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

Projekt Edible

Abschlussdokumentation

4.26.15

Inhalt

[1 Intention und Motivation 3](#_Toc417755038)

[1.1 Anwendungsbereich 3](#_Toc417755039)

[1.2 Zielgruppen 3](#_Toc417755040)

[2 Komponenten 3](#_Toc417755041)

[3 Anforderungen 4](#_Toc417755042)

[4 Architektur 4](#_Toc417755043)

[4.1 Backend 4](#_Toc417755044)

[4.1.1 Datenmodell 4](#_Toc417755045)

[4.1.2 Klassendiagramme 4](#_Toc417755046)

[4.1.3 Libraries und Framworks 4](#_Toc417755047)

[4.2 Backend 4](#_Toc417755048)

[4.2.1 Klassendiagramme 4](#_Toc417755049)

[4.2.2 Libraries und Frameworks 4](#_Toc417755050)

[4.3 HMD Applikation 4](#_Toc417755051)

[4.4 Klassendiagramme 4](#_Toc417755052)

[4.5 Libraries, Frameworks und Intents 4](#_Toc417755053)

[5 Bewertung der Ergebnisse 4](#_Toc417755054)

[6 Beschreibung des Projektablaufs 4](#_Toc417755055)

[6.1 Vorgehensmodell 4](#_Toc417755056)

[6.2 Rollen 4](#_Toc417755057)

[6.3 Testkonzeption 4](#_Toc417755058)

[7 Fazit und Ausblick 4](#_Toc417755059)

# Intention und Motivation

Das Ziel des Projektes Edible ist es eine Anwendung bereit zu stellen, mit der Allergiker ein Produkt auf seine Inhaltsstoffe prüfen können. Die Realisierung der Anwendung soll auf einem Head-Mounted Display (HMD) erfolgen, welches den Produktbarcode einliest und die Inhaltsstoffe per Web-Request von einem Server zugeschickt bekommt. 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.

# 

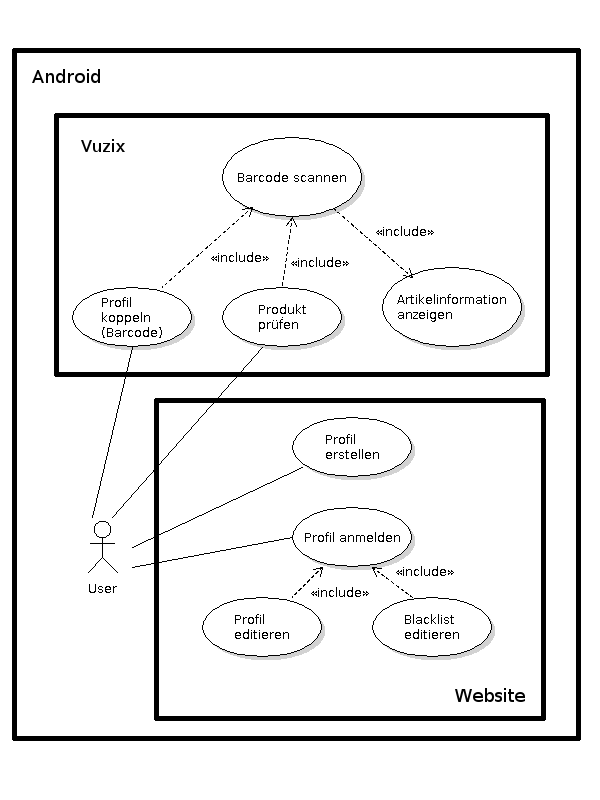
# Anforderungen

## 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

Auf Grund der Eingeschränkten Möglichkeiten an der HMD und dem Wunsch, dem Nutzer ein möglichst gutes Erlebnis mit seinem Gerät zu liefern, fiel die Entscheidung der dreiteiligen Architektur. Diese umfasst HMD Applikation, das Server Backend und die Webanwendung im Browser.

Die Bedienung komplexerer Anwendungen auf einem Wearable Device ist, auf Grund 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 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 tragbaren 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.

Als einziger Nachteil verbleibt bei dieser Aufteilung die benötigte Internetverbindung, während der Nutzer das Produkt verwendet. Wenn der Nutzer die Verbindung seines Handys benutzt, ist dies jedoch ein geringes 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 umgesetzt oder angepasst umgesetzt. Bis auf eine „Should have“ Anforderung wurden alle umgesetzt. Die Android-Smartphone Application wurde als „Nice to have“ formuliert und wurde zum größten Teil durch die Website in responsive Layout umgesetzt.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |

Tabelle 1: Funktionale Anforderungen

## 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. Dies geschieht mithilfe einer RESTful-API (Representational State Transfer Application Programming Interface), die über die Methoden GET, PUT, POST und DELETE des HTTP-Protokolls angesprochen wird. Zur Repräsentation von Daten wird JSON (JavaScript Object Notation) verwendet. Die Routen der API können im Anhang eingesehen 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.

### Klassendiagramme

Alle Klassen zur Realisierung der API-Routen (bereits erläuterte RESTful-API) sind in php implementiert. Dazu gehören vor allem API, DB, Ingredient, Product, Session und User. Desweiteren wurden hier zwei Klassen geschrieben, die zwischen internen (InternalError) und userspezifischen (UserError) Fehlern unterscheiden.

Details können dem Klassendiagramm im Anhang entnommen werden.

### Libraries und Framworks

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.

### 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. Apache HTTP Server (in der Version 2.4.10) deckt die Funktionalität des Webservers ab. MySQL-Server (5.4.42) die der Datenbank. Zu administrativen Zwecken wird zusätzlich phpMyAdmin (4.2.12) eingesetzt.

Um zusätzliche Sicherheit zu gewährleisten, sperrt Apache (mittels separaterr.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. Öffenlich erreichbar ist Port 80, zur Erreichbarkeit der Website bzw. API.

## Backend (...Frontend?)

### Klassendiagramme

### Libraries und Frameworks

## 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 verzerren 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 Application besteht aus zwei Activitys, zum einen, die Mainactivity zum anderen, die Scanactivity. Beide Activitys 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 weiterzuverarbeiten.

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

# Bewertung der Ergebnisse

Das Projekt ist sehr gut und besser als alle anderen

# Beschreibung des Projektablaufs

## 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 vom 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

|  |  |
| --- | --- |
| 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-Funktio  
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. Einige Anforderungen mussten zwar angepasst werden, wie beispielsweise F10.3 „…den spezifischen Sprachbefehl "scan" zum Starten des Produktscanners.“ oder auch F10.6 „…. Nach 10 Sekunden im Infoscreen kehrt die App automatisch in den Homescreen zurück.“. Gründe dafür sind zum einen die kostenpflichtige Erweiterung der Grammatik um den Sprachbefehl „scan“ realisieren zu können, zum anderen ein angepasster Workflow für eine verbesserte Usability. Da das Vorgehensmodell agil orientiert war, konnten diese 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 was den Anwendungsbereich erweitern würde und die Implementierung einer Android-Applikation für User ohne ein HMD.