C:\Users\Edwin\Google Drive\Studium\Studium\4 Semester\Software Engineering 1\Allergeek\Dokumentation\Konzeption\Corporate Design\Logo.png

Projekt Edible

Abschlussdokumentation

27.4.15

**Inhalt**

[1 Intention und Motivation 3](#_Toc417852574)

[1.1 Anwendungsbereich 3](#_Toc417852575)

[2 Anforderungen 3](#_Toc417852576)

[2.1 Zielgruppen 3](#_Toc417852577)

[2.2 Zielplattform 3](#_Toc417852578)

[2.3 Usecases 3](#_Toc417852579)

[3 Spezifikationen 4](#_Toc417852580)

[3.1 Genereller Aufbau und Systemkomponenten 4](#_Toc417852581)

[3.2 Funktionale und nicht funktionale Anforderungen 5](#_Toc417852582)

[3.3 Nicht funktionale Anforderungen 6](#_Toc417852583)

[4 Architektur 7](#_Toc417852584)

[4.1 Backend 7](#_Toc417852585)

[4.1.1 Schnittstellen 7](#_Toc417852586)

[4.1.2 Klassendiagramme 7](#_Toc417852587)

[4.1.3 Datenmodell 7](#_Toc417852588)

[4.1.4 Libraries und Frameworks 7](#_Toc417852589)

[4.1.5 Server 7](#_Toc417852590)

[4.2 Frontend 8](#_Toc417852591)

[4.2.1 Klassendiagramme 8](#_Toc417852592)

[4.2.2 Libraries und Frameworks 8](#_Toc417852593)

[4.3 HMD Applikation 8](#_Toc417852594)

[4.4 Klassendiagramme 8](#_Toc417852595)

[4.5 Libraries, Frameworks und Intents 9](#_Toc417852596)

[5 Bewertung der Ergebnisse 9](#_Toc417852597)

[6 Beschreibung des Projektablaufs 9](#_Toc417852598)

[6.1 Planung 9](#_Toc417852599)

[6.2 Vorgehensmodell 9](#_Toc417852600)

[6.3 Rollen 10](#_Toc417852601)

[6.4 Testkonzeption 10](#_Toc417852602)

[7 Fazit und Ausblick 11](#_Toc417852603)

# Intention und Motivation

Das Ziel von Edible ist es Allergikern zu helfen. Dafür prüft Edible Produkte auf ihre Inhaltsstoffe und informiert den User ob dieser gegen das Produkt allergisch reagieren könnte. Die Realisierung hat auf einem Head-Mounted Display (HMD) stattgefunden. Durch die Anwendung kann das HMD nun Produktbarcodes einlesen und die Inhaltsstoffe per Web-Request von einem Server abfragen. Durch den Abgleich einer Blacklist wird so dem Allergiker Auskunft gegeben ob er dieses Produkt konsumieren kann oder nicht.

## Anwendungsbereich

Der Hauptanwendungsbereich von Edible liegt im regelmäßigen Einkauf von Lebensmitteln der Anwender, bzw. auch vor dem Konsum eines Produktes.

# Anforderungen

## Zielgruppen

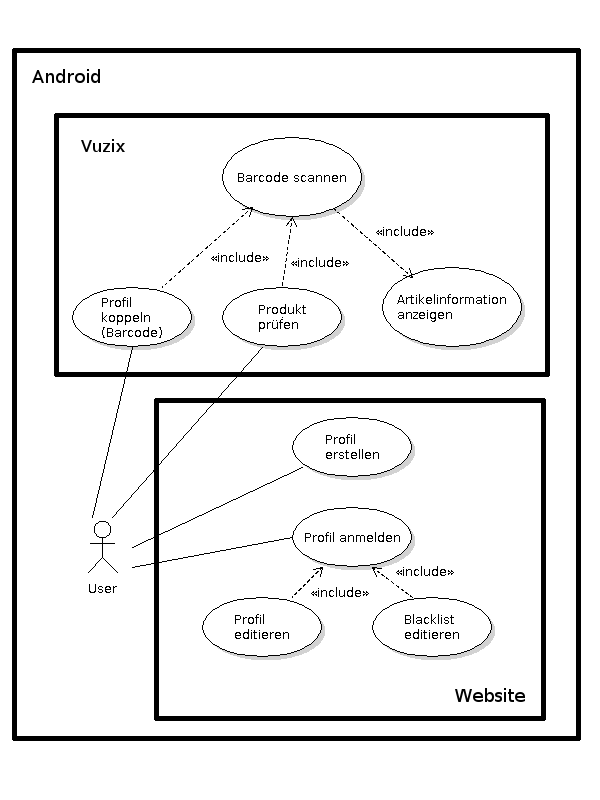
Die Hauptzielgruppe von Edible sind Allergiker die durch ihre Allergien gezwungen sind auf ihre Ernährung zu achten. Weiter kann die Anwendung von ernährungsbewussten Personen genutzt werden, die gewisse Inhaltsstoffe aus anderen Gründen meiden als Allergiker. Potenziell sind auch Personen auf Diäten eine Zielgruppe für die Anwendung, denn auch sie können durch die Anwendung gewisse Inhaltsstoffe meiden.

## Zielplattform

Edible ist als Anwendung für HMDs verfügbar und wurde für eine Benutzung auf der Vuzix M100 optimiert.

## Usecases

Die folgende Grafik zeigt die Usecaseübersicht für Edible.



# Spezifikationen

## Genereller Aufbau und Systemkomponenten

Aufgrund der eingeschränkten Möglichkeiten an der HMD und dem Wunsch, dem Nutzer ein möglichst gutes Erlebnis mit seinem Gerät zu liefern, fiel die Entscheidung auf eine drei geteilte Architektur. Diese umfasst HMD Applikation, das Server Backend und die Webanwendung im Browser.

Die Bedienung komplexerer Anwendungen auf einem Wearable Device ist, aufgrund der eingeschränkten Eingabemöglichkeiten, der Bedienung auf Geräten mit komplexeren Eingabeschnittstellen unterlegen. Eine komplette Verwaltung von Userdaten und Zutatenlisten innerhalb der mobilen Anwendung ist wenig komfortabel. Es bietet sich ein Webfrontend an, da ein Backend in jedem Fall für die Beschaffung von Artikeldaten benötigt wird. Die Zielgeräte verfügen außer den kleinen Bildschirmen nur geringe Speicherkapazitäten und Rechenleistung. Eine Artikeldatenbank, die permanent Updates erhält, ist auf einem solchen Gerät nicht ideal.

Die dreigeteilte Architektur bietet so dem Verbraucher maximalen Nutzen. Seine Daten werden gesammelt auf einem Server gehalten und sind von allen Geräten, die er auf dem Account registriert zugänglich. Er hat immer Zugang zu den aktuellsten Daten. Die Benutzerreingaben auf dem Endgerät sind bis auf ein Minimum ausgelagert. Das Design der App konzentriert sich dabei auf den einzelnen Anwendungsfall des Barcode Scannens und Beschaffen der Produktinformationen.

Durch diese Aufteilung, benötigt der Nutzer jedoch immer eine Internetverbindung. Wenn der Nutzer die Verbindung seines Handys benutzt, ist dies jedoch kaum ein Problem, da die meisten Personen mit ihrem Smartphone auch unterwegs permanent mit dem Internet verbunden sind.

## Funktionale und nicht funktionale Anforderungen

Die angestrebten funktionalen Anforderungen wurden in 3 Bereiche eingeteilt „Must have“ (1), „Should have“ (2), „Nice to have“ (3). Wie in der folgenden Tabelle zu sehen ist, sind alle „Must have“ Anforderungen realisiert oder angepasst worden. Bis auf eine „Should have“ Anforderung wurden alle umgesetzt. Die Android-Smartphone Applikation wurde als „Nice to have“ formuliert und wurde zum größten Teil durch die Website in responsive Layout implementiert.

### Funktionale Anforderungen

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Funktionalität | | Priorität | Ergebnis |
| F10 | HMD App |  | |  |
| F10.1 | Das Zielsystem der HMD App ist die Vuzix M100. Die App muss ohne Einschränkungen auf der Vuzix M100 lauffähig sein. | 1 | | Umgesetzt |
| F10.2 | Um sein mobiles Endgerät mit einem Account zu verbinden, scannt der User einen Barcode, der ihm im Webbrowser angezeigt wird. Der User muss sich dafür bereits auf der Website registriert haben. | 2 | | Umgesetzt |
| F10.3 | Die App verfügt sowohl über den Standard-Sprachbefehl "go home" als auch den spezifischen Sprachbefehl "scan" zum Starten des Produktscanners. | 1 | | Angepasst |
| F10.4 | Nach dem Start der App wird der Homescreen angezeigt. Hier wird auf den Sprachbefehl zum Scannen gewartet. | 1 | | Umgesetzt |
| F10.5 | Mit dem Sprachbefehl "scan" aktiviert der User im Homescreen den Scanner. Danach kann er über die Kamera den Barcode eines Produkts scannen. | 1 | | Umgesetzt |
| F10.6 | Wurde ein Produkt eingescannt, wird dem User auf dem Infoscreen das Ergebnis angezeigt. Nach 10 Sekunden im Infoscreen kehrt die App automatisch in den Homescreen zurück. | 1 | | Angepasst |
| F20 | Accountsystem |  | |  |
| F20.1 | Der User kann einen Account erstellen, der persistent im System gespeichert wird. (siehe F30.1) | 1 | | Umgesetzt |
| F20.2 | Der User kann beliebig viele mobile Endgeräte mit seinem Account verbinden. | 2 | | Umgesetzt |
| F30 | Webapplikation |  | |  |
| F30.1 | Der User kann einen neuen Account über ein Registrierungsformular erstellen. (siehe F20.1) | 1 | | Umgesetzt |
| F30.2 | Der User kann sich mit seiner Email-Adresse und seinem Passwort in einem Login Formular anmelden. | 1 | | Umgesetzt |
| F30.3 | Der User kann Einträge aus der Inhaltsstoffdatenbank zu seiner Blacklist sowohl hinzufügen als auch wieder entfernen. | 1 | | Umgesetzt |
| F30.4 | Der User kann HMDs koppeln und entkoppeln. | 2 | | Umgesetzt |
| F30.5 | Für die Verbindung mit der HMD-App wird ein Barcode im Web-Browser angezeigt. | 2 | | Umgesetzt |
| F30.6 | Der User kann seine Email-Adresse und sein Passwort ändern. | 2 | | Umgesetzt |
| F30.7 | Registriert sich ein User oder ändert er seine Emailadresse, muss er diese bestätigen. Zum Bestätigen wird eine Bestätigungslink an die Emailadresse versandt, die durch den User bestätigt werden muss. | 2 | | Nicht umgesetzt |
| F40 | Android-Smartphone App (durch responsive Layout realisiert) | | | |
| F40.1 | Der User kann HMDs koppeln und entkoppeln. | 3 | | Teilweise umgesetzt |
| F40.2 | Der User kann sich für die Nutzung des Dienstes in der Android-Anwendung registrieren. | 3 | | Teilweise umgesetzt |
| F40.3 | Der User muss sich zur Nutzung des Dienstes in der Android-Anwendung anmelden. | 3 | | Teilweise umgesetzt |
| F40.4 | Der User muss seine Email-Adresse und sein Passwort ändern können. | 3 | | Teilweise umgesetzt |
| F40.5 | Der User kann Einträge aus der Inhaltsstoffdatenbank zu seiner Blacklist hinzufügen und wieder entfernen. | 3 | | Teilweise umgesetzt |
| F40.6 | Der User scannt einen Produktbarcode und erhält eine Information über die Verträglichkeit. | 3 | | Teilweise umgesetzt |
| F40.7 | Der User kann sich alle Inhaltsstoffe des Produktes anzeigen lassen, sowie eine Websuche nach dem Produkt starten. | 3 | | Nicht umgesetzt |

### Nicht funktionale Anforderungen

|  |  |
| --- | --- |
| ID | Funktionalität |
| N10 | Fachliche Mengen |
| N10.1 | Bei der Anzahl der Nutzer wird zu Beginn von einer Nutzeranzahl von unter 1000 Nutzern ausgegangen. Die Anwendung sollte später auch für größere Nutzeranzahlen auslegbar sein. |
| N10.2 | Jeder Nutzer hat durchschnittlich 10 Inhaltsstoffe in seiner Blacklist. |
| N20 | Usability |
| N20.1 | Die Oberfläche muss selbsterklärend und einfach sein, damit es auch von Einsteigern ohne Einweisung verwendet werden kann. |
| N20.2 | Die Funktionen der Oberfläche werden in einem Handbuch dokumentiert. |
| N20.3 | Insbesondere auf dem niedrig auflösendem Bildschirm der Vuzix M100 muss auf eine gute Lesbarkeit der Inhalte geachtet werden. Auch auf allen anderen Plattformen muss eine Lesbarkeit gegeben sein. |
| N30 | Erscheinungsbild |
| N30.1 | Die Erscheinung der Oberfläche soll in Design und Bedienungskonzept einheitlich sein. |
| N40 | Performanz |
| N40.1 | Der Server soll für die Bearbeitung einer Anfrage maximal 100ms brauchen. |
| N40.2 | Es wird von maximal 60 Anfragen pro Minute unter Volllast ausgegangen. |
| N40.3 | Da die App vorerst nicht kommerziell betrieben wird, ist eine hohe Verfügbarkeit nicht wichtig. Eine Verfügbarkeit des Webservices von 98% im Jahreszeitraum ist damit ausreichend. |
| N50 | Wartbarkeit |
| N50.1 | Für die Qualität und Einheitlichkeit des Codes wird ein Styleguide definiert, dem der produzierte Code entsprechen muss. |
| N50.2 | Der Code muss mit ausreichend Kommentaren ausgestattet sein. Für eine Automatische Dokumentationserstellung werden spezielle Kommentare erstellt, die automatisch verarbeitet werden. Diese Kommentare werden im Styleguide festgehalten. |
| N60 | Tests |
| N60.1 | Anforderungen an die Testbarkeit werden in einem separaten Testkonzept festgelegt. |

# Architektur

## Backend

Der Zweck des Backends besteht darin, eine Schnittstelle zur Verfügung zu stellen, die sämtliche Funktionalitäten des Produktes abbildet. Diese Schnittstelle wird sowohl vom Web-Frontend, als auch von der HMD-Applikation verwendet.

### Schnittstellen

Die API arbeitet mit dem REST-Konzept (Representational State Transfer). Hierbei werden die Methoden GET, PUT, POST und DELETE des HTTP-Protokolls verwendet, um die API anzusprechen. Ein GET-Request auf eine Route, ggf. mit Parametern, liefert Daten zurück, mithilfe von PUT werden neue Datensätze angelegt, POST wird verwendet um Daten zu bearbeiten und mit DELETE werden nicht mehr benötigte Daten gelöscht. Zur Repräsentation von Daten wird JSON (JavaScript Object Notation) verwendet. Die Routen der API können im Anhang eingesehen werden.

### Klassendiagramme

Alle Klassen zur Realisierung der API-Routen sind in PHP implementiert. Die Klasse "API" stellt die Architektur der RESTful API zur Verfügung, "DB" verwaltet den Zugriff auf die Datenbank. Die Klassen "Ingredient", "Product", "Session" und "User" bilden die korrespondierenden Daten ab. Des Weiteren gibt es zwei Fehlerklassen, die zwischen internen (InternalError) und userspezifischen (UserError) Fehlern unterscheiden.

Details können dem Klassendiagramm im Anhang entnommen werden.

### Datenmodell

In der MySQL-Datenbank werden die Daten für das Backend gespeichert. Die Stammdaten des Systems sind Produkte und deren Inhaltsstoffe mit einer Kategorisierung gespeichert werden. Bewegungsdaten sind hauptsächlich das Accountsystem, das die User und ihre verknüpften Endgeräte umfasst, aber auch die Backlists, auf denen die Benutzer Inhaltsstoffe dynamisch hinzufügen und entfernen.

Details können dem ER-Diagramm im Anhang entnommen werden.

### Libraries und Frameworks

Für das Backend werden keine größeren Frameworks oder Libraries verwendet, da der Funktionsumfang der API am effektivsten durch vollständig eigene Implementierung umgesetzt werden kann.

Auf diese Weise können die Teammitglieder vorhandenes Wissen sowohl anwenden als auch vertiefen – mit vollständiger Kontrolle und Transparenz jeder implementierten Komponente. So werden Abhängigkeiten von externen Komponenten vermieden und unter anderem Performanceproblemen vorgebeugt.

### Server

Zur Umsetzung des Server-Backends wird Debian GNU/Linux eingesetzt. Version 8 (mit dem Releasenamen Jessie) befindet sich seit dem 05. November 2014 im Codefreeze. Änderungen im Code sind ab diesem Zeitpunkt kaum zu erwarten. Jessie bietet somit eine sowohl zuverlässige als auch aktuelle Grundlage für die Architektur der Web-Applikation darüber. Seit dem 26. April 2015 ist Debian Jessie "stable". Apache HTTP Server (in der Version 2.4.10) deckt die Funktionalität des Webservers ab. MySQL-Server (5.4.42) die des Datenbankmanagementsystems. Zu administrativen Zwecken wird zusätzlich phpMyAdmin (4.2.12) eingesetzt.

Um zusätzliche Sicherheit zu gewährleisten, sperrt Apache (mittels separater .htaccess-Restriktion) den Zugang mit einem eigenen Passwort. Darüber hinaus wird der Zugriff nur eingerichteten IP-Adressen gewährleistet. Die MySQL-Datenbank wird lediglich lokal dem Server zur Verfügung gestellt. Öffentlich erreichbar ist Port 80, zur Erreichbarkeit der Website bzw. API.

## Frontend

Das Frontend besteht aus einer Web-Applikation mit einem übersichtlichen Funktionsumfang, welche dem User die Einstellungsmöglichkeiten für die HMD-App ermöglicht. Der User kann so auf einem web-fähigen Gerät gewünschter Größe (z.B. ein Laptop, Tablet oder auch ein Smartphone) bequem alle Einstellungen für die App vornehmen, anstatt auf dem kleinen Display des HMD und mit dessen komplizierten Eingabemöglichkeiten hantieren zu müssen. Damit man das Frontend von verschiedenen Geräten verwenden kann, ist der Oberfläche entsprechend mit einem responsive Webdesign ausgestattet.

### Klassendiagramme

Für die Web-Applikation wurden zwei Klassen verwendet, User und Ingredient. Die Klasse User wird verwendet, um die Profileinstellungen zu realisieren. Mit der Klasse Ingredient werden die Einstellungen für die Allergien in der Blacklist umgesetzt.

Details können dem Klassendiagramm im Anhang entnommen werden.

### Libraries und Frameworks

Es wurde nur eine Library verwendet, diese ermöglicht die Darstellung des Barcodes zum koppeln des HMD. Da die Website nur einen geringen Funktionsumfang hat, wurden für das Frontend keine größeren Frameworks verwendet. Bei diesem Funktionsumfang lohnte sich eine Verwendung von großen Frameworks wie z.B. Angular oder Backbone nicht, vor allem aber auch weil die Teammitglieder noch keinerlei Erfahrung mit diesen Technologien haben. So besteht die Website grundlegend aus reinem HTML, CSS und JavaScript. Das Frontend greift über asynchrone Aufrufe mithilfe der AJAX-Technologie auf die in Kapitel 3.1 beschriebene RESTful API des Backends zu und bekommt so Zugriff auf die Daten in der Datenbank. Um die Implementierung zu vereinfachen, findet lediglich das Framework jQuery Verwendung. Außerdem finden keine Precompiler Anwendung.

## HMD Applikation

Die Vuzix M100 ermöglicht es dem Anwender mit dem Sprachbefehl „select“ einen Barcode einzulesen und damit zu überprüfen, ob der Anwender das Produkt verzehren darf oder nicht. Dies geschieht mit Hilfe eines RESTful-API Aufrufes, welcher den Authentifizierungscode und den gescannten Barcode mit der Datenbank vergleicht.

### Klassendiagramme

Die Vuzix Applikation besteht aus zwei Activities, zum einen, die Mainactivity zum anderen, die Scanactivity. Beide Activities greifen zusätzlich noch auf zwei weitere Klassen zurück.

Weitere Details können aus dem Klassendiagramm, im Anhang, entnommen werden.

### Libraries, Frameworks und Intents

Zum einscannen des Barcodes wurde ZXing („zebra crossing“) verwendet. Diese Library ermöglicht es den Scanner in einem neuen Intent zu starten und das Ergebnis weiter zu verarbeiten.

Außer der obengenannten ZXing Library wurden keine anderen Libraries oder Frameworks genutzt.

# Bewertung der Ergebnisse

Die zu Projektbeginn gestellten Anforderungen wurden, mit Ausnahme der „Nice to have“-Anforderungen, nur mit leichten Anpassungen umgesetzt. Die Gründe der Anpassungen lagen in Unterschiedlichen Quellen sind jedoch nicht so gravierend oder weitreichend um sie als verfehlt anzusehen. Die meisten „Nice to have“-Anforderungen der Android-Smartphone Applikation wurden durch das responsive Layout der Website abgedeckt.

Es ist zu erwähnen, dass eine „Should have“-Anforderung nicht umgesetzt wurde. F30.7 „Registriert sich ein User oder ändert er seine Emailadresse, muss er diese bestätigen. Zum Bestätigen wird eine Bestätigungslink an die Emailadresse versandt, die durch den User bestätigt werden muss.“. Diese Anforderung wurde aus technischen Gründen noch nicht umgesetzt, sollte sich aber das Produkt Edible verbreiten wäre dies ein wichtiger Punkt um die Sicherheit zu gewähren.

# Beschreibung des Projektablaufs

## Planung

Die Projektplanung begann mit der ersten Projektphase, der Planung. In dieser wurde einerseits evaluiert in welche Richtung wir mit dem ersten Prototyp, der aus der ersten Vorlesung existierte, fortfahren wollen. Außerdem wurden Meilensteine geplant mit groben Paketen, die bis zu den Meilensteinen fertig werden sollten. Wir hatten somit neben der Planung die Meilensteine Anforderungsanalyse, Design, Entwicklung und „Abnahme durch den Kunden“.

Für die Anforderungsanalyse haben wir zwei Wochen eingeplant. In dieser Phase wurde genauer definiert wie die Anwendung aussehen soll. Dafür wurden neben funktionalen und nichtfunktionalen Anforderungen, auch Aktivitäts- und UseCase-Diagramme angefertigt, sowie eine Aufwandsschätzung über die anstehenden Aufgaben. Damit konnte dann ein Pflichtenheft erstellt werden in dem die Aufgaben auch priorisiert wurden.

Die nächsten zwei Wochen waren eingeplant für die Architektur, sowohl der Software als auch der Hardware. Währenddessen stellten wir ebenfalls fest, dass wir ein Mockup für unsere Webseite benötigen, da zwar geplant wurde wie Webseite, Server und Vuzix miteinander interagieren können, aber nicht wie der User später mit der Webseite interagieren soll.

In der vierten Phase, die mit drei Wochen auch die längste war, wurden die einzelnen Komponenten erstellt. In dieser Phase wurde ein Server aufgesetzt, die Webseite mit Logik designt und programmiert, die API programmiert über die die Vuzix und die Webseite auf die Datenbank Zugriff bekommen und die App für das HMD. Der genauere Ablauf wird in Kapitel 4.2 dargelegt.

Bis zu „Abnahme durch den Kunden“ waren dann noch knappe zwei Wochen Zeit für Bugfixes, Nachziehen der Dokumentation, sowie für die Erstellung dieses Dokuments. Diese zwei Wochen waren absichtlich einkalkuliert, da wir aus bisherigen Projekten gelernt hatten, dass es sehr stressig wird, wenn der letzte Entwicklungstag und der Termin der Abgabe sehr nah bei einander liegen.

## Vorgehensmodell

Das Projekt wurde agil nach Scrum entwickelt. Dabei wurde sich streng an die Rollenverteilung und den Ablauf des Scrum-Frameworks gehalten. Leichte Anpassungen der Daily-Sprints wurden vorgenommen, da ein Projekt dieser Größe und dieses Umfangs keinen täglichen Daily-Scrum benötigt. Dies hat auch den Hintergrund, dass nicht täglich an den Issues gearbeitet wurde. Die Daily-Scrums wurden montags, mittwochs und freitags gehalten. Die Sprintwechsel wurden wöchentlich an Dienstagen gehalten.

Im Rahmen der Daily-Scrums wurden Fortschritt, Aussichten und Problematiken des aktuellen Issues angesprochen und bei Bedarf anschließend detaillierter besprochen. Die Besprechung der Problematiken wurde aus dem Daily-Scrum ausgelagert um diesen „kurz“ halten zu können und die Besprechung im Anschluss gezielt mit einzelnen Mitarbeitern durchführen zu können.

Die Sprintwechsel waren im Aufbau immer gleich. Zuerst wurden die Issues aus der Woche gemeinsam angesehen bzw. abgenommen, dadurch war jeder auf dem aktuellsten Stand und wusste genau über den Projektfortschritt Bescheid. Anschließend wurde eine Sprint-Retrospektive durchgeführt in der jeder Probleme, Schwierigkeiten oder Bedenken ansprechen konnte. Die Wünsche aus der Retrospektive wurden, wenn möglich, gleich im nächsten Sprint umgesetzt um die Zusammenarbeit im Team zu fördern.

Anschließend wurden neue Issues verteilt. Anfangs wurden die Issues horizontal verteilt, sodass ein Feature vom Backend bis hin zum Frontend in einem Sprint umgesetzt werden konnte. Diese Art der Verteilung hat sich nach dem ersten Sprint in dem Entwickelt wurde als nicht praktikabel herausgestellt, da manche Mitarbeiter auf das Ergebnis anderer warten mussten und es so zu zeitlichen Engpässen kam. Ab dem zweiten Sprint wurden Issues vertikal vergeben. So wurden in einem Sprint alle Issues aus einer Ebene verteilt umso die zeitlichen Abhängigkeiten von Mitarbeiter minimalisiert haben.

## Rollen

|  |  |
| --- | --- |
| Rolle | Mitarbeiter |
| Scrum Master | Marco Heumann |
| Communication Manager | Edwin Neubauer |
| Senior Requirements Analyst | Marco Schenkel |
| Documentation Manager | Edwin Neubauer |
| Senior Mobile Frontend Developer | Gregor Baumgärtner |
| Senior Web Frontend Developer | Hendrik Niemann |
| Senior Backend Developer | Moritz Bästlein |
| Test Developer | Marco Heumann |
| Security Administrator | Tim Bartel |
| Integration Manager | Hendrik Niemann |

## Testkonzeption

Ein agiles Projekt hatte auch agile Testmethoden zur Folge, weshalb in einem Sprint nicht nur die Funktion selber implementiert wurde sondern auch ein Test dazu geschrieben wurde. Allerdings ist keine Testabdeckung von 100% vorhanden. Außerdem war es nicht möglich zu allen Funktionen automatisierte Tests zu schreiben, was man an der API-Klasse erkennen kann. Des Weiteren wurden nur Backend-Funktion  
Applikation nur zwei Beispieltests geschrieben. Für das Frontend wurde sich nicht mit automatisierten Tests beschäftigt, da das nicht im Testkonzept vorgesehen war.  
  
Da das Backend in PHP programmiert wurde, sind die Tests mit dem Framework PHPUnit geschrieben worden. Es wurden keine komplizierteren Testmethoden benutzt sondern lediglich die Abfrage, ob das Ergebnis der Funktion dem erwarteten Ergebnis entspricht. Die Tests für die  
Applikation hingegen wurden mit JUnit geschrieben mit Android Erweiterung.  
  
Neben der Erstellung von Tests wurde beim Backend ebenfalls auf Continous Integration gesetzt. Unsere Tests wurden also bei jeder Änderung im Master-Branch wieder neu durchgeführt, um festzustellen ob Änderungen an einer Funktion auch Auswirkung hat auf andere Klassen  
und Funktionalitäten Travis, die das für Github-Projekte mit wenig Aufwand möglich  
machen.

# Fazit und Ausblick

Zusammenfassend lässt sich sagen, dass das Projekt Edible nach den festgesetzten Anforderungen erfolgreich umgesetzt wurde. Da das Vorgehensmodell agil orientiert war, konnten erforderliche Anpassungen problemlos umgesetzt werden.

Als zukünftiger Ausblick, um den Usern mehr Funktionalität und Umfang zu bieten, wären die nächsten Schritte eine Umsetzung zur Datenbeschaffung durch Crowdsourcing, eine Erweiterung der Blacklist nach Mengenangaben das den Anwendungsbereich erweitern würde und die Implementierung einer Android-Applikation für User ohne ein HMD.