn der Kunde nicht mehr eingeloggt ist.

Dass dies für den Kunden nachteilig wirkt, da er die Daten nicht umgehend zur Verfügung hat, ist unser meiner nach ein vernachlässigbarer Faktor. Wie oben bereits geschildert, stellen wir uns die Benutzung der App eher in ruhigen Bereichen vor, in denen in den meisten Fällen eine Internetverbindung existiert. Für den Self-Service gehen wir davon aus, dass die Bank ihren Kunden in der Filiale eine Internetverbindung zur Verfügung stellt. Sowieso sollte es eher im Interesse des Kunden sein, wichtige Finanzinformationen nur an Orten abzurufen, wo sie nicht vielen Menschen einsehbar sind.

Zudem würde in einer über einen Dummy hinausgehenden Implementierung ein Time-Out-Token (ähnlich einem Cookie im Web-Umfeld) implementiert, welche nach beispielsweise zehn Minuten ohne Interaktion die Anwendung beendete. Außerdem gewährleistet der Authentifizierungsmechanismus, dass kritische Features serverseitig deaktiviert werden, indem der entsprechende Benutzer wieder auf nicht authentifiziert geschaltet wird. Dieser Mechanismus kann einerseits bei einer Verlustmeldung greifen, aber auch bei in irgendeiner Weise verdächtigem Verhalten, also ungewöhnlich viele oder große Transaktionen in bestimmten Zeiträumen.

Wichtig ist außerdem, dass alle Daten direkt übertragen werden. Wir stellen uns ein direktes Protokoll wie eine REST-Schnittstelle oder ein SOAP-Verfahren zum Austausch der Informationen zwischen App und Server der Bank vor.

Insbesondere letzter Punkt kann es erforderlich machen, dass die Datenquelle anpassbar sein

muss. Eine Möglichkeit, das zu realisieren, erläutern wir im nächsten Abschnitt.

7.4 Dependency Injection

Eine zentrale Anforderung der App-Architektur für uns ist außerdem das Austauschen von einzelnen Kernkomponenten, wie etwa die Datenschicht. Denn hierdurch kann aus dem Click-Dummy eine vollwertige Banking-App erschaffen werden können, ohne große Änderungen am Code vornehmen zu müssen. Sie muss uns den Eindruck nehmen, an eine echte Bank gebunden zu sein.

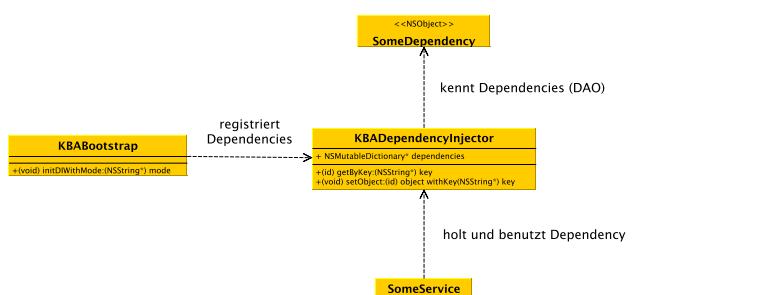


Abbildung 23: Klassendiagramm unserer Dependency Injection

Daher ist für uns Dependency Injection als Entwicklungsmuster die Lösung dieses Problems. Wir haben es in einer reduzierten Form selbst implementiert. Dabei wird an einer zentralen Stelle im Quelltext, nämlich im *Bootstrapping*, in den **KBADependencyInjector** wie in Abbildung 23 sichtbar registriert, welche konkrete Implementierung einer Abhängigkeit in der Anwendung verwendet werden soll. Beim **KBADependencyInjector** handelt es sich dabei nur um einen einfachen Schlüssel-Wert-Speicher.

Der Bootstrapping-Prozess unserer App ist im Programmausdruck 1 wiedergegeben. Er zeigt an, wie zunächst lokale Variablen zu den generalisierten Protokollen typisiert sind, dann allerdings in einer bedingten Verzweigung entweder mit einer Dummy- oder einer konkreten Implementierung instantiiert werden. Abschließend werden die instantiierten Abhängigkeiten in den **KBADependencyInjector** geschrieben. Beispielsweise wird für das Filial-Data Access Object (DAO) eine lokale Variable des Typs `id<KBABranchDao>` eingeführt, die im Entwicklungsmodus mit der Dummy-Implementierung `KBABranchDaoDummy` und produktiv mit dem `KBABranchDaoRest` instantiiert wird. Danach wird sie registriert.

Programmausdruck 1: Der Bootstrapping-Vorgang

```

/*
 * Bootstrapping der App-Abhängigkeiten.
 * `mode` ist entweder "dev", "prod" oder "test".
 */
+ (void) initDependencyInjectorWithMode:(NSString *)mode {
    // Lokale Variablen, die zu Protokollen typisiert sind
    id<KBABranchDao> branchDao;
    id<KBAExchangeRateDao> exchangeRateDao;
    id<KBACustomerDao> customerDao;
    id<KBAccountDao> accountDAO;
    id<KBATransactionDao> transDAO;
    id<KBACreditRatingDao> creditDAO;
    id<KBAMessageDao> messageDAO;

    if ([mode isEqualToString:@"dev"]) {
        // Dummy-Abhängigkeiten zum Testen und zur Entwicklung
        branchDao = [KBABranchDaoDummy new];
        customerDao = [KBACustomerDaoDummy new];
        accountDAO = [KBAccountDaoDummy new];
        transDAO = [KBATransactionDaoDummy new];
        creditDAO = [KBACreditRatingDaoDummy new];
        messageDAO = [KBAMessageDaoDummy new];
    }
    else {
        // Konkrete Abhängigkeiten für die produktive Anwendung
        creditDAO = [KBACreditRatingDaoRest new];
        branchDAO = [KBABranchDaoRest new];
        customerDAO = [KBACustomerDaoRest new];
        accountDAO = [KBAccountDaoRest new];
        transDAO = [KBATransactionDaoRest new];
        messageDAO = [KBAMessageDaoRest new];
    }

    // Eine Ausnahme: Der Währungskurs-DAO wurde von uns konkret
    // mit Daten der EZB implementiert.
    exchangeRateDAO = [KBAExchangeRateDaoRest new];

    // Speichern der Abhängigkeiten im Dependency Injector
    [KBADependencyInjector setObject:branchDAO forKey:@"branchDAO"];
    [KBADependencyInjector setObject:exchangeRateDAO forKey:@"rateDAO"];
    [KBADependencyInjector setObject:customerDAO forKey:@"customerDAO"];
    [KBADependencyInjector setObject:accountDAO forKey:@"accountDAO"];
    [KBADependencyInjector setObject:auth forKey:@"auth"];
    [KBADependencyInjector setObject:transDAO forKey:@"transDAO"];
    [KBADependencyInjector setObject:creditDAO forKey:@"creditDAO"];
    [KBADependencyInjector setObject:messageDAO forKey:@"messageDAO"];
}

```

Wie zu erkennen ist, besitzt jedes DAO ein Protocol, welches auf zwei Arten instanziert werden kann. Dadurch erreichen wir, dass ein Programm auf verschiedene Art und Weise ausführbar ist. Die zentrale Registrierung der Abhängigkeiten hat außerdem den Vorzug, dass für eine andere Bank, die über eine andere Schnittstelle als (in diesem Fall) Representational State Transfer (REST) verfügt, eine andere konkrete Implementierung nötig werden kann, die recht simpel nur

an dieser Stelle im Code eingefügt werden muss. Man muss sich keine Gedanken mehr über weitere Abschnitte des Quelltext bei so einer Änderung machen.

Ein weiterer Vorteil dieser Abstraktion ist die Testbarkeit der App. Dadurch, dass im Click-Dummy feste Daten hinterlegt sind, kann man sich zum Testen der App verschiedene Fälle erzeugen, die einfach in den jeweiligen **DaoDummies* benutzt werden. Ein Beispiel für Unit Tests finden Sie im Programmausdruck 2 in Kapitel 9.4.

Darüber hinaus erlaubt uns dieses Verfahren auch die erleichterte Austauschbarkeit einzelner Komponenten. Wenn man dieses Verfahren auf andere Funktionen der App anwendet, insbesondere dann, wenn benutzerspezifische Elemente erforderlich sind, so ist es schnell möglich, eine White-Label-App zu erzeugen. Das heißt also wir können eine App schaffen, die auf verschiedene Kunden zugeschnitten werden kann. Dies hat für unseren Kunden T-Systems MMS in seiner Eigenschaft als Vermarkter den Vorteil, ihre App leicht auf Endkunden anpassen zu können.

Wir haben festgestellt, dass die frühe Auseinandersetzung mit Fragen der Softwarearchitektur eine gute Entscheidung ist. Durch die Implementierung unserer eigenen Dependency-Injection und der zentralen Regelung von Abhängigkeiten an einem Ort haben wir eine dynamische und nachhaltige Methode geschaffen, unsere App sukzessive erweitern zu können.

8 Toolchain von Konstantin S. M. Möllers

Im Rahmen unseres Projekts wurde uns schnell klar, dass wir auf viele Tools angewiesen sein würden. So musste unser Quelltext versioniert, die grobe Struktur und Views festgehalten und Aufgabenpakete erstellt und koordiniert werden. Darüber hinaus war uns auch Design und Qualität wichtig. Auf unsere Erfahrungen in diesem Zusammenhang gehen wir im folgenden Abschnitt ein.

8.1 Versionsverwaltung und Informationsaustausch

Um kollaborativ an einer Software zu arbeiten ist es selbstverständlich erforderlich, mit Quelltextversionierung zu arbeiten. Da bei uns Team-Mitgliedern die Erfahrung mit Git⁵ aus dem Universitäts- und Arbeitsumfeld am größten war, viel unsere Wahl für ein VCS darauf.

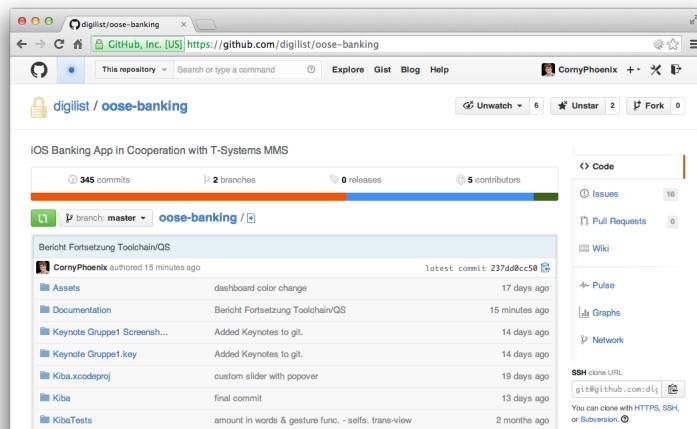


Abbildung 24: Repository-Hosting bei GitHub

Entscheidend dabei war auch der größere Komfort gegenüber Subversion. Einerseits ist die operative Geschwindigkeit von Git deutlich schneller⁶, sorgt also im Team für eine Produktivitätssteigerung. Dieser Punkt ist in sofern für und von Relevanz, als dass auch wir ein Problem mit größeren Dateien, wie den XIB-Views, hatten. Auf der anderen Seite erlaubt Git durch sein

⁵<http://git-scm.com/>, Abruf am 23.02.2014 um 17:55 Uhr.

⁶<http://biz30.timedoctor.com/git-mecurial-and-cvs-comparison-of-svn-software/>, Abruf am 23.02.2014 um 18:05 Uhr.

verteiltes System Branches pro Anwender zu führen, wodurch auch ermöglicht wird, Commits unabhängig vom Internet zu publizieren.

Darüber hinaus schuf Git auch die Möglichkeit, die eng verwandte Plattform „GitHub“⁷ als Repository-Hoster zu verwenden. Dieser bietet über den für die Versionsverwaltung erforderlichen Speicherplatz hinaus auch eine sehr gute grafische Bedienoberfläche, wie sie in Abbildung 24 zu sehen ist.

Ein weiteres nützliches Feature von GitHub für uns war außerdem das zuvor erwähnte Wiki, dargestellt in Abbildung 7. Ein Wiki hatte für uns den Vorteil eines demokratischen Informationsaustauschs; jeder konnte seine Erfahrung mit anderen Projektmitgliedern teilen, sodass alle davon profitierten. Daher tauschten wir in ihm erste Feature-Ideen oder auch Kontaktdaten aller Mitglieder aus. Allerdings nutzten wir es darüber hinaus auch für das Festlegen eines Arbeitsablaufs mit Xcode, ins Besondere das Code Styling oder das Verwenden bestimmter Plugins.

8.2 Issue-Tracking-System

Da uns freundlicherweise von C1 WPS die Möglichkeit gegeben wurde, eine Scrum-Zertifizierung zu erhalten, lag uns viel daran, den Scrum-Ansatz so gut es ging in unserem Projekt-Ablauf zu integrieren. Dementsprechend war für uns bei der Frage nach einer unterstützenden Software wichtig, inwiefern es Scrum-Iterationen unterstützt und erleichtert.

The screenshot shows the Pivotal Tracker interface for the project 'Projekt KiBa (Public)'. It features three main boards: 'DONE', 'CURRENT', and 'ICEBOX'. The 'DONE' board lists completed stories with their names and descriptions. The 'CURRENT' board shows the current iteration with 6 stories listed, including 'selfservice Redesign Self-Service Hauptseite (KSM)' and 'finder Navigation mit Apple Maps'. The 'ICEBOX' board lists stories waiting to be prioritized or assigned. A top navigation bar includes 'PROJECTS', 'DASHBOARD', 'REPORTS', 'HELP', and 'CORNYPHOENIX'. A search bar and a 'Select All' button are also visible.

Abbildung 25: Stories in PivotalTracker

⁷<https://github.com/>, Abruf am 23.02.2014 um 17:55 Uhr.

Dazu verwendeten wir – nach ersten Versuchen mit GitHub-Issues – die Software PivotalTracker. Mit ihr können wir, wie in Abbildung 25 dargestellt, einzelne Stories erstellen und überwachen. Ein großer Vorteil dabei war, dass Scrum-Iterationen automatisch von der Software generiert werden, berechnet durch unsere durchschnittliche Arbeitskraft (genannt „Velocity“)⁸ und unseren Vorhersagen für den Aufwand einer Story angegeben durch Punkte.

Zum anderen generiert PivotalTracker auch Burndown-Charts, welche für das Vorgehen während der Scrum-Iterationen unabdinglich sind. Denn so können wir am besten den Fortschritt unserer App messen und haben ein Monitoring für das Fortschreiten der Entwicklung unserer Features zu sehen ist.

8.3 Konzept- und Designwerkzeuge

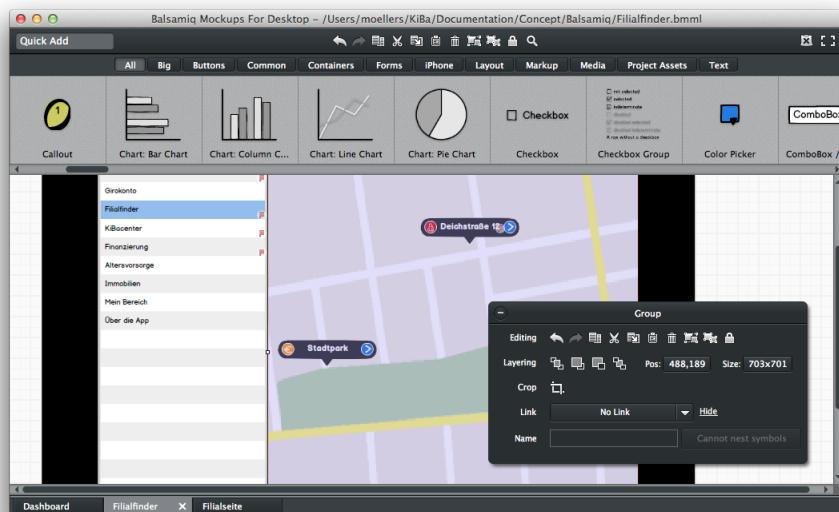


Abbildung 26: Mockup-Entwurf mit Balsamiq

Für das Anfertigen von qualitativen Mockups, die für eine zielführende Implementierung von besonderer Bedeutsamkeit sein können, benötigt man Software, die einem in diesem Vorhaben gut unterstützt.

Mit der in Abbildung 26 dargestellten Software „Balsamiq“⁹, die auch in einem Seminar der T-Systems MMS angesprochen wurde, fanden wir eine gute Lösung. Das interne Asset-

⁸<http://www.pivotaltracker.com/community/tracker-blog/velocity-matters>, Abruf 23.02.2014 um 18:00 Uhr.

⁹<http://balsamiq.com/>, Abruf am 23.02.2014 um 18:50 Uhr.

Management ermöglicht es, Grafiken, die in diversen Mockups benötigt werden, auszutauschen und zu konservieren. Eine weitere nützliche Funktionalität stellte das Verknüpfen von Mockups da, womit man klickbare Referenzen zwischen den Dokumenten herstellt und in einer einfachen Simulation einen Eindruck von der Bedienbarkeit der finalen Version bekommt.

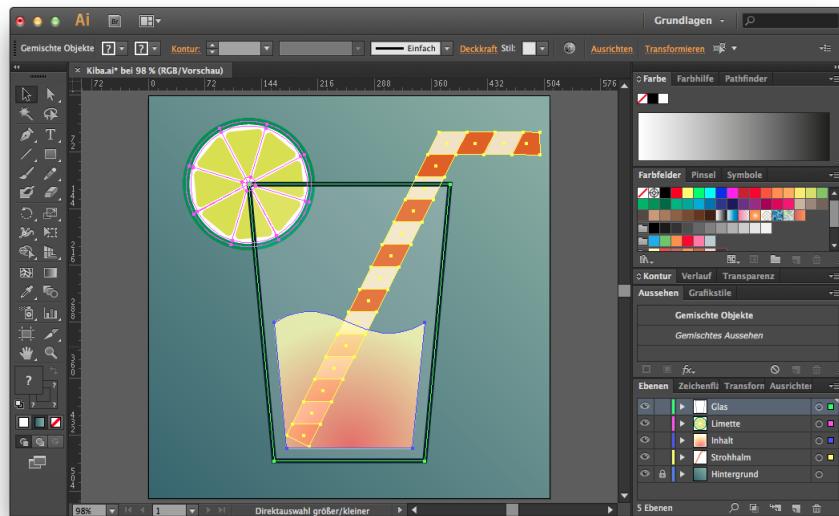


Abbildung 27: Designen des KiBa-Icons mit Adobe Illustrator

Zuletzt, für das Finalisieren von Grafiken und Inhalten, waren auch Adobe Illustrator und Photoshop von großer Bedeutung für uns. Wenn es beispielsweise um die Bearbeitung des Icons geht, welche in Abbildung 27 ersichtlich ist, wären frei verfügbare Bibliotheken für unsere Zwecke am Ende nicht mehr ausreichend. Daher haben wir auf qualitative Eigenproduktionen gesetzt.

9 Qualitätssicherung von Konstantin S. M. Möllers

Für große Projekte wird es zunehmend unabdinglich, über Planung und Ausführung hinaus auch ein Monitoring und Controlling zu besitzen, auf welche ich in Form der Qualitätssicherung in diesem Abschnitt eingehen werde.¹⁰

9.1 Entwicklungsprozess

Um Planungssicherheit zu haben benötigt man einen kontinuierlichen Prozess, damit man weiß, an wen welche Aufgaben zu verteilen sind und jeder Projektteilnehmer den Vorgangsablauf kennt. In unserem Fall war die Aufgabenverteilung bereits durch die T-Systems MMS anhand der Rollen Projektmanager, Konzepter, Berater und Entwickler vorgegeben, sodass wir uns auch versuchten, daran zu orientieren.

Die Feingliederung unserer Arbeitsprozesse war wichtig für die Qualität der Entwicklung. Die effektive Arbeitsteilung half uns, sich auf den jeweiligen Bereich fokussieren zu können und mit entsprechender Kenntnis voranzuschreiten zu können. Auch wenn die Umsetzung der Prozesse nicht immer problemfrei ablief, so haben wir es doch geschafft, unsere beiden Hauptfeatures Finder und Self-Service nach unseren Wünschen umsetzen zu können.

9.2 Qualität des Designs

Damit sichergestellt ist, dass wir ein konsistentes, qualitatives Design haben, versuchten wir uns grundlegend an den iOS7-Design-Richtlinien zu orientieren.¹¹ Da leider kein Mitglied von uns weitreichende Erfahrungen mit Design-Werkzeugen hatte, war die Umsetzung dieser zunächst schwierig. Insbesondere mangelte es noch am nötigen Feingefühl, im Sinne eines konsistenten Interaktionsdesigns vorzugehen. Zum Abschluss des Projekts wurden wir hier jedoch sicherer und experimentierten zunehmend mit eigenen Entwürfen.

¹⁰PROJEKT MANAGEMENT INSTITUTE: *A Guide to the Project Management Body of Knowledge*. Fifth Edition. New-ton Square, 2008. 589 Seiten.

¹¹<https://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/>

9.3 Umgang mit Feedback

Ein effektives Instrument zur Qualitätssicherung, dass uns im Projekt durch die Veranstalter zur Verfügung gestellt wurde, ist das Feedback von potentiellen Kunden. Umso mehr ist es für uns daher eine große Hilfe gewesen, dass auf Probleme und Wünsche bei der App in wöchentlichen Plenarveranstaltungen eingegangen wurde. Wir haben jedes Feedback umgehend in unserer Facebook-Gruppe dokumentiert und Lösungen ausdiskutiert.

Dieser Vorgang hat uns sehr in der Einhaltung der oben geschilderten Prozesse geholfen. Da wir uns jeden Dienstag und Donnerstag im Mac-Arbeitsraum getroffen haben, konnten wir erst neue Stories in PivotalTracker anhand des Feedbacks anlegen, diese dann unter uns aufteilen und bis zur nächsten Plenarveranstaltung abarbeiten. Man kann sagen, dass wir also die Plenarveranstaltung als „Motor“ für unsere Scrum-Iterationen benutzt haben.

Wir entwickelten somit über die Zeit ein immer besseres Gefühl für die Umsetzung von Feedback und auch für die Möglichkeit, Scrum als Methode des agilen Projektmanagements anzuwenden.

Zentral war dabei auch die Erkenntnis, dass eine Produktidee schon scheitern kann, wenn es uns als Entwicklern nicht gelingt, bestimmte Zusammenhänge nachvollziehbar zu machen. Insbesondere das Zusammenspiel zwischen Authentifizierung und erweiterter Funktionalität im Self-Service war bis zuletzt problematisch. Warum muss ein Kunde für die Authentifizierung speziell in die Bank gehen? Warum ist in die Self-Service-Station ein Kontoauszugsdrucker angebunden?

Hier wurde deutlich, dass ein Entwickler in derartigen Situationen zu einem gewissen Selbstmarketing fähig sein muss, dass eine Idee, deren Nutzen nicht prägnant wiedergegeben werden kann, schon an sich ein Problem haben könnte.

9.4 Ausblick

Über die vorangegangen Punkte hinaus haben wir uns außerdem überlegt, wie wir auch entwicklungstechnischer Sicht Qualitätssicherung umsetzen können. Unser nächster Schritt in der Entwicklung wäre daher auch das Einbinden von Unitests gewesen, um auch in Hinblick auf die

oben beschriebene Architektur eine Absicherung über die Prozesse geben.

Programmausdruck 2 zeigt ein Beispiel für so einen Unitest, wie er die Arbeitsweise der Dependency Injection verifizieren kann. Er demonstriert, wie beispielsweise das Einbinden der Authentifizierung getestet werden kann.

Programmausdruck 2: Beispiel für einen Unitest mit iOS

```
@implementation KibaTests

/**
 * Diese Methode wird vor jedem Test ausgeführt.
 */
- (void)setUp
{
    [super setUp];
    // Bootstrap der App
    [KBABootstrap bootstrap];
}

/**
 * Diese Methode wird nach jedem Test ausgeführt.
 */
- (void)tearDown
{
    [super tearDown];
}

/**
 * Testet, ob die Abhängigkeit für die
 * Authentifizierung korrekt eingebunden wurde.
 */
- (void)testAuthDependency
{
    // Ist die Abhängigkeit vorhanden?
    XCTAssert([KBAInjector hasDependency:@auth],
              @"Auth dependency is missing!");

    // Handelt es sich auch um ein KBAAuth-Objekt?
    id auth = [KBAInjector getByKey:@auth];
    XCTAssert([auth isKindOfClass:[KBAAuth class]],
              @"Auth dependency is not a valid Auth object!");
}

@end
```

10 Fazit von Michael Schaarschmidt

Das Projekt objektorientierte Softwareentwicklung hat für uns in gewisser Hinsicht eine Zäsur im Studium markiert. Bisherige Module oder Praktika legten den Fokus auf eine technisch möglichst saubere Implementierung, bei der im Zweifel das Interaktionsdesign oder der Funktionsumfang zurückgestellt wurden. Im Projekt wurde die Beherrschung objektorientierter Techniken bereits vorausgesetzt und erstmalig ein Software-Produkt entwickelt, das in seiner Gesamtheit auch industriellen Anforderungen genügen musste.

Mehrere Teilnehmer unserer Gruppe hatten bereits ein erstes Maß an Arbeitserfahrung vorzuweisen, etwa als Werkstudent oder selbständiger Webentwickler. Trotzdem wurde in den ersten Wochen des Projekts deutlich, dass im Bezug auf die Arbeitsorganisation als Auftragnehmer eines Produkts noch Raum für Verbesserungen war.

Dies galt insbesondere für den Umgang mit Designfragen und der Reihenfolge der Entwicklung. In einer Gruppe, die bis auf eine Ausnahme aus Informatikstudenten bestand, gingen wir zunächst in bekannter Manier vor. Um die Eigenheiten von iOS und Objective-C kennenzulernen, unternahmen wir erste Gehversuche in Funktionen wie einer Überweisungsansicht, die für das Endprodukt sekundär waren. Ebenso behandelten wir in den ersten Überlegungen das Design noch etwas stiefmütterlich. Eine Mentalität, der im Informatikstudium nicht selten Vorschub geleistet wird, ließe sich auf die Aussage „Erst wird programmiert, dann das Design nachgeschoben“ reduzieren. Im Plenum wurde uns aufgezeigt, dass ein solcher Ansatz nicht nur weniger effizient ist, sondern auch das eigene Produkt schlechter dastehen lässt.

Einerseits ist es in einem engen Zeitrahmen sinnvoller, sich voll und ganz auf die Funktionen zu konzentrieren, die Alleinstellungsmerkmale darstellen und den innovativen Kern des Konzepts bilden. Denn nur anhand dieser lässt sich ein Kunde von einem Produkt überzeugen. Zudem erfordert ein benutzerzentriertes Design, das erst nachträglich hinzugefügt wird, umständliche Änderungen an der Codebasis. Gleichzeitig können bereits wenige, minimalistische Designelemente in konsistenter Farbe und Anordnung einen enormen Unterschied in der Wahrnehmung der Produktqualität bedeuten.

Entsprechend haben wir unsere Arbeitsweise adjustiert und ein neues Verständnis von zeitgemäßer Softwareentwicklung gewonnen. Während innovative Funktionalität von technischer

Seite weiter geboten ist, hat sich der Anspruch der Benutzer an die Bedienbarkeit stark gewandelt; Apps, die dies nicht berücksichtigen, haben auf einem derart umkämpften Markt wenig Erfolgsaussicht.

Abkürzungsverzeichnis

DAO Data Access Object

REST Representational State Transfer

SOAP Simple Object Access Protocol

VCS Version Control System (Versionsverwaltung)

Programmausdrucke

Prog. 1:	Der Bootstrapping-Vorgang	38
Prog. 2:	Beispiel für einen Unitest mit iOS	46