Hochschule Luzern
Departement für Informatik

Projekt FoodPrint

Programmieren fürs IOS

Studierende: Frederico Fischer, Nico Iseli
Dozenten: Prof. Dr. Ruedi Arnold, Nicolas Märki
Abgabedatum: 13. Dezember 2020

iOS-App

Webservice

1 Einleitung

Das vorliegende Projekt hat zum Ziel, eine App zu entwickeln, um Menschen mehr zu unweltfreundlichen Einkäufen zu ermutigen. Die App soll dabei bedürfnisbasiert regionale und saisonale Produkte vorschlagen. Die Implementierung ist vorerst für Benutzer aus der Schweiz vorgesehen. Das Datenbankschema im Backend ist jedoch so aufgebaut, dass sich die App auch auf weitere Länder erweitern lässt.

2 Architektur

Die Architektur ist so aufgebaut, dass die benötigten Produktdaten über eine REST-API zur Verfügung gestellt werden. Die Produktdaten werden über HTTP-Requests geladen und auf der App mittels Core-Data persistiert wobei die Daten auch gleich abgeglichen werden. Neben den Produktdaten werden die User-Einstellungen ebenfalls mittels Core-Data persistiert, wobei man diese nur App-Intern verwaltet. Die User-Daten werden genutzt, um die Produkte userspezifisch zu filtern. Das User-Interface wurde mit dem UI-Kit aufgebaut. Die Architektur ist ebenfalls als Abbildung (vgl. Abbildung 1 auf Seite 2) festgehalten.

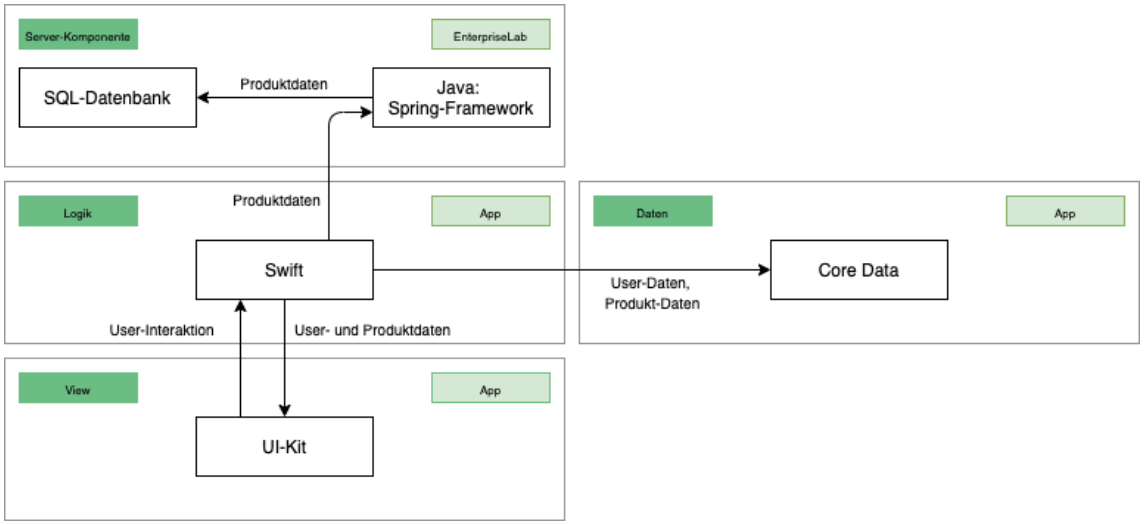

Abbildung 1: Architektur,
Quelle: Projektteam

3 Feature-List

- **User-Settings:** Bedürfnisorientierte Einstellungen, wie die Produkte gefiltert werden sollen
- **Userspezifische Produktauflistung:** Auflistung der gefilterten Produktdaten gemäss den User-Einstellungen
- **Produkt-Details:** Detail-View für jedes Produkt, in der ein vertiefterer Einblick möglich ist
- **About-View:** View mit Informationen über die Intention und Funktionsweise der App

4 Umsetzung technischer Anforderungen

4.1 Server-Komponente

Für die Server-Komponente wird eine Ressource im EnterpriseLab verwendet. Als Datenbank verwendet man MariaDb und als Java-Framework greift man auf SpringBoot zurück. Die Anbindung der Datenbank zum SpringBoot wurde mittels der JPA-Schnittstelle (OR-Mapper) umgesetzt. Bei der Datenbank wurde Wert darauf gelegt ein erweiterbares Schema zu definieren, da eine Weiterentwicklung der Applikation geplant ist. Über einen Rest-Controller werden die Produktdaten dann im JSON-Format zurückgegeben (ersichtlich unter: `http://ffischer-ios-h20.el.eee.intern:8080/api/v1/products/Schweiz`). Aktuell läuft der Server nicht in der DMZ. Daher ist ein Aufruf nur mit bestehender VPN-Verbindung im HSLU-Netzwerk möglich.

Code-Referenz: `foodprint-backend-distribution`

4.2 HTTP-Kommunikation

Die JSON-Daten des Webservers werden über den Aufruf der URL geladen und mittels des JSON-Decoders von Swift in ein Array von Swift-Objekten konvertiert. Dabei wird mit Data Transfer Objects gearbeitet, welche während der Konzipierung der API definiert wurden.

Code-Referenz:
`FoodPrint/FoodPrint/ViewControllers/ProductListViewController.swift`
In diesem File ist der entsprechende Code unter dem Abschnitt **„MARK: - API“** ersichtlich.

4.3 Persistierung mit Core-Data

Mittels Core-Data werden die User-Daten sowie die Produktdaten lokal persistiert. Die Produktdaten werden vor der Persistierung noch mit einer entsprechenden Logik verarbeitet da die gefetchten DTOs noch nicht dem entsprechend, was im User-Interface schlussendlich angezeigt werden soll. Die App verwaltet dabei ingesamt einen User und mehrere Produktdaten. Diese können gelesen (über die Fetch-Methode im NSManagedContext) und persistiert (über die Save-Methode im NSManagedContext) werden. Der persistierte User dient dazu, die eigenen Bedürfnisse zu ändern und demzufolge die entsprechenden Produktdaten zu filtern. Während die gesammten Produktdaten im Core-Data persistiert werden, wird die Filterung In-Memory vorgenommen.

Code-Referenz:
`FoodPrint/FoodPrint/Utils/CoreData.swift`

4.4 Unit-Tests

Unit-Tests kamen insbesondere bei der Logik zur Anwendung. Damit die Daten richtig gefiltert werden, war es wichtig, verschiedene Fälle zu berücksichtigen. Daher versuchte man, möglichst alle Ausnahmefälle zu evaluieren und somit verschiedene Tests für die unterschiedlichen Funktionen durchzuführen.

Code-Referenz:
`FoodPrint/FoodPrintTests/FoodPrintTests.swift`

4.5 Auto-Layout

Für das Projekt wurden die verschiedenen Inhalte der einzelnen View-Controller mittels Auto-Layout formatiert. Dafür wurden entsprechende Constraints gesetzt. Die Nutzung dieses Features wurde jedoch zu wenig gut geplant. Dieses wurde ohne grosses Vorwissen genutzt, was dazu führte, dass die Constraints mehrmals gelöscht und wieder neu erstellt werden mussten. Zudem wurden auch zu wenig Geräte getestet. Aktuell sieht die Formatierung auf neueren beziehungsweise grösseren Geräten gut aus. Hingegen auf kleineren Geräten (beispielsweise auf dem I Phone SE) überlappen sich einige Inhalte. Um dies zu beheben, wurde versucht, im Nachhinein noch eine Scroll-View zu integrieren. Diese kam dann jedoch mit den bestehenden Constraints in Konflikt. Daher ist die App aktuell nicht auf alle Geräte ausgelegt.

Code-Referenz:
`FoodPrint/FoodPrint/Utils/Main.storyboard`

Erkenntnisse

Das Erstellen einer eigenen App war eine sehr interessante Erfahrung. Das Wissen über die iOS-Plattform sowie die Programmiersprache „Swift“ hat sich im Team durch das Projekt stark verbessert. Während der Entwicklung sind viele kleine Fehler als auch architektonische Problem aufgetaucht, ohne die man im Nachhinein sicherlich effizienter gewesen wäre. Ein Beispiel dafür ist die Nutzung vom Auto-Layout. Ein Problem dabei war es, dass zu schnell damit begonnen wurde. Es mussten immer wieder Constraints gelöscht und neu erstellt werden. Ein anderes Beispiel ist das Datenmanagement. Hier wurden die bisherigen Produktdaten von Hand abgetragen, weil keine passende Quellen gefunden wurden. Kämen neue Produkte dazu, müsste die Datenerhebung überdacht werden. Eine weitere Hürde war auch das Aufsetzen der Logik. Diese wurde unterschätzt. Beim Filtern der Daten gab es diverse Ausnahmefälle, die betrachtet werden mussten. Dazu gehört beispielsweise die Auseinandersetzung mit den Saisonzeiten der Produkte. Einerseits wurde anfangs nicht berücksichtig, dass man dieses Array auch unsortiert erhalten könnte. Andererseits musste die Kalkulation der Saison mehrmals getestet werden, da der Übergang vom Dezember zum Januar ein spezieller Fall ist. Zuletzt war die Persistierung der Produktdaten ebenfalls eine Herausforderung. Damit die Daten der API auf der App nicht jedesmal neu persistiert werden, wurde eine entsprechende Überprüfung eingebaut. Aktuell ist es so, dass effektiv jedes Produkt nach deren Existenz im Core-Data überprüft wird. Dies könnte man effizienter lösen in dem man bspw. mit Hashes auf der Serverseite arbeitet. Abgesehen davon ist man aber mit dem Resultat sehr zufrieden und motviert, die App auch im Anschluss an die Projektabgabe weiterzuführen.