Technology
Arts Sciences
TH Köln

Entwicklungsprojekt interaktive Systeme
Wintersemester 2017/2018

Implementationsdokumentation

& Installationsdokumentation

Dozenten

Prof. Dr. Gerhard Hartmann
Prof. Dr. Kristian Fischer

Mentoren

Robert Gabriel
Sheree Saßmannshausen

Von

Michael Michel – (11111440)
Steffen Owtschinnikow – (11081610)

Inhaltsverzeichnis

Implementationsdokumentation

Architekturmodell

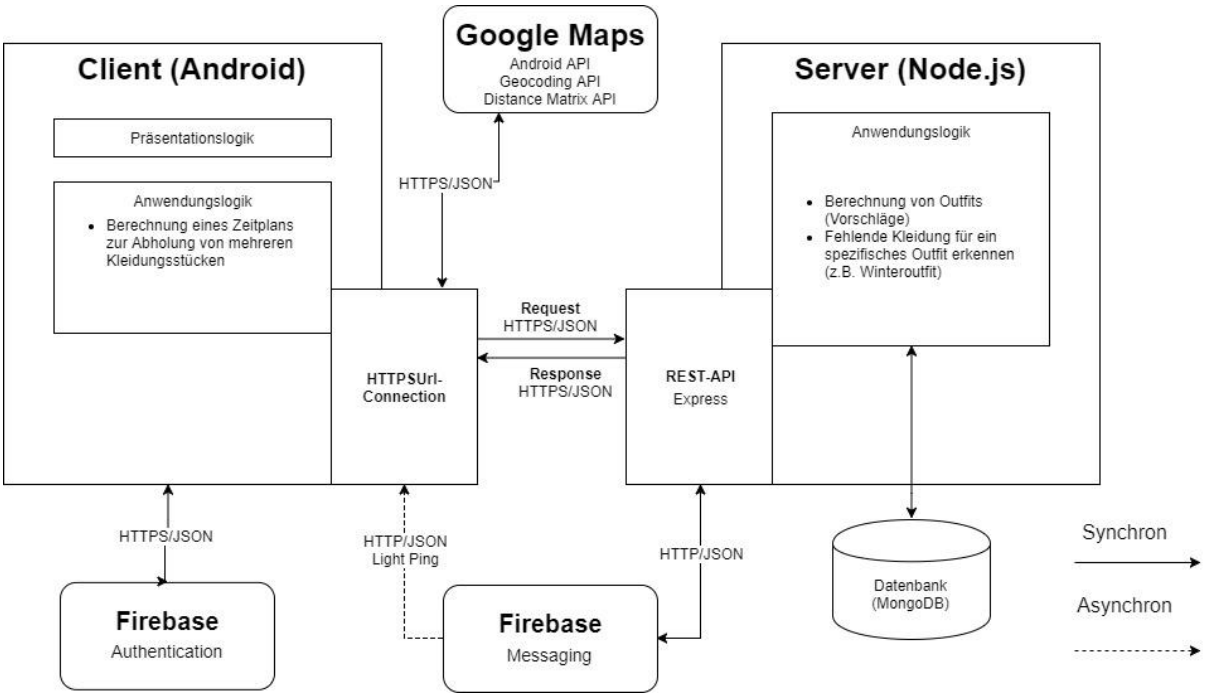

Abbildung 1: Geringe Iteration des Architekturmodells

Bei der Erstellung dieses Dokumentes und bei der erneuten Betrachtung des Architekturmodells wurde deutlich, dass bestimmte Anpassungen notwendig sind, um das System besser darzustellen.

Die Iteration des Modells beinhaltet nun eine Aufteilung des „Firebase Services“ in „Firebase Messaging“ und „Firebase Authentication“. Diese Repräsentation führt genauer die Komponenten auf, die von dem „Firebase Service“ genutzt werden.

Bei der „Google Maps API“ wurden ebenfalls die genauen API Namen aufgeführt, die genutzt wurden, da die „Google Maps API“ aus vielen unterschiedlichen Komponenten besteht. Eine Aufteilung in mehrere Blöcke wie bei Firebase ist hier aber nicht notwendig, weil nur der Client die Dienste nutzt.

Architekturmerkmale

Die Architekturmerkmale wurden schon ausführlich in der Projektdokumentation aufgeführt, im Folgenden sollen nun Abweichungen vom ursprünglichen Konzept erwähnt werden, die während der Implementation stattgefunden haben.

Es wurde unter anderem die Route (GET) „/clothingOptions“ eingeführt, die dazu dient die definierten Kleidungskategorien, Kleidungsgrößen usw. abzurufen. Diese Route wurde beim Client beispielsweise bei der Kleidungserstellung benutzt, um eine Auswahl zu bieten. Diese Variante ermöglicht es die verfügbaren Kategorien und Optionen an einer einzigen Stelle zu ändern und jeder Bereich, der diese Optionen erhält dann den selben Satz an Informationen.

Weitere Abweichungen betreffen die Suche nach Outfits. Dort wurde die Route angepasst, damit man die Kleidungsstücke nach der jeweiligen Kleidungsgröße und dem Geschlecht filtern kann, wodurch die Ergebnisse von größerer Relevanz für den Nutzer sind.

[/outfit/:Nutzungskontext/:Geschlecht/:Kopf-Größe/:Oberkörper-Größe/:Unterkörper-Größe/:Schuh-Größe/:Longitude/:Latitude/:Umkreis]

Hinzugefügt wurde ebenfalls eine Route um die Transaktionen eines Nutzers zu holen, diese Route ist vor allem relevant für den Zeitplan und die Anzeige der eigenen und fremden Anfragen.

[/user/:userID/:userToken/requests]

Routen und weitere definierte Merkmale, die enthalten sind werden hier nicht alle aufgeführt, aber sie umfassen die Funktionalitäten der Stiftungen und das Speichern von erstellten Outfits.

Absicherung von Routen

Zum Absichern der verschiedenen Routen wird der zum Userprofil gehörende Firebase Token verwendet. Dieser Token wird bei jedem erfolgreichen Login-Vorgang zum Server gesendet und dort gespeichert. Vor dem Aufruf bestimmter Aktionen wird der aktuell gültige Firebase Token angefordert und im Payload des Aufrufs aufgeführt. Durch einen Vergleich mit dem auf Serverseite vorhandenen Token kann überprüft werden, ob der Aufrufende einen gültigen Token vorweisen kann. Zusätzlich ist sicherzustellen, dass keine Token von Fremden eingetragen werden können, weshalb diese Aktion nur durch Beifügen des auf Client und Server hinterlegten 256-Bit-Schlüssel legitimiert wird. Diese Art der Absicherung ist keineswegs optimal. Eine anständige Implementierung eines bewährten Systems wie beispielsweise "JSON Web Tokens", sollte hier unbedingt eingefügt werden. Da auf das Verwenden von Bibliotheken weitestgehend verzichtet werden sollte und eine eigene Implementierung zu aufwendig gewesen wäre, wurde diese Art der schnellen Absicherung gewählt, um das System zum jetzigen Zeitpunkt nicht völlig schutzlos zu hinterlassen. Zum Einsatz kommt der temporäre Mechanismus zurzeit lediglich beim Aufruf von "Benutzer-Requests". Weitere Pfade müssen selbstverständlich ebenfalls geschützt werden.

Anwendungslogik

Zeitplan

Die Erstellung eines Zeitplans zur Abholung von Kleidungsstücken war der Anwendungslogikteil des Clients. Bei den ersten Prototypen wurde zunächst JavaScript genutzt, um eine groben Ablauf der Logik zu definieren. Nun musste es auf Java übertragen werden, was zunächst wegen Wissenslücken des einen Teammitglieds, problematisch war und mehr Zeit gekostet hat als gewollt. Bei der Umsetzung der Logik wurden dann aber mehrere Verbesserungen vorgenommen. Vor allem der letzte Schritt der Logik, die Festlegung der Termine, ist umfangreicher geworden. Eine Einbindung der Google Maps API für die Distanz zwischen zwei Koordinaten war geplant, wurde aber nun etwas anders eingesetzt. Nun wird mithilfe der API die Reise-Dauer zwischen zwei Koordinaten geholt. Mithilfe der Angabe der Dauer lassen sich die Termine präziser definieren. Da die API die vier Optionen „walking“, „driving“, „bicycling“ und „transit“ als Transportationsmethode bietet wurden diese auch genutzt und werden dem Nutzer als Option zur Verfügung gestellt.

Es wurde versucht möglichst viele Fälle mit dieser Logik abzudecken. Die wichtigsten Fälle, die aus Zeitgründen noch fehlen sind: Die momentane Uhrzeit in Betracht ziehen, Termine auf andere Tage verschieben und das Traveling Salesman Problem.

Outfitberechnung

Der Outfitberechner wurde eine Filterung von Kleidungsstücken basierend auf dem Geschlecht und der Kleidungsgrößen eingebaut, damit ein User seine Größen und sein Geschlecht eingeben kann und somit auch ein Outfit erhält, das ihm passt. Ebenfalls wurde der Distanzrechner in die Logik des Outfitrechners eingebunden und es ist nun möglich, auch einzelne Kleidungsstücke daraufhin zu prüfen, für welchen Nutzungskontext sie eingesetzt werden können, um Benutzern, denen bestimmte Kleidungsstücke fehlen zu benachrichtigen, wenn ein passendes Kleidungsstück eingestellt wurde.

Implementierung

Client

Beim Client lag der Fokus auf der Umsetzung des Prozesses der Weitergabe von ungenutzten Kleidungsstücken. Vor allem die Einbindung der Anwendungslogik war wichtig. Die *TimePlan.class* setzt die Anwendungslogik des Clients um und wurde schon im Abschnitt „Anwendungslogik“ thematisiert. Bei der *ShowOutfit.class* sollten die Ergebnisse der Anwendungslogik des Servers präsentiert werden und Möglichkeiten gegeben werden mit diesen Ergebnissen weiter zu handeln. Ein Filter wurde bereitgestellt, um Ergebnisse zu liefern die auch relevant sind (Entfernung zum User, Kleidungsgrößen). Der Code ist bei diesem Teil leider ziemlich aufgebläht, da für jeden Bereich des Outfits eine RecyclerView erstellt wurde und einzeln initialisiert und befüllt wurde. Ein Lösungsansatz der zeitlich nicht mehr möglich war, ist es einen Adapter zu schreiben, der sich darauf anpasst für welchen Bereich Ergebnisse geliefert wurden. So würde man viel Redundanz vermeiden. Problematisch war auch der Filterdialog. Dieser wurde ziemlich spät geschrieben und musste schnell fertig sein, deswegen ist dort auch sehr viel redundanter Code, der mit mehr Zeitaufwand eleganter lösbar ist. Der Bereich der *ShowRequest.class* lässt sich auch mit der TimePlan.class kombinieren, um es kompakter zu machen. Es wurde bei der Implementierung getrennt gehalten, um Übersichtlichkeit zu gewährleisten, aber kann mit mehr Zeit auf *ShowRequests.class* übertragen werden, weil beide Bereiche dieselben Informationen als Grundlage nehmen, nämlich die Transaktionen. Was leider auch zu spät bemerkt wurde war die Verwendung von *Fragmenten*. Aufgrund von keinerlei Android-Erfahrung wurden bei der Implementierung nur *Activities* benutzt und erst gegen Ende, wo es schon etwas zu spät war, wurde herausgefunden, dass es sowas wie Fragmente gibt.

Durch den hohen Zeitaufwand bei einigen Bereichen des Clients sind Bereiche wie die Bewertung der Benutzter und die Anzeige von Subscriptions zu kurz gekommen. Es konnten auch nicht alle Routen des Servers in die Anwendung eingebaut werden, z.B. Delete-Funktionen für das eigene Profil oder ein Kleidungsstück sind nicht implementiert. Planmäßig sind auch die Funktionen für die Stiftungen ausgefallen, da diese keinen Einfluss auf die Anwendungslogik hatten und die Prioritäten bei den Funktionen der Anbieter und der Suchenden lag.

Client Probleme

In diesem Abschnitt sollen die noch existierenden und entdeckten Probleme des Clients aufgeführt werden. Der Abschnitt soll Klarheit schaffen falls man beim eigenen Ausführen der Anwendung auf dieselben Probleme stößt.

1. Wenn man in den Bereich der Requests geht und dort beispielsweise etwas an einem Request in „Foreign Requests“ ändert und der gegenüberstehende User dieser Request im selben Moment im Bereich „My Requests“ ist und auch etwas an dieser Request ändern möchte, beispielsweise „Delete“, dann gibt es ein Problem, da Änderungen der Request nicht aktualisiert werden, wenn die Request schon einmal geladen wurden. So kann es passieren das Operationen auf Requests ausgeführt werden können obwohl sie eventuell schon gar nicht mehr existieren.
2. Beim Time Plan gibt es ein Problem, dessen Ursprung nicht gefunden werden konnte. Betritt man einmal diesen Bereich verlässt ihn und betritt ihn erneut, dann crasht die Anwendung.
3. Wenn man sich einen Account anlegt wird per E-Mail-Authentifizierung der Account bestätigt und aktiviert. Momentan gibt es noch keine Möglichkeit diese Mail erneut zu senden, falls das erste Mal nicht erfolgreich funktioniert hat.
4. Manche Bereiche wie „Search Clothing“ oder „Time Plan“ brauchen zu Beginn die eigenen Koordinaten und das Erhalten dieser Koordinaten ist zeitlich immer sehr unterschiedlich, deswegen dauert es manchmal noch viel zu lange.

5. Aufgrund der Problematik, dass HTTPsUrlConnection nicht ohne Weiteres mit ungeprüften SSL-Zertifikaten zusammenarbeitet, wurde auf Seite des Clients ein "Trust Manager" installiert, der jedes Zertifikat akzeptiert. Mit der Verwendung dieser Einstellungen ist die Sicherheit der Verschlüsselung nicht gewährleistet. Vor einer Verwendung des Systems muss der "Trust Manager" unbedingt entfernt und ein geprüftes SSL-Zertifikat verwendet werden.
6. Beim Login/Registrierung gibt es ein Problem, dass man sich zwei Mal einloggen kann, weil nicht geprüft wird, ob ein User schon eingeloggt ist.

Server

Der Server wurde größtenteils wie geplant umgesetzt, Änderungen und der gleichen wurden in vorherigen Abschnitten aufgeführt.

Server Probleme

Der Server zeigt im Bereich der Verteilung von Code Mängel. Viele Operation finden im Bereich der Datenbank (database.js) statt, die besser von Funktionen erledigt werden sollten, welche die einzelnen Datenbankabfragen initialisieren (getOutfit.js, getClothing.js,..). Darüber hinaus sollten wiederverwendbare und anpassungsfähige Funktionen für die einzelnen Datenbankabfragen entwickelt werden.

Design

Das Design konnte, so wie es im „Detailed User Interface Design“ definiert wurde, nicht eingehalten werden, weil unteranderem der Fokus auf den Funktionalitäten lag und so eine vollkommene Umsetzung zeitlich nicht möglich wäre. In der gegebenen Zeit wurde aber dennoch versucht die Designvorgaben, auch wenn nur im gering Umfang, umzusetzen. So ist beispielsweise die Activity für die Kleidungsstückerstellung relativ nah an der Vorgabe. Andere Activities weichen von den Vorgaben stark ab, weil die Implementierung der „Cards“ nicht auf Anhieb funktioniert hat und so ausgelassen wurde, um keine Zeit für die Problemlösung aufzuwenden.

Da die vorhandene Design-Zeit hauptsächlich für die Bereiche selber genutzt wurde, war es nicht mehr möglich ein Seitenmenü einzufügen. Die ganzen Funktionspunkte im Hauptmenü sollten in einem Seitenmenü platziert werden.

Installationsdokumentation

Client

1. Repository downloaden oder klonen https://github.com/micha997/EISWS1718MichelOwtschinnikow
2. Android Studio 3.0 installieren und das Projekt „MS3\Implementation\client“ öffnen
3. Nachdem Android Studio den workplace eingerichtet hat in der Datei „res/values/strings.xml“ bei „DOMAIN“ die Adresse auf die eigene IP ändern
4. Oben rechts auf den grünen „Run“-Pfeil klicken
5. Bei der Auswahl des „Deployment Targets“ empfiehlt sich ein per USB angeschlossenes Android Device *(Hinweis: Die App wurde hauptsächlich auf einem richtigen Android Device getestet und ausgeführt.)*
6. Nachdem die APK installiert wurde bei dem Android Device unter Settings>Apps>[App-Name]>Permission alle Berechtigungen aktivieren
7. (App sicherheitshalber neustarten)
8. Einloggen mit einem der folgenden Accounts oder einen eigenen Account erstellen
   - E-Mail: eistest1@endrix.org | Passwort: password
   - E-Mail: eistest2@endrix.org | Passwort: password
   - E-Mail: eistest3@endrix.org | Passwort: password

   Wird ein eigener Account erstellt, dann muss dieser nach der Registrierung zunächst per E-Mail-Authentifizierung aktiviert werden. Zum schnellen Testen empfiehlt sich https://temp-mail.org für eine temporäre E-Mail.

Server

1. Repository downloaden oder klonen https://github.com/micha997/EISWS1718MichelOwtschinnikow
2. In den Ordner \EISWS1718MichelOwtschinnikow\MS3\Implementation\server navigieren und Command Prompt öffnen
3. Den Befehl „npm install“ ausführen (node.js muss installiert sein) und die Module installieren
4. Das Zertifikat muss nun ausgetauscht werden. Dazu müssen die bereits eingefügten Dateien im Ordner /server/keys ersetzt werden. Die Datei server.crt enthält dabei die Zertifikatsdaten und key.pem den privaten Schlüssel.
5. Nachdem die node Module installiert und die Zertifikatsdaten ausgetauscht wurden den Befehl „node server.js“ ausführen
6. Der Server sollte nun unter der eigenen IP und dem Port 50262 erreichbar sein

Server Datenbank

Als Bereitsteller der Datenbank, die innerhalb des Systems zum Einsatz kommt, dient zum jetzigen Zeitpunkt die öffentliche Plattform www.mlab.com . Sollte die Benutzung einer anderen Datenbank gewünscht sein, kann diese in der Datei app.js im Wurzelverzeichnis des Serversystems mithilfe der Konstante "mongoUrl" definiert werden.