Dokumentation

Projektarbeit im Modul

Entwicklungsprojekt interaktiver Systeme

im Wintersemester 2016/2017

Niklas Reinhardt

Maximilian Krusch

Projektbetreuer:	Prof. Dr. Kristian Fischer Prof. Dr. Gerhard Hartmann Ngoc-Anh Dang Daniela Reschke
Abgabedatum:	05.12.2016

Inhaltsverzeichnis

1. Einleitung

Im Rahmen des Moduls „Entwicklung Interaktiver Systeme“, geht es um die Konzipierung und Umsetzung einer verteilten, multimedialen Anwendung unter Verwendung von Grundlagen der in zuvor erlernten Modulen des Studiums zu Verwenden. Es sollen Methoden und Techniken der Module „Web-basierte Anwendungen 2: Verteilte Systeme“ und „Mensch Computer Interaktion“ selbständig geplant und durchgeführt werden. Nach einer Ausarbeitung eines Konzepts, eines Rapid Prototype und eines Projektplans innerhalb der ersten Projektphase, befasst sich diese Dokumentation mit den weiterführenden Projektarbeiten. Ein großes Augenmerk sollte hier auf Inhalte der „Mensch Computer Interaktion“ und „Web-basierte Anwendungen 2: Verteilte Systeme“ geworfen werden und die Ergebnisse in dieser Dokumentation festgehalten werden.
Die Dokumentation ist in mehrere Teile strukturiert. Nach der allgemeinen Einführung folgt als erster Hauptbestandteil die Prozessdokumentation, die den zeitlichen Ablauf des Projektes fokussiert. Darin wird die Entwicklung des Projektes nach Konzept Abgabe, einzelne Arbeitsschritte und angewandte MCI Methoden mit den entstandenen Ergebnissen beschrieben. Es folgt die Systemdokumentation, welche die Funktionen des entwickelten Systems erklärt und Alternativen und Abwägungen, auf die Augenmerk geworfen werden kann. Welche Architektur gewählt wurde und warum man sich dafür entschieden hat. Welche Protokolle in unserem Verteilten System benutzt wurden, warum wir uns dafür entschieden haben und welche Aufgabe sie innerhalb des Verteilten System haben. Eine genaue Erklärung der Techniken die für dieses Projekt gewählt wurden und warum wir uns für diese entschieden haben. Am Ende geben wir noch ein Fazit und eine Projektaussicht, in der erklärt wird, was von den geplanten Funktionen schon Implementiert wurden und welche noch Implementiert werden.

1.1 Zielsetzung

Ziel ist eine Application zur Übermittlung personalisierter Werbung zu entwickeln. Dabei steht vor allem die Infrastruktur hinter der Anwendung im Mittelpunkt. Der User soll beim Einrichten der Application zuerst einen kurzen Fragebogen zu Kaufgewohnheiten und Vorlieben ausfüllen. Anhand dieser Daten kriegt der User fortan Empfehlungen zu aktuellen Angeboten. Diese Angebote können entweder von der Application mitgeteilt werden oder initiativ vom User abgefragt werden. Die Application bietet Unternehmen eine Schnittstelle, um ihre Sonderangebote zu pflegen und zu aktualisieren. Dabei setzen wir vor allem auf Supermärkte, aber prinzipiell lässt sich die Idee auf den gesamten Einzelhandel und E-Commerce anwenden. Die Datenhaltung über die User ist dabei strikt von teilnehmenden Unternehmen getrennt. Die Werbung wird so gesehen über die Application "vermittelt". Dadurch soll einerseits die Privatsphäre der User geschützt und eine höhere Akzeptanz der Werbung erreicht werden.

1.2 Rahmenbedingung

1.2.1 Funktional

Obwohl dies ein selbstständiges Projekt ist, sind bestimmte Rahmenbedingungen vorhanden, die wir der Vollständigkeit halber nennen wollen. Eine Voraussetzung ist unter anderem, dass ein verteiltes System entwickelt wird. Programmiersprache sollte Javaskript oder Java sein. Die Website darf keine Anwendungslogik beinhalten. Natürlich legt der Modulname fest, dass es sich um ein interaktives, also von Benutzeraktionen getriebenes System sein soll. Das System soll nach möglich ein Problem der Wirtschaft beheben.

1.2.2 Organisatorisch

Der Anteil an der Entwicklung soll pro Teammitglied 50% betragen. Als Nachweis gilt sowohl eine Arbeitsmatrix als auch das commit-log von Github, welches zur Versionierung und Entwicklung des Projekts benutzt wird.

1.3 Zielhierachie

Ziele werden festgehalten um das Handeln des Unternehmens oder des Teams an ihnen auszurichten. Sie helfen der Kommunikation und der Identifizierung mit dem Projekt. Ziele werden meist in Strategisch, Taktisch und Operativ eingeteilt, obwohl auch andere Einteilungsarten möglich sind, wie zum Beispiel anhand des magischen Dreiecks (Kosten, Zeit, Leistung) oder der STEP-Einteilung (Social, Technical, Economical, Political).

1.3.1 Strategisch

Strategische Ziele sind langfristige Ziele von einer Dauer ab 5 Jahren. An ihnen lassen sich alle taktischen und operativen Ziele messen. Meist sind solche Ziele eher abstrakt gehalten und geben eine Richtung vor in die sich das Unternehmen oder das Projekt entwickeln soll. Unsere strategischen Ziele sind:

- Engere Bindung der Kunden zu den Unternehmen
- Reduzierung der Papierwerbung

1.3.2 Taktisch

Taktische Ziele sind mittelfristige Ziele von einem Zeitrahmen von meist ein bis drei Jahren (vereinzelt auch fünf Jahre). Unsere taktischen Ziele sind:

- Kundenakzeptanz für Werbung erhöhen
- Gewinn erwirtschaften
    - Unternehmen akquirieren
    - Marketing

1.3.3 Operativ

Operative Ziele sind kurzfristige Ziele innerhalb eines Jahres. Sie sind quasi die Meilensteine des Unternehmens an denen man sich im Tagesgeschäft orientiert.
Unsere operativen Ziele sind:

- System fertig stellen und Anforderungen erfüllen
    - Planung
        - Domänenrecherche
        - Anforderungen festlegen
    - Useranalyse
        - Interviews und Personas
        - Use Cases anlegen
    - Design
        - Mockups und Konzepte anfertigen
    - Entwicklung
        - Sprintziele erreichen
    - Tests
        - Use Case testen

1.4 Marktrecherche

Im Zuge unserer Recherche konnten kauf.Da, MeinProspekt, Marktjagd und MarktGuru als Konkurrenzprodukte ausfindig gemacht werden. Alle Apps wurden heruntergeladen und besichtigt. Ihre Features werden im Folgenden erläutert.

1.4.1 kauf.Da und MeinProspekt

Beim Öffnen fällt auf, dass beide Apps identisch aussehen. Dieser Sachverhalt kann schnell aufgeklärt werden, denn 2014 wurde MeinProspekt von kauf.Da übernommen. Deswegen werden wir nur noch auf kauf.Da eingehen. Beim initialen Öffnen der App startet ein kurzer Monolog von fünf Slides mit jeweils einem Satz und Piktogram, der die App kurz erläutert und in dessen Verlauf man seinen Standort angeben muss. Positiv fällt auf, dass man mitverfolgen kann auf welcher Seite man ist. Sollte man die Berechtigung des Standort-Trackings verweigern, so wird man gefragt den Standort manuell einzugeben. Gibt man zum Beispiel nur Köln ein bleibt es auch bei Köln. MarktGuru bekommt das besser hin. Danach kann man sich seine Interessen zusammenklicken. Auffällig ist, dass hier u.a. Lidl und Rewe fehlen. Es fehlt außerdem der Hinweis, dass man mindestens einen der Läden anklicken muss um fortzufahren. Danach landet man auf der Hauptansicht. Nun kann man auf eines der Prospekte klicken die ganze Darstellung ist sehr unübersichtlich und uneinheitlich. Die Prospekte sind nicht responsive oder anderweitig Angepasst und nutzen so nicht immer den Bildschirm optimal aus. Weiterhin wird aus der Prospektübersicht nicht ersichtlich welches Prospekt interaktiv sind (eingefügte Hyperlinks zum Artikel auf der Unternehmenswebsite) und welche nicht. Eine Einkaufsliste ist zwar vorhanden, kann aber nicht mit Angeboten aus den Prospekten befüllt werden. Sie ist eher eine sehr umständliche Suchfunktion.

1.4.2 MarktGuru

Beim erstmaligen Einloggen in die Marktguru-App wird man nur nach seinem Standort gefragt. Danach landet man auf dem Startbildschirm der um einiges aufgeräumter wirkt als von kauf.Da. Alle Funktionen lassen sich von hier aus erreichen, einzig die Suchmaske in der oberen Leiste könnte deutliche sein. Auch MarktGuru bietet interaktive Prospekte. Allerdings kann man nicht erkennen welche Prospekte das sind und man muss es erst durch Gedrückthalten eines Angebots ausprobieren. Wenn es aber klappt, kann man den Artikel gleich in die Merkliste ziehen. Diese ist nach Unternehmen geordnet und zeigt auf Wunsch die nächstgelegenen Filialen an. Auch hier kann man nicht mehr als eine Merkliste haben.
Alles in allem lässt sich sagen, dass es schwerer wird MarktGuru zu übertreffen, es aber auch aufgrund unseres USPs möglich ist, sich weit genug abzugrenzen.

1.4.3 Marktjagd

Auch bei dieser App wird man nach seinem Standort gefragt. Das Auswahlmenü dieser App ist jedoch das überladenste. Die App bietet die Möglichkeit an auf die letzten Standpunkte erneut zuzugreifen. Die App selber bietet Benachrichtigungen bei neuen Prospekten und unzählige Kategorien. Allerdings bietet sie keine interaktiven Prospekte oder Merklisten für einzelne Artikel. Die App ist dementsprechend etwas grob, auch, weil man sich von dem Überangebot an Kategorien leicht erschlagen fühlt.

1.5 Usability Engineering

1.5.1 Nutzungskontext

Nutzungskontexte sind dafür da die Nutzung der Anwender zu modellieren und helfen dadurch die Erfordernisse festzuhalten und stellt damit den ersten Schritt der Implementierung dar. In unserem Fall kann der Benutzer Jeder sein der seinen Einkauf plant oder überhaupt Einkaufen geht, da auch nicht-Planen eine Form von Vorbereitung ist.
Momentan stehen den Einkaufenden gedruckte Prospekte, digitale Prospekte (Konkurrenz-Apps), und die Websites der einzelnen Läden zur Verfügung. Eine Anmerkung ist noch zusätzlich wichtig: die Effizienz mit der die Aufgabe erfüllt wird ist von Nutzer zu Nutzer sehr unterschiedlich und kommt auf die jeweiligen Prioritäten an. So kann für den Einen Nutzer die Kostenersparnis den Ausschlag geben und für den anderen die Zeitersparnis.

Nutzungskontext des Projekts

Benutzer	Einkaufende/r
Arbeitsaufgabe	Einkauf planen (Kriterien: Geld, Zeit, Bequemlichkeit)
Ausrüstung	Prospekt, Internet
Soziales Umfeld	Wohngemeinschaft, deren Vorräte aufgefüllt werden müssen
Physisches Umfeld	Wohnung, Einkaufsladen

1.5.2 Erfordernisse

Erfordernisse sind notwendige Voraussetzungen, die es ermöglichen, den in einem Sachverhalt des Nutzungskontextes enthaltenen Zwecks effizient zu erfüllen. [1]
In unserem Kontext konnten wir folgende Erfordernisse, welche das Minimum der zu erfüllenden Funktionen abdecken, identifizieren:

Erfordernisse

1	Der Einkaufende muss Informationen über Angebote aller für Ihn in Frage kommenden Läden zur Verfügung haben, um im für ihn gewünschten Kosten-, Bequemlichkeitsrahmen die Kosten möglichst gering zu halten.
2	Der Einkaufende muss die Läden in seiner Nähe und ihre Entfernung kennen, um die passenden Geschäfte zu finden.

1.5.3 Nutzungsanforderungen

Nutzungsanforderungen legen fest welche Aktionen der Nutzer am System durchführen können muss. Drei Arten von Aktionen sind dabei bei interaktiven Systemen erlaubt:

- etwas eingeben
- etwas erkennen
- etwas auswählen

In unserem Kontext sehen die Nutzungsanforderungen folgendermaßen aus:

[1] DAkkS „Leitfaden Usability“, unter http://www.dakks.de/content/leitfaden-usability (abgerufen am 5.12.16)

Nutzungsanforderungen

11	Der Nutzer muss **auswählen** können von welchen Läden er Angebote sehen möchte.
12	Der Nutzer muss **erkennen** können, wie sehr die Angebote reduziert sind.
13	Der Nutzer muss **erkennen** können, ob die Angebote noch aktuell sind.
21	Der Nutzer muss seinen Standort **eingeben** können.
22	Der Nutzer muss **erkennen** können, welche Laden in der Nähe sind.
23	Der Nutzer muss die Entfernung der in der Nähe befindlichen Läden **erkennen** können.

1.5.4 Systemanforderungen

Systemanforderungen legen fest welche Aktionen das System ausführen und unterstützen muss. Sie lassen sich meist leicht aus den Nutzungsanforderungen ableiten.

Systemanforderungen

111	Das System muss alle vorhandenen Läden anzeigen können.
112	Das System muss dem Nutzer die Möglichkeit bieten, einzelne Läden oder Kategorien an- und abzuwählen.
121	Das System muss sowohl den Original- als auch den reduzierten Preis anzeigen.
131	Das System muss das aktuelle Datum und das Ablaufdatum des Angebots kennen und mindestens Zweiteres auch anzeigen.
211	Das System muss dem Nutzer die Möglichkeiten bieten, seinen Standort einzugeben.
221	Das System muss in der Lage sein, die Adressen der Läden und die des Nutzers zu verarbeiten.
222	Das System muss die einzelnen, in der Nähe befindlichen Läden visualisieren können.
231	Das System muss die Entfernung der Läden zum Nutzer anzeigen können.

1.6 Risikoanalyse

Kunden/User bekommen Daten nicht

Es kann vorkommen, dass Kunden/User durch Technische Probleme keine Daten bekommen und somit die Werbung der Unternehmen den Endbenutzer (Kunden/User) nicht erreichen. Das könnte die Kunden/User zu einem Abgang von unserem System veranlassen. Eine Testphase kurz vorm Ende des Projektes sollte dieses Risiko jedoch minimieren können.

Aufwand für die Umsetzung

Da die Erfahrung in der Umsetzung von mobilen Applikationen im Team gering ist, kann die Aufwandschätzung zu gering ausfallen und so ein Risiko darstellen. Das Risiko kann durch gutes Zeitmanagement verringert werden.

Ablehnung der Anwendung der Unternehmen

Die Ablehnung der Unternehmen wurde von uns ebenfalls als Risiko identifiziert. Es liegt im Bereich des Möglichen, dass die Unternehmen bei unserem System nicht beteiligt sein wollen, da Sie keine Kundendaten bekommen um diese für Markt Studienzwecken zu benutzen. Dies könnte man mit Transparenten Gesprächen mit den Unternehmen klären und das Risiko minimieren. Eine eigene Art der Marktstudie die von uns Entwickelt wird und die Daten anonym an die Unternehmen weitergibt, könnte dieses Risiko auch minimieren.

Datensicherheit

Da Kunden sich bei unserem System Registrieren müssen und Daten wie zum Beispiel ihre Postleitzahl eingeben müssen, unterliegen diese Daten einer besonderen Sorgfalt. Deshalb ist es wichtig Maßnahmen zur Sicherheit dieser Daten zu gewährleisten. Wird die Sicherheit der Daten nicht gewährleistet entsteht ein Risiko des Missbrauchs der Daten durch eventuelle Dritte. Das Risiko kann durch Einhalten von Sicherheitsstandards, einer Regelmäßigen Überprüfung minimiert werden.

Datenbankausfall

Ein weiteres Risiko stellt der Ausfall einer Datenbank dar. Fällt diese aus können sich Kunden/User entweder nicht einloggen oder Werbedaten können nicht an den Kunden/User geschickt werden. Dieses Risiko kann mit einem Cluster-Datenbanksystem minimiert werden. Auf diesem Cluster-Datenbanksystem sind die Daten auf mehrere Datenbanken verteilt und so kann bei einem Ausfall auf eine Ausweich Datenbank zugegriffen werden.

Datenverlust

Durch unzureichende Sicherung des Programmier-Standes, kann es dazu kommen, dass zu einem bestimmten Zeitpunkt der aktuelle Projektstand verloren gehen kann. Dies würde zu einem erheblichen Risiko in Sachen Zeitmanagement kommen. Dies ist mit Sicherungen auf mehreren Systemen und in der Cloud zu Kompensieren und sollte, bei häufigen Sichern der Daten nicht vorkommen.

1.7 Bearbeitung der Proof-of-Concepts

Verbindung mit der Datenbank

Der Server muss sich mit der MongoDB verbinden und einfach CRUD-Operationen ausführen können. Die Daten sollen im JSON-Format übermittelt werden.

exit - Es entstehen keine Fehler und die Operationen erzielen das gewünschte Ergebnis. Alle Objekte werden in JSON-Format übermittelt und persistent in der MongoDB gespeichert.

fail - Es entstehen Fehler oder die Objekte werden nicht persistent gespeichert.

fallback - Sollte die Arbeit mit der MongoDB nicht funktionieren so muss die Architektur und die benutzten Produkte überdacht werden. Als alternative zu Node.js

bieten sich Ruby oder PHP an. MongoDB ließe sich eventuell durch MySQL oder Redis austauschen.

Zusammenstellung der Push-Nachrichten

Der Server stellt dem Nutzer eigenständig für ihn interessante Werbung zusammen um sie zum gewünschten Zeitpunkt zu verschicken.

exit - Es entstehen keine Fehler. Der Server kann sowohl auf die Nutzer- als auch auf die Angebotsdatenbank zugreifen und die Nachricht zusammensetzen.
fail - Die zu sendende Nachricht ist fehlerhaft oder wird zur falschen Uhrzeit abgeschickt.

fallback - Sollte das Problem an den vom Server erhaltenen JSON-Daten liegen, so muss die Anwendungslogik des Datenbankbereichs überprüft werden und gegebenenfalls zurück zu “Verbindung mit der Datenbank” iteriert werden. bei anderen Problemen ist lediglich die Anwendungslogik des Servers zu Korrigieren.

Eintragen der Angebote

Die Angebote werden über ein Formular an die Datenbank versendet und sind dann abrufbereit.

exit - Das Übersenden der Daten erzeugt keine Fehler und auch das erneute Auslesen des Angebots verläuft fehlerfei.

fail - Es entstehen Fehler oder die Objekte werden nicht persistent gespeichert oder dargestellt.

fallback - Die Präsentationslogik der HTML-Seite müssen korrigiert werden.

2. MCI

2.1 Einleitung

Bevor im zweiten Teil die Entwicklung vor allem aus WBA-Sicht besprochen wird, folgt zuvor eine Ansicht der Teile, die sich eher dem User Experience Design, kurz UXD zuordnen lassen. Doch wozu eigentlich UX-Engineering? Jahrzehntelang kam man zwar auch gut ohne gesonderten Blick auf den User aus, aber alleine Apple zeigte mit seinem Macintosh, aber auch mit seinen IPods und IPhones, wie wichtig gutes Design ist und dass gute Backend-arbeit allein nicht erfolgreich sein kann. Deshalb nannte schon Richard Monson-Haefe in seinem Buch “97 Things Every Software Architect Should Know” eines der Kapitel: „for the end-user, the interface is the system“ [2]. Diese Erkenntnis gewinnt in den letzten Jahren nur noch an Fahrt. Deshalb wollen wir zu Beginn einige der Paradigmen des UX-Engineerings vorstellen und evaluieren.

2.2 Vorgehensmodell

2.2.1 Gegenüberstellung

2.2.1.1 User Centered Design

User Centered Design, kurz UCD, ist ein Vorgehensparadigma, bei dem der User an erster Stelle steht. Die Software orientiert sich an den Fähigkeiten und Bedürfnissen des Endanwenders. UCD ist interdisziplinär und beruht auf Erkenntnissen aus Design, Softwareentwicklung aber auch Psychologie. Am Ziel des Prozesses steht ein leicht erlernbares, intuitives Produkt. Wichtig für dieses Vorgehen ist der Namensgebende User, sprich Endanwender. Es ist nötig eben diesen genau zu kennen, was er will aber auch was er kann. Nur so kann im Endeffekt eine gute Lösung für den Endanwender erreicht werden. Im Mittelpunkt des Vorgehens stehen die drei Bereiche Analyse, Design und Evaluation. Die Analyse dreht sich wie vorher besprochen um den Endanwender. Anhand der in der Analyse gewonnenen Erkenntnisse wird in der Designphase das Produkt designed. Es werden dabei viele unabhängige Mock-Ups

[2] Monson-Haefel, „97 Things Every Software Architect Should Know” unter http://shop.oreilly.com/product/9780596522704.do

angefertigt und evaluiert um dann das beste Design fortzuführen oder verschiedene Ansätze zu kombinieren. Das Besondere ist, dass die Evaluation so früh im Projektablauf stattfindet. Das verhindert, dass Denkfehler oder ähnliches erst zu spät entdeckt werden, was die Entwicklungskosten in die Höhe treiben würde. Ein weiterer Vorteil ist die breite Anerkennung des User Centered Design. Es gibt genügend digitale und analoge Quellen die Hilfestellung bei Fragen bieten.

2.2.1.2 Usage Centered Design

Das von Lockwood und Constantine entwickelte Usage Centered Design lässt sich gut mit dem User Centered Design vergleichen, unterscheidet sich jedoch in einem wichtigen Punkt. Es geht nicht primär um den Endanwender sonder um seine Interaktion mit dem System. Usage Centered Design hat weiterhin das Problem, dass es gemeinhin weniger benutzt wird. So ist es um einiges schwerer an Literatur zu kommen, die Veröffentlichungen von Lockwood und Constantine außen vor gelassen. Diese beschreiben zwar auch das Vorgehen, können sich aber nicht mit der Vielfalt an Unterlagen zum UCD messen.

2.2.1.3 Discount Usability Engineering

Dieses Vorgehensmodell wurde erstmals 1997 von Jakob Nielsen beschrieben.[3] Es geht um eine günstige Alternative zu den anderen Vorgehensmodellen, welche oftmals zu Teuer und Aufwendig für ungeübte Unternehmen sind. Dem liegt das Motto “Lieber schlechtes Usability Engineering, als gar keines” zu Grunde. Das ganze Vorgehensmodell lässt sich als eine große Heuristik beschreiben. Es baut vor allem auf der Expertenevaluation (siehe Evaluationsmethoden) auf. Wie in Abbildung 1 zu erkennen ist, ist die Funktion der gefundenen Fehler abhängig von den eingesetzten Experten asymptotisch zur 100% Marke. Die größten Fehler lassen sich laut Nielsen

[3] Jakob Nielsen: “Discount Usability for the Web” unter https://www.nngroup.com/articles/web-discount-usability/ (abgerufen am 5.12.16)

schon mit relativ Wenig Aufwand entdecken, während die UX-Perfektion des Systems nur mit übertriebenen Mitteln erreicht werden kann.

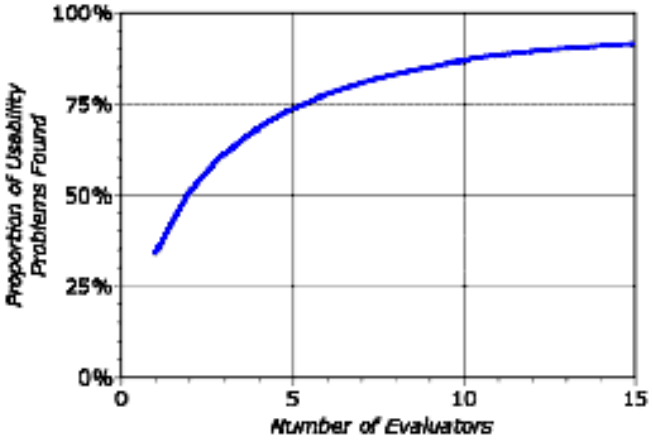

Abbildung 1: Expertenfunktion, Auf der Y-Achse wird der Anteil der im System gefundenen Fehler, auf der X-Achse Anzahl der UX-Experten dargestellt

2.2.1.4 Fazit

Trotz anfänglichem Interesse für das Usage Centered Design, haben wir uns nicht zuletzt wegen der mannigfaltigen Auswahl an Lektüre für das UCD entschieden. Diese erleichtert uns das Einarbeiten in die MCI-Schwerpunkte ungemein. Denn trotz Grundwissen aus den MCI-Vorlesungen, sind wir noch ungeübt bezüglich der Umsetzung. Durch das UCD können wir unsere Ressourcen für das weitere Projekt einfach besser verteilen. Obwohl wir der Meinung sind, dass das Discount Usability Problem vor allem kleinen Projekten und Gruppen enorm helfen kann, ist es nicht das Richtige für unser Projekt, da Usability Engineering schon von den Aufgaben her viel zu tief verwurzelt ist, als es nur oberflächlich abhaken zu können.

2.3 Analyse

Die Analyse ist der Erste der drei Bereiche des UCD. Es geht darum Wissensgrundlagen über die User für das spätere Design zu schaffen. Im Folgenden sollen einige mögliche Vorgehensweisen vorgestellt werden und unser eigenes Vorgehen beschrieben werden. Natürlich muss angemerkt werden, dass es noch unzählige weitere Methoden gibt. Dies soll nur eine Auswahl von Standardmethoden sein, die sehr häufig eingesetzt werden

2.4 Methoden

2.4.1 Fragebögen

Fragebögen haben das Ziel quantitative und repräsentative Aussagen zu ermitteln. Dafür braucht man eine ausreichend große Gruppe. Man kann entweder eigene Umfragebögen aufsetzen oder auch auf vorgefertigte zurückgreifen.
Vorteile:

- hohe Reichweite durch Onlineumfragen
- man gewinnt statistisch vergleichbare Daten
- man gewinnt repräsentative Daten

Nachteile:

- müssen methodisch korrekt sein um verwertbare Daten zu erhalten
- keine Nachfragen möglich
- braucht viele Nutzer
- Menschen tun und denken nicht immer das, was sie bei einer Befragung angeben
- fallen durch sozialen Druck oft zu Positiv oder zu Negativ aus[4]

2.4.2 Expertenevaluation

Expertenevaluation, auch heuristische Evaluation genannt, hat das Ziel Designschwächen ausfindig zu machen. Dafür werden drei bis fünf Experten eingeladen, die sich dann das System angucken und beurteilen. Dafür können zum Beispiel Nielsens 10 Heuristiken herangezogen werden.[5] Diese Faustregeln erlauben eine schnelle Suche nach offensichtlichen Designmängeln.
Vorteile:

- Benötigt keine User für den Test, ist also geeignet, wenn das Team keine Nutzer für Tests akquirieren kann

Nachteile:

- Experten können sich nicht immer in die Zielgruppe hineinversetzen
- findet nicht zwangsläufig alle Designschwächen

---

[4] UIG e.V. „Fragebogen“ unter http://www.usability-in-germany.de/definition/fragebogen
[5] Jakob Nielsen, „10 Usability Heuristics for User Interface Design” unter https://www.nngroup.com/articles/ten-usability-heuristics/

2.4.2 Personas

Eine Persona ist eine Zusammenfassung von mehreren Usern in eine Gruppe mit ähnlichen Eigenschaften. Das dient einerseits dazu die Nutzer eindeutig zu charakterisieren und somit die Bedürfnisse und Anforderungen zu konkretisieren. Andererseits dienen Personas der besseren Kommunikation, da man statt langer Gruppenbezeichnungen wie “alleinstehende Mütter über 30” einfach die Persona “Molly” ansprechen kann. Man fokussiert quasi die gesamte Gruppe auf eine einzige Persona.

Vorteile:

- fördern das Verständnis der spezifischen Bedürfnisse und Anforderungen der User
- hilft die Anzahl der verschiedenen Nutzergruppen auf einzelne Personas zu reduzieren

Nachteile:

- sind alleine ungeeignet als User-Analyse
- des besteht die Gefahr, dass die Entwickler ihre eigenen Stereotypen Vorstellungen mit einfließen lassen

2.4.3 Individuelle Interviews

Interviews erlauben ausführliche Gespräche mit einzelnen Personen. Sie werden meist als Ergänzung zu Umfragen genutzt, da sie genaueres Nachfragen erlauben und so unklare Sachverhalte klären können oder Nutzeraussagen präzisieren können.

Meist werden Interviews auch vor Umfragen durchgeführt um Themenfelder oder Probleme ausfindig zu machen die man dann in Umfragen besser ermitteln kann, indem man die Fragen nochmal an eben Jene anpasst.

Vorteil:

- Erlaubt es qualitative Aussagen über User-Sachverhalte zu treffen
- man kann Unklarheiten klären

Nachteil:

- man gewinnt keine repräsentativen Aussagen
- Interviewer kann durch tendenziöse Fragen das Ergebnis verfälschen
- aufwendiger pro Nutzer als Umfragen

2.4.4 Durchführung

Am Anfang unserer Überlegung stand der Entschluss auf eine aufwendige Umfrage verzichten zu wollen. Wir haben selber kaum Erfahrung im Umgang mit Umfragen und so würde das Planen, Erstellen, Durchführen und Auswerten ungeheure Mengen an Zeit verschlingen ohne garantieren zu können, dass im Nachhinein auch brauchbare Ergebnisse vorliegen. Wir erachteten es daher als sinnvoll, Umfragen zu vermeiden und mit Alternativen zu arbeiten.
Wir entschieden uns daraufhin auf persönliche Interviews auszuweichen, die uns immerhin qualitative Aussagen liefern. Zwar würde dabei sehr wahrscheinlich die Repräsentativität auf der Strecke bleiben, aber die wäre wie bereits erläutert auch bei Umfragen nicht garantiert gewesen.
Die Interviews verliefen dabei reibungslos und wir stellten fest, dass Interviews mit Bekannten oder Freunden zu Teilen leichter zu führen, weil sie bereitwilliger über ihre privaten Gewohnheiten sprachen als Fremde. Wir haben als Vorbereitung für die Interviews eine Liste von Fragen erstellt an der wir uns während des Interviews orientieren konnten. Gleichzeitig wurden in diesen Listen die entsprechenden Antworten festgehalten.
Beim Betrachten der Ergebnisse bestätigt sich was wir schon länger aufgrund eigener Erfahrungen vermuteten. Prospekte kommen bei den jüngeren Generationen nur sehr begrenzt an und landen meistens im Müll. Weiterhin richtet nur ein kleiner Teil seine Einkäufe nach Sonderangeboten aus und kauft vielmehr in seinen Lieblingsläden ein. Auch scheint der Onlinehandel eine kleinere Rolle zu spielen als wir von vornherein erwartet hätten. Die Generation, die am häufigsten in den Interviews vorkommt, zählt bereits zu den Digital Natives. Diese verbringen so viel Zeit am Smartphone, dass davon ausgegangen werden kann, dass sie alle die Standard Interaktions-Pattern vollkommen verinnerlicht haben.
Das könnte bei älteren Generationen nur begrenzt der Fall sein, obwohl wir auch dort vor allem erfahrenere Smartphoneuser angetroffen haben.
Weiterhin überlegten wir uns, die Ergebnisse der Interviews in Personas zu transferieren. Das würde es erlauben, ausgewerteten Erkenntnisse fokussiert auf einzelne Personas zur Verfügung zu stellen und so besser mit ihnen zu arbeiten. Das klappte ausgezeichnet und erlaubte uns Designentscheidungen viel schneller zu

evaluieren, da man nur noch die begrenzten Personas durchgehen muss. Zu der Art und Weise wie die Personas erstellt werden lässt sich relativ wenig sagen. Wir haben die Daten der Interviews nach übereinstimmenden Aussagen durchsucht, und diese dann in einzelne Personas zusammengefasst. Dabei achteten wir auch darauf, dass alle vorher definierten Zielgruppen berücksichtigt wurden. Später ermöglichte uns das sogar eine bessere Priorisierung, da viel ersichtlicher wurde, welche Persona wie sehr von unserem System profitiert und was noch ergänzt werden müsste.

2.4.3 Fazit / Kritik

Alles in allem sind wir durchaus zufrieden mit der Analyse und sind zuversichtlich, damit ein befriedigendes Design für unsere User entwerfen zu können. Allerdings wünschen wir uns, dass wir uns doch getraut hätten, eine Umfrage zu starten. Einfach weil uns diese Erfahrung nun gewissermaßen fehlt. Hier hätten wir also mehr Mut für Rückschläge haben sollen.
Die Personas empfanden wir als große Hilfe über den gesamten Entwicklungszeitraum hinweg und werden auch in zukünftigen Projekten auf jeden Fall davon Gebrauch machen.

2.5 Interaktionsparadigmen

Interaktionsparadigmen beschreiben Möglichkeiten, wie Nutzer mit dem System interagieren können. Dabei können sie unter verschiedenen Gesichtspunkten quantifiziert werden. Dazu gehören Erlernbarkeit oder auch Belastung des Lang- und Kurzzeitgedächtnisses (LZG und KZG).[6]

[6] Ilse Schmiedecke „7 Interaktionsparadigmen“ unter http://www.schmiedecke.info/HCI/Folien/HCI-08-Interaktionsparadigmen.pdf

2.5.1 Kommandosystem

Kommandosysteme sind Interaktionsparadigmen, in denen der User die Interaktion über eingegebene Kommandos unternimmt. Für jede Aktion gibt es ein mindestens ein Kommando. Ein Beispiel für Kommandosysteme sind die Command-Shells der Betriebssysteme. Solche Systeme sind in der Regel sehr mächtig was den Umfang der Interaktion angeht. Durch Overloading mit Argumenten und Parametern ermöglichen sie einzelne Aktionen in völlig verschiedene Kontexte zu integrieren. Allerdings sind Kommandosysteme nur sehr schwer zu erlernen und in heutigen Zeiten meist nicht Endanwender gerecht.

Tabelle 1 Quantifizierung von Kommandosystemen

Erlernbarkeit	sehr schwer durch viele verschiedene Befehle
LZG	starke Belastung durch das Auswendiglernen
KZG	geringe Belastung, da die vorherigen Arbeitsschritte und Ergebnisse einsehbar sind
Vorteile	• benutzergeführt • unterstützt Parametrisierung und Makrobefehle • gut protokollierbar
Nachteile	• Befehle müssen sich gemerkt werden • Schreibfehler verhindern ausführung • nach heutigen Standards wenig intuitiv
Anmerkungen	Nachteile für das LZG können durch Hilfestellungen und Default-Parameter abgeschwächt werden

2.5.2 Menümasken

Menümasken sind Interaktionsparadigmen, die Interaktionen vorgeben. Der Anwender sieht welche Parameter er ausfüllen kann und welche Aktionen er momentan ausführen kann. Menümasken können geführt oder Benutzer-navigiert sein. Geführte Menümasken werden auch als Wizards bezeichnet (siehe Installations-Wizards). Wizards sind zwar sehr angenehm zu benutzen, sind aber extrem unflexibel und müssen für jede unterschiedliche Aufgabe neu programmiert werden.

Tabelle 2: Quantifizierung von Menümasken

Erlernbarkeit	sehr leicht
LZG	**Wizard**: sehr gering, weil der Nutzer sich nichts merken müssen und die Felder „on-the-fly“ ausfüllt **Benutzer-navigiert**: mittel bis hoch, da der Nutzer trotz Vorgaben die einzelnen Kontexte und Aktionen finden muss
KZG	**Wizard**: sehr gering, weil der Nutzer sich nichts merken müssen und die Felder „on-the-fly“ ausfüllt **Benutzer-navigiert**: mittel bis hoch, da der Nutzer trotz Vorgaben die einzelnen Kontexte und Aktionen finden muss
Vorteile	• leicht zur Erlernen da die Aktionen vorgegeben sind • richtig programmiert kommt es nur zu sehr wenigen Fehlern
Nachteile	• nicht sehr mächtig oder flexibel
Anmerkungen	Nachteile der Benutzer-navigierten Masken können durch Navigationshilfen, Ergebnis- und Historienvisualisierung gemindert werden

2.5.3 Direkte Manipulation

Bei der direkten Manipulation werden sowohl Arbeitsobjekt als auch Werkzeuge als Metaphern dargestellt. Mit diesen Metaphern können dann Aktionen ausgeführt werden. Solche direkten Manipulationssysteme sind durch die Metaphern in der Regel leicht zu erlernen. Belasten wegen der Visualisierung das Langzeitgedächtnis kaum. Dabei wird immer ein Kompromiss zwischen Mächtigkeit und erlernbarkeit eingegangen. Je mächtiger das System, desto komplexer die Aktionen, desto schwieriger das Programm.

Tabelle 3 Quantifizierung von direkter Manipulation

Erlernbarkeit	leicht
LZG	gering durch Visualisierung
KZG	ähnlich der Benutzer-navigierten Menümaske
Vorteile	• leicht erlernbar • Aktionen intuitiv umkehrbar • Ergebnis unmittelbar sichtbar
Nachteile	• meist keine Komplexen Aktionen möglich
Anmerkungen	Benötigt zwingend eine Undo-Funktion da sonst zu unintuitiv

2.5.4 Hypermedia

Hypermedia steht für ein Paradigma, bei dem Ressourcen durch sogenannte Hyperlinks miteinander verknüpft sind. Das führt auch zu einer kontextuellen Verknüpfung der Informationen. Da man diesen kontextuellen Pfaden einfach folgen kann, belastet das Hypermedia-Paradigma kaum das Langzeitgedächtnis, wo hingegen das KZG ohne entsprechende Hilfen, wie Navigationshilfen o. ä. ziemlich belastet werden kann, da es durchaus passiert, dass man die Orientierung verlieren kann.

Tabelle 4 Quantifizierung von Hypermedia

Erlernbarkeit	sehr leicht
LZG	gering durch Visualisierung der aktuellen Möglichkeiten
KZG	kann unter Umständen überfordert werden
Vorteile	• leicht erlernbar • assoziative Navigation • Nutzer wird unterstützt und nicht geführt • mehrere Wege zum Ziel
Nachteile	• erlaubt nur begrenzt Orientierung • keine Suche ohne weiteres • Zielfindung kann sich als schwierig erweisen
Anmerkungen	Nachteile können durch Designanpassungen abgeschwächt oder ganz aufgehoben werden

Wir haben uns bei der Umsetzung unseres Systems für einen Kompromiss aus verschiedenen Paradigmen entschieden. Zum Beispiel wollen wir beim Einrichten des Accounts einen Wizard benutzen, der den Anwender durch alle nötigen Schritte leitet. Das hilft den Start in das System so einfach und angenehm zu gestalten wie nur möglich. Dabei sollten wir jedoch im Auge behalten, den Wizard so kurz und prägnant wie möglich zu halten, um den User nicht beim ersten Eindruck zu langweilen. Für das normale Benutzen des Systems setzen wir auf direkte Manipulation. Dazu werden Angebote wie Objekte behandelt, die manipuliert, also zum Beispiel in die Einkaufsliste oder den Mülleimer gezogen, werden können. Das ist auch deshalb nützlich, weil es am ehesten die Datenstruktur unseres RESTful Services wiedergibt. Die Nachteile der direkten Manipulation versuchen wir durch die bereits erwähnten Maßnahmen, wie Ergebnis-Visualisierung zu reduzieren.

2.6 Mockups

Die Mockups zeigen einen deutlichen Prozess vom ersten Entwurf, über praktische Überlegungen, hin zu vielversprechenden Design-Prototypen. Dabei möchten wir vor allem auf die Platzierung des Menübalkens hinweisen, der entgegen gängiger Konventionen am unteren Bildschirmrand angesiedelt ist, wodurch die Benutzung und Erreichbarkeit erleichtert wird. Außerdem kritisch hinterfragt, wurde die Visualisierung der einzelnen Artikel. Kacheln sind eher unpraktikabel da sie beim links-rechts-Swipen nicht genügend Lauffläche für den Finger bieten und beim Hoch- Runter wischen im Konflikt mit dem Hoch- und Runterscrollen stehen. Das könnte zwar mit gedrückthalten des Artikels behoben werden was dann aber wiederum den Flow der App stören würde.
Der Rest der Mockups bezieht sich auf rein visuelle Designentscheidungen und ist auch nicht endgültig geklärt.

3. WBA

3.1 Anforderungen

Anforderungen die mit der Interaktion des Systems auftauchen.

Benutzerinteraktion

Muss-Kriterien	A100	Das System muss den Usern/Kunden die Möglichkeit bieten Werbung für bestimmte Produkte abzuschalten.
	A101	Das System muss den Usern/Kunden die Möglichkeit bieten Werbung für bestimmte Produkte bevorzugt zu bekommen.
	A103	Das System muss dem User/Kunden fragen stellen um ihn in eine Kategorie einzuordnen.
	A104	Das System muss auf Anfrage dem Kunden Angebote schicken.
	A105	Das System muss den User/Kunden die möglichkeit bieten einzustellen, zu welchem Zeitpunkt er die Werbung bekommen soll.
Soll-Kriterien	A106	Das System soll dem User/Kunden Einzelhandelsunternehmen in seiner Nähe anzeigen.
	A107	Das System soll dem User/Kunden Informationen über die Inhaltsstoffe der Produkte anzeigen.
	A108	Das System soll dem User/Kunden Spezialangebote anzeigen.

Selbständige Interaktion

Muss-Kriterien	A200	Das System muss selbständig Daten an den Benutzer schicken und Ihn über Angebote informieren
	A201	Das System muss dem User/Kunden die Werbung zu einem von ihm bestimmten Zeitpunkt schicken.
Soll-Kriterien	A202	Das System muss erkennen welche Produkte der User häufig auf seine Einkaufsliste setzt und ihm diese bevorzugt anzeigen.
Kann-Kriterien	A203	Das System kann anhand eines Algorithmus die Produkte selbstständig in die vordefinierten Kategorien einteilen.

3.2 Begriffsdefinition

User/Kunden:

Ein User ist ein Kunde der die Werbung erhält. Er kann dabei beeinflussen was für eine spezielle Werbung er kriegen soll.

Unternehmen:

Unternehmen können gezielte Werbung für den Kunden/User in die App eintragen. Sie können dabei jedoch nicht auf die User/Kundendatenbank zugreifen.

Artikel:

Artikel sind die Produkte, welche momentan im Angebot sind und von den Unternehmen in das System eingepflegt werden.

RESTful Service:

Ein Service die Ressourcen als URL darstellt, diese Ressourcen können mit der richtigen HTTP Request Methode aufgerufen, verändert oder upgedatet werden.[7]

HTTP

Ist ein Transportprotokoll im Internet was definiert wie Nachrichten formatiert sind und wie sie Transportiert werden. Es entscheidet außerdem welche Aktionen ein Webserver und Browser als Antwort auf verschiedene Kommandos bekommen.[8]

HTTP CRUD-Operations:

Ein Ausdruck für die HTTP Operationen Create (Put oder Post), Read (Get), Update (Patch oder Put) und Delete (Delete) die auf Datensätze im System erfolgen können.

HTML:

Ist dafür da den strukturellen Aufbau einer Internetseite zu regeln.

---

[7] Stefan Tilkov „Rest – Der bessere Web Service?“ unter https://jaxenter.de/rest-der-bessere-web-service-8988

[8] Vangie Beal „HTTP – HyperText Transfer Protocol“ unter http://www.webopedia.com/TERM/H/HTTP.html

3.3 Datenstrukturen

3.3.1 JSON

Ein schlankes Datenaustauschformat, welches für Menschen einfach zu lesen und zu schreiben ist. Für Maschinen ist dieses Format sehr einfach zu parsen und zu generieren.
Es ist ein Textformat war unabhängig von Programmiersprachen ist, jedoch vielen Konventionen von, aus der Familie der C-basierten Sprachen, bekannt sind. Es baut auf zwei Strukturen auf: Name/Wert Paaren und geordneten Listen von Werten, welches universelle Datenstrukturen sind, auf die alle modernen Programmiersprachen in der einen oder anderen Form unterstützt werden.[9]

3.3.2 XML:

Textbasierte Format für den Austausch strukturierter Informationen. Ist von dem älteren Standard SGML abgeleitet für die Anwendungen im Web. Es kann wohlgeformt sein, indem es die Syntaktischen Regeln von XML einhält oder gültig sein, wenn es wohlgeformt ist und alle Regeln der Dokumenttypen-Definition einhält. Für XML-Dateien können Schemata erstellt werden die die Art und Struktur von XML-Dokumenten definiert und zur Validierung genutzt werden.[10] [11]

3.3.3 Entscheidung für JSON

Wir haben uns in unserem Projekt für das Datenformat JSON entschieden, da wir hier mit der Erfahrung aus dem Modul Web-basierte Anwendungen 2: Verteilte Systeme arbeiten konnten. Da MongoDB die Datensätze in einem JSON ähnlichen Format speichert, ist es für uns besser geeignet als XML. Weil JSON für JavaScript Object Notation und somit auf JavaScript Datentypen zugreift, welche von unserer MongoDB Datenbank unterstützt werden, macht JSON zum optimalen Dateiformat für unsere Anwendung.[12]

---

[9] http://www.json.org/json-de.html
[10] Torsten Horn „XSD“ unter http://www.torsten-horn.de/techdocs/java-xsd.htm
[11] https://wiki.selfhtml.org/wiki/XML
[12] http://nodecode.de/mongodb

3.3.4 JSON Schema

Im Folgenden zeigen wir das JSON Schemata, dass bestimmt wie Daten in die MongoDB Datenbank gespeichert werden sollen. An diese Schemata soll sich jeder Datensatz halten und dieses Erfüllen.

```
{
    "$artikelschema": "/artikel/",
    "title": "Artikel",
    "description": "Ein Artikel im Angebot",
    "type": "object",
    "properties": {
        "name": {
            "description": "Der Name des Artikels",
            "type": "String",
        },
        "Firma": {
            "description": "Die Firma die den Artikel herstellt",
            "type": "String",
        },
        "Laden": {
            "description": "Der Laden in dem der Artikel im Angebot ist",
            "type": "String",
        },
        "Adresse": {
            "description": "Die Adresse des Ladens",
            "type": "String",
        },
        "Angebotstag": {
            "description": "Ab wann ist das Produkt im Angebot",
            "type": "integer",
        },
        "Angebotspreis": {
            "description": "Der Angebotspreis",
            "type": "integer",
        },
        "Normalpreis": {
            "description": "Der preis den der Artikel normalerweise kostet",
            "type": "integer",
        },
        "Kategorie": {
            "description": "Die Kategorie des Artikel z.B. TiefkÃ¼hl, Obst etc.",
            "type": "String",
        },
        "Inhaltsstoffe": {
            "description": "Die Inhaltsstoffe des Artikels",
            "type": "String",
        },
    },
    "required": ["name", "Firma", "Laden", "Adresse", "Angebotstag", "Angebotspreis", "Normalpreis", "Kategorie", "Inhaltsstoffe"]
}
```

Abbildung 2 JSON-Schema für Artikel 3.1

3.4 Systemarchitektur

Die Systemarchitektur wird mithilfe eines Architekturmodell und eines Kommunikationsmodells erklärt und soll aufzeigen wie die einzelnen Komponenten in unserem verteilten System miteinander kommunizieren.

3.4.1 Architekturmodell

Wie in Abbildung 3 zu sehen ist, besitzt unser verteiltes System zwei Server: den Main-Server und den Artikel Server. Der Artikel Server hat die Aufgabe Artikel auszugeben und eine Blacklist zu erstellen, die von den Kunden erstellt werden kann. Diese soll verhindern, dass User Artikel angezeigt bekommen, die ihnen nicht gefallen. Zwar werden die User durch eine vorherige Abfrage in unterschiedliche Kategorien eingeteilt, trotzdem können in den verschiedenen Kategorien, Artikel vorhanden sein, die dem Kunden nicht zusagen.
Die Kunden Datenbank wird vom Main-Server verwaltet und kann auch nur über diesen Abgerufen werden, was zur Sicherheit der Nutzerdaten dienen soll. Als Datenbank wird MongoDB verwendet. Der Main-Server kann mit dem Artikel Server oder einer App / Website kommunizieren. Dabei wird jeweils über das HTTP Protokoll kommuniziert. Als zu übertragendes Datenformat wird JSON anstelle von XML benutzt, da wir durch vorherige Projekte mit diesem Format mehr Erfahrung haben. Die App / Website schickt nur GETs auf den Main-Server um dort Informationen aus der Datenbank zu erhalten um diese dem Kunden zu Präsentieren. Die App / Website ist zum Erstellen von Einkaufslisten und dem Verwalten der Kundendaten zuständig. Hier können Kunden persönliche Daten ändern, die Änderungen werden dann an den Main-Server weitergeleitet. Außerdem ist die App / Website zur Darstellung der vom Main-Server gesendeten Daten zuständig. Die Einkaufslisten sollen es dem Kunden erleichtern, sich die Artikel die im Angebot sind besser zu merken, sodass diese beim nächsten Einkauf auch mitgenommen werden. Die Einkaufslisten werden lokal innerhalb der App / Website gespeichert. Um die Artikel zu bekommen gibt es den Artikel Server mit einer eigenen Datenbank. Dieser Artikel Server ist ein RESTful Service und speichert alle Artikel in der Datenbank ab. Artikel werden vom Server anhand eines Algorithmus in Kategorien unterteilt. So nimmt er sich die Marken, Inhaltsstoffe und Preise um die Artikel in Kategorien wie z.B. Kostengünstig, Fast Food oder Markenprodukte, zu unterteilen. Die Artikel werden vom Main-Server anhand eines GET abgefragt und werden vom Artikel Server anhand eines HTTP Response zurückgeschickt. Alle Artikel die der Artikel Server verwalten soll, werden durch eine Website welche nur für Unternehmen erreichbar ist, eingepflegt.
Auf dieser Website existiert ein Formular um einzelne Artikel einzupflegen, es soll aber auch möglich sein, direkt ein JSON Dokument hochzuladen, um so mehrere Artikel

gleichzeitig einzupflegen. Unternehmen werden die schon eingepflegten Artikel angezeigt, damit man keinen Artikel zweimal ins System einfügt. Die Unternehmen können auf dieser Website, Artikel löschen, bearbeiten oder neu erstellen.

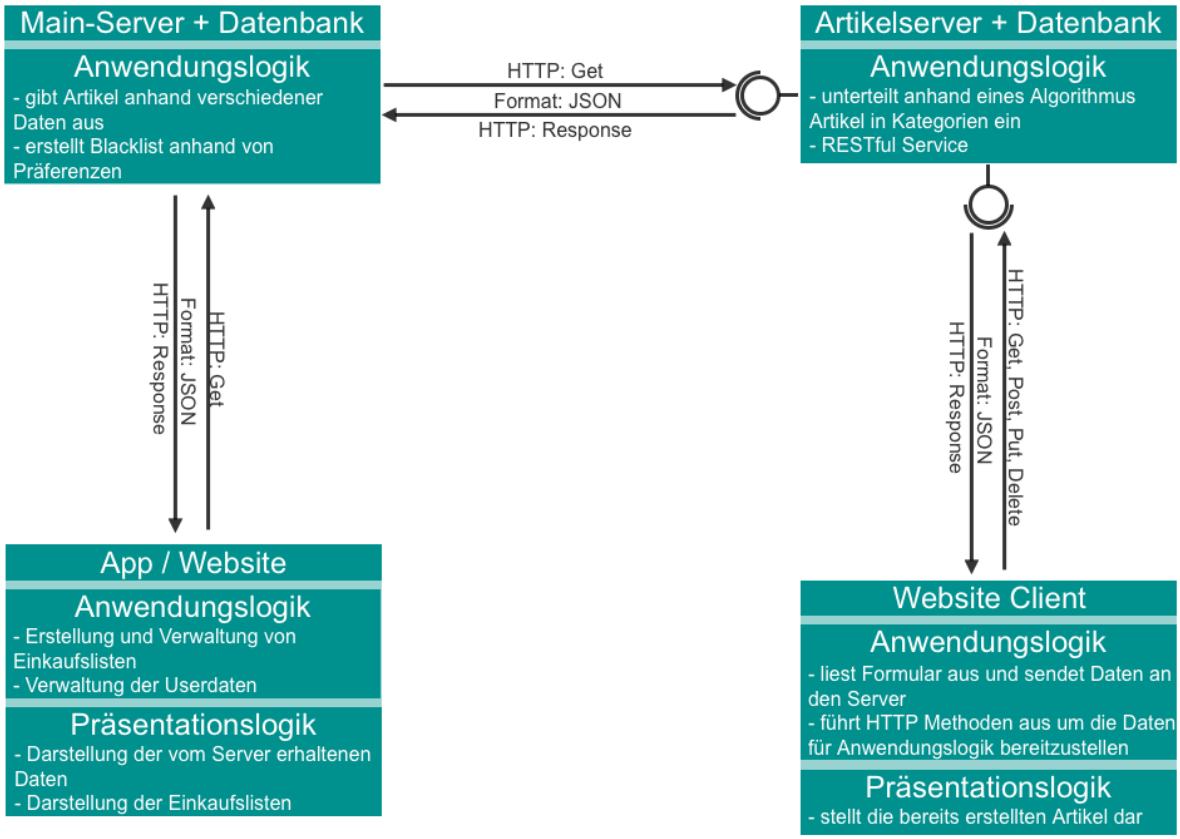

Abbildung 3 Architekturmodell zu Punkt 3.2

3.4.2 Kommunikationsmodell

Unsere Stakeholder können, wie in Abbildung 4 zu sehen ist, unterschiedlich mit dem System interagieren. Der Stakeholder “Kunde”, kann beim System die neuesten Angebote abfragen und sich den genauen Preis anzeigen lassen. Er kann seine Einkaufslisten abrufen um diese zu bearbeiten. Sind Produkte im Angebot, kann er die genaue Entfernung zum Laden anfragen um sich diese anzeigen zu lassen. Werden neue Angebote vom Stakeholder “Unternehmen” eingetragen, wird der Kunde durch einen externen Dienst, den Firebase Cloud Messaging, darüber per Push-Nachricht informiert. Das Unternehmen ist es auch möglich seine Momentanen Angebote einzusehen, die Angebote der Konkurrenz sind ihm dabei jedoch nicht ersichtlich.

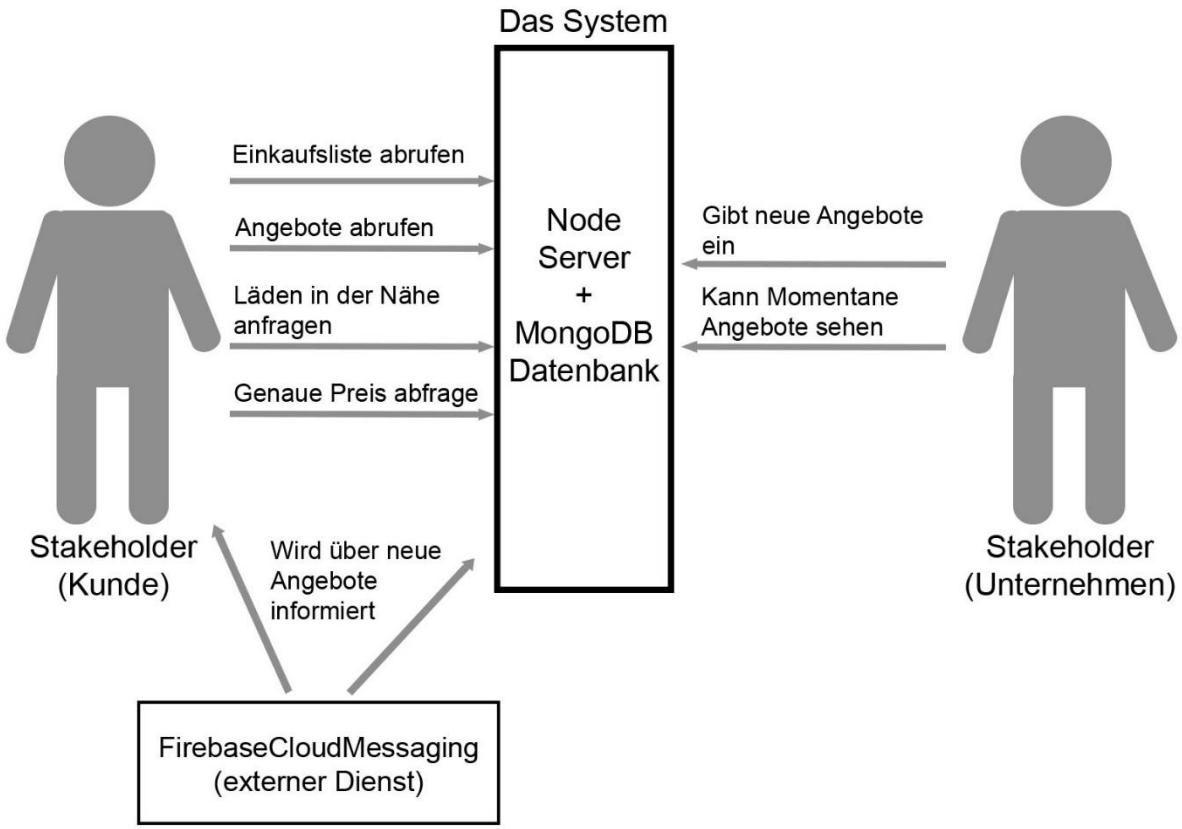

Abbildung 4 Kommunikationsmodell 3.3

3.4.3 WBA-Modellierung

Die WBA-Modellierung ist fast genauso wie unser Architekturdiagramm aufgebaut, diesmal jedoch mit dem Push-Nachrichtendienst Firebase Cloud Messaging. Mit diesem Dienst muss sich jeder Nutzer der App bei der Anmeldung einen Token registrieren, damit das Gerät für den Dienst eindeutig identifiziert werden kann. Dieser Token wird dann mit in die Client-Datenbank gespeichert. Anhand des Tokens kann eine Message als Push-Nachricht versendet werden, dabei wird die zu sendende Message mit einem Token an den Firebase Cloud Message Dienst geschickt. Dieser identifiziert anhand des Tokens das Gerät und sendet die Nachricht an den Client der die Message dann anzeigt.

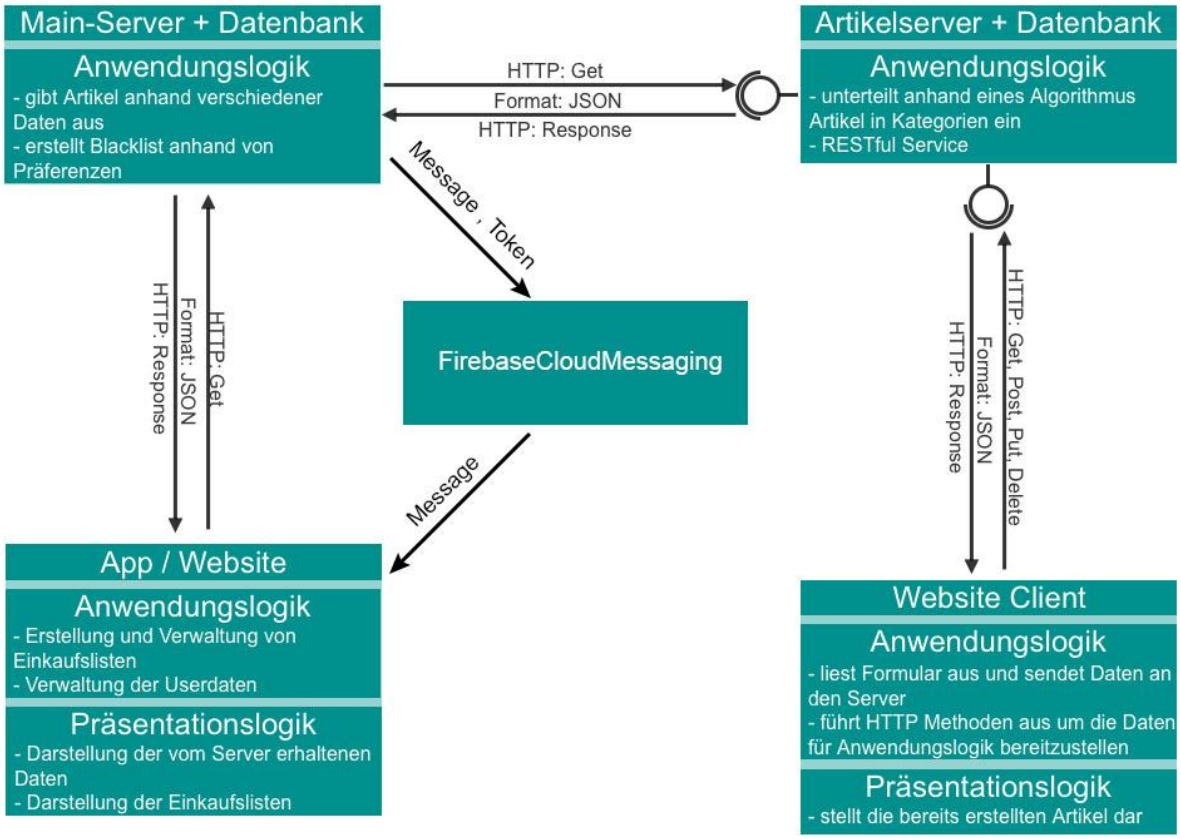

Abbildung 5 WBA-Modellierung 3.4

3.4.4 Architekturbegründung

3.4.4.1 Abwägungen Form der/des Clients

Der Nutzer-Client hat das Ziel für möglichst viele Nutzer erreichbar zu sein. Aus diesem Grund haben wir uns für eine Web-App entschieden, die als zusätzliches Angebot über eine entsprechende Entwicklungsumgebung als hybride App umgesetzt wird. Damit wollen wir einerseits die große Verfügbarkeit einer Website, andererseits die Bequemlichkeit einer Smartphone-App anbieten. Dabei hat die hybride App den Vorteil, nicht die Präsentationslogik aus dem Internet downloaden zu müssen.
Generell sehen wir den Vorteil, den Code nur einmal schreiben zu müssen und mit relativ wenig Aufwand auf verschieden Plattformen portieren zu können (siehe Hybride Apps).
Der Firmen-Client ist so konzipiert, dass einzelne Unternehmen den REST webService über die angebotene Schnittstelle erreichen. Wir wollen weiterhin eine Web-App anbieten, um auch kleinen Betrieben die Chance zu bieten, unser Angebot über eine vorgefertigte Eingabe-Plattform zu nutzen.

3.4.4.2 Abwägung Server

Bei der Auswahl der Serverarchitektur spielt der spätere Einsatz des Systems eine entscheidende Rolle. Bei unserem System handelt es sich um eine dynamische Anwendung, bei der der Server eine große Anzahl von asynchron auftretenden Verbindungen performant abarbeiten muss. In den letzten Jahren hat sich Node.js in diesem Bereich enorm bewährt und stellte bei der Konzipierung und Implementation von heutigen abfragen intensiven webbasierten Anwendungen die Grundlage zur Verfügung. Node ist dabei eine JavaScript-basierte Plattform für Webanwendungen, agiert serverseitig und setzt auf der performanten „V8-Laufzeitumgebung“ auf. Durch seine ereignisgesteuerte („nonblocking I/O model“) Strukturen bietet es eine ressourcenschonende Architektur, die im Kontrast zu threadbasierten Strukturen („blocking I/O model“) einen sparsamen Umgang mit dem wertvollen Arbeitsspeicher. Des Weiteren besitzt Node durch seine JavaScript-Basis eine native Möglichkeit JSON zu verarbeiten. Für die weitere Integration von Funktionen zu dieser Plattform dient NPM, welches die derzeit größte Open Source Bibliothek darstellt. All diese Kernaspekte von Node zeigen eine besondere Eignung der Plattform für skalierbare, hoch performante und echtzeitfähige Webanwendungen und stellen damit eine gute Grundlage für die Entwicklung unseres Systems dar

3.4.4.3 Abwägung Datenbank

Eine wichtige Rolle bei einer Webanwendung stellt das persistente Speichern von Daten, und somit ein Datenbanksystem, dar. Unterschieden wird dabei grundsätzlich zwischen einer schemafreien (NoSQL) und einer relationalen (SQL) Datenbank. Bei einer relationalen Datenbank werden die zu speichernden Daten in Tabellen hinterlegt, bei schemafreien hingegen in einer Sammlung von einzelnen Dokumenten. Durch diese schemaorientierte Speicherung bei SQL Datenbanken stoßen sie bei enorm großen Datenmengen an ihre Grenzen. Dokumentorientierte NoSQL Datenbanken sind besonders bei der schnellen Verarbeitung von einer großen Anzahl von Daten performant, stellen somit eine gute Ausgangslage für die Skalierbarkeit und Umsetzung unseres Systems dar. Als eine der beliebtesten NoSQL Datenbanken der

heutigen Zeit gilt die im Jahre 2007 entwickelte Datenbank MongoDB. Die Speicherung im BSON Format, das auf der Basis von JSON arbeitet und somit gute Verbindung zu einem JSON-basierten REST-Dienst mittels Node.js bietet. Auch die starke Skalierbarkeit und schnelle Performanz, die eine Echtzeitanwendung wie Neice voraussetzt sind gegeben. Die Wahl der passenden Datenbankarchitektur kann die serverseitige Anwendungslogik deutlich vereinfachen und kann mögliche Fehlerquellen vermindern, und stellt daher einen qualitätssichernden Aspekt des Systems dar.

3.4.4.4 Abwägung Messaging

Um eine möglichst Echtzeit nahe Kommunikation zwischen den Anbietern und dem Endverbraucher zu ermöglichen gibt es mehrere Möglichkeiten, um dies zu erreichen. Mögliche Varianten sind zum einen das Polling, Push-Benachrichtigungen und eine Kommunikation über eine Publish/Subcribe-Paradigmas. Polling beschreibt vom Client initiierte Anfragen auf Veränderung an den Server und dessen direkte oder zeitlich versetzte Antwort. Push Benachrichtigungen sind vom Server erstellte Mitteilungen an den Client. Eine weitere Möglichkeit um die Kommunikation zwischen Client und Server zu gewährleisten ist eine Verbindung über ein Publish/Subscribe-Paradigmas. Dieses beschreibt die bei Veränderung einer Publikation ausgelöste Mitteilung an alle Abonnenten. Zumeist wird dieses Paradigma durch das XMPP Protokoll, welches sich auf das Instantmessaging spezialisiert hat, und Push-Benachrichtigungen realisiert. Für unseren Anwendungsfall scheint eine Implementierung via Push-Paradigma am geeignetsten.

3.4.4.5 Abwägung Geo-API

Um dem Nutzer mögliche Läden, in denen die beworbenen Angebote zu finden sind zu vermitteln, werden Standortdaten benötigt. Um die Daten auszuwerten bieten sich Kartenservices wie Google Maps oder Openstreetmap an, da aufgrund der mangelnden Zeit zur Realisierung des Projektes eine eigene Erstellung eines Kartenservice ausgeschlossen ist. Da es sich bei dem Standort eines Nutzers um höchst sensible Daten handeln kann, ist eine genauere Untersuchung des Datenschutzes auf Seiten dieser Drittanbieterdienste nötig. Weiterhin muss der

Kartenservice aufgrund der Grundsätzlichen Architektur des Interaktiven Systems über eine REST-Konforme API verfügen, da sonst das Bestreben eines zustandslosen, auf HTTP basierenden Systems nicht mehr gewährleistet ist. Aufgrund dieser Kriterien kommt das Verwenden von Googles Kartenservice Maps unmöglich. Gründe hierfür sind die mangelnden Defizite des Datenschutzes und der nicht vorhandenen REST-Konformen API. Das auf OpenSource basierende Kartographie Projekt Openstreetmap erfüllt alle geforderten Kriterien. Es ist möglich, auf schon vorgerenderte Karten oder auf Rohdaten in Form von XML-Files über eine API herunterzuladen. Des Weiteren gibt es mehrere andere API´s, wie Overpass API, welche es ermöglichen diese XML-Dateien in ein für das zu entwickelnde System lesbares JSON-Format umwandeln

3.5 Hybride Apps

3.5.1 Was sind Hybride Apps?

Hybride Apps sind eine Mischung aus Web App und nativer App. Sie gleichen die Vor- und Nachteil der nativen- und Web App aus. Hybride Apps sind Web-Apps die auf Web-Technologien setzen und es ermöglichen plattformübergreifende Apps zu entwickeln die nicht direkt von nativen Apps zu unterscheiden sind. Die Web-App wird hierbei durch Hybrid-Frameworks in einen Container gepackt. Verschiedene Plattformspezifische Funktionen werden dem Entwickler dabei durch JavaScript bereitgestellt. Dabei kann die Hybride App auf viele Funktionen zugreifen, die sonst nur der Nativen App zur Verfügung stehen. Ein großer Vorteil von Hybriden Apps ist, dass die App wie eine Web App Programmiert werden kann und auf die verschiedenen Plattformen Kompiliert wird. Dadurch wird der Aufwand nicht erhöht und man muss sich nicht auf eine Plattform festlegen. Die Hybriden Apps können dabei auf dem App-Store der jeweiligen Plattform angeboten werden. Trotzdem kommt man dem “Look and Feel” von Nativen Apps sehr nahe. Für die Erstellung einer Hybriden App werden Frameworks wie Ionic, Intel XDK, Onsen UI, Sencha Touch, Mobile Angular UI oder Phonegap benutzt. Viele dieser Frameworks erzeugen jedoch nur native UI-Komponenten und basieren nicht direkt auf Web-Apps. Benutzt ein Framework jedoch Web-Technologien, so nutzten Hybriden-Technologien die sogenannte “WebView”,

eine in die native Anwendung eingebetteter Browser, durch den die App dargestellt wird.

Viele Firmen setzen schon jetzt auf Hybride Apps aufgrund der Kosten die bei der Entwicklung gespart werden können. Firmen wie Apple haben schon 2013 die HTML5-Funktionen deutlich ausgebaut und hat es Entwicklern ermöglicht auf Beschleunigungssensoren zuzugreifen. So gibt es viele Berühmte Apps die auf Hybrid-Technologien setzen, z.B. JustWatch, ZenMate oder ADAC Mitfahrclub.[13]

Mit 33 Prozent (stand 2015) werden hybride Apps von App-Entwicklern, im Vergleich zu nativen Apps (25 Prozent) und Web-Apps (19 Prozent), bevorzugt.[14]

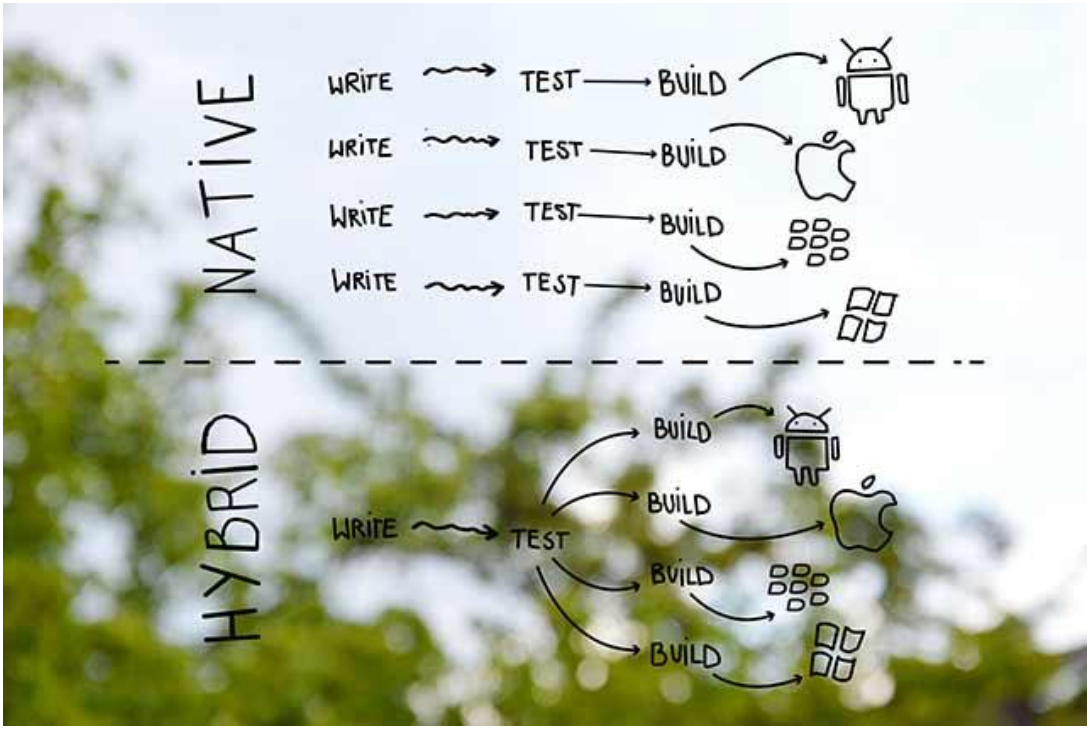

Abbildung 6 Hybrid-App Ansatz (Quelle: FLYACTS)

3.5.2 Unterschied zwischen Hybride App, Native App und Web-App

Eine Native App entwickelt man mit Hilfe von Tools und Sprachen, welche von der entsprechenden Plattform unterstützt werden, z.B. XCode und Objective C für iOS Apps oder Android Studio und Java für Android. Diese Apps laufen auch nur auf der Zielplattform auf der sie entwickelt wurde. Die Native App ist auch Offline Nutzbar,

[13] http://hybridheroes.de/blog/2015-11-12-die-besten-hybrid-apps-deutschlands/
[14] Oliver Schonscheck „Hybride Apps zwischen Chancen und Risiken“ unter http://www.zdnet.de/88248234/hybride-apps-zwischen-chancen-und-risiken/

auch Daten der Nativen App können Lokal gespeichert werden und benötigen nicht unbedingt eine Internetverbindung. Durch die Hardware Nähe der Nativen App ist die Performance besser und die Hardware Ressourcen schonender behandelt werden, als bei Hybriden und Web-Apps. Das Gegenteil davon ist eine Web App oder auch HTML5 App genannt. Diese sind plattformübergreifend und laufen im Browser des Gerätes. Es werden moderne Web-Technologien wie HTML5, CSS3, JavaScript benutzt. Diese Apps können jedoch nicht auf Plattformspezifische Funktionen des Gerätes Zugreifen, oder nur sehr begrenzt. Dafür ist die Installation nicht vorhanden und eine Web-App kann wie eine Webseite aufgerufen werden. Des Weiteren sind Web Apps für Endkunden einfacher zu finden und zu benutzen. Im Gegensatz zu einer Nativen App, kostet eine Web-App bis zu 15-20% weniger in der Entwicklung. Hybride Apps vereinen wie in Punkt 3.5.1 beschrieben alle Vorteile einer Web-App und Nativen App.

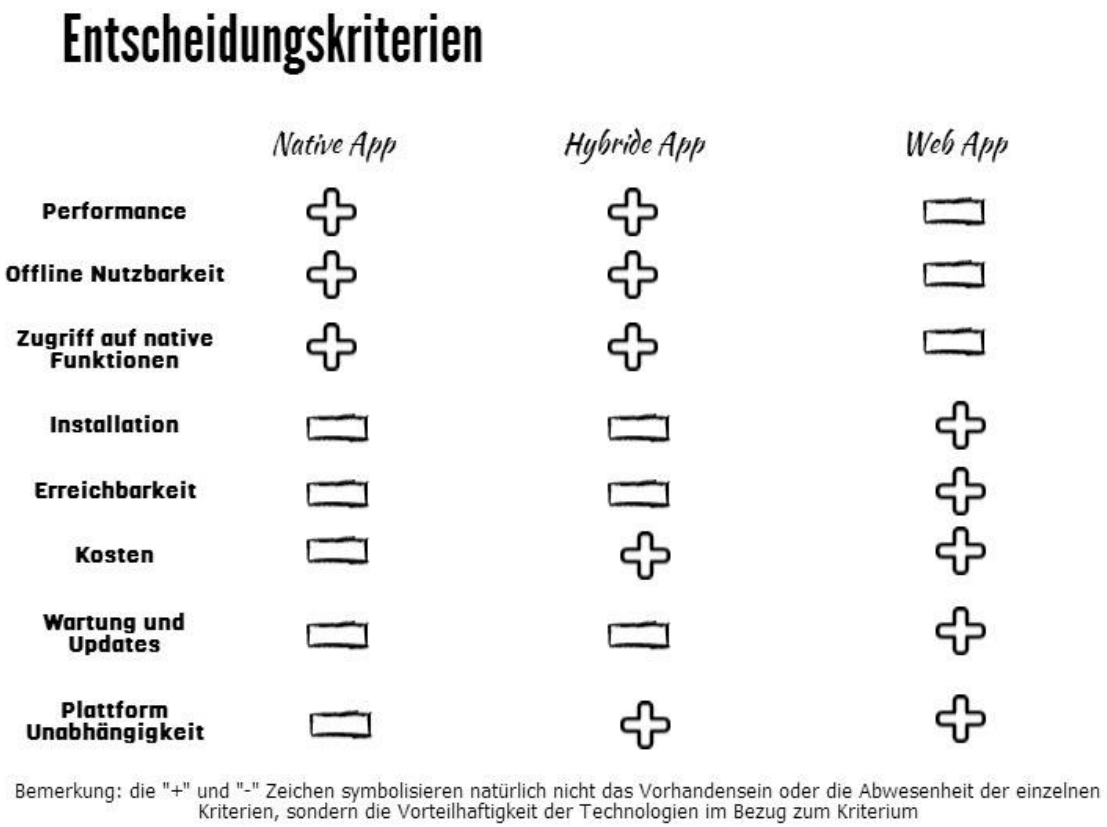

Abbildung 7 Entscheidungskriterien (Quelle: app3null)

3.5.3 Entscheidung für Hybride App

Wir haben uns am Ende für eine Hybride App entschieden, da die Vorteile überwiegen. Da wir nur begrenzte native Funktionen benutzen wollen, bietet sich eine Hybride App

für unserer Verteiltes System an. Dabei haben wir uns an der Abbildung 7 Orientiert die für uns mit dem Vorhandensein der nativen Funktionen und der Plattform Unabhängigkeit den richtigen Kompromiss bieten. Die Performance einer Hybriden App hat sich in den letzten Jahren auch verbessert und der Unterschied zu einer Nativen App ist nicht mehr allzu groß. Wir können so eine App, die eine gutes “look and feel”-Gefühl an den Endnutzer überträgt, für verschiedene Plattformen, mit weniger Ressourcen, umsetzen.
Durch das Vorhandensein vieler verschiedener Frameworks mit großen Dokumentationen haben wir hier eine wirklich gute Auswahl, und so die Möglichkeit genau das Framework für unsere Bedürfnisse zu finden.

3.6 Push-Nachrichten

3.6.1 Firebase Cloud Messaging

Zum Versenden wird der externe Dienst Firebase Cloud Messaging[15] genutzt. Wir haben uns für diesen Dienst entschieden, da er sowohl Node.js als auch REST Unterstützung bietet.[16] Eine Authentifizierung ist hierbei auch ohne richtigen Server Code möglich, was gut zum Debuggen und ausprobieren von Funktionen ist. Dabei reagiert der Dienst als ein Kerndaten Ersatz. Firebase besitzt neben den Push-Notifications noch mehr Tools, wie z.B. eine Echtzeit-Datenbank, Statistik-Tools und Hosting. Firebase ist dabei eine weiterentwicklung vom Google Cloud Messaging und wurde in Firebase dabei integriert. Es ist einfacher zu Verwenden und hat einen verbesserten Funktionsumfang als Google Cloud Messaging. Viele Funktionen lassen sich auch durch die Notifications Console, welche eine grafische Benutzeroberfläche bietet, testen. Damit ist es auch mögliche, direkte Push-Notifications an Nutzer unserer App zu versenden. Daneben gibt es die normalen Firebase Notification.
Dies ist eine out-of-the-box Lösung für das Versenden von Push-Notifications. Dabei kümmert sich Firebase selbständig um das Versenden von Nachrichten, wodurch kein zusätzlicher Server benötigt wird. Durch eine Analysefunktion kann dabei sogar eingesehen werden, wie viele Benutzer die Nachricht erhalten haben und auch

[15] https://firebase.google.com/docs/reference/
[16] Kishin Manglani „Top 5 Parse Alternatives” unter https://www.raywenderlich.com/126098/top-5-parse-alternatives

geöffnet haben, was eine Interessante Information für uns und die Unternehmen darstellt. So lassen sich gute Teststudien durchführen, die uns Zeigen wie viele Benutzer die App regelmäßig benutzen. Zur einfachsten Einbindung von Firebase ist zu dem nicht mal eine Zeile Code Notwendig. Dafür muss nur ein neues Projekt im Browser angelegt werden und eine Abhängigkeit zu einer im Browser erstellten Konfigurationsdatei hergestellt werden. Danach können Push-Notifications an Nutzer unserer App gesendet werden. Diese Form wird nur im Anfangsstadium unserer App genutzt zum Debuggen. Für eine App-Umgebung mit Push-Notifications mit einem eigenen App-Server ist das komplette Firebase Cloud Messaging System notwendig. Dies bietet eine weitere Schnittstelle zur Kommunikation mit einem App-Server, welcher dann automatisch die Nachrichten versendet. Trotzdem ist es auch möglich über die Notifications Console Nachrichten manuell zu versenden. Es gibt dabei zwei Arten von Nachrichten, welche mit dem Typen “Notification”, die nur eine begrenzte Größe haben dürfen, und dem Typen “Data”, welcher es erlaubt auch doppelt so große Nachrichten zu versenden. Die größe von Push-Notification begrenzt sich dabei auf ein Bild mit einer maximalen Größe von 80x80 Bildpunkten und einer Textlänge von 200 bis 250 Zeichen.[17] Nachrichten vom Typen “Data”, bleiben dabei jedoch für den Nutzer unsichtbar und werden im Hintergrund empfangen und werden nicht al Einblendung oder im Notification Center angezeigt. Die Daten werden alleine von der App ausgewertet und entsprechende Aktionen damit ausgeführt. Dies kann dazu benutzt werden um bestimmte Inhalte in der App zu aktualisieren, z.B. wenn ein Angebot fehlerhaft ist und nicht den korrekten Angebotspreis zeigt.[18]

[17] Martin Aschoff “Web Push Notifications: Ein neuer Kommunikationskanal“ unter http://www.email-marketing-forum.de/Fachartikel/details/1639-Web-Push-Notifications-Ein-neuer-Kommunikationskanal/135084

[18] Bluesource „Firebase Cloud Messaging” unter http://www.app-entwicklung.info/2016/11/firebase-cloud-messaging/

Literaturverzeichnis

DAkkS, S.36, 2016 http://www.dakks.de/content/leitfaden-usability

Monson-Haefel, „97 Things Every Software Architect Should Know" unter http://shop.oreilly.com/product/9780596522704.do

Jakob Nielsen: "Discount Usability for the Web" unter https://www.nngroup.com/articles/web-discount-usability/ (abgerufen am 5.12.16)

UIG e.V. „Fragebogen" unter http://www.usability-in-germany.de/definition/fragebogen

Jakob Nielsen, „10 Usability Heuristics for User Interface Design" unter https://www.nngroup.com/articles/ten-usability-heuristics/

Jakob Nielsen, „10 Usability Heuristics for User Interface Design" unter https://www.nngroup.com/articles/ten-usability-heuristics/

Stefan Tilkov „Rest – Der bessere Web Service?" unter https://jaxenter.de/rest-der-bessere-web-service-8988

Vangie Beal „HTTP – HyperText Transfer Protocol" unter http://www.webopedia.com/TERM/H/HTTP.html

http://www.json.org/json-de.html

Torsten Horn „XSD" unter http://www.torsten-horn.de/techdocs/java-xsd.htm

https://wiki.selfhtml.org/wiki/XML

http://nodecode.de/mongodb

http://hybridheroes.de/blog/2015-11-12-die-besten-hybrid-apps-deutschlands/

Oliver Schonscheck „Hybride Apps zwischen Chancen und Risiken" unter http://www.zdnet.de/88248234/hybride-apps-zwischen-chancen-und-risiken/

https://firebase.google.com/docs/reference/

Kishin Manglani „Top 5 Parse Alternatives" unter https://www.raywenderlich.com/126098/top-5-parse-alternatives

Martin Aschoff "Web Push Notifications: Ein neuer Kommunikationskanal" unter http://www.email-marketing-forum.de/Fachartikel/details/1639-Web-Push-Notifications-Ein-neuer-Kommunikationskanal/135084

Bluesource „Firebase Cloud Messaging" unter http://www.app-entwicklung.info/2016/11/firebase-cloud-messaging/