ERNST-ABBE-HOCHSCHULE JENA

Fachbereich Wirtschaftsingenieurwesen

Professur E-Commerce/E-Business

Bachelorarbeit zum Thema

Untersuchung von Cross-Compiler-Frameworks für die mobile App-Entwicklung und Darstellung der Machbarkeit anhand der Stundenplan-App der EAH

Betreuender Hochschullehrer:	Prof. Dr. M. Stepping
Bearbeiter:	Michael Schulz Carl-Born-Straße 6 07749 Jena
Matrikel-Nr.:	638777
Eingereicht am:	18.09.2019

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DOM	Document Object Model
EAH	Ernst-Abbe-Hochschule
HTML	Hypertext Markup Language
JSON	JavaScript Object Notation
PWA	Progressive Web App
URL	Uniform Resource Locator

Codebeispielverzeichnis

Glossar

Cloud

Serversystem, dass verschiedene Dienste und Funktionen zum Speichern von Daten bereitstellt

Cookie

Textinformation, die im Browser auf dem Computer des Betrachters zu einer besuchten Webseite gespeichert wird

Frontends

Stellt die Präsentationsebene der Applikation meistens in Form einer grafischen Benutzeroberfläche dar

HTML-Markup

Auch HTML-Tags genannt, legen sie eine Struktur über Text und Bilder der Webseite, um den Browser mit Informationen über den Aufbau zu versorgen

HTTP-Header

Erlauben es dem Client und Server zusätzliche Informationen an eine Anfrage oder eine Antwort zu übergeben

HTTP-Request

Stellt ein Paket aus Informationen dar, das zwischen verschiedenen Computern hin- und her gesendet werden kann

POST-Befehle

POST-Befehle können große Datenmengen (wie Bilder oder Formular-Daten) zur weiteren Verarbeitung zum Server senden

Proxy Server

Ein Proxy Server dient als Vermittler zwischen dem Internet und einem zu schützenden Netzwerk

Template

Vordefiniertes Gerüst für das Layout von Dokumenten oder Ansichten

Web Inspectors

Ein von Browsern bereitgestelltes Tool, um Internetauftritte untersuchen und bearbeiten zu können

1 Einleitung

Mobilität und Kommunikation spielen eine immer größere Rolle in unserer Gesellschaft. Smartphones, Tablets und tragbare Geräte wie Smartwatches sind kaum noch aus dem Alltag wegzudenken. Während zur Anfangszeit der Smartphones im Jahr 2009 die Anzahl der Smartphone-Nutzer in Deutschland noch bei 6,3 Millionen lag, so konnte diese Zahl bis zum Jahr 2018 auf 57 Millionen ansteigen.[1] Eine der Hauptfunktionalitäten eines solchen Smartphones stellt die Fähigkeit zur Installation von Apps dar. Diese sind dabei entweder auf dem jeweiligen Endgerät vorinstalliert oder über die verschiedenen App Stores, wie dem „Google Play Store“ und „iOS App Store“ erhältlich. Mittlerweile existieren über 4,3 Millionen Applikationen[2] in diesen Stores und es kommen täglich welche dazu. Da der aktuelle Markt zu 100% von Google und Apple dominiert wird, ist es kaum noch möglich als Anbieter eines alternativen Betriebssystems Fuß zu fassen. In einer Prognose bis zum Jahr 2023 soll lediglich das Verhältnis der Marktanteile beider Konkurrenten zu einander variieren, nicht aber der eines anderen Marktteilnehmers relevant werden.[3] Um ein möglichst breites Kundenspektrum abdecken zu können, versuchen viele Unternehmen Angebote für beide Plattformen zu veröffentlichen. Allerdings ist zu beachten, dass die Programmierung für die beiden Betriebssysteme separat erfolgen muss, da beide eine unterschiedliche Grundsprache verwenden. Aufgrund der vielen unterschiedlichen Möglichkeiten, eine Anwendung für Android oder iOS zu entwickeln, zeigt sich dadurch oft ein Ressourcenproblem bei jungen, aber auch eingesessenen Unternehmen. Durch den doppelten Aufwand, der für beide Plattformen separat erfolgen muss, sind Spezialisten von Nöten, die über entsprechendes, fundiertes Wissen verfügen. Um den genannten Problemen entgegen wirken zu können, liegt die Entwicklung von hybriden Lösungen nahe, die mittels nur eines Codestandes die Implementierung für Android und iOS gleichzeitig übernehmen. Da die komplette Wertschöpfungskette einer App-Entwicklung im Verlauf nur einmalig durchgeführt werden muss, können Kosten und Zeit eingespart werden. Dabei gibt es verschiedene Möglichkeiten, um dieses Ziel zu erreichen. Welche Ansätze dabei von den Cross-Plattform-Frameworks der Anbieter verfolgt werden und welche Vor- und Nachteile sich durch die Entwicklung einer Cross-Plattform-Anwendung ergeben, wird im weiteren Verlauf der Arbeit erläutert.

---

[1] Statista, Anzahl Smartphone-Nutzer

[2] Statista, Anzahl Apps

[3] Reith, Worldwide Smartphone Shipment

1.1 Zielsetzung

Die Arbeit soll im Folgenden die Machbarkeit einer Stundenplan-App mit Hilfe eines Cross-Plattform-Frameworks prüfen sowie weitere Optionen zur Umsetzung vorstellen. Die Grundfunktionen sollen dabei auf Basis der Leitbilder der Anwendung beruhen, die ehemals durch die EAH Jena vertrieben worden ist. Einige Funktionen erhalten dabei Erweiterungen, die während der Konzeption ausgearbeitet werden. Dabei handelt es sich im Spezifischen um die Anzeige der Stundenpläne, weitere Unterseiten finden jedoch ebenfalls Verwendung. Auf diese Weise kann untersucht werden, ob ein solches Framework mit einmaliger Entwicklung dazu im Stande ist, native Applikationen für die Plattformen Android und iOS zu realisieren. Für den Prozess selbst wird aufgrund vorangehender Erfahrungen des Studenten das Framework Nativescript verwendet, da bereits bekannte Webtechnologien, wie JavaScript, TypeScript und CSS für die Nutzung eingesetzt werden können. Da immer mehr große Unternehmen wie beispielsweise Tesla, Pinterest, Skype und Uber[4] auf den Einsatz einer hybriden Lösung bei der Entwicklung setzen, liegt es im Interesse des Studenten entsprechende Möglichkeiten zu evaluieren. Die Untersuchung des Frameworks und der Entstehungsprozess zur Umsetzung der Stundenplan-App sollen während der Arbeit vorgestellt werden. Das Ziel ist es, einen Prototyp zu kreieren, der nicht nur für Studenten einen Mehrwert bietet, sondern auch plattformübergreifend mit einem Codestand funktioniert.

1.2 Struktur der Arbeit

In den folgenden Kapiteln befasst sich die Arbeit detailliert mit den einzelnen Schritten, die bei der Umsetzung einer Stundenplan-App durchlaufen wurden.

Das Kapitel „Grundlagen“ dient dabei als Einstieg in das Thema und liefert wichtige Grundbegriffe, die für das weitere Verständnis wichtig sind. Im ersten Schritt werden all jene Möglichkeiten besprochen, die einem Entwickler bei der Implementierung einer App für ein Smartphone zur Verfügung stehen. Daran schließt sich eine Vorstellung der Frameworks an, die mittels eines Codestandes Anwendungen für mehrere Zielplattformen ermöglichen.

[4] Github, React Native Apps

Das Kapitel „Verwendete Technologie“ beschreibt dabei im Detail das genutzte Framework und liefert weitere technische Hintergründe und Informationen über die dabei verwendeten Sprachen.

In der „Analyse“ geht es vor allem darum zu prüfen, was vorhanden ist und was noch gebraucht wird. Dabei beschreibt der erste Schritt die aktuelle Situation an der Hochschule. Es folgt eine nähere Betrachtung der Schnittstellen, um die Machbarkeit der Umsetzung zu untersuchen. Mit einer Analyse der ehemaligen App werden Funktionen beschrieben, die bei der Umsetzung als grundlegende Idee genutzt werden können. Mit dieser Vorarbeit ist es möglich, explizit funktionale und nichtfunktionale Anforderungen an die App zu formulieren, die den Implementierungsprozess unterstützen.

Das Kapitel „Konzeption“ geht dabei im Einzelnen auf konzeptionelle Entscheidungen bei der Gestaltung der Nutzerführung ein. Gleichzeitig spielen auch Wireframes für die Strukturierung des Layouts eine Rolle sowie weitere Festlegungen, die das User-Interface betreffen.

Die „Implementierung“ geht auf die eigentliche Umsetzung der Anwendung ein und die Probleme, die sich während des Entwicklungsprozesses gezeigt haben. Dabei werden verschiedene, wichtige Teile des erstellten Codes und deren Funktionen und Einsatz näher erläutert.

Im 7. Kapitel, dem Abgleich, werden die an die App gestellten funktionalen und nichtfunktionalen Anforderungen aufgegriffen und auf ihre Erfüllung überprüft.

Zukünftige Erweiterungen, Änderungen oder Anpassungen der erstellten Anwendung werden im Kapitel „Fazit“ beschrieben.

2 Grundlagen

In diesem Kapitel werden zunächst die Grundlagen erklärt, die für das weitere Verständnis der Arbeit essentiell sind. Als erstes werden verschiedene Möglichkeiten der App-Entwicklung im Allgemeinen vorgestellt. Anschließend geht es um aktuell moderne Frameworks, die zur Cross-Plattform-Entwicklung fähig sind, sowie um die Vorteile, die durch den jeweiligen Hersteller beworben werden.

2.1 Möglichkeiten der mobilen App-Entwicklung

2.1.1 Native App

Native Apps nutzen eine native Sprache des mobilen Betriebssystems, auf dem sie operieren. Beispielsweise wird so unter Android Java als Grundsprache verwendet sowie Objective-C unter iOS. Der Code der jeweiligen App wird direkt im Betriebssystem ausgeführt und bietet hinsichtlich der Performance und der nutzerspezifischen Erfahrung die besten Ergebnisse. Die Entwicklung selbst gestaltet sich aber aufgrund ihrer Diversität in der jeweiligen Plattform komplexer und folglich auch wesentlich kostspieliger.[5] Für jede Zielplattform muss ein komplett eigener Code gepflegt werden, der wiederum unterschiedliche Kompetenzen voraussetzt.

Ihren größten Nutzen entfalten native Apps jedoch bei mehrfachem, beziehungsweise generellen Zugriff auf die systemeigene Hardware. In dieselbe Kategorie fällt auch die Berechnung aufwendiger Aufgaben. Da keine zusätzlichen Übersetzungen von Code stattfinden, kann so die gesamte Rechenleistung des Endgerätes voll ausgenutzt werden.[6]

Über die Jahre hinweg hat sich eine duopole Einteilung des Marktes der mobilen Betriebssysteme entwickelt. Statistiken zufolge beruht der Marktanteil von Handy-Verkäufen, die nicht mit einem Android oder iOS Betriebssystem ausgeliefert werden auf 0% und aktuell zeigt kein Trend die Möglichkeit an, dass sich diese Situation in naher Zukunft verändern könnte.[7] Somit hat sich auch der gesamte Fokus der Entwickler, die für mobile Endgeräte programmieren möchten, auf diese beiden Plattformen verteilt.

---

[5] Long, 2016, S. 168

[6] ebd., S.168

[7] Reith, Worldwide Smartphone Shipment

Die native Entwicklung bringt neben den bereits genannten Vorteilen auch eine Reihe weiterer unterstützender Maßnahmen mit sich, die im Laufe der Jahre unverzichtbar für die generelle App-Entwicklung geworden sind. Es gibt ein eigenes Development Kit, das sowohl seitens Apple als auch von Google ständig weiterentwickelt wird. Dadurch werden sämtliche Tools, die ein Entwickler für die native App-Entwicklung braucht, gebündelt angeboten. Dazu gehören eine eigene Entwicklungsumgebung (Android Studio[8] und Xcode[9]) und Plattformen für das Speichern von Daten in der Cloud (Firebase[10] und CloudKit[11]). Google bietet unter anderem auch eine Kollektion von vorgeschriebenen und zum Großteil schon getesteten Komponenten an, die direkt für die eigene Entwicklung in Form von Bibliotheken genutzt werden kann (Android Jetpack[12]). Mit enthalten sind Design Pattern, die bei der Entwicklung als Hilfestellung dienen. Es werden sämtliche Voraussetzungen geschaffen, um den Entwicklungsprozess zu unterstützen.

2.1.2 Progressive Web-App

Im ursprünglichen Sinne waren Web-Apps Anwendungen, die direkt über den Browser aufgerufen wurden und somit auf jedem internetfähigen Gerät lauffähig sind. Es wurde quasi ein Lesezeichen erstellt, das direkten Zugriff auf die jeweils verwiesene und oftmals für mobile Geräte optimierte Webseite erlaubte. Die Programmierung wurde sowohl damals als auch heute mit CSS, JavaScript und HTML realisiert. Da Web-Apps nach dem klassischen Client-Server-Modell agieren, ist eine Installation auf dem jeweiligen Endgerät nicht nötig. Die Auswertung, Verarbeitung und Bereitstellung der Daten erfolgt auf einem Webserver oder der Cloud.[13] Dadurch konnte zwar generell der Eindruck einer mobilen, nativen App erweckt werden, jedoch stellte sich der Zugriff auf die Hardware oft als problematisch dar. Der typische Smartphone-Nutzer zählt einzelne Hardware-Funktionen, wie den Zugriff auf die Kamera oder das Ablegen von Daten im Speicher zum Standard, deshalb war dies die erste Hürde, die überwunden werden musste.

Da sich mobile Webseiten ständig in ihren Möglichkeiten weiterentwickeln, spricht man aktuell von progressiven Web-Apps, in der Arbeit PWA genannt. So werden die Grenzen zu

[8] Android, Android Studio
[9] Apple, Xcode
[10] Google, Firebase
[11] Apple, CloudKit
[12] Google, Android Jetpack
[13] Augsten, Web App

nativen Applikationen immer weiter aufgebrochen und Funktionen angeboten, die vorher nur nativen und Cross-Plattform-Anwendungen vorenthalten waren. Gerade Google hat verschiedene Konzepte entwickelt und bereitgestellt, die in herkömmliche Webseiten implementiert werden können. Anforderungen und Funktionen wie das Bereitstellen von Push-Benachrichtigungen oder Zugriff auf die Kamera sind nun realisierbar. Solche Webseiten sind, insofern sich ein PWA-fähiger Browser auf dem Endgerät befindet, ebenfalls offline nutzbar. PWA's erscheinen auf dem bekannten Smartphone-Homescreen und sind direkt aufrufbar.[14] Diese sollen den Anschein einer regulären App erwecken, unterscheiden sich jedoch signifikant in ihrer Bau- und Funktionsweise. Im Hintergrund läuft nämlich immer noch der reguläre Browser, wenn auch in einer verkleinerten Variante, diesmal ohne die visuellen Steuerelemente. Allerdings unterstützt bisher einzig Chrome solche Funktionen innerhalb einer Webseite, wodurch sich diese Art von App lediglich auf Android-Geräte beschränkt.[15] Somit bleibt abzuwarten, ob Apple im weiteren Verlauf dem Trend folgt und Safari mit ähnlichen Möglichkeiten ausstatten wird.

2.1.3 Hybride App

Hybride Apps versuchen die Vorteile aus den bereits genannten Lösungen zu vereinen und so eine Kombination aus Webinhalten und nativen Darstellungen zu ermöglichen. Ähnlich, wie bei Web-Apps, kommen bekannte Technologien wie JavaScript und HTML zum Einsatz. Ein spezifisches Fachwissen über die jeweilige Zielplattform wird bei einer üblichen hybriden Entwicklung nicht vorausgesetzt. Die App selbst wird jedoch nicht über den Browser aufgerufen und dargestellt, sondern ganz normal über den jeweiligen Store des Gerätes installiert und direkt auf dem Betriebssystem ausgeführt. Im Anschluss werden eine mobile Seite und der dazugehörige JavaScript-Code in einem nativen Container ausgeführt. Folglich können bekannte Konzepte aus der Webentwicklung angewendet werden. Zugriff auf die Hardware und einzelne Sensoren ist gegeben, obwohl keine Hochsprache wie Java durch den Programmierer eingesetzt werden muss.[16] Viele Entwickler bekommen so die Möglichkeit auch plattformunabhängig zu agieren und sind nicht mehr mit einem Code an ein einzelnes Betriebssystem gebunden.[17] Ursprünglich war die Performance zwar unterhalb einer nativen, aber oberhalb einer webbasierenden App einzuordnen, da neben dem nativen

[14] Google Developers, Web Apps
[15] Majchrzak, 2018, S. 5735
[16] Xianghao, 2014, S. 1262
[17] Delia, 2015, S. 2

Teil auch noch Web-Code interpretiert werden muss. Die Wartung von nur einem Codestand war für viele Firmen jedoch ein entscheidender und der ausschlaggebende Vorteil.[18]

2.2 Moderne Cross-Plattform-Frameworks

Neben den genannten Möglichkeiten gibt es noch einige alternative Ansätze, die bisherige Optionen erweitern oder verschiedene Technologien miteinander verbinden. Oft sind diese unter dem Namen Cross-Plattform-Frameworks zu finden. Allerdings gibt es gravierende Unterschiede in ihrer Bauweise und den Fähigkeiten, die ein Entwickler für die Umsetzung benötigt. Da es sich bei dem für die Stundenplan-App verwendeten Framework um ein solches Modell handelt, werden im folgenden Abschnitt einige vorgestellt.

2.2.1 React Native

React Native ist ein von Facebook bereitgestelltes Framework, mit dem durch möglichst einen Codestand Cross-Compiler-Fähige Applikationen geschrieben werden können und ein natives Rendering möglich ist. React Native basiert sehr stark auf dem Framework React, das von den Entwicklern derselben Firma herausgegeben wurde. Der Unterschied zwischen beiden ist maßgeblich die Zielplattform. Während React lediglich für Browser verwendbar ist, ist React Native nur für Android und iOS verfügbar.[19]

Entwicklern wird der Umstieg jedoch sehr leicht gemacht, da für das User-Interface die XML-ähnliche Syntax beibehalten worden ist. Es wird eine native Brücke genutzt, um den Code tatsächlich als native Komponente rendern zu können. Die komplette Business-Logik von React Native wird in schwach typisiertem JavaScript realisiert.[20]

In React Native wird die Logik auf einem einzelnen Thread namens JavaScript-Thread (kurz: JS-Thread) ausgeführt. Die Ausführung des User-Interface erfolgt auf einem separaten Thread (kurz: UI-Thread). Der UI-Thread ist für das Rendern des aktiven Layouts verantwortlich und reagiert konstant nur auf Benutzerinteraktionen. Seine einzige Aufgabe besteht darin zu reagieren, um so flüssige Animationen zu erlauben. Der JS-Thread

[18] Delia, Multi-Platform Analyses S. 6
[19] ebd., S.7
[20] Bonnie, 2015, S. 1

wiederum verarbeitet konstant sämtliche Eingaben des Nutzers und kümmert sich um jegliche Logik, die zur Laufzeit erfolgen muss.[21]

Facebook, beziehungsweise React Native arbeitet mit sogenannten Modulen und Plug-Ins, die die herkömmliche Funktionsweise des Frameworks erweitern. Somit kann bereits nativ programmierte Software in das eigene React-Projekt eingebunden werden.
Diese sind immer in der jeweiligen Sprache der Zielplattform, Java für Android und Objective-C für iOS, geschrieben in Form von Bridge-APIs. Ist ein Zugriff aus dem eigenen Code heraus auf Methodenaufrufe innerhalb der Plug-Ins notwendig, muss beispielsweise unter Android innerhalb des Java-Codes die Annotation @ReactMethod eingefügt werden.[22] Erst dann existiert diese auch im Namespace des eigenen React Native Projektes. Wenn man sich also während der Entwicklung eines eigenen Projektes auf ein bereits vorhandenes, natives Plug-In eines Dritten verlassen möchte, so muss oftmals sowohl nativer Code geschrieben werden als auch eine zusätzliche React Komponente, die Nutzen von diesem nativen Code macht. Das setzt nicht nur eine umfassende Kenntnis der jeweiligen Zielplattform voraus, sondern resultiert - samt Lernprozess - auch in längerer Entwicklungsdauer, die oftmals nicht gegeben ist.[23]

2.2.2 Xamarin

Xamarin ist ein von Microsoft bereitgestelltes Framework, mit dem Cross-Plattform-Entwicklung möglich ist. Über die Jahre hinweg hat sich Xamarin aufgrund seiner Portierbarkeit auf verschiedene Plattformen bemerkbar gemacht. Hierzu zählen neben den im Vordergrund stehenden mobilen Plattformen auch verschiedene Desktopvarianten für Windows und Linux zur Verfügung. Das Framework nutzt hauptsächlich C#, eine statisch typisierte Sprache. Das .NET-Framework wird im Verbund für mobile Plattformen eingesetzt.[24]

Bei C# handelt es sich, wie bei anderen Hochsprachen, wie Java und Objective-C um eine strikt typisierte Sprache. Das Framework selbst bietet die Möglichkeit “Bindungs-Bibliotheken” zu erstellen, die bereits vorhandene Java-Bibliotheken mit C#-Wrappern

[21] React Documentation, Performance
[22] Github, React Native Android
[23] React Documentation, Android Modules
[24] Altexsoft, Cross-Platform Comparison

umschließt, die dann wiederum einfach aufgerufen werden können.[25] Aufgrund der langjährigen Existenz von Xamarin wurden viele Bindungs-Bibliotheken online bereitgestellt, die direkt für den Android und iOS Code wiederverwendet werden können. Dabei müssen Entwickler einzelne Schnittstellen schreiben, sodass mit C#-Code dasselbe erreicht werden kann, was in dem jeweiligen nativen Pendant möglich wäre. Wenn diese Brücke jedoch gegeben ist, kann mit C# Code geschrieben werden, der bei der Kompilierung in nativem Code resultiert und auf dem mobilen Endgerät wiedergeben werden kann.[26]

Zu berücksichtigen ist, dass trotz des hardwarenahen Codes dennoch Performance-Einbußen bei typischen Operationen, wie CRUD-Befehlen (Create, Read, Update, Delete auf der Netzebene) oder lokalem Speicherzugriff vorkommen können. Hinzu kommt, dass der Ansatz sich einen Codestand zwischen allen Plattformen zu teilen, nur auf die Business-Logik zutrifft. Xamarin versucht die Apps so nativ wie möglich erscheinen zu lassen, deshalb muss sowohl für Android als auch iOS eine separate User-Interface-Ebene geschrieben werden. Dadurch entsteht zusätzlicher Entwicklungsaufwand, der zwar dem Framework in seiner Performance und Benutzerführung hilft, aber wesentlich mehr Entwicklungszeit in Anspruch nimmt.[27]

2.2.3 Nativescript

Nativescript ist ein Cross-Plattform-Framework, das wie React Native dazu im Stande ist, ein natives User-Interface für Android und iOS zu rendern. Allerdings steht in diesem speziellen Fall keine große Firma wie Facebook oder Microsoft hinter dem Produkt, sondern die Progress Software Corporation.[28]

Als Entwicklungssprache ist JavaScript fest verankert. Allerdings bietet das Framework die Option, sowohl mit Angular, einem Web-Framework, als auch mit TypeScript, einer Art typisierten JavaScript, zu arbeiten. Diese sind ebenfalls fester Bestandteil des Nativescript-Ökosystems und haben ihren eigenen Platz in Form einer separaten Dokumentation, die der einfachen JavaScript Dokumentation in nichts nachsteht.[29] Welche Vorteile sich durch den

[25] Xamarin Documentation, Binding Java
[26] ebd.
[27] Altexsoft, Performance Comparison
[28] Nativescript Documentation, Getting Started
[29] Nativescript Angular Documentation, Getting Started

Einsatz der zusätzlichen Sprache und des Frameworks ergeben und wie sehr diese im Detail zusammenhängen, wird im nachfolgenden Kapitel geklärt.

Die Progress Software Corporation, beziehungsweise Nativescript, wirbt mit einem hundertprozentigen Zugriff auf die native Plattform API (Android oder iOS), die direkt mit JavaScript aufgerufen werden kann. Auf diese Weise muss kein nativer Code vorgeschrieben werden, wie beispielsweise bei React Native und es wird signifikant Entwicklungs- und vor allem Weiterbildungszeit gespart.[30] Wenn in Nativescript Plug-Ins geschrieben werden sollen, die auf bereits vorhandene Android Plug-Ins zugreifen, so konvertiert die Nativescript Runtime sämtliche vorhandenen Typen in ein jeweiliges Pendant der anderen Sprache, um auf nativen Code zugreifen zu können. Zum Beispiel werden folgende Typen “int, long, float, double” aus der Objective-C Welt lediglich in den Typ “number” aus der JavaScript Welt übersetzt, um den Code aufrufen zu können. Dies gestaltet sich bei entsprechend komplexeren Datenstrukturen komplizierter. Durch diese Vereinfachung können auch Detailinformationen verloren gehen, die den Grenzen der Programmiersprache geschuldet sind. Integer-Zahlen aus Objective-C, deren Zahlenwert größer als $\pm 2^{53}$ sind, würden so innerhalb von JavaScript einiges an Schärfe verlieren, da hier Zahlenwerte auf ein Maximum von 53-Bit beschränkt sind.[31]

Zusätzlich gibt es noch die Option, während der Entwicklung ein sogenanntes Hot-Module-Replacement-Feature zu nutzen. Dieses gibt Nativescript die Fähigkeit, Änderungen am Code direkt einbinden zu können, ohne dass die komplette App neu geladen und gebaut werden muss.[32] Dabei ist völlig egal, ob die Zielplattform direkt auf dem Rechner in Form eines Android Emulators/iOS Simulator läuft oder direkt auf einem verbundenen mobilen Endgerät ausgeführt wird. Dadurch ist eine Zeitersparnis während der Entwicklung möglich.

[30] Nativescript Presentation, Features
[31] Nativescript Documentation, Marshalling
[32] Nativescript Blog, Hot Module Replacement

3 Verwendete Technologie: Nativescript

Dem Studenten stehen zum Zeitpunkt der Anfertigung der Bachelorarbeit bereits verschiedene Kenntnisse im Umgang mit typisierten Sprachen zur Verfügung. Deshalb ist die Wahl naheliegend, ein Framework zu nutzen, das eine solche verwendet. Da an der aktuellen Arbeitsstelle des Studenten verschiedene Möglichkeiten für die Cross-Compiler-Entwicklung evaluiert und getestet worden sind, bestand ein grundlegendes Interesse darin, bestehende Erfahrungen anwenden zu können. Das Nativescript-Framework ist dabei besonders aufgefallen, da kein plattformspezifisches Wissen akquiriert werden musste. So liegt es im Interessenbereich des Studenten zu prüfen, ob mit der genannten Technologie dieselben Ergebnisse wie mit einer rein nativen Option erzielt werden können.

Im folgenden Kapitel werden zunächst einzelne Bestandteile von Nativescript aufgegriffen und erklärt, bevor im Detail mit Hilfe einer technischen Übersicht beschrieben wird, wie durch Einsatz eines Web-Frameworks nativer Code auf einem mobilen Endgerät entstehen kann. Beendet wird das Kapitel mit einer detaillierten Vorstellung der ausgewählten Technologie für die Umsetzung der Stundenplan-App. Dabei werden sowohl die eingesetzten Sprachen als auch der technische Hintergrund des Frameworks thematisiert.

3.1 TypeScript

TypeScript ist eine optional typisierte Programmiersprache, die mit jeder JavaScript-Engine lauffähig ist, solange mindestens die Version ES3 des JavaScript-Sprachkerns unterstützt wird. Sobald TypeScript kompiliert wird, entsteht vollständig funktionierender JavaScript-Code. Das bedeutet, dass TypeScript jederzeit mit JavaScript kommunizieren kann und umgekehrt.[33]

3.2 Angular

Angular ist ein Framework, mit dem Client-Applikationen für den Browser geschrieben werden können. Für das Layout wird standardisiertes HTML/CSS genutzt und als Grundsprache stehen sowohl TypeScript als auch JavaScript zur Verfügung. [34]

[33] Typescript, Description

[34] Angular Documentation, Architecture

Angular Apps werden realisiert, indem der Client diese HTML-Layouts nutzt und durch einzelne Komponenten ansteuert. Sobald Logik geschrieben werden muss, die mehr erfordert, als einfach nur UI-Elemente ein- und auszublenden, wird empfohlen, diese neben den Komponenten in sogenannte Services auszulagern.[35] Im weiteren Sinne wird auch davon gesprochen, dass eine Angular-Komponente die kontrollierende Klasse eines einzelnen Views widerspiegelt. Die Klasse steuert das Verhalten der Ansicht bei Benutzerinteraktionen.[36]

Nativescript kennt zwar keine HTML-Elemente und kann diese deshalb nicht nutzen, dennoch werden viele Möglichkeiten und Konzepte übernommen, die das Programmieren schneller und effizienter gestalten. Ebenfalls kann ohne viel Aufwand vorhandener Nativescript Code in eine reine Angular-Web-App umgewandelt werden. Lediglich die eben beschriebene User-Interface-Ebene benötigt in diesem Fall weitere Anpassungen.[37] Jedoch muss separat darauf geachtet werden, inwieweit die Hardware des Endgerätes angesprochen wird, da diese Funktionalität nicht portiert werden kann.

3.3 Template Syntax (XML & CSS)

Die Ansicht eines NativeScript Projektes wird mit XML in Form von Nativescript-UI-Elementen dargestellt. Anstelle von typischen Web-Elementen wie “” oder “ ” werden alternative Tags angeboten wie “<ListView>”, “<Button>” oder “<GridLayout>”.[38] Ein einfaches Textfeld-Element würde man beispielsweise folgendermaßen darstellen:

```
1   <StackLayout orientation="vertical">
2       <Label text="Hello World" class="big-text"></Label>
3       <Label text="Goodnight World" *ngIf="bool"></Label>
4   </StackLayout>
```

Codebeispiel 1: Implementierung eines "Hello World"-Label (Eigene Darstellung)

Der Codeabschnitt zeigt dabei ein einfaches Beispiel für einen Layout-Container, der zwei weitere Textelemente beinhaltet. Für den Nutzer sind dabei lediglich die Texte untereinander sortiert sichtbar. Den Elementen können dabei in Form von Attributen weitere Eigenschaften

[35] Angular Documentation, Services
[36] Angular Documentation, Component Architecture
[37] Nativescript Documentation, Angular Code Sharing
[38] Nativescript Documentation, Template Syntax

zugewiesen werden. Im Codebeispiel ist im zweiten Label-Aufruf auch ein Beispiel für eine Angular-Template-Syntax dargestellt. Die meisten Befehle dieser Art werden mit *ng abgekürzt und sind dazu im Stande die UI zu manipulieren. Als Veranschaulichung wird hier auf eine boolesche Variable namens “bool” verwiesen, die innerhalb der Logik den Wert true oder false zurückgeben muss. Je nach Ergebnis wird der komplette zweite Label-Aufruf überhaupt erst in die DOM geschrieben oder komplett weggelassen, wie bei einer klassischen If-Anweisung.

Nach Projekterstellung befindet sich eine .html-Datei im Hauptordner, die für das Einpflegen des User-Interface vorgesehen ist. Im selben Ordner ist ebenfalls eine .css-Datei vorhanden, in der Änderungen in typischer Syntax, wie „font-size: 20“ oder „color: #000000“ getroffen werden können.[39] Der Aufruf auf die CSS-Klasse geschieht innerhalb der Abbildung mit class=”big-text”.

3.4 Technische Übersicht des Frameworks

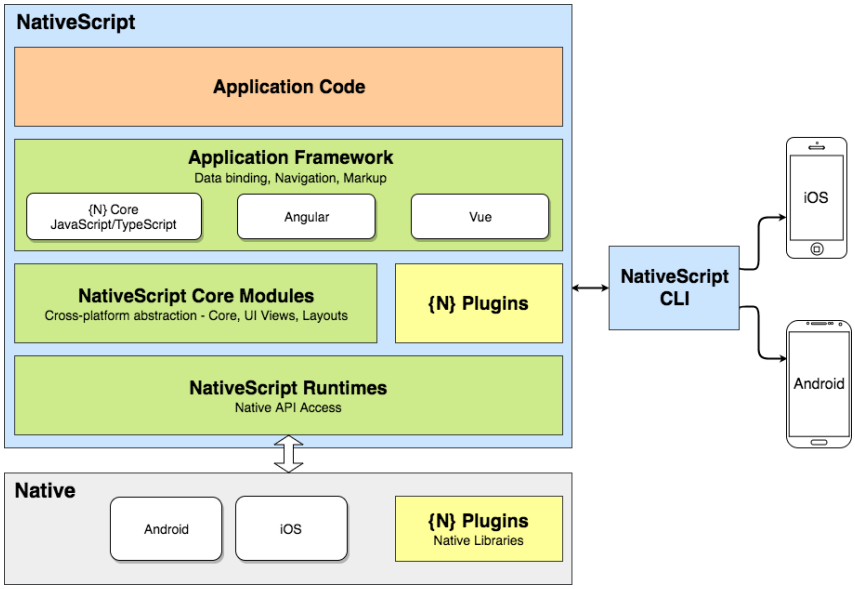

Abbildung 1: Technische Übersicht des Frameworks[40]

Nativescript setzt sich aus einer Vielzahl verschiedener Technologien zusammen. Erst dadurch ist es möglich Applikationen zu entwickeln, die nativ kompiliert werden. Die Blöcke, die innerhalb der Abbildung mit “Application Code” und “Application Framework” beschrieben wurden, sind bereits in Kapitel 3.1 TypesScript, 3.2 Angular und 3.3 Template

[39] Nativescript Documentation, UI Styling
[40] Nativescript Documentation, Technical Overview

Syntax erklärt worden. Welche Technologien an welcher Stelle eine Rolle spielen um den gesamten Prozess zu ermöglichen, soll im folgenden Abschnitt geklärt werden.

Der dritte Block, die **NativeScript Core Modules** stellen die Ebene dar, in der die Abstraktion geschieht, die benötigt wird, um auf die zugrunde liegende native Plattform zugreifen zu können. Die Dokumentation führt als Beispiel das Gesten-Modul an, wodurch Funktionen, wie “Pinch-To-Zoom” oder “Swipe”-Gesten ermöglicht werden. Im Modul ist eine standardisierte JavaScript-API definiert, die den TypeScript/JavaScript-Code der App übersetzt, um die native Gesten-API aufrufen zu können.[41]

Die Übersetzung von TypeScript/JavaScript Code geschieht mithilfe der **NativeScript Runtime**, dem vierten Block innerhalb der Abbildung. Die Runtime erlaubt es die API des Android und iOS Frameworks mit JavaScript aufzurufen. Um dies zu ermöglichen nutzt das Framework eine JavaScript Virtual Machine. Unter Android kommt Google’s V8-Engine zum Einsatz, unter iOS wird die JavaScriptCore Implementierung des WebKits genutzt.[42]

NativeScript bietet einen eigenen Marktplatz[43] an für jegliche Art von **Plug-Ins**.[44] Diese können entweder gekapselte Funktionen beinhalten, um Entwicklern Arbeit zu ersparen oder enthalten dank der Runtime direkten Zugriff auf native Bibliotheken.[45]

Die **Nativescript CLI**, dem rechten und letzten Block innerhalb der Abbildung, ist ein von Progress bereit gestelltes Kommandozeilen-Tool, mit dem Nativescript Apps erstellt, gebaut und gestartet werden können. Das Tool läuft unter Windows, Linux und Mac. Hiermit wird der gesamte Prozess in der Kette automatisiert gestartet, der alle Technologien vereint. Ein einfacher Befehl, wie “tns run android” oder “tns run ios” reicht nach Installation der CLI. Der Code wird je nach Zielplattform an Android Studio oder xCode zum Bauen weitergeleitet, anschließend wird die App entweder im Emulator/Simulator geöffnet oder direkt auf dem Endgerät ausgeführt.[46]

---

[41] Nativescript Documentation, Core Modules
[42] Nativescript Documentation, Runtimes
[43] Nativescript, Marketplace
[44] Nativescript Documentation, Finding Plug-Ins
[45] Nativescript Documentation, Plug-Ins
[46] Nativescript Documentation, CLI

4 Analyse

Dieser Abschnitt befasst sich mit den verschiedenen Untersuchungen und Analysen, die getroffen werden müssen, bevor mit der Entwicklung einer App begonnen werden kann. Sowohl die aktuelle Situation an der Hochschule für die Beschaffung der Stundenpläne als auch die Erforschung von möglichen Schnittstellen ist im Anschluss Thema. Sobald alle Vorarbeiten getroffen sind, kann ein Anforderungsdokument für die Anwendung erstellt werden, die den Rahmen der Entwicklung im Prozess vorgeben.

4.1 Analyse des Ist-Stands

Die einst angebotene offizielle Applikation ist über die Jahre sowohl aus dem Android als auch dem iOS Store verschwunden und somit für Studenten nicht auf üblichem Weg erhältlich. Wenn diese ihre aktuellen Modulpläne an der Ernst-Abbe-Hochschule Jena überprüfen möchten, müssen sie gezwungenermaßen die dafür angebotene Internetpräsenz betreten.[47] Eine Google-Suche verweist zumindest unter Android Endgeräten auf eine ehemalige Version in einem alternativen App Store, bei der aber nicht verifizierbar ist, ob es sich um die letzte erhältliche Version handelt oder eventuell schadhafte Manipulationen am Code vorgenommen wurden.[48] Eine Überprüfung der aktuellen Internetpräsenz zeigt mithilfe eines sogenannten Web Inspectors, unter dem Browser Chrome auch DevTools[49] genannt, dass weder ein Cookie gesetzt noch ein einzelner Wert in den lokalen Speicher geschrieben wird. Dadurch ist das Endgerät, das den aktuellen Stundenplan anzeigt, nicht in der Lage die Eingaben des Nutzers zu speichern. Studenten müssen also folglich bei jedem Besuch der Internetseite dieselben sechs Schritte wiederholen:

1. Wahl des gewünschten Studiengangs
2. Wahl des gewünschten Semesters
3. Wahl der Arbeitsgruppe innerhalb des Semesters
4. Wahl der anzuzeigenden, zeitlichen Spanne
5. Bestätigen durch Klick auf „Anzeigen“
6. Warten, bis Weiterleitung und Seitenaufbau abgeschlossen sind

[47] EAH, Stundenplanung
[48] Apkmonk, EAH Jena
[49] Google Developers, DevTools

Nicht zu vernachlässigen ist auch die Tatsache, dass sich dieser Prozess fast täglich wiederholen kann. Gerade Studenten, die von weit her anreißen, wollen über mögliche Ausfälle informiert bleiben. Aufgrund der hohen Anzahl angebotener Studiengänge und vorhandener Arbeitsgruppen, gestaltet sich die Navigation schwierig. Es wurde mit Hilfe des browserinternen Web-Inspectors und eines physischen Smartphones (Samsung Galaxy S7, Android 8.0.0) überprüft, ob der Internetauftritt beim Betreten an die Größe des jeweiligen Endgerätes anpasst wird. Jedoch bleibt die Ansicht komplett unverändert und der Benutzer ist gezwungen, den Inhalt mit einer Zoom-Geste zu vergrößern, da es sonst kaum lesbar, geschweige denn bedienbar ist.

Alternativ wird angeboten, die Stundenpläne vollständig in Form einer einzelnen Datei herunterzuladen, damit diese anschließend in ein kompatibles Kalenderprogramm importiert werden können. Das löst zwar das Problem mit den sich wiederholenden, erschwerten Nutzereingaben, macht die Pläne allerdings komplett statisch. Wenn sich die Startzeit oder Raumnummer eines einzelnen Moduls innerhalb des Stundenplanes verändert, würde der Endnutzer davon nichts mitbekommen. Deshalb muss oftmals weiterhin die Webseite genutzt werden, möchte ein Studierender über sämtliche Änderungen im Lehrplan informiert bleiben.

Zusätzlich gibt es aus eigener Erfahrung noch einige Fälle, die zwar im Alltag der Hochschule vorkommen, aber im aktuellen Design des Stundenplans nicht wiedergespiegelt sind. Dazu gehören unter anderem Studenten, die eine Prüfung nicht im ersten Durchlauf bestanden haben oder beschließen ein Modul zu einem späteren Zeitpunkt abzulegen. In beiden Fällen müssten Studierende, um über sämtliche Änderungen aller Vorlesungen informiert zu bleiben, konstant zwei verschiedene Stundenpläne beobachten. Diese können in ihrer Auswahl zwischen Semestern und sogar Studiengängen variieren. Falls der eigene Studiengang im Folgesemester das vergangene Modul nicht anbieten kann, so müssen Alternativen aus dem eigenen Fachbereich überprüft werden.

4.2 Soll-Zustand

Mit der zu entwickelnden Applikation soll überprüft werden, ob die Funktionen, die ehemals in der offiziellen App angeboten worden sind, auch innerhalb eines Cross-Compiler Frameworks umsetzbar sind und der Performance einer nativen Applikation gleichkommen. Neben der technischen Machbarkeit steht die Erweiterung bestehender Funktionalitäten im

Fokus. Die bisherige Anzeige des Stundenplans soll durch den Studenten angepasst werden können. Anstatt zwei verschiedene Stundenpläne verfolgen zu müssen, ist das Ziel ein bearbeitbares Modell anzubieten, bei dem der Endnutzer selbst die Wahl hat, welche Module angezeigt werden sollen. Auf diese Weise soll der Problematik des Ist-Standes (siehe Kapitel 4.1) entgegengewirkt werden. Die Ansicht der Speisepläne lässt in der ursprünglichen App ebenfalls zu wünschen übrig. Aushänge und Preise von Gerichten der lokalen Kantine konnten zwar eingesehen werden, beschränkten sich allerdings auf ein Mensagebäude zur selben Zeit. Um die Ansicht zu ändern, musste diese umständlich durch den Nutzer gewechselt werden. Wie aber am Beispiel der EAH Jena zu sehen ist, rotiert eine Vielzahl von Studenten mindestens zwischen zwei Mensen, da beide sich auf demselben Hochschulgelände befinden (Cafeteria EAH und Mensa Carl-Zeiss-Promenade). Deshalb ist auch hier das Ziel, die bestehende Funktionsweise um einen modularen Ansatz zu erweitern, sodass nicht hin und her geschaltet werden muss.

4.3 Akquirieren der Rohdaten

Um überhaupt Datensätze, wie Stunden- und Speisepläne anzeigen zu können, bedarf es einer Schnittstelle, die entsprechende Daten ausliefert. Da während des Analyseprozesses keine Informationen über die Existenz einer solchen Schnittstelle bekannt waren, mussten offizielle Frontends untersucht werden. So kann zumindest die Machbarkeit des Projektes durch die Bereitstellung der Stundenplan-Rohdaten analysiert werden. Dabei steht zum einen die Webseite, die bisher genutzt wurde, zur Verfügung. Desweiteren soll in einer späteren Analyse die ehemalige App der EAH für die Anforderungen an die Cross-Plattform-App herangezogen werden. Da beide die Daten der Hochschule anzeigen, eignen sie sich für eine Überprüfung auf mögliche Datensätze. Die Untersuchung der Webpräsenz erfolgte mit den Chrome DevTools.[50] Hauptaugenmerk lag hier auf möglichen HTTP-Requests, die ausgelesen werden können oder lokal gespeicherten Daten, die entsprechende Datensätze liefern. Es existiert zwar eine Liste, die auf die verschiedenen IDs der Studiengänge verweist, mit deren Hilfe zumindest eine Ziel-URL für einen Stundenplan aufgebaut werden könnte. Jedoch sind auf der Unterseite keine weiteren Rohdaten aufzufinden. Falls lediglich die Webseite zur Verfügung stünde, um an Modulpläne und Raumnummern zu gelangen, wäre eine Option, den gesamten Quellcode mit Hilfe eines Skriptes abzugreifen. Zusätzlich bedarf es Logik, die den Code von seinem HTML-Markup

[50] Google Developers, DevTools

befreit und anschließend in eine Datenbank sortiert. Auf diesem Weg können allerdings viele Fehler entstehen und der Entwicklungsaufwand steigt exorbitant.

Für die Untersuchung der ehemaligen App wurde das Tool „Charles Proxy“ [51] genutzt. Dabei handelt es sich um einen HTTP-Proxy, der zwischen das Endgerät und dem Internet geschaltet werden kann, um sämtlichen Datenverkehr, der versendet und empfangen wird, auszulesen.[52] Um die Untersuchung beginnen zu können, bedarf es zunächst einiger Vorkehrungen. Das Programm ist unter Linux, Windows und macOS erhältlich und bietet die Möglichkeit, eine IP-Adresse auszugeben, die als Proxy-Server für das Smartphone dient. Diese wird nun innerhalb der WiFi-Einstellungen des verbundenen Netzwerkes auf dem Smartphone unter dem Feld Proxy-Server eingegeben. Im Anschluss wird sämtlicher Datenverkehr mitgeschnitten und angezeigt.

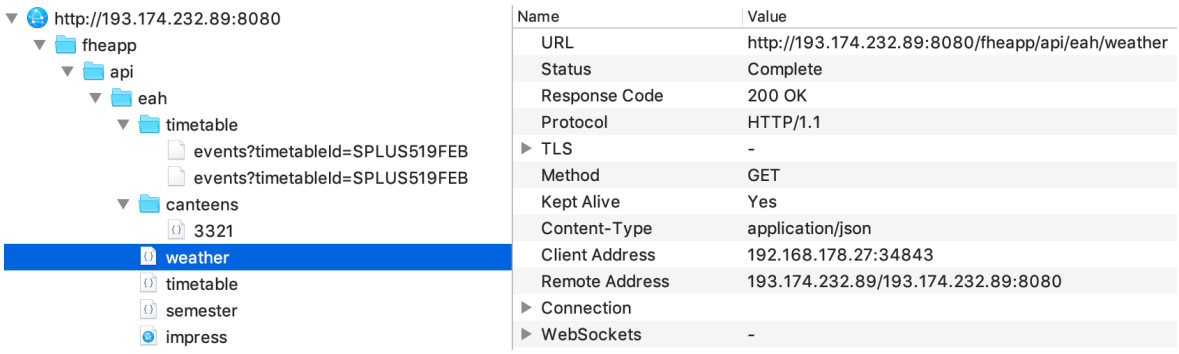

Abbildung 2: Mitschnitt der ehemaligen EAH-App in Charles

Abbildung 2 zeigt im Detail, wie die Ausgabe des Programmes aussieht. In den Details auf der rechten Seite ist zu sehen, dass für die gesamte Kommunikation GET-Befehle versendet werden. Die Zieladresse ist ebenfalls vollständig ersichtlich. Nach einer kurzen Überprüfung im Browser ist klar, dass die Schnittstellen bis auf einige Ausnahmen, die im nachfolgenden Kapitel im Detail geklärt werden (siehe Kapitel 4.4), funktionieren und auch sinnvolle Daten zurückgeben. Als Ausgabe-Typ des Inhalts werden Daten im JSON-Format zurückgegeben, was die Kurzform für „JavaScript Object Notation“ ist. Da es sich dabei um ein gängiges Format handelt, das gerade in der Angular und der generellen Web-Frontend Welt Verwendung findet, kann die Schnittstelle bedenkenlos in die Cross-Plattform-App implementiert werden. Nachdem sämtliche GET-Requests extrahiert und notiert worden sind, finden jene in einer separaten Datei Platz. Auf diese Weise sind die Links per Aufruf

[51] Charles, Web Proxy

[52] Charles, About Charles

über die gesamte App hinweg zentralisiert nutzbar und müssen bei Änderungen nur an einer Stelle manipuliert werden.[53]

Hierdurch wird die Tatsache deutlich, dass keine dedizierte Schnittstelle existiert, an die sogenannte POST-Befehle versendet werden können, um beispielsweise an einzelne, ausgewählte Module zu kommen. Stattdessen existiert einzig eine GET-Schnittstelle, die mit der passenden ID ein ganzes Set an Modulen zurückgibt. Die ID ergibt sich aus der Vorauswahl des Einrichtungsassistenten und kann ebenfalls über eine Schnittstelle als Liste abgefragt werden. Es wird nun klar, dass zumindest Schnittstellen existieren. Im nachfolgenden Abschnitt wird dabei die App, die genannte Endpunkte nutzt, untersucht. So kann gleichzeitig die Qualität und Sinnhaftigkeit der empfangenen Daten überprüft werden.

4.4 Analyse der ehemaligen EAH-App

Um Funktionalitäten zum Erreichen des Soll-Zustands implementieren zu können, wird zunächst untersucht und geprüft, ob die genutzten Schnittstellen weiterhin verwendet werden können. Im Folgenden werden die wesentlichen Funktionen der App den einzelnen Menüpunkten der App-Navigation entnommen.

4.4.1 News

Die App beginnt auf der Startseite mit den Pressemitteilungen der Ernst-Abbe-Hochschule. Das Selektieren der gewünschten Nachricht zeigt den gesamten Text in einer weiteren Unterseite. Auf der News-Seite wird das obere Drittel der Anzeige den Daten der Wetterstation vor Ort gewidmet und bietet einen Überblick über die aktuelle Temperatur.

4.4.2 Stundenplan

Unter diesem Punkt erreichbar kann die Konstellation aus Studiengang, Semester und Arbeitsgruppe bearbeitet und als Favorit gespeichert werden. Anschließend erscheint der dazugehörige Plan. Navigiert wird mit Wisch-Gesten, wodurch die Wahl der aktuellen Kalenderwoche editiert wird. Ein Zahnrad in der oberen, rechten Ecke erlaubt die Manipulation des eingestellten Favoriten.

[53] Siehe Anhang: Inhalt der CD-ROM, Programmcode, eahjena/src/app/shared/config.ts

4.4.3 Personen

Über „Personen“ erlaubt die Ansicht die Eingabe eines Vor- und Nachnamens. Es wird keine weitere Information darüber geliefert, ob die Suche sämtliche Mitarbeiter der EAH einschließt oder sich nur auf bestimmte Personengruppen beschränkt. Nach kurzer Überprüfung mit einigen Namen von Dozenten, die auf der Hauptseite aufgeführt sind, steht fest, dass die Schnittstelle nicht mehr ansprechbar ist. Die Anzeige bleibt vollständig weiß.

4.4.4 Mensa

Hier wird die Anzeige jeweils eines Speiseplans realisiert. Die aktive Wahl lässt sich über ein Zahnrad in der oberen, rechten Ecke bearbeiten und gibt Zugriff auf neun verschiedene Kantinen in Jena. Auf Wisch-Gesten wird verzichtet, stattdessen gibt es eine Liste, die ohne weiteres Zutun bis zum Ende der gebotenen Daten gescrollt werden kann. Gegliedert wird nach Tagen sowie Gerichten. Neben der Beschreibung des Gerichts stehen auch die Preise.

4.4.5 Campus

Eine Liste gibt Übersicht über sämtliche Gebäude der EAH Jena. Nach Auswahl wird auf eine schematische Zeichnung des Gebäudes verwiesen, die je nach Bedarf vergrößert und verkleinert werden kann. Detailinformationen zu Raumnummern sind nicht gegeben. Hier handelt es sich lediglich um Bilder der einzelnen Grundrisse.

4.4.6 Termine

Jede Hochschule hat eine Reihe von Semestereckdaten, die verschiedene Meilensteine im Semester anzeigen. Dazu gehören neben dem Semesterbeginn auch der Beginn des Prüfungszeitraums, Feier- und Brückentage, sowie weitere wichtige Termine. Das Frontend bietet nur Einsicht bis Anfang 2019, damit scheint die Schnittstelle veraltet zu sein.

4.5 Anforderungen an die hybride App

4.5.1 Funktionale Anforderungen

Für gewöhnlich definiert der jeweilige Auftraggeber der zu entwickelnden Software die verschiedenen Anforderungen innerhalb eines Lastenhefts. Auf diese Weise können in Zusammenarbeit mit den beauftragten Entwicklern Unstimmigkeiten von vornherein geklärt werden. Da hingegen versucht wird eine Alternative für die alte Applikation zu finden, die auf den beiden großen Plattformen läuft, dient die Analyse der bisherigen Stundenplan-

Variante als Vorlage für die funktionalen Anforderungen. Die Liste der Funktionen wird allerdings um drei Bereiche erweitert. Erstens wird nun eine Startseite hinzugefügt, die Informationen anzeigt, die nur den heutigen Tag (bzw. den nächstmöglichen gelieferten Zeitpunkt der Schnittstelle) betreffen. Des Weiteren werden sowohl die Modul- als auch die Speisepläne durch den Studenten editierbar sein.

In der Literatur findet man sogenannte User Storys als Beispiel für die Formulierung von Anforderungen im Requirements Engineering. Diese werden häufig in agilen Entwicklungsmodellen, wie Scrum eingesetzt und vollständig aus der Sicht des Anwenders formuliert.[54] Als Muster für die Beschreibung bietet sich der Entwurf von Mike Cohn an: „Als (Akteur) möchte ich (folgende Funktion durchführen), um (daraus folgenden Nutzen zu ziehen).“ [55] Der Akteur wird durch die Entität „Student“ im weiteren Verlauf dargestellt. Hier sind verschiedene Rollen möglich, um alle Bedürfnisse zu erfüllen. Da Studenten die Hauptzielgruppe sind, bleibt es bei einer Interessengruppe. Auf diese Weise können Anforderungen schnell formuliert werden. Der Detailgrad wird zwar durch die Art der Formulierung verringert, jedoch kann Zeit in die Implementierung investiert werden. Wenn User Storys einen komplexen Detailgrad aufweisen, können diese meist folglich in viele weitere Kleinere aufgeteilt werden. Solche Stories, die über der Sammlung von Kleineren stehen, werden auch Epics genannt.[56] Während der Implementierung diente die Tabelle als Übersicht des Entwicklungsfortschrittes. Gegliedert wird nach Menüpunkten in der App, die genutzt werden sollen, um strukturiertes Vorgehen zu ermöglichen.

4.5.1.1 Hauptseite

Nr.	Typ	Beschreibung
US#1	Epic	Als Student möchte ich nach App-Start über heutige Speisen und bevorstehende Fächer informiert werden, die mich betreffen, damit ich meine Zeit nicht mit langen Klickpfaden verschwende.
US#1.1	User-Story	Als Student möchte ich auf der Hauptseite über alle Fächer des bevorstehenden Tages informiert werden, die ich eingespeichert

[54] Rau, 2016, S. 21
[55] Cohn, 2009, S. 246
[56] ebd., S. 252

		habe, damit ich schnell einen Überblick über Raumnummer und Uhrzeit bekomme.
US#1.2	User-Story	Als Student möchte ich auf der Hauptseite über alle Speisen informiert werden, die meine gespeicherten Kantinen betreffen, damit ich schnell einen Überblick über Zutaten, Preis und Ortschaft bekomme.

4.5.1.2 Stundenplan

Nr.	Typ	Beschreibung
US#2	Epic	Als Student möchte ich eine Übersicht aller Studienfächer in einer Ansicht, die ich frei bearbeiten kann, damit ich nicht mehr mehrere Stundenpläne nacheinander überprüfen muss.
US#2.1	User-Story	Als Student möchte ich innerhalb meiner Studienübersicht Informationen über Raumnummer, Dozenten und Uhrzeit erhalten, damit ich die wichtigsten Informationen schnell aufnehmen kann.
US#2.2	User-Story	Als Student möchte ich einzelne Module meiner Studienübersicht entfernen können, damit ich nur Fächer sehe, die mich betreffen.
US#2.3	User-Story	Als Student möchte ich einzelne Module aus anderen Studiengängen und Semestern zu meiner Übersicht hinzufügen können, damit ich nicht zwischen verschiedenen Plänen hin- und herwechseln muss.

4.5.1.3 Speiseplan

Nr.	Typ	Beschreibung
US#3	Epic	Als Student möchte ich in einer Übersicht alle Speisen meiner gespeicherten Kantinen überblicken, damit ich nicht mehr zwischen mehreren Kantinen hin- und herwechseln muss.

US#3.1	User-Story	Als Student möchte ich innerhalb meiner Kantinenansicht Einblick in Preise, Zutaten und Kantine erhalten, damit ich die wichtigsten Informationen schnell aufnehmen kann.
US#3.2	User-Story	Als Student möchte ich die angezeigten Kantinen meines Speiseplans bearbeiten können, damit ich nur Mensen sehe, die mich betreffen.

4.5.1.4 Neues

Nr.	Typ	Beschreibung
US#4	User-Story	Als Student möchte ich eine Übersicht aller Nachrichten der Hochschule Jena, damit ich während meines Studiums informiert bleibe.

4.5.1.5 Wetter

Nr.	Typ	Beschreibung
US#5	User-Story	Als Student möchte ich eine Übersicht der Daten der Klimastation an der Hochschule, damit ich weiß, welche Temperatur gerade vor Ort ist.

4.5.1.6 Rechtliches

Nr.	Typ	Beschreibung
US#6	User-Story	Als App-Entwickler möchte ich ein Impressum einpflegen, damit ich rechtlich abgesichert bin.

4.5.1.7 Einstellungen

Nr.	Typ	Beschreibung
US#7	User-Story	Als Student möchte ich einstellen können, ob ich über Änderungen oder bevorstehende Veranstaltungen benachrichtigt werde, damit ich diese nicht verpasse.

4.5.2 Nichtfunktionale Anforderungen

Während funktionale Anforderungen oftmals festlegen, was genau ein bestimmtes System bewerkstelligen soll, beschreiben nichtfunktionale Anforderungen, welche Qualität ein System zu erfüllen und wie es zu arbeiten hat.[57]

Für die Klassifizierung der einzelnen nicht-funktionalen Anforderungen wird die Norm ISO/IEC 9126 verwendet.[58] Diese Norm bezieht sich nicht auf einzelne Funktionalitäten, sondern auf Qualitätsmerkmale und die gesamte Leistung des Systems. Es werden Aspekte beschrieben, die mehrere oder alle funktionalen Anforderungen auf einmal betreffen können.

4.5.2.1 Funktionalität

Bei der Funktionalität geht es nicht nur darum, die in Kapitel 4.5.1 beschriebenen Funktionen umzusetzen, sondern auch die Sicherheit und Genauigkeit zu wahren. Damit ist unter dem Aspekt der Sicherheit gemeint, die App- und firmeninternen Daten vor Dritten zu schützen. Da in dieser App nicht mit sicherheitskritischen Informationen, wie personenbezogenen Daten gearbeitet wird, spielt die Sicherheit hier keine übergeordnete Rolle. Die Genauigkeit soll sicherstellen, dass bei bereitgestellten Funktionen auch die richtigen Ergebnisse zurückkommen. Wenn Informationen über Studiengänge aufgerufen werden, wird erwartet auch diese Informationen zu bekommen und nicht etwa die eines anderen Studiengangs.

4.5.2.2 Zuverlässigkeit

Innerhalb des Punktes Zuverlässigkeit soll sichergestellt werden, dass die App ein bestimmtes Leistungsniveau aufrechterhält, wenn sie typischen Situationen eines studentischen Alltags ausgesetzt ist. Dazu gehören Funktionen, wie das Bereitstellen von

[57] Balzert, 2009, S. 464
[58] ebd., S. 468

Fächern im Falle eines Internetabbruchs. Aus eigener Erfahrung kann gesagt werden, dass die Netzabdeckung der Hochschule in einigen Gebäuden nur lückenhaft funktioniert.

4.5.2.3 Benutzbarkeit

Dem Nutzer soll ein konstantes User-Interface vorgelegt werden, das sich intuitiv benutzen lässt und an die Hochschule erinnert. Es soll schnell überschaubar sein, welche Funktionalitäten angeboten werden und wie diese zu bedienen sind. Das App-Design selbst soll dabei möglichst die bekannten Nutzungsmechanismen verwenden, die Endnutzer eines Smartphones gewohnt sind. Gerade bei einer Cross-Compiler-App steht im Fokus, die verschiedenen betriebssystemspezifischen Elemente zu nutzen, damit beim Kunden ein vertrautes Gefühl entsteht. Eines der Hauptziele ist es, alle Ansichten mit möglichst wenigen Klicks erreichen zu können, um der bisherigen Situation an der Hochschule entgegenzuwirken.

4.5.2.4 Effizienz

Die App soll auf den beiden genutzten Betriebssystemen möglichst effizient funktionieren. Hier muss zwischen mehreren Aspekten differenziert werden. Gerade bei der Performance ist sehr wichtig, wie schnell die Daten vom System bereitgestellt und genutzt werden können. Da aber während der Entwicklungsphase kein Einfluss auf die Leistung der Schnittstelle ausgeübt werden kann, wird keine Rücksicht auf dieses Thema genommen. Anders verhält es sich mit der plattformspezifischen Performance, die maßgebend vom genutzten Framework abhängt. Der Endnutzer soll nicht merken, dass es sich bei der Implementierung um eine Cross-Compiler-Variante handelt, sondern das Gefühl haben, es handelt sich um eine eigens für die Plattform konzipierte Applikation.

4.5.2.5 Wartbarkeit

Die Wartbarkeit beschreibt die Fähigkeit, den Code auch in Zukunft warten und pflegen zu können. Die Nutzung von bewährten Entwurfsmustern soll ermöglichen, dass die App weiterhin modifizierbar bleibt. Dadurch soll zum einen Stabilität gewährleistet werden, denn mit gut strukturiertem Code ist eine reibungslose Fehlersuche möglich. Zum anderen kann schnell auf spontane Änderungen reagiert werden. Das Gliedern von Funktionalitäten in Ansichten und verschiedene Komponenten, samt einer sinnvollen Ordnerstruktur ist ebenfalls erforderlich.

4.5.2.6 Portabilität

Da für die Entwicklung der Stundenplan-App Nativescript als Grundlage genutzt wird, stellt die Portabilität einen zentralen Punkt der Anforderungen dar. Der Anspruch besteht darin, dass der Code mit möglichst wenig plattformspezifischen Kenntnissen auf den Betriebssystemen Android und iOS installierbar und vollständig lauffähig ist.

5 Konzeption

5.1 Einrichtungsassistent

In Kapitel 4.3 „Akquirieren der Rohdaten“ wird beschrieben, woher die Rohdaten zur Darstellung der verschiedenen Stundenpläne kommen. Im Anschluss wird klar, dass es keine dedizierte Schnittstelle gibt, um an einzelne Fächer zu gelangen, die in den eigenen Stundenplan eingepflegt werden können. Es wird lediglich eine vollständige Liste aller Fächer übergeben, die einem bestimmten Studiengang/Semester angehören. Um an ein Set von Modulen zu kommen, muss jedes Mal eine Vorauswahl getroffen werden, die aus Studiengang, Semester und der zugeteilten Arbeitsgruppe besteht. Deshalb muss diese Auswahl auch dementsprechend bei der Konzeptionierung des Ablaufs der verschiedenen Ansichten beachtet werden.

Es wurde die Entscheidung getroffen, einen entsprechenden Einrichtungsassistenten zu implementieren, der zu Beginn des ersten App-Starts eines Nutzers eingeblendet wird. Da jedem Studenten ein Studiengang an der EAH Jena zugeteilt ist, kann er diesen direkt einstellen. Auf diese Weise besteht die Möglichkeit, weitere Auswahlmöglichkeiten innerhalb der App abzufangen. Neben der Wahl des passenden Studiengangs steht die Option zur Verfügung, Kantinen auszuwählen, die im Interesse des Studenten liegen. Zusätzlich können verschiedene Push-Benachrichtigungen an- und ausgeschaltet werden. Das Ganze wird im Assistenten innerhalb von drei Stufen dargestellt.

Durch die Wahl eines entsprechenden Modells ergeben sich verschiedene Vorteile. Zum einen sind sofort Daten auf der Startseite vorhanden und das Layout kann entsprechend gestaltet werden. Zum anderen können die entsprechenden Ansichten, die innerhalb des Assistenten Verwendung finden, in der App wiederverwendet werden. Die Kantinenwahl dient der regulären Kantinenansicht und lässt den Studenten seine gespeicherten Kantinen bearbeiten. Auf dieselbe Weise kann, sobald der Stundenplan bearbeitet werden soll, der Studienwahl-Assistent angezeigt werden, um entscheiden zu können, welche Fächer zusätzlich erscheinen sollen. Da das bis jetzt die einzige Möglichkeit war, um Informationen über Vorlesung anderer Studiengänge zu erhalten, sind die Studenten mit dem entsprechenden Prozedere vertraut. Das Umschalten der Push-Benachrichtigungen kann in den Menüpunkt „Einstellungen“ eingepflegt werden.

5.2 Wireframe

Mit Hilfe eines konzeptionellen grafischen Entwurfes, dem sogenannten Wireframe, soll der generelle Ablauf der einzelnen Ansichten skizziert werden. Ebenfalls Bestandteil ist der grundlegende Aufbau einzelner UI-Elemente im Verhältnis zueinander. Dabei sollen keine Details des Designs geklärt werden, wie die Farbgebung oder Schriftart, sondern die Einteilung der einzelnen Bereiche und der Aufbau einer jeweiligen Ansicht. Diese können dann zusammen mit den User Storys aus Kapitel 4.5.1 verwendet werden, um den Entwicklungsprozess zu vereinfachen.

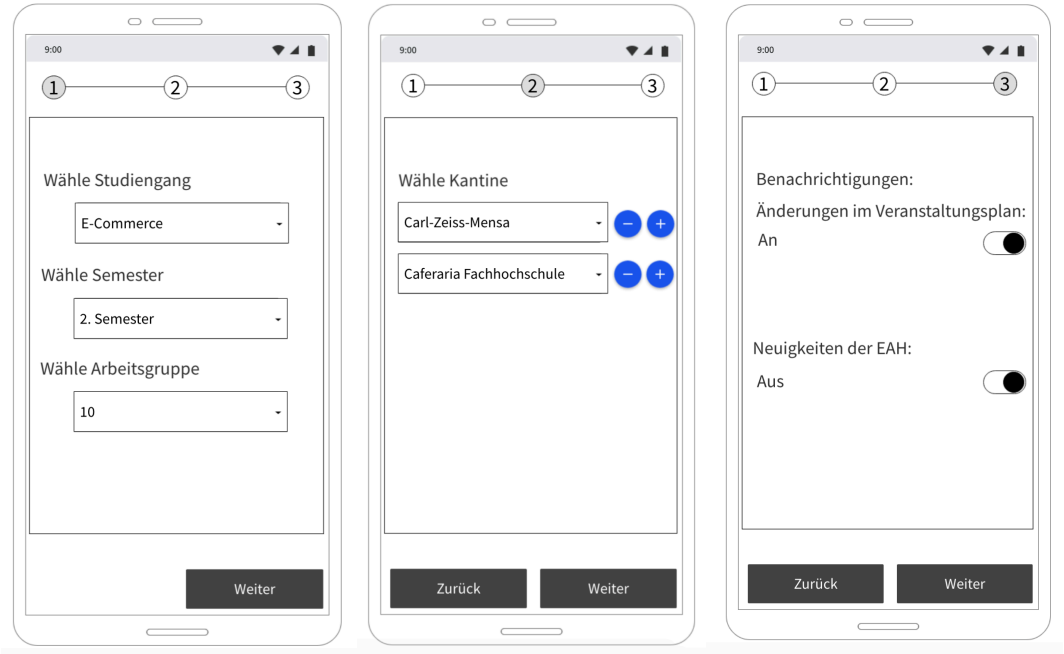

Abbildung 3: Wireframe des Einrichtungsassistenten

Abbildung 3 zeigt hier im ersten Schritt die einzelnen, zu durchlaufenden Schritte des Einrichtungsassistenten. Innerhalb des abgebildeten Smartphones umfasst das äußerste, eingezeichnete Rechteck bestimmte Bereiche des User-Interface. Bei diesen handelt es sich um diejenigen Teile des Layouts, die in späteren Schritten wiederverwendet werden können. Damit der Student seine jeweilige Auswahl tätigen kann, wird für die Darstellung im User-Interface ein sogenannter Dialog genutzt. Mit Dialogen werden bestimmte Interaktionen zwischen Nutzer und Endgerät ausgelöst. Dieser kann entweder mit einem Warnhinweis auf verschiedene Begebenheiten hinweisen und wird anschließend nur bestätigt oder es wird eine entsprechende Auswahl erwartet. Nativescript bedient sich dabei der zugrunde liegenden nativen Funktion von Android und iOS und zeigt das dazugehörige User-Interface im Stil des Betriebssystems an.

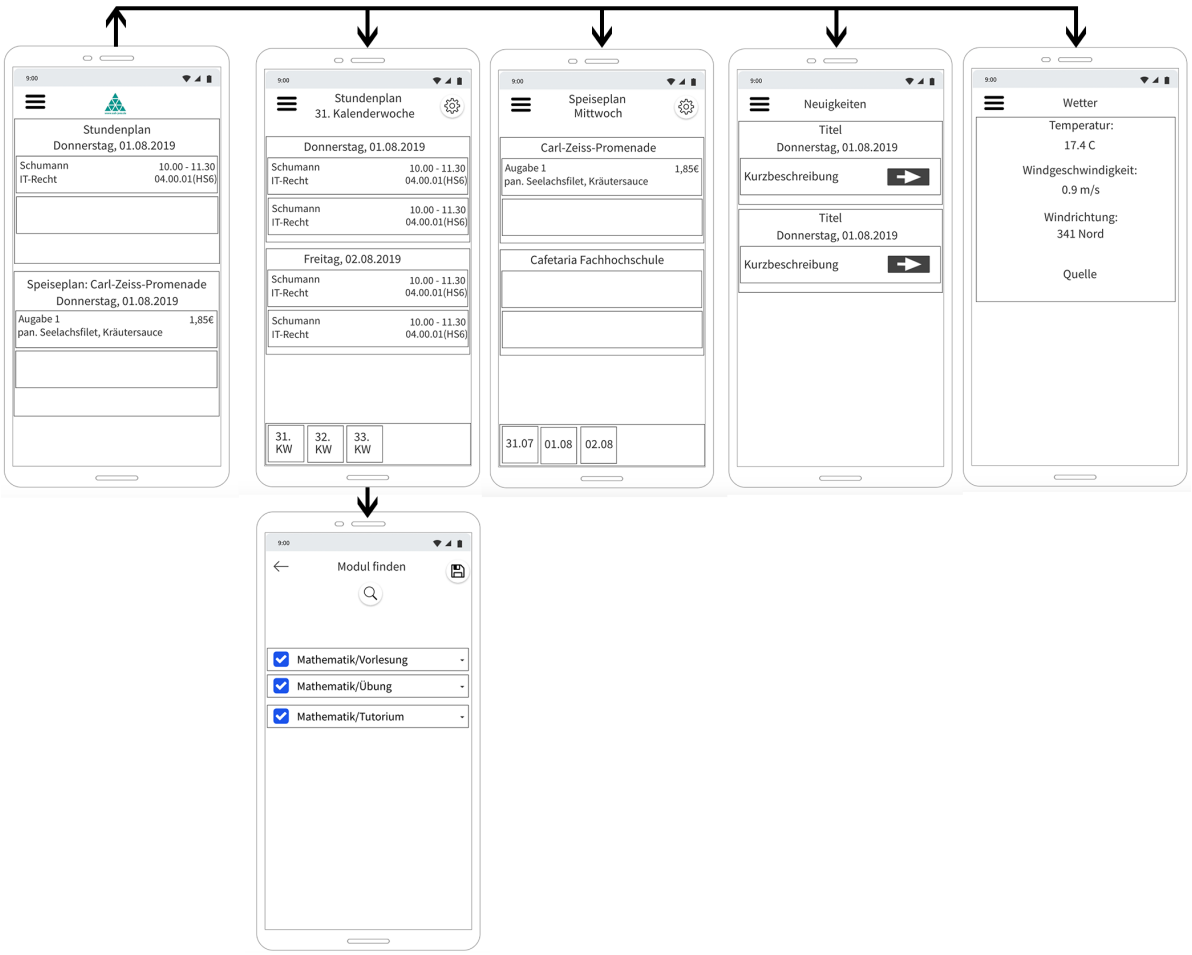

Abbildung 4: Wireframe der Cross-Plattform-App

Der restliche Verlauf der Applikation wird durch das Wireframe in Abbildung 4 dargestellt. Die Ansichten ergeben sich durch die festgelegten, funktionalen Anforderungen. Die erste Abbildung ganz links zeigt dabei die Startseite, die durch den Nutzer erreicht wird, sobald die Einrichtung abgeschlossen ist. Von ihr sind alle weiteren Unterseiten über die Navigation, dargestellt mit einem Icon in der oberen linken Ecke zu erreichen. Das Layout beschränkt sich hier auf zwei Bereiche, die in Form von separaten Listen dargestellt werden. Die Elemente aktualisieren sich entsprechend der getätigten Anpassungen, die auf der Unterseite der jeweiligen Rubrik getroffen werden können.

Das nächste Ziel des Pfeiles beschäftigt sich mit der Darstellung des Stundenplans. Dabei wird zum einen die entsprechende Hälfte der Startseite für Inhalte des jeweiligen Tages verwendet. Jedoch sollen pro Ansicht sämtliche Veranstaltungen der aktuellen Kalenderwoche eingeblendet werden. Über eine verschiebbare Leiste im unteren Bereich des Layouts wird ausgewählt, welche Woche dargestellt werden soll. Wisch-Gesten können

an dieser Stelle bei der Navigation zwischen den Ansichten helfen. Die aus dem vorherigen Schritt bekannten nativen Dialoge sollen auch hier wieder Verwendung finden. Das Betätigen eines einzelnen Moduls liefert dabei die Möglichkeit, die ausgewählte oder alle Veranstaltungen dieser Reihe auszublenden. Beispielsweise kann so ein Student entscheiden, ob er lediglich die Mathematikvorlesungen oder auch die dazugehörigen Tutorien anzeigen lässt.
In der oberen rechten Ecke signalisiert ein Icon dem Nutzer, dass der Stundenplan weitere Bearbeitungsoptionen liefert. Das Betätigen des Icons führt den Nutzer zu der Ansicht, die mit dem Pfeil unter dem Stundenplan in Abbildung 4 markiert wurde. Im Wireframe ist bereits eine Liste von Fächern zu sehen, die der Studierende in seinen Plan einarbeiten kann. Da die Schnittstelle aber eine Vorauswahl durch den Nutzer verlangt, wird der erste Schritt des Einrichtungsassistenten eingeblendet (siehe Abbildung 3). Sobald der Nutzer sich entschieden hat, fährt nach Bestätigung die Ansicht nach oben und zeigt das entsprechende Wireframe. Das Suchfeld kann mit einem Knopfdruck auf das Lupen-Icon wieder in den Vordergrund gebracht werden. Im Anschluss kann der Nutzer dann einzelne Module mit Hilfe einer Checkbox auswählen und in der oberen rechten Ecke speichern. Ein Klick auf die entsprechenden Module zeigt in einer Detailansicht, die sich darunter erstreckt, weitere Informationen, wie Tage und Uhrzeiten der jeweiligen Veranstaltung.

Der Aufbau des Speiseplans weist große Ähnlichkeiten zur Einteilung der Stundenpläne auf. Hier wird bewusst die Entscheidung getroffen, nicht, wie im Fall vorher eine jeweilige Kalenderwoche auf einmal anzuzeigen. Nur ein einzelner Tag erscheint in der Ansicht. Die Speisepläne der Mensen liefern zum einen wesentlich konstantere Daten, als ein Stundenplan. Es ist nicht unüblich, dass ein Student an einem Tag frei oder nur wenige Veranstaltungen hat. Zum anderen werden hier viel mehr Informationen auf einmal angezeigt, da die Titeltexte im Schnitt wesentlich länger sind. Während die Einträge im vorherigen Beispiel sich auf den Namen eines Dozenten und das dazugehörige Fach beschränken, werden im jetzigen oftmals alle Zutaten des jeweiligen Gerichtes ausgeschrieben. Ebenfalls liefert die dazugehörige Schnittstelle meistens nur Speisepläne für bis zu zwei Wochen im Voraus. Die Leiste im unteren Bereich der Ansicht würde sich also lediglich auf zwei Einträge beschränken und in dem Fall obsolet erscheinen. Auf die Zeichnung des Wireframes für die Ansicht nach Bestätigen des Zahnrad-Icons wurde hier verzichtet, da der innere Teil des zweiten Schrittes im Einrichtungsassistenten (siehe Kapitel 5.1) in einer separaten View angezeigt wird.

Neuigkeiten und Wetter weisen keine weiteren Besonderheiten auf. Um die entsprechenden Informationen der Hochschule anzuzeigen, wurde zumindest für die Neuigkeiten ein ähnliches Modell, wie in der ehemaligen App verwendet. Neben dem Titel wird eine Kurzbeschreibung angezeigt, die aus den ersten Sätzen des dazugehörigen Textes bestehen. Ein Icon signalisiert dem Nutzer, dass er eine Detailansicht öffnen kann, um den Text über dem gesamten Bildschirm erstreckt lesen zu können. Beim Akquirieren der Rohdaten für die Wetter-Schnittstelle wurden Informationen wie die Windrichtung und die Windgeschwindigkeit gesichtet, die vorher keine Verwendung gefunden haben. Da eine dedizierte Ansicht für die Wetter-Daten existiert, können die neuen Information unterhalb der Temperaturanzeige platziert werden. In einer einfachen Ansicht mit zentralen Textelementen erhält der Student Informationen über das Wetter an der Hochschule.

5.3 User-Interface

5.3.1 Farbauswahl

Für die Wahl der zu nutzenden Farben innerhalb der Cross-Plattform-App wurde auf zwei Dinge besonders Wert gelegt. Da viele Informationen auf einmal eingeblendet werden und der Nutzer eine schnelle Übersicht über sämtliche Veranstaltungen haben möchte, darf das Design nicht mit zu vielen Farben überladen werden. Dennoch soll die App weiterhin an die EAH Jena erinnern und sich nicht zu viele Freiheiten in ihrer Gestaltung nehmen, da eine staatliche Einrichtung repräsentiert wird. Über das gesamte Layout hinweg werden nur drei Farben verwendet, wobei eine davon nur ein einziges Mal vorkommt. Die genauen Farben und dazugehörige Hex-Codes stammen aus dem Quelltext der offiziellen Webseite der EAH Jena.

Das Grün, das in Abbildung 5 verwendet wird, ist dabei die Akzentfarbe und unterstreicht UI Elemente, beziehungsweise unterteilt bestimmte Bereiche. Die zweite Farbe findet lediglich im ausfahrbaren Seitenmenü Verwendung und wird als Hintergrundfarbe genutzt. Ansonsten wird die dritte Farbe, ein schlichtes Weiß im weiteren Verlauf als Hauptfarbe für den Hintergrund der Ansichten festgelegt. Dadurch entsteht ein minimalistischer Eindruck und der Fokus liegt auf den dargestellten Informationen.

Mit dem Release von Android-10[59] und iOS-13[60] haben die Hersteller Google und Apple einen „Dunkelmodus“ in ihr Betriebssystem eingepflegt. Dabei handelt es sich um eine Repräsentation des Layouts, bei dem der Schwerpunkt auf besonders dunklen Farbtönen liegt. Auf diese Weise wird die Belastung der Augen in einer dunklen Umgebung reduziert.[61] Da das Konzept der Cross-Plattform-App auf ein schlichtes Layout mit wenigen Farben setzt, besteht die Möglichkeit mit wenigen Schritten in einem späteren Iterationsschritt eine entsprechende dunkle Version nachzuliefern. Dabei soll der Endnutzer selbst entscheiden können, welche Darstellung bevorzugt wird.

5.3.2 Icons

Über den gesamten Verlauf des Wireframes werden verschieden Icons genutzt, die dem Nutzer eine bestimmte Interaktion signalisieren sollen. Für gewöhnlich werden jene von einem Designer bereitgestellt, der diese explizit für das zu entwickelnde Produkt entwickelt. In diesem Fall jedoch wird ein sogenanntes Icon-Font-Set, namens „Font Awesome Free“[62] genutzt. Icon-Font-Sets haben den Vorteil, dass diese nicht als Bilder in bekannten Dateiformaten, wie .jpg oder .png in das Projekt importiert werden, sondern direkt als Schriftart zur Verfügung stehen. Dadurch können diese unendlich größer skaliert und verwendet werden, ohne die Qualität des jeweiligen Icons im Blick behalten zu müssen. Font Awesome ist unter der SIL Open Font Licence 1.1 lizensiert, die den Nutzer dazu berechtigt, die entsprechende Software auch zu kommerziellem Zwecke weiter zu verteilen. Der Programmierer hat zwar den Nachteil, auf ein bestimmtes Set an Icons beschränkt zu sein, jedoch kann so mehr Zeit in die Implementierung der App investiert werden. Ebenfalls sind pro Icon mehrere Varianten vorhanden, wodurch eine Auswahl geboten ist.

[59] Google, Android-10 Features
[60] Apple, iOS-13 Features
[61] Material, Dark Theme
[62] Font Awesome, Free Package

6 Implementierung

In diesem Bereich geht es um den praktischen Teil der Arbeit. Um die Implementierung einer Hochschul-App mit dem Framework Nativescript umsetzen zu können, müssen zunächst eine ganze Reihe von Programmen installiert werden, die in ihrem Zusammenspiel das Bauen einer nativen App ermöglichen. Folgend ist sowohl die Implementierung des User-Interfaces als auch die Kommunikation zwischen App und Server Thema. Zuletzt werden Performanceprobleme aufgrund der Bauweise und Implementierungsdifferenzen näher beschrieben, die sich bei der Entwicklung für die jeweilige Zielplattform ergeben haben.

6.1 Projekt Setup

Der komplette Entwicklungsprozess fand auf einem „MacBook Pro (2017)“ statt. Bei dem genutzten Betriebssystem handelte es sich um „macOS Mojave 10.14.6“. Der Grund dafür bestand darin, dass Apple gewisse Restriktionen vorgibt, um native Apps für die iOS Plattform entwickeln zu können. Dazu gehört der Besitz eines offiziellen Apple Gerätes, da sonst die Entwicklungsumgebung xCode nicht installiert werden kann, die exklusiv für Mac OS entwickelt worden ist.[63] Die Umgebung selbst wird zum finalen Bauen auf der Zielplattform nach Schreiben des Codes benötigt. Da untersucht werden soll, ob die Entwicklung einer Cross-Plattform-App mit dem ausgewählten Framework möglich ist, war die Verwendung eines solchen Gerätes Voraussetzung. Als Hauptentwicklungsumgebung wurde das Programm „Visual Studio Code“ verwendet, bei dem es sich um einen von Microsoft bereitgestelltes, plattformübergreifenden Programm handelt.[64]
Zu den Hauptpaketen, die für die Implementierung einer Android App installiert werden müssen, gehören die folgenden Programme:

- „Google Chrome“ um eine Nativescript App debuggen zu können
- „Node.js LTS“, eine serverseitige Plattform für Netzwerkanwendungen
- „Open JDK 8“, die offizielle freie Implementierung der Java-Plattform
- „Android SDK“, ein Kit, das Entwicklern bei Java-basierten Anwendungen hilft
- „Androids- und Googles Support Repository” stellen Bibliotheken dar, die für den Entwicklungsprozess genutzt werden können

[63] Nativescript Documentation, Build iOS
[64] Microsoft, Visual Studio Code

- „Android SDK Build Tools", eine Komponente, die für das Bauen nach Entwicklung der Applikation zuständig ist
- „Android Studio", die offizielle Entwicklungsumgebung
- Die Einrichtung eines Android Emulators in Android Studio

Um die iOS Applikation realisieren zu können, müssen ebenfalls neben den genannten noch zusätzlichen Tools installiert werden. Dazu gehören unter anderem:

- „Cocoapods", ein Tool, um externe Bibliotheken in xCode-Projekte einbinden zu können
- „Xcode", die offizielle Entwicklungsumgebung
- „Command-line-tools for Xcode", eine Erweiterung, die die Steuerung der Entwicklungsumgebung über die Kommandozeile ermöglicht

Sowohl unter Windows[65] als auch MacOS[66] werden auf der jeweiligen Nativescript Dokumentationsseite verschiedene Möglichkeiten angeboten, um den Installationsprozess zu vereinfachen. Dabei werden Programme und Kommandozeilenbefehle empfohlen, die alle bisher genannten Pakete und mehr auf dem jeweiligen Zielbetriebssystem mit dem Einfügen von nur einer Zeile installieren. Wenn der Installationsprozess abgeschlossen worden ist, muss der Nutzer über die Kommandozeile den Befehl „npm install -g nativescript" ausführen. Dabei werden zum einen die Paketmanager-Funktionen von node.js in Form eines „npm install" genutzt, zum anderen wird direkt global Nativescript auf dem Rechner installiert. Das wiederrum hat zur Folge, dass nun der Befehl „tns" im Terminal als Umgebungsvariable nutzbar ist. Nachdem die Installation eben aufgelisteter Tools von System zu System unterschiedlich verlaufen kann, empfiehlt es sich zum Schluss den Befehl „tns doctor" auszuführen, der wiederum überprüft, ob alle Programme und Bibliotheken installiert sind. Ebenfalls wird sichergestellt, dass die lokal installierten Versionsnummern den jeweiligen Mindestanforderungen des Frameworks entsprechen.

Nachdem nun das Grundsetup abgeschlossen worden ist, kann mit Hilfe der Nativescript CLI (siehe Kapitel 3.4) begonnen werden, ein tatsächliches Projekt zu erstellen. Dabei muss

[65] NativeScript Setup, Windows
[66] NativeScript Setup, macOS

innerhalb der Kommandozeile zu dem entsprechenden Ordner navigiert werden, in dem das Projekt angelegt werden soll. Im Anschluss reicht der Befehl „tns create <name>“ aus, um die Ordnerstruktur für die Cross-Plattform-App zu initialisieren. Dieser Prozess kann einige Sekunden dauern, da alle benötigten Abhängigkeiten heruntergeladen werden. Die gerade eben erstellte Applikation ist nun auch sofort lauffähig und wird mit dem Befehl „npm start android“ oder „npm start ios“ ausgeführt. Das heißt in diesem Fall speziell, dass für die jeweilige Zielplattform entweder die Android SDK build-tools oder xCode im Hintergrund gestartet werden, damit die native App gebaut werden kann. Wenn das Setup erfolgreich abgeschlossen worden ist, erscheint nun entweder der Android Emulator oder der iOS Simulator mit der gerade erstellten Applikation. Falls ein physisches Gerät am Computer angeschlossen ist, reagiert die CLI entsprechend und installiert die Applikation nach dem Bauen direkt auf dem Endgerät.
Seit Nativescript 5.3 wird nach Initialisieren des Projektes ebenfalls das „Hot Module Replacement“[67] aktiviert. Dadurch lässt sich der Entwicklungsprozess massiv optimieren, denn die Funktion tauscht einzelne Codezeilen und ganze Dateien zur Laufzeit aus. Die Änderungen und Updates, die im Code vorgenommen wurden, erscheinen innerhalb von Sekunden nach dem Speichern auf dem jeweiligen Zielgerät und es muss kein neuer Build des Projektes angestoßen werden.

6.2 Implementierung eines Kommunikations-Service

Die Angular Dokumentation[68] empfiehlt aufwendige Logik, wie das Ansprechen einer Schnittstelle mit HTTP-Requests in eine einzelne Service-Dateien auszulagern. Die geschriebenen Komponenten, die für die Anzeige des User-Interface wichtig sind, sollen nichts über das Akquirieren, Speichern und Bearbeiten von Daten wissen. Auf diese Weise kann eine strikte Trennung der Datenlage erfolgen und Komponenten, die eher für visuelle Aufgaben gemacht sind, bleiben frei von Request Headern und ähnlichen Dingen.[69]

Der folgende Abschnitt befasst sich mit dem Aufbau eines Service und wie die dazugehörige Implementierung aussieht. Als Beispiel wird die Klasse genutzt, die für die Beschaffung der Wetter-Daten zuständig ist. Da keine weiteren Manipulationen am Inhalt der empfangenen Daten nötig sind, kann der Fokus auf den generellen Aufbau gelegt werden. Begonnen wird

[67] NativeScript Blog, Hot Module Replacement
[68] Angular Documentation, Services
[69] ebd.

jedoch mit dem Daten-Typ, der durch die Requests zurückgegeben wird, um die genutzten Methoden besser verstehen zu können.

6.2.1 Observable

Bei einem Observable handelt es sich um einen Daten-Typ, der von RxJS, einer Bibliothek für den Umgang von asynchronen oder ereignisbasierten Daten[70] bereitgestellt wird. Sämtliche Methoden, die im Anschluss für die verschiedenen GET-Requests genutzt werden, geben den Inhalt der Schnittstelle dabei innerhalb eines solchen Observable-Objektes zurück. Sobald eine Funktion geschrieben wurde, die verschiedene Werte in der genannten Form bereitstellt, wird diese nicht ausgeführt, bis ein Konsument sie abonniert hat. Der Konsument ist dabei in der Komponente zu finden, die Gebrauch von dem Service zur Bereitstellung der Daten macht. Abonnierte Observables stellen dabei einen ständigen Fluss aus Daten dar, der erst beendet wird, sobald eine bestimmte Bedingung des Senders erfüllt wurde oder der Abonnent diese Verbindung explizit schließt. Natürlich spielen auch äußere Einflüsse, wie eine stabile Internetverbindung eine Rolle beim Vorlegen der Daten. Mit diesem ständigen Fluss ist ein Echtzeit-Abbild der Schnittstelle möglich, wenn in der Anwendung nichts anderes definiert wurde. Aufgrund der vielfältigen Nutzbarkeit finden Observables in vielen Teilen des Angular Framework Verwendung.[71]

[70] RxJS, Documentation

[71] Angular Documentation, Observables

6.2.2 weather.service.ts

```
1   import { Injectable } from "@angular/core";
2   import { HttpClient, HttpHeaders } from "@angular/common/http";
3   import { Observable } from "rxjs";
4   import { weather } from "~/app/shared/config";
5   import { WeatherModel } from "../model/weather.model";
6
7   @Injectable()
8   export class WeatherService {
9       private weatherUrl: string = weather;
10
11      constructor(private http: HttpClient) { }
12
13      getWeatherData(): Observable<WeatherModel> {
14          let headers = this.createRequestHeader();
15          return this.http.get<WeatherModel>(this.weatherUrl, { headers: headers });
16      }
17
18      private createRequestHeader() {
19          let headers = new HttpHeaders({
20              "Content-Type": "application/json",
21          });
22
23          return headers;
24      }
25  }
```

Codebeispiel 2: Implementierung eines Service für die Datenbeschaffung

Die Klasse für die Kommunikation mit der Wetter-Schnittstelle der Hochschule ähnelt im Aufbau einer typischen Klasse, unterscheidet sich aber um das Stichwort „@Injectable()“ (siehe Zeile 7). Dadurch wird dem Framework verständlich gemacht, dass die gerade erstellte WeatherService-Klasse als Abhängigkeit verfügbar ist und damit später in UI-Komponenten „injiziert“ werden kann. Damit die Klasse im Stande ist, verschiedene Requests zu versenden, wird im Konstruktor ein entsprechendes HttpClient-Modul deklariert, das Zugriff auf die durch das Framework bereitgestellten Methoden erlaubt (siehe Zeile 11). Dadurch ist die Anweisung „this.http“ im gesamten Umfang der Klasse verfügbar. Das HttpClient-Modul selbst wird durch das Angular Framework bereitgestellt (siehe Zeile 2). Im nächsten Schritt wird die Zieladresse definiert, die benötigt wird, um eine Verbindung aufbauen zu können. Dazu importiert der Entwickler eine zuvor erstellte Datei[72] in das Projekt (siehe Zeile 4), die sämtliche Zieladressen der Schnittstellen beinhaltet. Im Anschluss wird diese als globale Variable innerhalb der Klasse bereitgestellt (siehe Zeile 9). Das Versenden von Anfragen mit dem HttpClient-Modul sieht laut Klassendefinition das Einbinden eines HttpHeaders-Objekts vor. Dadurch wird die Möglichkeit geboten, Schlüssel

[72] Siehe Anhang: Inhalt der CD-ROM, Programmcode, eahjena/src/app/shared/config.ts

in die Anfrage zu inkludieren, die eventuell von der Schnittstelle zur Authentifizierung benötigt werden. Allerdings hat eine Untersuchung (siehe Kapitel 4.3) gezeigt, dass sämtliche Daten unverschlüsselt auf dem Server liegen. Infolgedessen sind entsprechende Felder unnötig. Es wird bloß der Typ des Inhaltes der Schnittstelle (siehe Zeile 20) definiert, damit das Modul weiß, wie mit den empfangenen Daten umzugehen ist. Das eben deklarierte Objekt wird innerhalb einer Methode (siehe Zeile 18 bis 24) zurückgegeben, wodurch ein späterer Aufruf erleichtert wird. Nachdem das Grundgerüst für die Implementierung eines Http-Get-Requests abgeschlossen ist, beginnt der eigentliche Hauptteil des Services. Dazu wird eine neue Methode definiert, die zuständig für die Beschaffung und Übergabe der Daten ist (siehe Zeile 13 bis 16). Als erstes folgt der Aufruf der vorher definierten Methode für den Http-Header innerhalb einer lokalen Variable (siehe Zeile 14). Im Anschluss gibt die Methode des HttpClient-Moduls das Ergebnis des Requests direkt wieder an die Methode zurück (siehe Zeile 15). Der GET-Request wird mit „this.http.get()“ definiert und erwartet im Methodenaufruf sowohl den Header, als auch die Zieladresse, wie im Beispiel zu sehen ist. Die spitzen Klammern stellen hier eine Besonderheit dar. Dabei handelt es sich um eine Erweiterung, die durch TypeScript[73] zwar angeboten wird, aber in der Nativescript Dokumentation[74] keine Erwähnung findet. „Type Assertions“ sind ein Weg, dem Compiler zu sagen, wie der Inhalt der empfangenen Daten aussieht, bevor sie innerhalb eines Observables (siehe Kapitel 6.2.1) zurückgegeben werden. Es findet eine Typisierung der einzelnen Felder statt, eine Restrukturierung des Inhaltes entfällt dagegen. Um das realisieren zu können, wird eine zusätzliche Klasse erstellt. Diese Klasse findet sich ebenfalls im Rückgabe-Typ der Methode wieder (siehe Zeile 13) und definiert für den weiteren Verlauf den Inhalt der zurückgegebenen Daten.

[73] Typescript Documentation, Type Assertion
[74] Nativescript Documentation, HTTP Get

```
export class WeatherModel {
    temperature: string; // 16.7 °C
    windSpeed: string; // 1.2 m/s
    windDirection: string; // 167° Süd
    chill: string; // not available
    provider: string; // Klimastation Ernst-Abbe-Hochschule
    code: string; // FAIR_DAY
    backgroundId: number; // 4
    iconId: number; // 9

    constructor(options: any) {
        this.temperature = options.temperature;
        this.windSpeed = options.windSpeed;
        this.windDirection = options.windDirection;
        this.provider = options.provider;
    }
}
```

Codebeispiel 3: Modell der Wetter-Schnittstelle

Die Fertigstellung der Service-Klasse erfordert die abgebildete WeatherModel-Klasse. Diese enthält Kopien der Felder aus der Schnittstelle mit einer passenden Typisierung. Einzelne Auszüge der empfangenen Daten sind als Kommentar im Code enthalten, um bei der Programmierung einen Überblick des Inhaltes zu liefern. Dadurch entsteht auch der Vorteil, dass bei der Entwicklung der visuellen Komponenten die automatische Code-Vervollständigung der Entwicklungsumgebung zur Verfügung steht, da nun der Inhalt des Observables bekannt ist.

6.3 Implementierung der Logik einer App-Ansicht

Es folgt eine Vorstellung des Codes, der für die Logik der visuellen Wetter-Komponente verantwortlich ist. Diese erhält Zugriff auf die WeatherService-Klasse (siehe Kapitel 6.2.2) und stellt Methoden zum Umgang mit der Benutzeroberfläche bereit. Die Nativescript CLI (siehe Kapitel 3.4) hilft dem Entwickler, Code automatisch zu generieren, der wiederholt in jeder Unterseite der App auftritt. Der Befehl „tns generate component <component-name>“ kann im gewünschten Ordner in die Kommandozeile eingegeben werden, um dem Prozess automatisiert zu starten. Es entsteht eine Ordnerstruktur für eine Beispielkomponente mit dem übergebenen Namen. Innerhalb dessen ist die Datei „weather.component.ts“ zu finden. Diese wird im Anschluss vorgestellt.

```
1   import { Component, OnInit } from "@angular/core";
2   import { WeatherService } from "../shared/service/weather.service";
3   import { WeatherModel } from "../shared/model/weather.model";
4   import { RadSideDrawer } from "nativescript-ui-sidedrawer";
5   import * as app from "tns-core-modules/application";
6   import { Page } from "tns-core-modules/ui/page/page";
7
8   @Component({
9       selector: "Weather",
10      moduleId: module.id,
11      templateUrl: "./weather.component.html",
12      styleUrls: ['./weather.component.css'],
13      providers: [WeatherService]
14  })
15  export class WeatherComponent implements OnInit {
16      public requestFinished: boolean = false;
17      public weather: WeatherModel;
18
19      constructor(private page: Page,
20          private _weatherService: WeatherService) {
21      }
22
23      ngOnInit(): void {
24          this.page.actionBarHidden = true;
25          this.extractWeatherData();
26      }
27
28      openDrawer(): void {
29          const sideDrawer = <RadSideDrawer>app.getRootView();
30          sideDrawer.showDrawer();
31      }
32
33      extractWeatherData(): void {
34          this._weatherService.getWeatherData()
35              .subscribe((result: WeatherModel) => {
36                  this.weather = result;
37                  this.requestFinished = true;
38              }, (error) => console.log(error));
39      }
40  }
```

Codebeispiel 4: Implementierung der Logik für die Wetter-Ansicht

Abschnitte, die im Codebeispiel 4 mit einem dunklen Balken markiert worden sind, zeigen den im Rahmen der Entwicklung ergänzten Code. Stellen, die nicht markiert wurden, kommen direkt durch die Codegenerierung zustande. Innerhalb des Beispiels können vor Beginn der Klassen-Definition externe Dateizugriffe festgelegt werden (siehe Zeile 8 bis 14). Der Verweis auf eine .css-Datei für das Layout ist dabei nicht zwingend (siehe Zeile 12), hilft aber den Code im späteren Verlauf sauber zu halten. Im Anschluss wird die WeatherService-Klasse aus dem vorherigen Kapitel als Abhängigkeit der visuellen Komponente definiert (siehe Zeile 13). Damit das Framework weiß, wo die Abhängigkeit zu finden ist, muss sie zunächst in die Komponente importiert werden (siehe Zeile 2). Nach Festlegung der externen Zugriffe, folgt die Beschreibung einiger globaler Variablen. Zu Beginn steht ein Boolean namens „requestFinished“ (siehe Zeile 16), das den Wert „false“

zurückgibt. Dieses Objekt soll als Schalter für den Abschluss des Requests herhalten, um gegebenenfalls später eine Lade-Ansicht steuern zu können. Als nächstes wird ein noch leeres Objekt definiert, das dem Typ „WeatherModel“ (siehe Codebeispiel 3) entspricht (siehe Zeile 17). Inhalt des Objektes sollen nach erfolgreichem Request die Wetter-Daten sein. Die Bereitstellung des Typs erfolgt ebenfalls mit einem Import-Befehl (siehe Zeile 3). Im Konstruktor steht die WeatherService-Klasse (siehe Zeile 20), die zuvor als Abhängigkeit definiert wurde. Da der Service bereits importiert worden ist, erhält der Entwickler Zugriff auf die deklarierten Methoden mit „this._weatherService“.

Zu diesem Zweck folgt die Implementierung der Methode, die letztlich zuständig für das Abonnieren des Observables ist (siehe Kapitel 6.2.1). Nach der Benennung des gesamten Aufrufes erhält diese den Rückgabe-Typ „void“ (siehe Zeile 33). Das signalisiert wiederum demjenigen, der den Code aufruft, dass nach Ausführung der Funktion kein Objekt zurückerwartet wird. Für den Entwickler besteht nun die Möglichkeit, die vordefinierte Methode aus dem Service um die Methode „.subscribe((result: WeatherModel) => {})“ zu erweitern. Dieser Teil der Programmierung ist dafür zuständig, dem Framework zu sagen, dass das Abonnement des Observables erfolgen soll. Dabei wird der Inhalt dessen innerhalb einer Variable zurückgegeben. Erst das „subscriben“ öffnet und ermöglicht den Datenfluss. Aufgrund der Bauweise dieser Methode erlaubt der Code den Umgang mit den empfangenen Daten innerhalb der geschweiften Klammern, obwohl es sich um asynchrone Daten handelt. Dabei können weitere Zuweisungen oder Manipulationen, die für das User-Interface relevant sind, Anwendung finden. Genauso wird im Beispiel das Ergebnis der asynchronen Daten dem leeren WeatherModel-Objekt zugewiesen (siehe Zeile 36), das vorher deklariert worden ist (siehe Zeile 17). Im Anschluss wird das ebenfalls vorher deklarierte Boolean-Objekt „requestFinished“ auf „true“ gesetzt (siehe Zeile 37), um einer Ladefortschritt-Komponente die Befüllung des WeatherModel-Objektes zu signalisieren. Die Subscribe-Methode bietet neben dem „result“-Argument auch eine Ausgabe für den Fall eines Fehlschlags bei der Datenbeschaffung. In der aktuellen Implementierung (siehe Zeile 38) wird der Inhalt der Fehlernachricht in der Konsole ausgegeben. Um die Logik der visuellen Wetter-Komponente vervollständigen zu können, fehlt noch die Umsetzung eines letzten Schrittes, der direkt im Anschluss folgt.

Der Lebenszyklus einer einzelnen Komponente, die eine eigene Ansicht in der App darstellt, wird dabei durch das Framework gesteuert. Dieser Lebenszyklus kontrolliert das Erstellen, Erneuern und Zerstören einer Komponente. Dabei besteht die Möglichkeit in verschiedenen Phasen des Zyklus einzugreifen. Im Framework wird das durch sogenannte „Hooks“ (engl.

Haken) ermöglicht. Nachdem eine neue Ansicht über die Kommandozeile erstellt worden ist, enthält diese standardmäßig eine Implementierung der „ngOnInit“-Methode (siehe Zeile 23). Dabei handelt es sich um die erste Hook, die beim Initialisieren einer neuen Ansicht ausgeführt wird. Damit die Logik der Datenerfassung funktionieren kann, muss innerhalb dieser Phase die Ausführung des Service-Code erfolgen. Zu diesem Zweck wird lediglich die passende Methode durch „this.extractWeatherData()“ aufgerufen (siehe Zeile 25).

6.4 Implementierung des User-Interface

Um die in Abbildung 4 vorgegebene Darstellung anhand des Beispiels der Wetter-Daten realisieren zu können, bedarf es noch einer entsprechenden Festlegung innerhalb der „weather.component.html“-Datei. Die genannte Datei ist nach Erstellung der Komponente durch die Kommandozeile innerhalb des Ordners zu finden, in der schon die Implementierung der „weather.component.ts“ stattgefunden hat. Allerdings ist diese nicht wie im Beispiel vorher befüllt, sondern vollständig leer. Ein Blick in die Dokumentation zeigt, dass verschiedene UI-Elemente geboten sind, die bei der Gestaltung der Ansicht zur Verfügung stehen.[75] Diese müssen im Endeffekt innerhalb der Datei mit den entsprechenden Bindungen an die Daten der Logik versehen werden. Da es sich bei dem gewählten Beispiel um eine unkomplizierte Darstellung der Daten handelt, hält sich die Diversität der genutzten Elemente in Grenzen. Das Wireframe aus Abbildung 4 zeigt dem Entwickler, dass etwas zur Darstellung von Textfeldern benötigt wird. Diese Textfelder müssen im Anschluss innerhalb eines Layout-Containers untereinander dargestellt werden können.

```
1  <StackLayout>
2      <StackLayout *ngIf="!requestFinished">
3          <ActivityIndicator busy="true" width="100" height="100"></ActivityIndicator>
4      </StackLayout>
5      <StackLayout *ngIf="requestFinished">
6          <Label text="Temperatur:" class="footnote"></Label>
7          <Label [text]="weather.temperature"></Label>
8          <Label text="Windgeschwindigkeit:" class="footnote"></Label>
9          <Label [text]="weather.windSpeed"></Label>
10         <Label text="Windrichtung:" class="footnote"></Label>
11         <Label [text]="weather.windDirection"></Label>
12         <Label [text]="'(Quelle: ' + weather.provider + ')'" class="footnote"></Label>
13     </StackLayout>
14 </StackLayout>
```

Codebeispiel 5: Implementierung des User-Interface

[75] Nativescript Documentation, User-Interface

Das abgebildete Code-Beispiel zeigt, wie der innere Container der Wetter-Ansicht innerhalb des Wireframes (siehe Abbildung 4) implementiert wurde. Um eine Unterseite realisieren zu können, ist Voraussetzung des Frameworks, alles innerhalb eines einzelnen Layout-Containers unterzubringen ist (siehe Zeile 1 und 14). Für diese und alle anderen Unterteilungen wurde das sogenannte „StackLayout“ ausgewählt. Das „StackLayout“ zeigt alle Kind-Elemente, die sich innerhalb dessen befinden, nacheinander an. Dabei geschieht dies in horizontale Richtung, sofern nichts anderes definiert wurde. Alternativ steht auch eine vertikale Anordnung zur Auswahl. Die Größe und Abstände werden durch den Inhalt selbst bestimmt. Nachdem nun der Hauptcontainer der Ansicht ausgewählt worden ist, folgen zwei weitere StackLayout-Container. Dabei ist der Erste für die Anzeige einer Ladeansicht zuständig (siehe Zeile 2 bis 4) und der Zweite (siehe Zeile 5 bis 13) für die tatsächliche Anzeige der Daten aus der Wetter-Schnittstelle. Beide Container sind mit einer Erweiterung der Angular-Template Syntax (siehe Kapitel 3.3) versehen (siehe Zeile 2 und 5). Das „*ngIf“ entscheidet dabei, gebunden an eine Variable aus der Logik (siehe Kapitel 6.3), welcher der beiden aktiv gezeichnet und angezeigt werden soll. Innerhalb des ersten Teils befindet sich eine Ladeansicht-Komponente, die durch das Framework[76] bereitgestellt wird. Dabei handelt es sich um eine native Komponente, die unter Android auf die Klasse „ProgressBar“[77] und unter iOS auf die Klasse „UIActivityIndicatorView“[78] zurückzuführen ist. Es besteht die Möglichkeit, die bereitgestellte Komponente durch verschiedene Attribute zu steuern. Da aber der gesamte Container ausgeblendet wird, bedarf es keiner weiterer Änderungen. Das Beispiel aus der Dokumentation kann also vollständig übernommen und eingepflegt werden (siehe Zeile 3). Nach Fertigstellung der Ladeanzeige folgt die tatsächliche Wetter-Anzeige. Zu diesem Zweck stellt das Framework das „Label“-Element zur Verfügung. Im Prinzip beschränkt sich die gesamte Funktionalität auf das Anzeigen von Objekten des Types „String“. Dabei können einzelne Textbausteine direkt in den Code eingepflegt werden (siehe Zeile 6). Ansonsten besteht die Möglichkeit dem Framework zu signalisieren, dass im Anschluss kein Text folgt, sondern der Verweis auf eine Variable. Um das zu ermöglichen, wird das Textfeld des Labels mit eckigen Klammern umschlossen (siehe Zeile 7). Nachdem dies geschehen ist, erhält das Label die Fähigkeit auf die Variablen der Logik zu zugreifen. Es können nun einzelne Felder der Wetter-Daten direkt angesprochen werden, wie es im abgebildeten Beispiel mit der Temperatur (siehe Zeile 7),

[76] Nativescript Documentation, Activity Indicator
[77] Android Documentation, ProgressBar
[78] iOS Documentation, UIActivityIndicatorView

Windgeschwindigkeit (siehe Zeile 9) und Windrichtung (siehe Zeile 11) gemacht wird. Damit die Daten visuell besser auseinanderzuhalten sind, erhalten die Titel noch den Verweis auf die „footnote“-Styleklasse (siehe Zeile 6). Dabei handelt es sich um eine von vielen CSS-Eigenschaften, die für Design-Anpassungen im Layout durch das Framework zur Verfügung stehen.[79] Im aktuellen Beispiel würde der Titel in einer kleinen, grauen Schrift erscheinen. Es stehen aber weitere Möglichkeiten, wie verschiedene Abstandshalter zur Verfügung. Alternativ können diese manuell in einer externen CSS-Datei definiert und eingepflegt werden.

6.5 Implementierungsdifferenzen zwischen Android und iOS

Aufgrund der Tatsache, dass jedem Element des User-Interface eine separate, native Klasse zu Grunde liegt, kann unterschiedliches Verhalten auf der jeweiligen Plattform auftreten. Deshalb sind einige Dinge in der Entwicklungsphase zu beachten, die vor allem durch exploratives Testen aufgefallen sind.

Ein einfaches Beispiel stellt dabei das Label-Element dar, das schon in Kapitel 6.4 für die Darstellung der Daten genutzt wurde. Beide Plattformen haben unterschiedliche Richtlinien und Standardverhalten. Android nutzt für die Darstellung der Text-Komponente die Klasse „android.widget.TextView“ und iOS die Klasse „UILabel“. Während die Klasse sich unter Android innerhalb eines anderen Layout-Containers befindet, muss explizit deklariert werden, wenn Inhalte zentriert dargestellt werden sollen. Ansonsten beginnt das Betriebssystem immer in der oberen, linken Ecke des Layout-Containers. Unter iOS ist allerdings das gegenteilige Verhalten der Fall. Damit muss innerhalb der Implementierung eine Festlegung für beide Elemente stattfinden, da es sonst zu unterschiedlichen Ansichten auf den Endgeräten kommt. Um das zu ermöglichen, stehen verschiedene Optionen zur Auswahl. Das Framework erstellt zu Beginn eine globale CSS-Datei zur Verfügung, die mit verschiedenen Anforderungen an das Design versehen werden kann. Eine Möglichkeit ist die Definition einer spezifischen Regel, die alle Label-Elemente innerhalb der App zentriert. Optional stehen auch verschiedene Layout-Container in der Dokumentation zu Verfügung, die jeweils unterschiedliches Verhalten für ihre Inhalte bieten.

[79] Nativescript Documentation, CSS

6.6 Performanceprobleme

Während der Entwicklungsphase zeigten sich bei der Nutzung der App im Android-Emulator bereits einige Schwierigkeiten. Es kam an mehreren Stellen zu Performanceeinbußen, die sich durch ein Einfrieren der Bedienelemente bemerkbar machten. Der iOS Simulator zeigte im gesamten Verlauf keinerlei vergleichbare Probleme. Dabei sind vor allem die Übergänge der Navigation auf eine Unterseite negativ aufgefallen. Eine weitere Baustelle war die Ansicht, in der Studenten ihre Stundenpläne anzeigen lassen und verwalten können. Abbildung 4 zeigt im Wireframe für die Stundenplan-Ansicht eine Leiste im unteren Bereich, die der Student nutzen kann, um die aktuelle Kalenderwoche zu verändern. Sobald die genannte Leiste betätigt wird, antwortet die App mit langsamen Reaktionen auf die Benutzerinteraktion. Dennoch muss beachtet werden, dass der Android-Emulator nur bedingt dazu im Stande ist, Aussagen über die Performance zu treffen. Selbst vonseiten Googles wird die dringende Empfehlung ausgesprochen, sämtliche Apps auf einem physischen Endgerät zu testen.[80]

Um verlässliche Aussagen über die Performance treffen zu können, wird neben den Smartphones zusätzlich ein Tool benötigt, das entsprechende Messungen liefern kann. Eine Google-Recherche hat gezeigt, dass Nativescript eine sogenannte Performance-Monitor-Erweiterung[81] anbietet, die für die Evaluierung der App zur Verfügung steht. Hierfür reicht das Implementieren einer Start-Methode auf der jeweiligen Ansicht. Innerhalb der Konsole erscheinen im Anschluss sämtliche Änderungen der Frames-per-Second. Dabei handelt es sich um einen Wert, der den angezeigten Bildern pro Sekunde entspricht. Dieser stagniert laut der Beschreibung des Plug-Ins[82] ungefähr bei 60 Bildern pro Sekunde für Android und iOS. Wenn dieser Wert erreicht wird, erscheint die Interaktion mit dem User-Interface dem Endnutzer gegenüber als flüssig.

Für die weitere Evaluierung der Problematik wurde eine explorative Testreihe durchgeführt, die verschiedene Android-Smartphones mit Hilfe des beschriebenen Monitors überprüfte. Ziel war es, herauszufinden, ob die genannten Probleme aufgrund der Nutzung eines Emulators zu Stande kommen. Für den Testablauf standen dem Studenten zu dem Zeitpunkt der Anfertigung die folgenden Geräte zur Verfügung:

---

[80] Android Documentation, Run Device

[81] Nativescript Plug-In, Performance Monitor

[82] ebd.

- Samsung Galaxy S7, Android 7.0, Marktstart 2016
- Nokia 5.1, Android 8.1, Marktstart 2018

Der Ablauf des Tests definierte sich durch das händische Ansteuern der kritischen Bereiche auf den genannten Endgeräten. Die Konsolenausgaben der Erweiterung wurden im Anschluss in ein separates Dokument[83] eingepflegt, um die verschiedenen Einzelergebnisse miteinander vergleichen zu können. Wenn die Werte nun gegenübergestellt betrachtet werden, zeigt sich, dass die Performance teilweise auf unter zehn Bilder die Sekunde fiel. Eine Ansicht mit sehr vielen gespeicherten Fächern oder Speisen begünstigt das Auftreten. Damit überprüft werden kann, ob das Verhalten wiederholt auftritt, geschah der Testablauf für die Endgeräte in zweifacher Ausführung. Das Samsung Galaxy S7 zeigte zwar Probleme im ersten Durchlauf, konnte sich aber im Zweiten besser schlagen. Ganz anders sah es bei dem Nokia-Gerät aus. Hier sind die Performance-Abfälle wesentlich massiver aufgetreten, als bei dem zuvor genutzten Handy. Es darf allerdings nicht außer Acht gelassen werden, dass die Auswahl der Geräte in dem Fall nicht repräsentativ ist, da eines der verfügbaren Smartphones veraltet ist und das andere der Mittelklasse angehört. Es besteht durchaus die Möglichkeit einer Verbesserung der Performance bei der Nutzung eines aktuellen Oberklasse-Handys.

Um den genannten Problemen entgegenzuwirken, findet sich auf der Webseite des Frameworks eine Unterseite mit Ansätzen und Lösungen zur Steigerung der Performance.[84] Einige davon wurden im Verlauf der Entwicklung im Code eingepflegt. Inwieweit die Lösungen nützlich waren oder nur einen Zeitaufwand für den Entwickler generierten, wird im folgenden Abschnitt besprochen.

6.6.1 Anhalten des Seitenaufbaus

Eine der gebotenen Lösungen, bezieht sich explizit auf die Performanceprobleme, die nach dem Betreten einer Unterseite auftreten. Das Problem entsteht laut dem Beitrag[85] dadurch, dass die Animation zum Anzeigen der Unteransicht und das Aufbauen der Ansicht auf der jeweiligen Unterseite gleichzeitig zu viel Last auf der CPU des Smartphones aufbauen. Bei dem gebotenen Vorschlag handelt es sich dabei nicht im eigentlichen Sinne um eine Lösung

[83] Siehe Anhang: Testreihe: Frames-Per-Second-Messung der Navigation in Stundenplan-App
[84] Nativescript Blog, Performance
[85] Nativescript Blog, Performance

des Problems, anstatt dessen wird eine Illusion für den Endnutzer erzeugt. Der Vorschlag besagt, den aktuellen Seitenaufbau um ungefähr 300 bis 500 Millisekunden zu verzögern, damit die Animation für die Navigation auf die Unterseite ihren Prozess abschließen kann. Erst im Anschluss generiert das Framework die tatsächliche Ansicht. Dem Nutzer wird in der kurzen Zeit eine Ladekomponente angezeigt. Auf diese Weise kann die Navigation flüssig ausgeführt werden.

6.6.2 Optimierung des Layouts

Eine andere Lösung bezieht sich explizit auf Performance-Probleme in den jeweiligen Unterseiten. Dabei gibt es die Möglichkeit die Komplexität einer Ansicht in der jeweiligen Implementierung des User-Interface zu reduzieren (siehe Kapitel 6.4). Wie dort bereits gezeigt wurde setzt sich die Ansicht aus verschiedenen „Layout-Containern“ zusammen. Die Seite erklärt dabei, dass jeder weitere Container innerhalb eines anderen eine zusätzliche Last für das Endgerät darstellt. Es werden verschiedene Ideen aufgeführt, wie eine Ansicht mit so wenig Elementen wie möglich realisiert werden kann. Dabei ist von einer Verringerung der Komplexität die Rede. Diese Lösung wurde in einem Versuch implementiert und auf ihre Wirksamkeit untersucht. Es stellte sich allerdings heraus, dass durch das Verringern der genutzten Layout-Elemente die Lesbarkeit des Codes massiv gelitten hat. Der Test mit dem Performance-Monitor (siehe Kapitel 6.6) zeigte, dass der Aufbau der Seite von fünf auf acht Bilder pro Sekunde optimiert werden konnte. Da diese Wertsteigerung für den Endnutzer keinen Unterschied aufweist, wurde auf die Implementierung dieser Lösung verzichtet.

6.6.3 Listenansicht

Die aktuelle Implementierung der einzelnen Ansichten für die Cross-Plattform-App ähnelt in weiten Teilen der Umsetzung des vorgestellten Beispiels (siehe Kapitel 6.4). Dabei setzen sich die UI-Elemente aus verschiedenen Layout-Containern zusammen und zeigen die platzierten Daten. Im Prinzip präsentiert der Code alle Informationen nach Betreten der Unterseite auf einmal. Dabei ist es gleichgültig, ob der Nutzer die aktuellen Daten sieht oder diese sich außerhalb des Bildschirms befinden. Sowohl Android als auch iOS bieten ein „Listview“-Element an. Wie der Name schon sagt, kann diese native Komponente für die Darstellung von Listen genutzt werden. Der gravierende Vorteil ergibt sich aus der „Load-

on-Demand“-Funktionalität.[86] Dabei werden Inhalte, die außerhalb des Sichtbereiches liegen, erst gezeichnet, sobald der Nutzer sich dem entsprechenden Datensatz nähert. Dementsprechend verringert sich auch die Last der Ansicht auf dem Endgerät.

Rückblickend betrachtet, wäre die Listview-Komponente ein nützliches Element gewesen, um sich innerhalb der Cross-Plattform-App einem nativen und flüssigen Erlebnis anzunähern. Um diesen Teil des Frameworks nutzen zu können, hätte jedoch schon in der Konzeptionsphase, während der Erstellung des Wireframes besonderer Wert auf die Gestaltung der Oberfläche gelegt werden müssen. Denn erst während des Versuchs einer Implementierung zeigte sich eine Schwäche, die mit keinem Wort in der Dokumentation des Listview-Elementes erwähnt wurde. Damit das Framework im Stande ist, bestimmte Inhalte dynamisch zu laden, müssen alle Elemente der Listenansicht die gleiche, bzw. vorher festgelegte Höhen-Werte haben. Das Wireframe sieht allerdings vor, sämtliche Modulbeschreibungen, Dozenten und Gerichte auf einmal anzuzeigen. Demnach ist die Höhe der jeweiligen Ansicht nicht vorherzusagen und wird vollständig durch den eigenen Inhalt dynamisch gesteuert.

[86] Nativescript Documentation, Load on Demand

7 Abgleich

Das folgende Kapitel beschäftigt sich mit dem Abgleich der Anforderungen, die in Kapitel 4.5 an die App gestellt worden sind.

7.1 Abgleich mit funktionalen Anforderungen

In Kapitel 4.5.1 wurden verschiedene User Storys formuliert, die während der Entwicklung die verschiedenen Anforderungen an die Unterseiten der Cross-Plattform-App definiert haben. Diese konnten fast vollständig umgesetzt werden. Dabei wurde zu großen Teilen die Methodik eingesetzt, die schon im Kapitel 6 der Implementierung näher erklärt worden sind. Einzig die User Story, die sie sich mit der Einstellungs-Ansicht beschäftigte, ist gescheitert. Dabei sollte der Student die Möglichkeit bekommen, über verschiedene Änderungen in seinem Stundenplan benachrichtigt zu werden. Eine Logik zu implementieren, die prüft, ob sich die Termine bestimmter Fächer geändert haben oder neue hinzugekommen sind, hätte den zeitlichen Rahmen der Arbeit gesprengt.
Weiterführend konnte evaluiert werden, dass bereits eine Erweiterung für das Framework existiert, die Push-Nachrichten über die App an das Smartphone senden kann.[87] Somit besteht zumindest die Grundlage, um die Funktion in einem späteren Entwicklungsschritt nachliefern zu können.

7.2 Abgleich mit nichtfunktionalen Anforderungen

Im Anschluss werden die in Kapitel 4.5.2 beschriebenen nicht-funktionalen Anforderungen in einer tabellarischen Ansicht aufgeführt und anschließend kommentiert. Dabei wird entweder die Erfüllung bestätigt oder der Fehlschlag intensiver evaluiert.

Anforderung	**Kommentar**
Funktionalität	Konnte nur zum Teil erfüllt werden. Im Laufe der Entwicklung ist offensichtlich geworden, dass die durch die Schnittstelle zurückgegebenen Stundenpläne im Laufe des Semesters um Veranstaltungen erweitert werden können, die vorher nicht enthalten waren. Dazu zählen unter anderem verschiedene Prüfungsleistungen,

[87] Nativescript Plug-In, Firebase

	deren Termin am Anfang des Semesters noch nicht feststand. Dieser spezifische Fall wurde während der Entwicklung nicht beachtet. Der Grundstundenplan, der durch den Einrichtungsassistenten (siehe Kapitel 5.1) festgelegt wird, enthält dabei automatisiert sämtliche neuen Termine. Davon sind die einzelnen Module, die durch den Nutzer in den eigenen Stundenplan eingepflegt werden, nicht betroffen. Die Veranstaltungen können zwar durch eine Interaktion in den eigenen Plan hinzugefügt werden, der Nutzer erhält jedoch keinen Hinweis dazu. Damit ist die Anforderung an die Genauigkeit der Daten im Verlauf des Semesters nicht vollständig erfüllt.
Zuverlässigkeit	Konnte nur zum Teil erfüllt werden. Da die Stundenplan-Daten aufgrund der möglichen Änderungen des Nutzers lokal gespeichert werden, sind diese auch im Falle eines Internetabbruchs verfügbar. Die Ansicht für Speisepläne bleibt allerdings leer, da keine Bearbeitungen der Daten stattfinden. Hier sollte noch nachgeliefert werden, um die Anforderung vollständig erfüllen zu können.
Benutzbarkeit	Konnte erfüllt werden.
Effizienz	Konnte nur zum Teil erfüllt werden. Kapitel 6.6 geht dabei im Detail auf verschiedene Performance-Probleme ein. Da diese allerdings nur unter Android auftreten, ist die Anforderungen zumindest für iOS erfüllt worden.
Wartbarkeit	Konnte erfüllt werden.
Portabilität	Konnte erfüllt werden.

8 Fazit

In den letzten Kapiteln standen die Implementierung und Einhaltung der vorgegebenen Anforderungen an die Cross-Plattform-Anwendung im Fokus. Der folgende Abschnitt fasst die Ergebnisse noch einmal zusammen und skizziert mögliche Erweiterungen, die in einer weiteren Entwicklungsphase oder separaten Arbeit umgesetzt werden könnten.

8.1 Zusammenfassung

Die Zielstellung der Bachelorarbeit war es, ein ausgewähltes Framework auf seine Fähigkeit zu untersuchen, eine plattformunabhängige Anwendung zu entwickeln. Der umgesetzte Prototyp sollte eine Einschätzung darüber geben, ob mit der eingesetzten Technologie native Apps programmiert werden können, die auch der Performance einer solchen gleich kommen.

Das Nativescript-Framework bot dabei die verschiedenen Grundvorrausetzungen, um eine Stundenplan-App für die Hochschule ermöglichen zu können. Durch den Zugriff auf die nativen Komponenten des jeweiligen Betriebssystems (Android und iOS) entsteht bei den Nutzern ein vertrautes Gefühl, da das plattformspezifische Erscheinungsbild entsteht. Ebenfalls kann auf jeweilige Gerätefunktionen zugegriffen werden. Um die Umsetzung mit dem genannten Framework beginnen zu können, musste zu Beginn eine Analyse durchgeführt werden. Ein wichtiger Schritt war es, zu prüfen, ob überhaupt Schnittstellen existieren, die entsprechende Daten liefern. Ebenfalls darf die Analyse der aktuellen Situation an der Hochschule nicht außer Acht gelassen werden. Weitere Untersuchungen befassten sich mit den Funktionen der App, die durch die ehemals vertriebene Applikation der Hochschule angeboten wurden. Erst durch das Ergebnis beider Analysen konnten Probleme festgestellt und bestehende Funktionen entsprechend in ihrer Grundidee erweitert werden. Das Ergebnis wurde in Form einer Liste von Anforderungen festgehalten und zusammengefasst. Der nächste Schritt im Entwicklungsprozess ging auf die Konzeptionierung ein. Im Fokus stand die Einteilung der Layout-Elemente und die Gestaltung der Nutzerführung. Die Implementierungsphase machte es möglich, sämtliche Vorbereitungen mit den vorgestellten Methoden umsetzen zu können. Wie im Abgleich festgestellt wurde, konnten die User Storys fast vollständig mit nur einem Code umgesetzt werden. Allein die nichtfunktionalen Anforderungen ließen im Gesamten zu wünschen übrig. Trotz der Verwirklichung einer nativen Android und iOS App macht der Verlauf der Arbeit deutlich, dass weiterhin Optimierungen an der Entwicklung benötigt werden. Obwohl ein großer Teil der Anforderungen umgesetzt werden konnte, bleibt die Anwendung an

verschiedenen Stellen erweiterungsfähig. Einerseits lag der Grund dafür bei fehlenden Erfahrungen in der Handhabung des Frameworks, anderseits hätten verschiedene Schwächen bereits in der Konzeption beachtet werden müssen, die eine gewisse Erfahrung im Umgang voraussetzen. Dennoch lässt sich sagen, dass die Nutzung einer Option, wie Nativescript für viele Kunden und Programmierer attraktiv bleibt, da plattformunabhängig eine voll funktionsfähige App für beide Plattformen umsetzbar ist. Im Anschluss folgt ein Ausblick auf Dinge, die weiterhin ausgebaut werden können.

8.2 Ausblick

8.2.1 Verhalten bei Internetabbruch

Die aktuelle Implementierung der Cross-Plattform-App zeigt im Falle eines Internetabbruchs die Daten des Stundenplanes an. Diese Umsetzung der Datenspeicherung findet allerdings nicht für die Speisepläne und Nachrichten der Hochschule statt. Dort werden die Informationen direkt nach der Kommunikation mit der entsprechenden Schnittstelle angezeigt. Das führt zu einem inkonsistenten Verhalten für den Endnutzer und resultiert in leeren Ansichten auf den jeweiligen Unterseiten. Um das beschriebene Problem zu lösen, stehen verschiedene Optionen zur Auswahl. Es besteht die Möglichkeit, Warnmeldungen zu implementieren, die zumindest erklären, weshalb das Fehlverhalten auftritt. Während der Implementierung des manipulierbaren Stundenplans ist jedoch klar geworden, dass entsprechende Datensätze mit relativ wenig Aufwand gespeichert werden können. Nativescript bietet verschiedene Optionen und Erweiterungen an, die Genanntes bewerkstelligen können. Die Überlegung ist, einen Service an zentraler Stelle zu schreiben (siehe Kapitel 6.2), der vor den Requests an die Schnittstellen geschaltet wird. Dieser müsste im Anschluss Daten speichern, zurückgeben und auf ihre Aktualität überprüfen können.

8.2.2 Vollständigkeit der Daten

Beim Umgang mit den Schnittstellen der Hochschule ist klargeworden, dass weiterhin Sonderfälle auftreten, die bei der Implementierung nicht bedacht worden sind. Dabei ist die Rede von Prüfungsleistungen, die möglicherweise im Stundenplan unterschlagen werden (siehe Kapitel 7.2). An dieser Stelle muss die Applikation nachziehen und den Studenten zumindest mit einer Warnmeldung versorgen. Eine einfache Lösung an dieser Stelle wäre ein Hinweis, der dem Nutzer sagt, dass neue Modulfächer mit dem Namen PL/APL

(„Prüfungsleistung“ und „Alternative Prüfungsleistung“) vorhanden sind und in den aktuellen Plan eingepflegt werden können.

8.2.3 Performance

Die Performance innerhalb der Cross-Plattform-App lässt auf dem Betriebssystem Android mit der aktuellen Implementierung zu wünschen übrig. Dabei ist das Problem auf die Bauweise des User-Interface zurückzuführen (siehe Kapitel 6.6.2). Aufgrund der hohen Anzahl von Daten, die dem Nutzer auf einmal präsentiert werden, ist die Bauweise auf die vorgestellte Art (siehe Kapitel 6.4) eher hinderlich als hilfreich. Um dem Problem entgegen zu wirken, sollten native Listen-Komponenten genutzt werden, da auch eine native Performance von der Applikation erwartet wird. Es muss allerdings schon während der Konzeptionierung darauf geachtet werden, die Ansicht entsprechend den Einschränkungen der Komponente aufseiten von Nativescript zu gestalten (siehe Kapitel 6.6.3).

8.2.4 Verbesserung der Schnittstellen

Die entwickelte App ist aktuell zwar im Stande zur Anzeige der Stunden- und Speiseplan-Daten, darüber hinaus gibt es weitere Informationen, die für einen Studenten wünschenswert wären. Dazu gehört beispielsweise die Anzeige der Semester-Eckdaten, sowie gesonderte Informationen zu Dozenten, wie E-Mail-Adressen und Sprechzeiten. Da beide Endpunkte jedoch entweder keine oder falsche Daten anzeigen, konnten diese Funktionen nicht in die Anwendung eingepflegt werden. Deshalb besteht selbst nach Nutzung der App für viele Studierende weiterhin der Bedarf, Informationen online von offizieller Stelle einzuholen. Damit diese grundlegenden Informationen geliefert werden können, wäre eine Aufwertung der Schnittstellen in einem späteren Entwicklungsschritt auch für andere Projekte von Vorteil.

Literaturverzeichnis

Buchquellen

[Ater, 2017]

T. Ater, *Building Progressive Web Apps*. O'Reilly Media Inc, 2017.

[Bonnie, 2015]

E. Bonnie, *Learning React Native: Building Native Mobile Apps with JavaScript*. O'Reilly Media, Inc., 2015.

[Balzert, 2009]

Prof. Dr.-Ing. habil. Helmut Balzert, *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. Spekrum Akademischer Verlag, 2009.

[Cohn, 2009]

Mike Cohn, *Succeeding with Agile: Software Development using Scrum*. Addison Wesley, 2009.

[Rau, 2016]

Karl-Heinz Rau, „Überblick zu Vorgehens- und Prozessmodellen", in *Agile objektorientierte Software-Entwicklung: Schritt für Schritt vom Geschäftsprozess zum Java-Programm*, Springer Vieweg, 2016.

Internetquellen

[Altexsoft, Performance Comparison]
Altexsoft, „Performane Comparison Xamarin iOS vs Android vs Native Apps“, 22-Aug-2017. Verfügbar unter: https://www.altexsoft.com/blog/engineering/performance-comparison-xamarin-forms-xamarin-ios-xamarin-android-vs-android-and-ios-native-applications/. [abgerufen am: 28.08.2019].

[Altexsoft, Cross-Platform Comparison]
Altexsoft, „Xamarin vs React Native vs Ionic vs NativeScript: Cross-platform Mobile Frameworks Comparison“, 10-Apr-2018. Verfügbar unter: https://www.altexsoft.com/blog/engineering/xamarin-vs-react-native-vs-ionic-vs-nativescript-cross-platform-mobile-frameworks-comparison/. [abgerufen am: 26.08.2019].

[Android Documentation, ProgressBar]
„Android Documentation: ProgressBar“. Verfügbar unter: https://developer.android.com/reference/android/widget/ProgressBar.html. [abgerufen am: 11.09.2019].

[Android Documentation, Run Device]
„Android Documentation: Run Testing on Physical Device“. Verfügbar unter: https://developer.android.com/studio/run/device. [abgerufen am: 16.09.2019].

[Angular Documentation, Architecture]
„Angular Documentation: Component Architecture“. Verfügbar unter: https://angular.io/guide/architecture. [abgerufen am: 11.09.2019].

[Angular Documentation, Component Architecture]
„Angular Documentation: Component Architecture“. Verfügbar unter:
https://angular.io/guide/architecture-components.
[abgerufen am: 11.09.2019].

[Angular Documentation, Observables]
„Angular Documentation: Observables“. Verfügbar unter:
https://angular.io/guide/observables#observables.
[abgerufen am: 09.09.2019].

[Angular Documentation, Services]
„Angular Documentation: Tutorial Services“. Verfügbar unter:
https://angular.io/tutorial/toh-pt4#why-services.
[abgerufen am: 07.09.2019].

[Apkmonk, EAH Jena]
„Apkmonk, EAH Jena App“. Verfügbar unter:
https://www.apkmonk.com/app/de.fhe.eahmobile/.
[abgerufen am: 15.09.2019].

[Apple, CloudKit]
„Apple, CloudKit“. Verfügbar unter:
https://developer.apple.com/documentation/cloudkit.
[abgerufen am: 15.09.2019].

[Apple, iOS-13 Features]
„Apple, iOS-13 Feature Presentation“. Verfügbar unter:
https://www.apple.com/de/ios/ios-13/features/.
[abgerufen am: 15.09.2019].

[Apple, Xcode]
„Apple, Xcode“. Verfügbar unter:
https://developer.apple.com/xcode/.
[abgerufen am: 15.09.2019].

[Augsten, Web App]
S. Augsten, „Was ist eine Web App“. Verfügbar unter: https://www.dev-insider.de/was-ist-eine-web-app-a-596814/. [abgerufen am: 25.08.2019].

[Charles, About Charles]
„Charles: About Charles“. Verfügbar unter: https://www.charlesproxy.com/overview/about-charles/. [abgerufen am: 15.09.2019].

[Charles, Web Proxy]
„Charles: Web Debugging Proxy Application“. Verfügbar unter: https://www.charlesproxy.com. [abgerufen am: 15.09.2019].

[EAH, Stundenplanung]
„Stundenplanung der EAH Jena“. Verfügbar unter: https://stundenplanung.eah-jena.de. [abgerufen am: 15.09.2019].

[Font Awesome, Free]
„Font Awesome, Free Offer“. Verfügbar unter: https://fontawesome.com/icons?d=gallery&m=free. [abgerufen am: 15.09.2019].

[Github, React Native Apps]
„Github, Who's using React Native?“. Verfügbar unter: https://facebook.github.io/react-native/showcase.html. [abgerufen am: 14.09.2019].

[Google, Android-10 Features]
„Google, Android-10 Feature Presentation“. [Online]. Verfügbar unter: https://www.android.com/android-10/.
[abgerufen am: 15.09.2019].

[Google, Android Studio]
„Google, Android Studio“. Verfügbar unter: https://developer.android.com/studio.
[abgerufen am: 15.09.2019].

[Google, Firebase]
„Google, Firebase“. Verfügbar unter: https://firebase.google.com.
[abgerufen am: 15.09.2019].

[Google Developers, DevTools]
„Google Developers: Tools for Web Developers: Chrome DevTools“. Verfügbar unter:
https://developers.google.com/web/tools/chrome-devtools/.
[abgerufen am: 15.09.2019].

[Google Developers, Web Apps]
„Google Developers: Progressive Web Apps“. Verfügbar unter: https://developers.google.com/web/progressive-web-apps/.
[abgerufen am: 15.09.2019].

[Github, React Native Android]
„Github, React Native: Native Modules Android“. Verfügbar unter: https://facebook.github.io/react-native/docs/native-modules-android#the-toast-module.
[abgerufen am: 15.09.2019].

[iOS Documentation, UIActivityIndicatorView]
Apple, „iOS Documentation: UIActivityIndicatorView“. Verfügbar unter: https://developer.apple.com/documentation/uikit/uiactivityindicatorview. [abgerufen am: 11.09.2019].

[Material, Dark Theme]
„Material, Dark Theme Usage“. [Online]. Verfügbar unter: https://material.io/design/color/dark-theme.html#usage. [abgerufen am: 15.09.2019].

[Microsoft, Visual Studio Code]
„Microsoft: Visual Studio Code“. Verfügbar unter: https://material.io/design/color/dark-theme.html#usage. [abgerufen am: 16.09.2019].

[Nativescript, Marketplace]
„Nativescript, Marketplace“. Verfügbar unter: https://market.nativescript.org. [abgerufen am: 16.09.2019].

[NativeScript Angular Documentation, Getting Started]
„Nativescript Angular Documentation: Getting Started“. Verfügbar unter: https://docs.nativescript.org/angular/start/introduction. [abgerufen am: 29.08.2019].

[NativeScript Blog, Performance]
„Nativescript: Performance Tips & Tricks“. Verfügbar unter: https://www.nativescript.org/blog/nativescript-angular-performance-tips-tricks. [abgerufen am: 12.09.2019].

[NativeScript Blog, Hot Module Replacement]
„Nativescript: Performance Tips & Tricks“. Verfügbar unter: https://www.nativescript.org/blog/nativescript-hot-module-replacement [abgerufen am: 12.09.2019].

[NativeScript Documentation, Activity Indicator]
„Nativescript Angular Documentation: Activity Indicator“. Verfügbar unter: https://docs.nativescript.org/angular/ui/ng-components/activity-indicator. [abgerufen am: 11.09.2019].

[Nativescript Documentation, Build iOS]
„Nativescript Documentation: Build for iOS“. [Online]. Verfügbar unter: https://docs.nativescript.org/sidekick/getting-started/build#23-build-for-ios. [abgerufen am: 30.08.2019].

[NativeScript Documentation, Angular Code Sharing]
„Nativescript Documentation: Angular Code Sharing“. Verfügbar unter: https://docs.nativescript.org/angular/code-sharing/intro. [abgerufen am: 15.09.2019].

[NativeScript Documentation, Core Modules]
„Nativescript Core Documentation: Core Modules“. Verfügbar unter: https://docs.nativescript.org/core-concepts/technical-overview#core-modules. [abgerufen am: 30.08.2019].

[NativeScript Documentation, CLI]
„Nativescript Core Documentation: CLI“. Verfügbar unter: https://docs.nativescript.org/core-concepts/technical-overview#nativescript-cli. [abgerufen am: 30.08.2019].

[NativeScript Documentation, CSS]
„Nativescript Core Documentation: CSS for Nativescript Apps“. Verfügbar unter: https://docs.nativescript.org/ui/theme#css-themes-for-nativescript-apps. [abgerufen am: 11.09.2019].

[NativeScript Documentation, Finding Plug-Ins]
„Nativescript Core Documentation: Finding Plug-Ins“. Verfügbar unter: https://docs.nativescript.org/core-concepts/Plug-Ins#finding-Plug-Ins. [abgerufen am: 30.08.2019].

[NativeScript Documentation, Getting Started]
„Nativescript Core Documentation: Getting Started“. Verfügbar unter: https://docs.nativescript.org/start/introduction. [abgerufen am: 29.08.2019].

[NativeScript Documentation, HTTP Get]
„Nativescript Core Documentation: HTTP Get“. Verfügbar unter: https://docs.nativescript.org/angular/ng-framework-modules/http. [abgerufen am: 30.08.2019].

[NativeScript Documentation, Marshalling]
„Nativescript Core Documentation: Marshalling“. Verfügbar unter: https://docs.nativescript.org/core-concepts/ios-runtime/Marshalling-Overview#numeric-types. [abgerufen am: 30.08.2019].

[NativeScript Documentation, Load on Demand]
„Nativescript Core Documentation: RadListView: Load on Demand“. Verfügbar unter: https://docs.nativescript.org/ui/components/RadListView/overview#load-on-demand. [abgerufen am: 12.09.2019].

[NativeScript Documentation, Plug-Ins]
„Nativescript Core Documentation: Plug-Ins“. Verfügbar unter: https://docs.nativescript.org/core-concepts/technical-overview#nativescript-Plug-Ins. [abgerufen am: 30.08.2019].

[NativeScript Documentation, Runtimes]
„Nativescript Core Documentation: Runtimes“. Verfügbar unter: https://docs.nativescript.org/core-concepts/technical-overview#runtimes. [abgerufen am: 30.08.2019].

[Nativescript Documentation, Technical Overview]
„Nativescript Core Documentation, Technical Overview“. Verfügbar unter: https://docs.nativescript.org/core-concepts/technical-overview. [abgerufen am: 16.09.2019].

[NativeScript Documentation, Template Syntax]
„Nativescript Angular Documentation: Template Syntax“. Verfügbar unter: https://docs.nativescript.org/angular/core-concepts/angular-application-architecture#template-syntax. [abgerufen am: 30.08.2019].

[NativeScript Documentation, UI Styling]
„Nativescript Core Documentation: UI Styling“. Verfügbar unter: https://docs.nativescript.org/ui/styling#page-specific-css. [abgerufen am: 30.08.2019].

[NativeScript Documentation, User-Interface]
„Nativescript Angular Documentation: User-Interface“. Verfügbar unter: https://docs.nativescript.org/angular/ui/basics. [abgerufen am: 11.09.2019].

[Nativescript Plug-In, Firebase]
Github, „Nativescript Firebase Plug-In: Features“. Verfügbar unter: https://github.com/EddyVerbruggen/nativescript-Plug-In-firebase#features. [abgerufen am: 15.09.2019].

[Nativescript Plug-In, Performance Monitor]
„Naticescript Plug-In, Performance Monitor“. Verfügbar unter: https://github.com/EddyVerbruggen/nativescript-performance-monitor. [abgerufen am: 13.09.2019].

[NativeScript Presentation, Features]
„Nativescript Presentation: Features“. Verfügbar unter: https://www.nativescript.org/native-api-access. [abgerufen am: 29.08.2019].

[NativeScript Setup, macOS]
„NativeScript Advanced Setup: macOS“. Verfügbar unter: https://docs.nativescript.org/angular/start/ns-setup-os-x. [abgerufen am: 05.09.2019].

[NativeScript Setup, Windows]
„NativeScript Advanced Setup: Windows“. Verfügbar unter: https://docs.nativescript.org/angular/start/ns-setup-win#nativescript-advanced-setup-windows. [abgerufen am: 05.09.2019].

[React Documentation, Android Modules]
„React Native Documentation: Native Android Modules“. Verfügbar unter: https://facebook.github.io/react-native/docs/native-modules-android.html. [abgerufen am: 29.08.2019].

[React Documentation, Performance]
„React Native Documentation: Performance“. Verfügbar unter: https://facebook.github.io/react-native/docs/performance.html. [abgerufen am: 29.08.2019].

[Reith, Worldwide Smartphone Shipment]

R. Reith und M. Chau, „IDC, Worldwide Smartphone Shipment“, Verfügbar unter: https://www.idc.com/promo/smartphone-market-share/os. [abgerufen am: 25.08.2019].

[RxJS, Documentation]

„RxJS Library: Documentation“. Verfügbar unter: https://rxjs-dev.firebaseapp.com/guide/overview. [abgerufen am: 29.08.2019].

[Statista, Anzahl Apps]

„Statista, Anzahl der Apps in den Top App Stores“. Verfügbar unter: https://de.statista.com/statistik/daten/studie/208599/umfrage/anzahl-der-apps-in-den-top-app-stores/. [abgerufen am: 14.09.2019].

[Statista, Anzahl Smartphone-Nutzer]

„Statista, Anzahl der Smartphone-Nutzer in Deutschland“. Verfügbar unter: https://de.statista.com/statistik/daten/studie/198959/umfrage/anzahl-der-smartphonenutzer-in-deutschland-seit-2010/. [abgerufen am: 14.09.2019].

[Typescript, Description]

„TypeScript, Description“. Verfügbar unter: https://www.typescriptlang.org. [abgerufen am: 14.09.2019].

[TypeScript Documentation, Type Assertion]

„TypeScript Documentation: Type Assertion“. Verfügbar unter: https://www.typescriptlang.org/docs/handbook/basic-types.html#type-assertions. [abgerufen am: 08.09.2019].

[Xamarin Documentation, Binding Java]

Microsoft, „Xamarin-Dokumentation“, *Binding Java Library*, 05-Jan-2017. Verfügbar unter: https://docs.microsoft.com/de-de/xamarin/android/platform/binding-java-library/#overview. [abgerufen am: 28.08.2019].

Whitepaper

[Delia, 2015]

L. Delia, N. Galdamez, P. Thomas, und Leonardo Corbalan, „Multi-Platform Mobile Application Development Analysis“. Institute of Research in Computer Science III-LIDI. Verfügbar unter: https://www.researchgate.net/profile/Lisandro_Delia/publication/285643425_Multi-platform_mobile_application_development_analysis/links/5c6ad876299bf1e3a5b24bcf/Multi-platform-mobile-application-development-analysis.pdf
[abgerufen am: 26.08.2019].

[Majchrzak, 2018]

T. A. Majchrzak, „Progressive Web Apps: the Definite Approach to Cross-Platform Development?“ Proceedings of the 51st Hawaii International Conference on System Sciences. Verfügbar unter: https://pdfs.semanticscholar.org/6723/24aaa5de7c41a57d951898d2a7dd21b15f04.pdf
[abgerufen am: 25.08.2019].

[Long, 2016]

Q. Y. J. Long, „Research on the Development Technology of Cross Platform Hybrid Mobile Application Based on HTML5“. Atlantis Press. Verfügbar unter: https://pdfs.semanticscholar.org/a53c/e48a1bc12a46420a3e8acbfcc9b888d37559.pdf
[abgerufen am: 12.08.2019].

[Xianghao, 2014]

Y. Xianghao und W. Li, „Research and Design of Mobile Payment Platform Based on Hybrid APP Technology“. Trans Tech Publications. Verfügbar unter: https://www.scientific.net/AMR.1044-1045.1262
[abgerufen am: 15.09.2019].

Anhang

Testreihe: Frames-per-Second-Messung der Navigation in Stundenplan-App

	Samsung Galaxy S7		Nokia 5.1	
	1. Versuch	**2. Versuch**	**1. Versuch**	**2. Versuch**
Startseite	**59.9**	**59.9**	**56.9**	**56.9**
	59.8	59.8	56.9	56.9
	59.8	59.8	56.9	56.9
	59.9	60.4	56.9	56.9
	59.8	59.8	56.9	56.8
	59.8	48.2	16.6	20.8
Stundenplan	**1.8**	**33.6**	**3.6**	**1.7**
	27.9	37.9	0.3	0.4
	37.8	57.8	0.4	0.4
	59.8	59.8	0.4	0.5
	59.8	59.8	0.5	0.1
	59.9	59.9	31.4	53.3
	59.8	31.9	56.8	8.7
Speiseplan	**5.4**	**29**	**6.9**	**3.8**
	25	20.2	29.4	47.2
	58.7	58.7	56.9	56.9
	59.8	59.8	54.9	56.9
	43.8	59.8	56.9	17.1
Neues	**17.4**	**33.9**	**29.4**	**11.8**
	59.9	59.9	56.9	56.8
	59.8	59.8	54.8	56.8
	59.8	59.8	56.9	56.8
	59.9	59.9	56.9	54.8
	59.8	59.8	56.9	56.9
Wetter	**33.9**	**31.9**	**23.5**	**25.5**
	57.9	53.8	56.9	56.9
	59.8	59.8	56.8	56.8
	59.8	59.8	57	56.9
	59.8	59.8	37.9	47.2
Rechtliches	**15.5**	**43.8**	**0.5**	**5.3**
	49.4	59.8	43.2	43.1
	53.8	59.8	56.9	56.9
	59.8	59.9	56.9	52.9
	59.8	59.8	56.8	56.9
	59.8	59.8	47.1	56.8
Einstellungen	**27.6**	**30.8**	**15.5**	**11.4**
	51.2	45.8	51	45.1

Inhalt der CD-ROM

Bachelorarbeit elektronisch:

Bachelorarbeit_638777_MichaelSchulz.pdf

Quellen:

Literaturverzeichnis/Internetquellen

Literaturverzeichnis/Whitepaper

Programmcode:

Quellcode/eahjena/

Hinweis: Innerhalb des Quellcode-Ordners ist eine „README.md“-Datei hinterlegt. Diese enthält weiterführende Details des hinterlegten Programmcodes. Dazu zählen eine Beschreibung der Ordnerstruktur, Inhalte der Ordner und Informationen zur Handhabung der Cross-Plattform-App.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Sämtliche Stellen der Arbeit, die benutzten Werken im Wortlaut oder dem Sinn nach entnommen sind, habe ich durch Quellenangaben kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen und dergleichen sowie für Quellen aus dem Internet.

Bei Zuwiderhandlung gilt das Seminar als nicht bestanden.

Datum und Unterschrift der Verfasserin / des Verfassers