Fortgeschrittenenpraktikum

Portierung der Bavarian App in Ionic

Patrick Frömel, Tim Romonath

Unter Anleitung von Prof. Christoph Bockisch und PD. Dr. Hanna Fischer

30. Oktober 2021

Inhaltsverzeichnis

1 Kurzbeschreibung

Die App „Welcome to Bavaria“ in der Version 3.0 ist eine Portierung der gleichnamigen Version 2.0 auf das Ionic-Framework um diese auf verschiedenen Betriebssystemen - insbesondere iOS und Android - mit geringen Entwicklungsaufwand zur Verfügung zu stellen. Die App wurde dabei im Zeitraum von Juli bis Oktober 2021 Entwickelt.

Die App beruht auf dem gedrucktem Heft „Migraboarisch - Von uns fia eich. Ein Wegweiser durch den Dialekt“ welcher im Schuljahr 2016/17 von Kelheimer Schülern Entwickelt wurde. Ziel war es dabei Neuankömmlingen in Bayern den Bayrischen Dialekt näher zu bringen.

Die App wurde in der Version 1.0 von Marburger Studenten 2019/20 entwickelt und um verschiedene Sprachaufnahmen ergänzt. Für die Version 2.0 wurden weitere Sprachaufnahmen und ein primitives Archivement-System eingebracht.

1.1 Zur Portierung

Es wurde sich für Ionic mit dem Angular-Framework entschieden um die Vorteile der Webentwicklung bezüglich Ihrer Anpassungsfähigkeit in verhältnismäßig einfachen Frameworks zu nutzen und damit die Reichweite der App zu erweitern. Sogenannte „Web-Apps“ sind weit verbreitet und stellen - vereinfacht ausgedrückt - Websites in speziellen Browsern dar so dass der Quellcode auf unterschiedlichen Betriebssystemen stets der selbe bleiben kann. So müssen bei der Wartung nicht verschiedene Programme parallel Entwickelt werden.

Ionic mit Angular wird in dieser Kombination von quasi allen Seiten propagiert und wurde in unserer unbedarftheit dankend angenommen. Ionic stellt viele mehr oder minder komplexe Bausteine zur Webentwicklung bereit (von Textfeldern über Knöpfen bis hin zu Slidern mit Animation) während Angular zum einen HTML um viele nützliche „Funktionen“ erweitert (Loops, If-Else,...) und zum anderen komplexe Mechanismen im Hintergrund verwaltet so dass beispielsweise eine Website mit verschiedenen wechselnden Komponenten hoch dynamisch gestaltet werden kann.

In der Portierung wurde versucht sich an S.O.L.I.D-Prinzipien zu halten, auch wenn dies mit Type-Script als Programmiersprache bisweilen kompliziert gestaltete. Während die Datenbank-Mechanismen noch verhältnismäßig gut ohne Abhängigkeiten funktionieren und dank Schnittstellen (Interfaces) auch einfach ausgetauscht werden können sind die Seiten der UI sehr eng an verschiedene „Services“ gebunden welche wiederum untereinander stark abhängig sind. Dies ist aber keine Notwendigkeit und das refactoren dieser kann Gegenstand eines Nachfolgeprojekts sein. Kern der Bemühungen ist ein „Request-Response“-Pattern in welchen ein Request über ein Interactor gestellt wird welcher wiederum von den Datenbankmechanismen eine Response-Methode aufrufen. Sehr großer Vorteil ist hier das sich die asynchrone Programmierung in Grenzen hält - eine Aufgabe kann und wird auch erst dann zuende gebracht wenn eine Antwort vorhanden ist.

Alle Daten der App liegen als Exceltabellen vor. Um diese besser verarbeiten

zu können werden die Daten in einer Datenbank abgespeichert. In der App 2.0 wurde dies durch eine SQL-Datenbank realisiert. Daher war auch unsere erste Herangehensweise eine SQL-Datenbank zu verwenden, um die Modellierung der DB und der Abfragen von der Version 2.0 wiederverwenden zu können. Diese Idee wurde aber verworfen, da eine NoSQL-Datenbank (IndexedDB) schon in den Browser eingebaut ist und somit wesentlich einfacher zu erreichen und auch zu testen ist. Dies ruht daher, dass wir so auch die App in einem normalen Browser ausführen konnten, anstatt sie jedes mal für ein Mobilgerät neu „builden“ zu müssen. Des weiteren hat sich herausgestellt, dass die benötigten Anfragen auf die Datenbank eine relativ niedrige Komplexität aufwiesen.

1.2 Das Team

Leitung

Hanna Fischer

Koordination und Betreuung

Hanna Fischer, Christoph Bockisch, Peter Kaspar

Systementwicklung

Version 3.0: Patrick Frömel, Tim Romonath
Version 2.0: Lea Fischbach
Version 1.0: Benedikt Batton, Lester Brühler, Lea Fischbach, Hannah Greß, Anh-Thy Truong, Qi Yu

Mitarbeit

Milena Gropp

Sprachaufnahmen

Melanie Bajrami – Regensburg (Nordmittelbairisch)
Christina Böhmländer – Zwiesel (Nordmittelbairisch)
Felicitas Erhardt – Walleshausen (Schwäbisch-Mittelbairisches Übergangsgebiet)
Christian Ferstl – Regensburg (Nordmittelbairisch)
Edith Funk – Krumbach (Schwäbisch)
Markus Kunzmann – Mühldorf am Inn (Mittelbairisch)
Barbara Neuber – Altenstadt an der Waldnaab (Nordbairisch)
Michael Schnabel – Bayreuth (Ostfränkisch)

Übersetzung

Arabisch: Clara Mikhail
Englisch: Jeffrey Pheiff
Tschechisch: Andrea Königsmarková

Grafiken

Udo Butler

Beteiligte Lehrerinnen und Schülerinnen

Peter Kaspar, Daniela von Schultz
Karam Alayoubi, Sali AlBoulad, Yazan Aljleilati, Ahmad AlShalabi, Mustafa AlShalabi, Arreeya Banlo, Anita Blaha, Nicolas Bruckmaier, Florian Fahle, Nico Fichte, Lukas Fischer, Tina Föhre, Nico Horn, Aisa Jusic, Alexej Kobzev, Karina Kluger, Max Kohlmeier, Samuel Konschelle, Damian Kowalski, Andreas Lening, Laura Lohwasser, Markus Müller, Felix Neuhauser, Franziska Nigl, David Nguyen, Samuel Noy, Daniel Rißmann-Toledo, Lena Ritzinger, Anna Roidl, Joshua-David Roland, Heba Sandafi, Julian Schanderl, Andreas Stemmer, Wahyu Susilo, Liviu Vasiu, Mihai Vasiu, Maria Waldhier, Markus Weber, Stefanie Weber, Amadeus-Daniel Wiselka, Burak Yılmaz, Andreas Zellner, Tobias Zinkl, Thomas Zirmer

2 Entwicklungsprozess

Der Entwicklungsprozess selbst wurde zunächst keiner komplexen Systematik unterworfen. Es gab die Trennung, dass Patrick Frömel die Datenbankmechaniken und Interfaces inkl. Interactor entwickelt während Tim Romonath sich in Ionic und Angular einarbeitet und das User-Interface entwickelt.

Die ersten 5 Wochen haben wir an den Wochentagen in den Semesterferien uns morgens getroffen, den aktuellen Stand besprochen und dann 6 - 8 Stunden entwickelt. Anfangs haben wir mit Agantty (Webtool zur Aufgabenplanung) gearbeitet, aber schnell festgestellt dass dies nicht unseren Bedürfnissen entsprach. Danach folgte eine Pause von 4 Wochen in welcher wir anderen Dingen nachgehen mussten. Nach dieser Pause haben wir nur schwer wieder den Einstieg in das Projekt gefunden. Zur Unterstützung haben fortan mit Trello gearbeitet, ein einfaches Webtool für Kanban-Boards.

2.1 Agiles vorgehen

Unser vorgehen kann am ehesten noch als „agil“ beschrieben werden. Wir haben die zugrundeliegende App Stück für Stück nachgebaut - zunächst einen Funktionierenden Kern (Datenbankmechaniken, auslesen und übertragen dieser in die UI - Soundwiedergabe, grundlegendes Routing usw.), dann wichtige Kernfeatures (Bilder, Fortschritt, Archievements, usw.) und nach und nach alle anderen implementierten Features (Dialektauswahl, App zurücksetzen, usw.).

Allerdings ist unser Vorgehen nicht besonders inkrementell und iterativ gewesen - eher modular chaotisch. Wir haben im Grunde gemacht was gerade anfiel und für den Großteil des Projekts auf Projektmanagement verzichtet. Das hat ob der geringen Komplexität und einer verhältnismäßig guten Codequalität funktioniert - ist aber für zukünftige Projekte nicht ratsam. Ein wirkliches, agiles Vorgehen kam erst am Ende mit Trello auf - dort wurden Aufgaben in einem Backlog gesammelt und nach und nach fertig gestellt. Allerdings gab es zu diesem Zeitpunkt nur noch ein Zwischenziel: Fertig werden.

2.2 Verwendete Tools

Agantty

Agantty ist ein Webtool um Projektmanagement zu unterstützen. Es implementiert dabei ein Gantt-Diagramm, welches zeitliche Aktivitäten als Balken darstellt. Wir haben Agantty sehr schnell verworfen, da unser prinzipiell agiles Vorgehen nicht gut mit dieser Art Diagram - welches besser zu fest geplanten Projekten mit klaren Zeiten und Zwischenzielen passt (vgl. Wasserfallmodell).

Gittea Kanban-Boards

Vor der Etablierung von Trello haben wir uns erkundigt wie die Kanban-Boards von Gittea genutzt werden können. Diese sind im Grunde eine grafische Organisationsmöglichkeit von Issues was durchaus viele Vorteile mit sich bringt - wie beispielsweise alle bekannten Gittea-Tools zum Monitoring oder die Aufgabenverteilung direkt im Repository - aber auch einen großen Nachteil weswegen wir

uns dagegen entschieden haben: Für jedes Issue bekomme Beobachter des Repositorys eine Email. Beim Sammeln der fehlenden Features wäre eine regelrechte Email-Flut auf die betreuenden Personen losgebrochen.

Trello

Wir haben zum Ende des Projekts mit Trello gearbeitet - eine einfache Implementation eines Kanban-Boards. Zu dem Zeitpunkt der Einführung von Trello existierte bereits ein grober Kern der App in Form von Datenbankmechaniken, dem grundlegenden User-Interface, ordentlichen Interfaces für Datenklassen und einer Schnittstelle zwischen Datenbank und User-Interface. Das Vorgehen Fehlende oder unvollständige Features zusammen mit Dokumentationsaufgaben in einem Backlog zu sammeln welches auch weiterhin mit Bugs und anderen Auffälligkeiten gefüllt wurde hat sehr zur Übersichtlichkeit des Projekts beigetragen. Wir haben mit 3 Bereichen gearbeitet: Dem Backlog - einem Pool an noch zu erfüllenden Aufgaben, einem Bereich für Aufgabe an denen gerade gearbeitet wird und einem für Aufgabe wo die Weiterarbeit gerade nicht möglich ist.

2.3 Verlauf

Die erste, „modular chaotische“ Phase war durch hohe Arbeitsbereitschaft geprägt. Durch ständiges und langes arbeiten am Projekt waren wir stets auf der Höhe des Projekts und hatten damit keine größeren Probleme mit fehlender Organisation. Wir haben uns direkt zu Beginn auf bestmögliches Einhalten der S.O.L.I.D.-Prinzipien geeinigt und daher viel Wert auf sinnvolle Interfaces gelegt. Wir haben zwar versucht mit Agantty eine Art Projektmanagement zu betreiben, dies ist aber am falschen Tool gescheitert.

Die zweite, „optimistische“ Phase trat nach einer arbeitsbedingten, mehrwöchigen Pause ein. Wir hatten den Überblick über das Projekt verloren könnten diesen aber mit Trello zurückgewinnen. Leider waren wir beide auch in dieser Zeit im Broterwerb eingespannt wodurch das Projekt nun zwar organisiert, aber langsam voran ging.

Die dritte, „crunchige“ Phase war von der eiligen und unsauberen (im Sinne von „schlecht wartbarer“) Implementierung letzter Features und dem fixen einiger Bugs geprägt. Nebenbei musste noch dieser Bericht geschrieben werden und es stellte sich eine gewisse Müdigkeit ein.

2.4 Fazit

Es wäre sehr viel Sinnvoller gewesen sich gleich zu Beginn des Praktikums mit Projektmanagementmethoden zu beschäftigen um Vorzüge etablierter Herangehensweisen zu nutzen. Mit Trello zu beginn des Projekts und von SCRUM adaptierten Sprints wäre das Ziel ein wenig klarer gewesen.

Die lange Pause zwischen der ersten und zweiten Projektphase war ein großer Fehler - wenn auch kein vermeidbarer. Es fiel uns sehr schwer wieder in dem Projekt Fuss zu fassen. Dabei garnicht so sehr vom Fachlichen her sondern eher rein von der Motivation. Das Projekt wirkte bereits so fern und so erledigt was

ein großer Trugschluss war. Wir sind überzeugt mit 6 oder 7 Wochen am Stück mehr erreicht haben zu können als wir schlussendlich erreicht haben.

3 Verwendete Technologien

In diesem Projekt kamen einige unterschiedlichen Frameworks zum Einsatz, welche als npm-Pakete in Node.js installiert wurden. Als IDE haben wir JetBrains WebStorm genutzt, eine zur Webentwicklung modifizierten Version von Intellij. Das Management der Daten erfolgte in IndexedDB und LocalStorage.

3.1 Node.js

Node.js ist eine JavaScript-Laufzeitumgebung um JavaScript-Code außerhalb von Webbrowsern ausführen zu können. Durch die ereignisgesteuerte Struktur von JavaScript erwachsen einige Performancevorteile beispielsweise auf Webservern. Node.js benutzt den Paketmanager npm (Node Package Manager) um weitere Module zu installieren.

3.2 Wichtige Node.js Module

Ionic

Ionic ist ein Framework um Hybride Apps als Progressive Web App zu entwickeln. Es konzentriert sich auf das Front-End, welches wir hier im Bericht als User-Interface bezeichnen. Ionic stellt verschiedene interaktive Elemente wie beispielsweise Buttons, Slider, Grids usw. zur Verfügung.

Angular

Angular ist ein Framework um die Entwicklung des Front-Ends dynamischer, TypeScript basierter Webb Apps zu unterstützen. Es wird der HTML-Code mit direktiven erweitert um sich wiederholende Aufgaben optimiert zu implementieren (Schleifen, If-Else).

Cordova

Cordova ist ein Framework welches aus JavaScript und HTML-Code äquivalenten, gerätespezifischen Code generiert (in unserem Fall Java und XCode).

3.3 Datenbanken

IndexedDB

Die Datenbank die wir verwendet haben um die Daten der Exceltabellen zu strukturieren heißt IndexedDB. Diese ist eine NoSQL-Datenbank, welche im Grunde eine Liste von javascript-Objekten indexieren kann. Wir haben hier aber im Falle der Vokabeln nicht alle Daten hineingeschrieben, sondern nur die momentan ausgewählte Sprache und Dialekt, damit die Datenbank nicht erweitert werden muss, wenn neue Sprachen und Dialekte hinzukommen.

LocalStorage

Um alle Benutzerdaten, wie Einstellungen und Fortschritt zu persistieren haben wir uns entschieden LocalStorage zu benutzen. Diese Datenbank, ist wesentlich simpler und einfacher zu benutzen als IndexedDB, allerdings kommt sie mit

einigen Einschränkungen. Da aber unser zu Speichernden daten sowieso nicht kompliziert sind reicht diese vollkommen aus. Ein der größten Vorteile von LocalStorage ist, dass der Zugriff im Gegensatz zu IndexedDB, nicht in neuen Threads stattfindet und somit der Code leicht verständlich bleibt

4 Architektur

In diesem Abschnitt wird die grundlegende Idee hinter den einzelnen, mehr oder weniger voneinander getrennten Teilen des Projekts beschrieben. Es handelt sich dabei nciht um eine Dokumentation im eigentlichen Sinne. Diese kann dem Wiki im Git-Repository und den Kommentaren im Code entnommen werden.

4.1 User-Interface

Das User-Interface also solches ist im Grunde nur eine HTML-Seite welche die Anordnung von Elementen bestimmt und eine dazugehörige TypeScript-klasse die unterschiedliche Events handelt. Die Datenmanipulation findet, bis auf einige Ausnahmen, findet in den Services statt.

4.2 Services

Die Services sind einzelne Klassen (Intern vermutlich als eine Art JavaScript-Äquivalent zu Singletons implementiert) welche für bestimmte, komplexere Aufgaben zuständig sind. Am wichtigsten ist hier der Controller, welcher die Kommunikation mit der Datenbank übernimmt. Wenn eine Aufgabe kompliziertere Logik erfordert wurde dafür ein Service implementiert auf welchen vom TypeScript-Code der jeweiligen Seite zugegriffen wird.

Zusätzlich zu den eigenen Services existieren einige Dienste Standardmäßig von Angular aus (bspw. Router) und können genau so auch mit Plugins nachinstalliert werden (in unserem Fall nativeAudio und ngx-language).

4.3 Strings

Die Strings der App sind in einem JSON-Objekt gespeichert auf welches mit ngx-language zugegriffen wird. Der bedachte Vorteil war hier dass mit dem Sprachpaket sehr einfach Übersetzungen als weitere JSON-Objekte hinzugefügt und einfach mit ngx-language eingebunden werden können. Die Übersetzungen beziehen sich im Moment lediglich auf die Sprachausgabe bzw. die Anzeige unterschiedlicher Vokabeln.

4.4 Datenstrukturen

Die Datenobjekte, die wir im Projekt auch als Entitäten beschreiben, sind die grundlegendsten Datenstrukturen die in der App verarbeitet werden. Da diese ganz unten in unserer Schichten-Architektur sind, dazu später mehr, sind fast alle anderen Module von diesen Abhängig. Daher können kleine Veränderungen an diesen sich durch das ganze Projekt ziehen. Um dies zu vermeiden und trotzdem noch flexibel zu sein haben wir uns hier für das bekannte Pattern der Abstract Factories entschieden. Daraus folgt, dass jeder Datentyp eine Abstrakte Fabrik für diesen besitzt und diese beim Aufbau des Programmbaums sehr leicht ausgetauscht werden kann. Daher können wir nun neue Funktionen einbauen ohne in den Datenstrukturen eine Zeile Code ändern zu müssen. Lediglich müssen wir beim Aufbau des Programmbaums eine neue Factory-Implementierung erzeugen.

4.5 Datenbank und Excel

Wie oben schon beschrieben liegen alle Daten der App als Exceltabellen vor. Somit müssen diese zuerst in eine Datenbank übertragen werden. Dies erfolgt zum einen mit der Bibliothek xlsx und der Datenbank IndexedDB. Wir haben darauf geachtet, dass so gut wie keine Abhängigkeiten zwischen dem einlesen der Exceltabellen und dem eintragen in die Datenbank bestehen, damit im Zweifel sehr einfach eine andere Datenbank oder wahrscheinlicher ein anderes Format für die Daten genutzt werden kann. Um die Daten aus dem Excel-Modul in das DB-Modul zu transferieren benutzen wir eine eigene Datenstruktur, die aber im Grunde nur ein 2-Dimensionales Array mit extra Kopfzeile ist. Die Zugriffe auf die Datenbank und das auslesen der Exceltabellen erfolgt größtenteils asynchron, was den Code zum Teilen aufgebläht wirken lässt. Wir haben trotzdem versucht durch zB einfache Interfaces den Code übersichtlich zu behalten.

4.6 Interactor und grundlegende Architektur

Die grundlegende Architektur ist so aufgebaut, dass wir unsere App in Schichten aufgeteilt haben. Wobei ein Modul in einer Schicht nur von Modulen in den Schichten darunter abhängig ist (also es existieren Code-Abhängigkeiten) Dies führt dazu dass die App sehr Modular und leicht erweiterbar aufgebaut wird. Das Herzstück der App ist der Interactor, er verbindet alle Module miteinander. Er ist dafür zuständig, dass alle Anfragen die der User über die UI erstellt, auf die dementsprechenden Module aufgeteilt werden um diese zu verarbeiten. Daraus folgt, dass die wichtigsten Proceduren der App in dieser Schicht stehen und diese keine Abhängigkeiten zu Ein- und Ausgabe haben. Zumindest war es so gedacht, in der Praxis sind leider momentan die Interfaces die im Interactor benutzt werden zu Anwendungsspezifisch als das sie leicht ausgetauscht werden könnten. Dieser Punkt ist uns aber erst später bewusst geworden und kann daher nur als Erkenntnis für zukünftige Projekte mitgenommen werden.

5 Bekannte Probleme, Anregungen und Hilfestellung

Im folgenden möchten wir auf fehlende Features und bekannte Probleme mit der App aufmerksam machen sowie einige Anregungen und Hilfestellungen bieten um die Qualität der App auf ein Niveau zu heben welches ein potentielles Deployment rechtfertigt.

5.1 Fehlende Features

Glückwunschtexte am Ende einer Lektion

Die Glückwunschtexte am Ende einer Lektion sollten sich im Falle der Quizes von der erreichten Punktzahl abhängig voneinander unterscheiden.

Assets und Oberfläche

Die Assets der App sollten vor der Lektion vorgeladen werden während der User in einem Ladebildschirm wartet.

Die Oberfläche der App sollte von Ihrer Größe her an die gängisten Smartphones angepasst werden so dass man möglichst nciht scrollen muss.

Außerdem muss sichergestellt werden dass die Cards mit Bildern und Texten in Quiz immer die selbe Größe haben.

Zudem sollte das Farbschema einmal passend zum Momentanen Blau überarbeitet werden.

Nicht freigeschaltete Level

Im Moment werden nicht freigeschaltete Level einfach ausgeblendet. Es wäre schöner wenn diesen die Interaktivität genommen wird und die Level ausgegraut angezeigt werden.

Karte für Dialekte

Es ist eine graphische Darstellung der Herkunft der unterschiedlichen Dialekte im Dialektauswahlmenü gewünscht. Schön wäre wenn diese Grafik interaktiv ist und die Dialektauswahl darüber funktioniert.

Menü- und Glückwunschsounds

Die Sounds der Menüführung und der Gratulation sind nicht implementiert.

Verschiedenfarbige Haken

Die Haken bei der Levelauswahl sollten aussagekräftiger sein - möglicherweise ist auch eine Darstellung des Fortschritts mit Zahlen o.ä. sinnvoll.

Initiales Routing

Beim ersten Appstart sollte der Benutzer kurz durch die Sprach- und Dialektauswahl geführt werden um diese einzustellen. Ein kurzes "Tutorial"welches die unterschiedlichen Modi und die Funktionen kurz erklärt wäre auch vorteilhaft.

Share-Button

Es sollte ein Share-Button welcher die App in social media teilt implementiert werden.

Bugs

Wenn man ein Quiz erneut spielt wird der erreichte Fortschritt dieses Quizes überschrieben.

Die Dialektauswahl funktioniert in Android- und iOS-Build nicht korrekt.

5.2 Anregungen

Improved Angular

Im Moment ist Angular nur sehr rudimentär genutzt. Die Möglichkeit mit unterschiedlichen Komponenten sich wiederholende Aufgabe mittels übergabe von Parametern zu regeln wurde quasi nciht genutzt.

Dienste

Außerdem sollten die Dienste umfangreich refactored werden - vorzugsweise dass TypeScript-Klassen der Seiten auf Dienste zugreifen und die Dienste auf den Controller. Im Moment greift das ein bisschen alles auf alles zu.

TypeScript der Seiten

Die Klassen welche die Events der Seiten steuern sind zu umfangreich. Hier sollte wo möglich wirklich nur ein Event auf einen Service zeigen.

5.3 Hilfestellung

Die Webwelt ist anfangs durchaus sehr verwirrend, daher hier ein paar Worte die den Einstieg vereinfachen sollen.

Node.js

Node.js ist der Kern dieses Projekts. Mit npm werden alle benötigten Pakete installiert. „Pakete“ ist dabei eine sehr schöne Analogie, denn wenn man ein wenig wie Linux-Pakete betrachtet erscheint es garnicht mehr so verwirrend wie wenn man aus der IDE verzweifelt versucht dinge zu installieren und Konsolenbefehle aus Stack Overflow kopiert.

Am Anfang steht die Installation von Node.js, alles andere kann dann mit dem Befehl `npm install xxx` installiert werden.

Ionic und Angular

Es ist Ratsam zunächst eines dieser 3 - 4 Stunden Tutorials von YouTube durchzuarbeiten um die Grundlagen von Ionic und insbesondere auch Angular zu lernen. Die Grenzen dieser beiden Frameworks verschwimmen sehr, da Ionic sich häufig direkt auf Angular bezieht. Das ist aber nicht weiter wild, im Grundsatz kann man sich erstmal denken: Ionic macht die Objekte und Angular macht die HTML-Seiten inklusive Routing und darunterliegender Struktur (auch wenn gerade beim Routing Ionic bei einigen Dinge mitredet).

IDE

Wir haben Webstorm genutzt da dies eine modifizierte Version von Intellij ist. Es wird in Tutorials immer auf Visual Studio Code verwiesen - hier ist es wirklich einfach Geschmackssache. Wichtig in Windows ist das ausführen der IDE als Andministrator, damit man die Konsole in der IDE mangels `sudo` trotzdem benutzen kann.

Builden (oder auch: Umgebungsvariabelnhölle)

Für das Builden mit Android muss eine Java JDK 1.8.x und eine Android JDK installiert sein und von den Umgebungsvariabeln referenziert werden. Für die Android JDK am besten Android Studio isntallieren, da der entstehende `platforms\android`-Ordner direkt als Android-Studio-Projekt geöffnet werden kann und dort dann auch der Emulator genutzt werdne kann. Gerade in Windows sind die Umgebungsvariabeln ein wenig... schwierig. Man bekommt es aber gegoogelt - ein kurzer Tipp gegen graue Haare: Die Variabeln werden erst nach dem Schließen des Einstellungsmenüs übernommen.
Das Builden für iOS funktioniert ganz ähnlich, hier muss eine XCode-IDE installiert werden mit dem dann das entsprechende Projekt geöffnet werden kann. Bei Apple ist das setzen von Umgebungsvariabeln nciht notwendig.

Es funktioniert eigentlich ganz gut sich von Fehler zu Fehler zu hangeln.

6 Anhang

6.1 Script der Vorstellung

Die Bavarian App ist ein Lernprogramm um verschiedene Bayrische Dialekte ausgehend von unterschiedlichen Sprachen wie Deutsch, Arabisch oder Tschechisch zu lernen. Es gibt zwei Modi, einen um den ausgewählten Dialekt zu lernen und einen um sich diesem Dialekt in einem Quiz zu stellen. Ein Dialekt ist in verschiedene Kategorien aufgeteilt welche wiederum 8 Leveln untergeordnet sind. Wenn man alle Quizes eines Dialekts erfolgreich abgeschlossen hat - erfolgreich bedeutet hier mehr als 90% der Antworten sind richtig - bekommt man ein Archievment welches sich in den Einstellungen betrachten lässt.

Die voliegende App wurde mit Ionic und Angular in Typescript und HTML geschrieben. Ionic ist dabei ein Framework das verschiedene Elemente wie Buttons oder Slider zur verfügung stellt und Angular erweitert HTML zum einen um viele nützliche Elemente wie Loops oder If/Else-Abfragen und setzt zum anderen eine völlig neue Abstraktionsebene auf, indem mit Komponenten gearbeitet wird welche dynamisch auf einer HTML-Seite ausgetauscht werden können. Ziel war dabei die Bavarian App mit neuer Technologie in einfacher Entwicklung sowohl für iOS als auch Android zur verfügung stellen zu können.

Im Kern funktioniert die App bereits - so wird im Hintergrund eine lokale Datenbank verwaltet die Vokabeln aus Exceltabellen beinhaltet, Daten persistiert und Assets zuordnet. Im Vordergrund wird die Datenbank abgefragt, die UI befüllt und Eingaben des Users wieder an die Datenbank zurückgegeben.

Die noch zu Erfüllenden Aufgaben teilen sich auf vier große Bereiche auf:
Zuallererst fehlen noch einige Features wie zum Beispiel der Share-Button oder die Gratulationen am Ende eines Kurses welche im Detail dem Praktikumsbericht entnommen werden können.
Dann sind die Anforderungen lose der zugrundeliegenden „Bavarian App“ entnommen. Wir haben uns im Grunde angeguckt was die kann und uns zuletzt ein Backlog angelegt mit Fehlenden oder unvollständigen Features welche wir sukzessive Implementiert haben. Es wäre Vorteilhaft die Anforderungen der App einmal schriftlich in einem Pflichtenheft zu erfassen und dies mit Systemtests zu überprüfen.
Außerdem hat die UI noch einige Schwächen. So sind die Bilder nicht immer gleich groß, allgemein sind die einzelnen Größen nicht unbedingt an Smartphones angepasst und gelegentlich wird der User über die Vorgänge in der App nur unzureichend Informiert (Mit Toasts, Alerts oder auch mit ungeschickter Menüführung). Für diesen Punkt muss die App selbstständig analysiert werden.
Schlußendlich kann mit ein wenig Erfahrung in Angular ein umfassendes Refactoring der Dienste (Services) der UI erfolgen. Wir haben die UI und die Datenbank strikt getrennt und versucht uns soweit möglich an S.O.L.I.D-Prinzipien zu halten - die Dienste aber haben sich zum Schluss immer mehr verknotet. Hier Abhägigkeiten zu eleminieren ist wichtig wenn die App in späteren Projektphasen noch erweitert werden soll.

Der schwierigste Teil dieses Praktikums wird das Einarbeiten zum einen in

Ionic mit Angular sowie Angular selbst und zum anderen in unser Projekt sein. Da bereits alle Mechanismen bereit stehen ist eine Weiterentwicklung der Datenbank-Mechanismen vermutlich nciht unbedingt notwendig. Gegebenenfalls kann der Interactor als Schnittstelle zwischen den Datenbankmechanismen und dem Front-End erweitert werden. Das schön dabei ist, dass alles unangetastete dann mit dem alten Interface einfach weiterfunktionieren kann. Da der Fokus in diesem Projekt jetzt auf Qualitätssicherung liegt wird der Großteil der Aufgaben „theoretischer“ Natur sein - soll heißen viel Analyse der zugrundeliegenden Programmen und weniger Implementieraufwand.