

Semesterprojekt

in

AIN

„Angular Universal“

zur Performancesteigerung
von Angular Webanwendungen

Vorgelegt am: 26.01.2018

Vorgelegt von: Gießler, Nathalie, 251433
nathalie.giessler@hs-furtwangen.de
Keßler, Jan, 252241
jan.kessler@hs-furtwangen.de

Abstract

Classical Angular web applications are rendered on the client side. Universal uses server-side rendering. This should greatly improve the performance. The goal of the project is to run and evaluate comparable performance tests for both variants of a self written Angular web application.

The web application should be composed of common elements to generate representative test results. This can be fulfilled by an image portal, which includes uploading and looking at images, selecting an image and commenting it. The web application also has an authentication system and a database for storing users and images.

Mit Angular Universal können klassische Angular Webanwendungen bereits serverseitig gerendert werden. Damit soll die Performance enorm gesteigert werden können. Ziel des Projekts ist es, anhand einer eigenen Beispielwebanwendung vergleichbare Performancetests bei beiden Varianten der Webanwendung durchzuführen und auszuwerten.

Die Webanwendung soll aus verschiedenen Seiten mit üblichen Elementen bestehen um repräsentative Tests liefern zu können. Daher ist die Entscheidung für ein Bilderportal gefallen, bei dem Bilder angeschaut, hochgeladen und kommentiert werden können. Zudem verfügt die Anwendung über ein Authentifizierungssystem und eine Datenbank.

Inhaltsverzeichnis

Abstract.....	I
Inhaltsverzeichnis	III
Abbildungsverzeichnis	V
Tabellenverzeichnis	VII
Abkürzungsverzeichnis	IX
1 Einleitung.....	1
2 Angular	3
2.1 Die Architektur	3
2.1.1 Modules.....	4
2.1.2 Components.....	4
2.1.3 Templates	4
3 Angular Universal	5
3.1 Funktionsweise	6
4 Beispiel-Webanwendung	7
4.1 Planung.....	7
4.1.1 Spezifikation der Anforderungen	7
4.1.2 Anwendungsfalldiagramm	7
4.1.3 Grafische Benutzeroberfläche.....	9
4.2 Verwendete Software-Technologien.....	9
4.3 Implementierung	10
4.3.1 Einrichten Entwicklungsumgebung.....	10
4.3.2 Generieren der Components und Services	10
4.3.3 Models.....	11
4.3.4 Firebase	12
4.3.5 Aufbau Components und Services (Beispiel).....	12

4.4	Umstellung auf Angular Universal	13
4.4.1	Verwendung Universal-Starter	13
4.4.2	Manuelle Umstellung	14
5	Test	15
5.1	Auswahl des Tools	15
5.2	Testszenario	16
5.3	Aufsetzen und Implementierung von Protractor	17
5.3.1	conf.js	18
5.3.2	spec.js	18
5.4	Navigation Timing API	21
5.5	Chrome Developer Tools	23
5.6	Auswertung der Testergebnisse	24
5.6.1	Navigation Timing API	24
6	Fazit	26
	Literaturverzeichnis	27
	Eidesstattliche Erklärung	28
A.	[Anhang]	29

Abbildungsverzeichnis

Abbildung 1: Angular Architektur Übersicht	3
Abbildung 2: Anwendungsfalldiagramm Image Board	8
Abbildung 3: Components und Services	11
Abbildung 4: Models	11
Abbildung 5: Firebase Imports	12
Abbildung 6: Front-Page Component.....	12
Abbildung 7: AngularFireDatabase	13
Abbildung 8: conf.js Datei	18
Abbildung 9: Öffnen der Startseite	18
Abbildung 10: Öffnen der Login-Seite durch Buttonclick.....	19
Abbildung 11: Durchführung des Logins	19
Abbildung 12: Anklicken des ersten Bildes	20
Abbildung 13: Kommentar einfügen und absenden	20
Abbildung 14: Logout.....	20
Abbildung 15: Festlegen des Treibers	21
Abbildung 16: Methoden für Seitenladezeit und Processing-Zeit.....	22
Abbildung 17: Übersicht der Attribute des Timing-Interfaces	22
Abbildung 18: Syntax der API in Java.....	23
Abbildung 19: Aufruf der Methoden zur Zeitbestimmung	23
Abbildung 20: Ergebnisse von Angular	24
Abbildung 21: Ergebnisse von Angular Universal	24

Tabellenverzeichnis

Tabelle 1: Auswertung Developer Tools	25
---	----

Abkürzungsverzeichnis

SPA	Single Page Application
API	Application Programming Interface (Schnittstelle zur Anwendungsprogrammierung)
DOM	Document Object Model (API für HTML und XML Dokumente)

1 Einleitung

Lange Zeit zeichneten sich gute Web-Anwendungen dadurch aus so viele Berechnungen wie möglich serverseitig zu erledigen. Inzwischen ist das nicht mehr der Fall und viele Web-Anwendungen setzen auf die clientseitige Verarbeitung ihrer Aufgaben, in vielen Fällen mit Hilfe von JavaScript. Das steigert die Benutzerfreundlichkeit und ermöglicht eine einfache Anpassung der Auflösung an die vielen unterschiedlichen mobilen Endgeräte.

Single-Page-Webanwendungen (SPA) sind zurzeit ein äußerst beliebter Architekturstil für Web-Anwendungen. Das SPA-Konzept sieht vor, dass eine Web-Anwendung nur aus einem einzelnen HTML-Dokument besteht. Weitere benötigte Dateien werden per direktem HTTP-Zugriff durch die SPA mit Wirken von JavaScript angefordert.

Bei Angular handelt es sich um ein Webapplikationsframework, das sowohl auf clientseitige Verarbeitung, als auch auf das SPA-Konzept setzt. Die clientseitige Verarbeitung hat auch Nachteile; so leidet die Endgerät-Performance darunter. Auch der Webcrawler von Google liefert fehlerhafte Ergebnisse beim Erstellen von Vorschaubildern, wie sie beim Veröffentlichen von Links auf sozialen Plattformen üblich sind.¹

Hier kommt Angular Universal ins Spiel, wo wieder auf das Konzept des serverseitigen Renderns zurückgegriffen wird. Wie groß die Vorteile von Universal (insbesondere was die Performance betrifft) wirklich sind, wird im Laufe dieses Projektes erläutert.

¹ Vgl. Angular Universal. In <https://universal.angular.io/>, zugegriffen am 19.01.2018

2 Angular

Angular ist ein Framework, das zum Erstellen von Webanwendungen in HTML und JavaScript genutzt wird. Anstatt JavaScript zu verwenden, können auch andere Sprachen, die zu JavaScript kompiliert werden, genutzt werden (z.B. TypeScript).

Beim Erstellen einer Anwendung mit Angular, werden normale HTML-Templates, mit Angular spezifischer Syntax versehen. Zum Steuern der Templates werden Components erstellt. Die Logik der Anwendung befindet sich in Services. Alle Komponenten werden in einem oder mehreren Modules gebündelt und verwaltet.²

2.1 Die Architektur

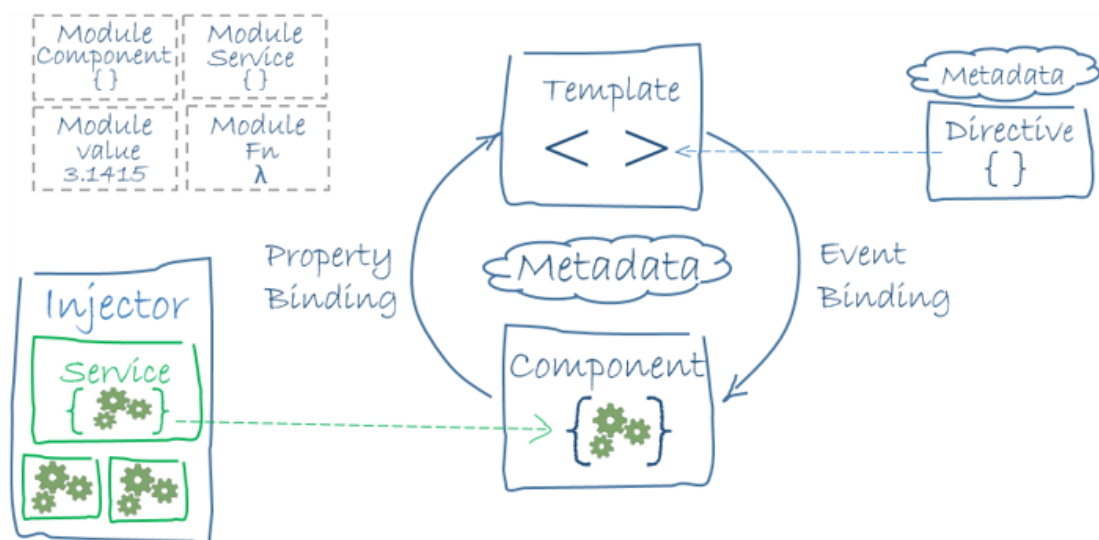


Abbildung 1: Angular Architektur Übersicht

Quelle: <https://angular.io/generated/images/guide/architecture/overview2.png>
zugegriffen am 18.12.2017

² Vgl. Architecture Overview. In <https://angular.io/guide/architecture>, zugegriffen am 18.12.2017

2.1.1 Modules

In der Angular Module-Klasse wird beschrieben, wie die einzelnen Teile einer Anwendung zusammenspielen. Jede Anwendung hat mindestens ein Module, das sogenannte Root-Module. Dieses wird zum Starten der Anwendung genutzt. Gerade bei kleineren Anwendungen reicht meistens ein Module aus. Bei größeren Projekten kann das Verwenden von mehreren Modules zu einer besseren Übersichtlichkeit beitragen.²

2.1.2 Components

Ein Component kontrolliert dem ihm zugeteilten View; also die für den Benutzer sichtbaren Bestandteil einer Web-Anwendung. Die Components interagieren wiederum mit den Services, von denen sie die benötigten Daten erhalten (z.B. von Datenbanken). Angular erstellt, aktualisiert und zerstört Components, während sich der Nutzer durch die Web-Anwendung bewegt.²

2.1.3 Templates

Die von den Components kontrollierten Views werden in Templates definiert. Das Template ist eine Art HTML-Dokument, das Angular darüber aufklärt wie ein Component gerendert werden soll. Dabei werden typische HTML-Elemente verwendet, aber auch Angular eigene Syntax.²

² Vgl. Architecture Overview. In <https://angular.io/guide/architecture>, zugegriffen am 18.12.2017

3 Angular Universal

Angular Universal ist als eigenständiges Projekt gestartet mit dem Ziel, Angular komplett plattformunabhängig auszuführen. Seit Angular 4 wurde Universal in das Hauptframework integriert und erlaubt serverseitiges Rendering.

Clientseitiges Rendern der Webseiten führt dazu, dass Suchmaschinen Probleme haben auf die Website zuzugreifen, da sie reines HTML erwarten. Sie können nicht bestimmen, wann das JavaScript Framework das Rendern abgeschlossen hat und sehen daher nur einen sehr kleinen Teil der Webseite. Die Lösung ist daher, dass der Client eine Anfrage an den Server schickt und eine fertige Seite präsentiert bekommt. Zusätzlich wird die Menge an Daten, die der Client herunterladen muss, im Rahmen gehalten. Bei dem clientseitigen Rendern müssen erst alle JavaScript Dateien heruntergeladen werden, was vor allem auf mobilen Endgeräten zu Problemen führen kann. Bei einer Angular Universal App muss das ausführende Gerät JavaScript nicht unterstützen, da man für diese User eine serverseitig gerenderte JavaScript-freie Version anbieten kann.³

Ein großer Vorteil von Angular Universal ist außerdem, dass die erste Seite der Webapplikation sehr schnell angezeigt werden kann. Dies ist vor allem für mobile Geräte sehr wichtig, da diese den Ladevorgang einer Seite nach etwa 3 Sekunden abbrechen.

³ Vgl. Angular Universal: server-side rendering. In <https://angular.io/guide/universal>, zugegriffen am 18.12.2017

3.1 Funktionsweise

Die Installation des „platform-server“ Pakets, welches die Serverimplementierung der DOM „XMLHttpRequest“ und weitere, vom Browser unabhängige Funktionalitäten enthält, wird zur Erstellung von Angular Universal Applikationen benötigt. Nun kann die normale Client Applikation, anstatt mit dem „platform-browser“ Modul, mit dem für Universal installierten „platform-server“ Modul kompiliert werden. Dadurch entsteht eine Angular Universal Applikation, welche nun auf dem Server ausführbar ist. Nutzeranfragen für Seiten der Applikation werden nun vom Server an die „renderModuleFactory“ Funktion weitergeleitet.

Diese Funktion nutzt ein HTML-Template (normalerweise index.html), ein Angular Modul mit Komponenten sowie eine Route, welche die anzuzeigende Komponente bestimmt. Damit wird die Seite innerhalb des <app> Kennzeichens gerendert und dem Nutzer vom Server eine fertige HTML-Seite präsentiert. Jede Nutzeranfrage bestimmt die Route und führt zu der passenden Ansicht.³

Bei der Programmierung der Angular Universal Applikation muss beachtet werden, dass einige Funktionalitäten der Browser-API auf dem Server fehlen. So kann man zum Beispiel nicht auf browsereigene Objekte wie „window“, „document“, „navigator“ und „location“ zugreifen. Stattdessen muss man mit Angular Abstraktionen arbeiten, welche die Funktionen der Browser-API ersetzen. Falls es keine geeignete Abstraktion gibt, muss selbst eine geschrieben werden, die dann gegebenenfalls die Aufgaben an die Browser-API weitergibt, sobald die Applikation im Browser geöffnet ist oder eine zufriedenstellende Alternativimplementierung für den Server bietet.³

Bei serverseitig gerenderten Seiten kann der Nutzer nur Links klicken, weshalb es Sinn macht, die klientseitige Applikation im Hintergrund zu laden und den Nutzer schnellstmöglich darauf wechseln zu lassen. Diese Vorgehensweise erfordert allerdings ein paar Anpassungen im Code der Applikation.⁴

³ Vgl. Angular Universal: server-side rendering. In <https://angular.io/guide/universal>, zugegriffen am 18.12.2017

⁴ Vgl. Angular Universal: Modify the Client App. In <https://angular.io/guide/universal#transition> zugegriffen am 31.12.2017

4 Beispiel-Webanwendung

4.1 Planung

4.1.1 Spezifikation der Anforderungen

Aus der Projektbeschreibung geht hervor, dass an die Webanwendung folgende Anforderungen gestellt werden:

- Basiert auf Angular 4 und Node.js
- Vordergründig für Performance-Tests ausgelegt.
- „Übliche Elemente“ sollen vorhanden sein (große Tabellen, Schaltflächen, Eingabefelder, Bilder).
- Erweiterbar um Angular Universal
- Angular 4 und Universal müssen vergleichbar sein

4.1.2 Anwendungsfalldiagramm

Um allen Anforderungen gerecht zu werden und trotzdem ein realitätsnahes Beispiel zu haben, wurde beschlossen ein „Image Board“ zu entwickeln. Dieses soll ähnlich aufgebaut sein wie die Website „imgur.com“, aber nur insoweit funktional sein, wie es die Anforderungen verlangen. Um die daraus folgende Webanwendung besser planen zu können, wurde ein Anwendungsfalldiagramm erstellt:

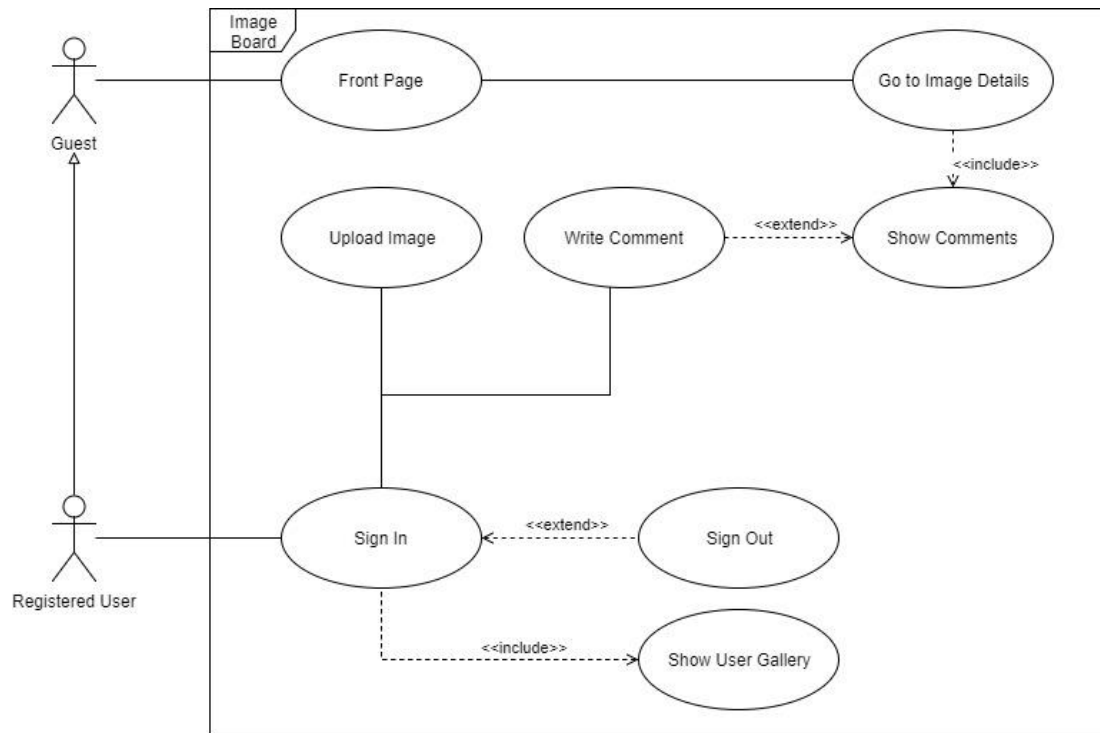


Abbildung 2: Anwendungsfalldiagramm Image Board

4.1.3 Grafische Benutzeroberfläche

Da die Beispielanwendung vordergründig für Performance-Tests verwendet wird, spielt das genaue Aussehen keine große Rolle. Um trotzdem eine funktionstüchtige Webanwendung simulieren zu können und die Benutzung während der Tests zu erleichtern, werden die Standardklassen des freien CSS-Framework Bootstrap verwendet.

4.2 Verwendete Software-Technologien

- **Visual Studio Code** ist ein Quelltext-Editor von Microsoft. Er bietet eine gute TypeScript-Unterstützung (TypeScript ist ebenfalls von Microsoft), sowie Funktionen zu Versionsverwaltung mit Git. Außerdem ist VS Code Open Source und für Windows, Mac Os und Linux erhältlich.⁵
- **GitHub** wird zur Versionsverwaltung des Projekts per Git verwendet. Der Online-Dienst ist gratis nutzbar und alle Teammitglieder hatten bereits Erfahrung damit.⁶
- **Firebase** dient als Back End, bzw. genauer gesagt zur Nutzer-Authentifizierung, Daten-Speicherung und Datenbank. Der Online-Dienst ist ebenfalls kostenlos nutzbar und erleichtert die gemeinsame Arbeit an der Webanwendung enorm. Dazu kommt, dass Firebase wie Angular von Google ist und eine breite Anzahl von Plugins zur gemeinsamen Verwendung bietet.⁷

⁵ Vgl. Setting up Visual Studio Code. In <https://code.visualstudio.com/docs/setup/setup-overview>, zugegriffen am 18.12.2017.

⁶ Vgl. Hello World. In <https://guides.github.com/activities/hello-world/>, zugegriffen am 18.12.2017

⁷ Vgl. Add Firebase to your JavaScript Project. In <https://firebase.google.com/docs/web/setup>, zugegriffen am 18.12.2017.

4.3 Implementierung

4.3.1 Einrichten Entwicklungsumgebung

Es wurde stichwortartig wie folgt vorgegangen:

- Aufsetzen des GitHub Repository.
- Anlegen eines Accounts auf <https://firebase.google.com/>; diesem ein neues Projekt hinzufügen.
- VS Code und Node.js auf dem jeweils verwendeten Betriebssystem installieren.
- Außerdem wird zum Anlegen eines neuen Angular-Projekts bzw. zum Erstellen von Components und Services die Node.js Erweiterung Angular CLI genutzt. Die Installation erfolgt beim Ausführen des folgenden Terminal-Befehls:

```
npm install -g @angular/cli
```

- Abschließend wird mit:

```
ng new semester-project-angular-universal
```

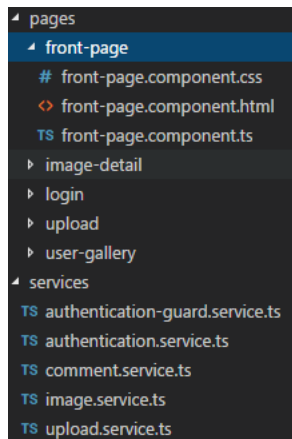
ein neues Angular-Projekt erstellt. Dieses wird dann zum GitHub Repository hinzugefügt.

4.3.2 Generieren der Components und Services

Im neu erstellten Projekt wurden alle benötigten Components und Services mit Hilfe von CLI generiert. Die Components konnten von dem Anwendungsfalldiagramm abgeleitet werden, jede aufrufbare Seite hat ihr eigenes Component. Generieren lassen sich die Components und Services mit dem Befehl

```
ng generate
```

Die so neu generierten Elemente werden automatisch im app.module.ts eingetragen. Nach dem händischen ordnen der Elemente ergibt sich folgende Dateistruktur:



Zu jedem Component wurde eine eigene HTML- und CSS-Datei erstellt. Die Services sind in dieser Webanwendung für die Kommunikation zwischen den Components und Firebase zuständig.

Abbildung 3: Components und Services

- **Authentication-guard** sichert die Seiten ab, die nur für eingeloggte Benutzer sichtbar sein sollen.
- **Authentication** wird vom Login und Logout verwendet.
- **Comment** trägt neue Kommentare in die Datenbank ein und kann diese auch abrufen.
- **Image** kann die Bilder von Firebase samt zugehörigen Datenbankinformationen abrufen.
- **Upload** dient zum hochladen der Bilder, sowie dem eintragen derer Informationen in die Datenbank.

4.3.3 Models

Es werden vier Models verwendet: „Comment“, „GalleryImage“, „Upload“ und „User“. Diese stellen die Objekte dar, die zwischen Firebase und der Client-Anwendung ausgetauscht werden.

```
1 export interface GalleryImage {  
2     $key?: string;  
3     name?: string;  
4     url?: string;  
5     uid?: string;  
6 }  
7
```

Hier beispielhaft das „GalleryImage“. Der \$key? dient zur eindeutigen Identifikation in der Datenbank. Die anderen Angaben sind Objektspezifisch.

Abbildung 4: Models

4.3.4 Firebase

Zur Kommunikation mit Firebase werden folgende Node.js-Erweiterungen verwendet:

- **Firebase** <https://github.com/firebase/firebase-js-sdk>
- **AngularFire** <https://github.com/angular/angularfire2>

Beide Erweiterungen wurden in das „package.json“ eingetragen, um sie anschließend per Kommando zu installieren. In das Root-Module konnten jetzt folgende Funktionalitäten importiert werden:

```
import { AngularFireModule } from 'angularfire2';
import { AngularFireDatabaseModule } from 'angularfire2/database';
import { AngularFireAuthModule } from 'angularfire2/auth';
```

Abbildung 5: Firebase Imports

4.3.5 Aufbau Components und Services (Beispiel)

Um die Zusammenarbeit und Arbeitsweise der Components und Services besser zu veranschaulichen, folgt eine Funktionsbeschreibung des Component front-page:

```
export class FrontPageComponent implements OnInit, OnChanges {
  images: Observable<GalleryImage[]>; //list of all Images

  constructor(private ImageServis: ImageService) { }

  ngOnInit() {
    this.images = this.ImageServis.getImages();
  }

  ngOnChanges() {
    this.images = this.ImageServis.getImages();
  }
}
```

Abbildung 6: Front-Page Component

„images“ ist eine Liste aller Bilder, die für die Startseite bestimmt sind. Sobald das Component aufgerufen wird (ngOnInit), wird aus dem Service die aktuelle Liste abgerufen. Sollte eine Änderung stattfinden (ngOnChanges) wird erneut ein Abruf durchgeführt.


```
getImages(): Observable<GalleryImage[]> { // return a map with GalleryImage objects: map key = Image key
// value = object
return this.db.list('uploads').snapshotChanges().map(actions => {
  return actions.map(action => {
    const $key = action.payload.key;
    const data = { $key, ...action.payload.val() };
    return data;
  });
});
}
```

Abbildung 7: AngularFireDatabase

Bei „db“ handelt es sich um ein „AngularFireDatabase“-Objekt. In einer „map“ werden der Key und die Adresse jedes Bildes abgelegt und an die aufrufende Methode verschickt.

Das komplette Projekt ist hier einsehbar: <https://github.com/jannks/semester-project-angular-universal> (zugegriffen am 19.01.2017)

Bilder der fertigen Webanwendung befinden sich im Anhang.

4.4 Umstellung auf Angular Universal

Um das eigene Angular 4 Projekt auf Universal umzustellen, gibt es im Wesentlichen zwei Möglichkeiten:

4.4.1 Verwendung Universal-Starter

Bei Verwendung des Universal-Starter wird ein neues Universal-Projekt erstellt, was ein vorkonfiguriertes Grundgerüst darstellt. Hier müssen alle, an dem über CLI generierten Projekt, durchgeführten Änderungen in das neu erstellte Projekt kopiert werden. Für die Webanwendung des Projekts wurde dieser Weg gewählt. Gerade für kleine Web-Anwendung dürfte dieses Vorgehen das schnellste sein. Der Universal-Starter ist hier zu finden: <https://github.com/angular/universal-starter>. (zugegriffen am 19.01.201)

4.4.2 Manuelle Umstellung

Bei größeren Web-Anwendungen ist die manuelle Umstellung zu bevorzugen. Aber auch hier kann CLI genutzt werden, weswegen sich der Aufwand in Grenzen hält. So können alle benötigten Pakete wie folgt installiert werden:⁸

```
npm install --save @angular/platform-server @nguniversal/module-map-ngfactory-loader ts-loader
```

Anschließend sind noch ein paar kleinere Änderungen im vorhandenen Projekt nötig. Zusätzlich müssen die Dateien, die die Konfiguration des Render-Servers enthalten, erstellt werden. Die exakte Vorgehensweise, inklusive Code-Beispiele kann im offiziellen Repository eingesehen werden: <https://github.com/angular/angular-cli/wiki/stories-universal-rendering>. (zugegriffen am 19.01.2017)

⁸ Vgl. Angular Universal Integration. In <https://github.com/angular/angular-cli/wiki/stories-universal-rendering>, zugegriffen am 19.01.2017.

5 Test

5.1 Auswahl des Tools

Als Testsoftware haben wir uns zuerst für das Tool Protractor entschieden. Protractor ermöglicht das Schreiben von automatisierten e2e (End-to-End) Tests, die einen Nutzer simulieren. Es wurde für AngularJS entwickelt und für die neue Angular Version aktualisiert. Es ist auf WebDriverJS aufgebaut, welcher systemeigene Events und spezifische Treiber für Browser nutzt, um mit der Angular Applikation zu interagieren, wie es ein Nutzer tun würde. Aufgrund der Unterstützung für Angular speziellen Locator Strategien, können Angular-Elemente ohne zusätzlichen Aufwand einfach getestet werden.

Ein weiterer Vorteil des Tools ist die automatische Waiting-Funktion. Anstatt eigene Wartezeiten oder Sleep-Funktionen einzubauen, wird der nächste Schritt des Tests automatisch ausgeführt, sobald der vorherige abgeschlossen ist.

Leider kann Protractor nur sehr rudimentäre Ergebnisse im Bereich Performancetesting liefern, weshalb wir uns nach einer weiteren Testmöglichkeit umgesehen haben. Dabei sind wir auf die Navigation Timing API gestoßen, welche präzise Daten für das Auswerten der Performance bietet. Mithilfe dieser API ist es möglich, automatisierte Tests mit Aufzeichnung der Performancewerte aus der Perspektive des Clients zu erstellen. Zudem ist es möglich gezielt die für das Rendern benötigte Zeit zu messen.⁹

⁹ Vgl. Navigation Timing API. In https://developer.mozilla.org/en-US/docs/Web/API/Navigation_timing_API, zugegriffen am 12.01.2018.

5.2 Testszenario

Um einen umfassenden Performancetest machen zu können, soll jede Funktionalität ausgeführt werden. Daher durchläuft das Testszenario alle möglichen Schritte:

1. Aufruf der Startseite
2. Klick des „Log In“-Buttons in der Navigationsleiste
3. Eingabe und Absenden der Login-Daten, danach erfolgt eine automatische Weiterleitung zur Startseite
4. Klick des „Upload“-Buttons
5. Eingabe einer JPG-Datei und bestätigen des Uploads
6. Auswahl eines Bildes
7. Eingabe und Absenden eines Kommentars
8. Klick des „Log Out“ Buttons

Diese Schritte sollen mehrfach durchlaufen werden um Durchschnittswerte zu erhalten.

5.3 Aufsetzen und Implementierung von Protractor

Installiert wird das Tool Protractor ganz einfach über den JavaScript Packagmanager npm. Diesen haben wir praktischerweise aufgrund Node.js schon installiert, da das der standard Packagmanager für Node.js ist.

```
npm install -g protractor
```

Damit haben wir zwei Kommandozeilentools installiert: „protractor“ und „webdriver-manager“. Letzteres ist ein Hilfstool, welches uns ermöglicht einen Selenium-Server aufzusetzen. Damit können Browser automatisiert werden. Dies ist in erster Linie für Testzwecke gedacht, kann allerdings auch für das automatisierte Ausführen von wiederholten Aufgaben eingesetzt werden.

Zur Überprüfung der Installation ist es sinnvoll, die Version zu überprüfen. Dies kann mit dem Befehl

```
protractor --version
```

einfach bewerkstelligt werden.

Danach muss der Seleniumserver durch den Webdrivermanager aktualisiert und gestartet werden.

```
webdriver-manager update  
webdriver-manager start
```

Auf der Browserseite <http://localhost:4444/wd/hub> kann der Status des Servers eingesehen werden.

Der Protractor-Test besteht aus zwei Dateien. Zum ersten aus der `conf.js` Datei, welche die Konfiguration des Tests enthält und zum anderen aus der `spec.js` Datei, welche die eigentlichen Tests enthält.¹⁰

¹⁰ Vgl. Setup. In <http://www.protractortest.org/#/>, Eingesehen am 18.12.2018

5.3.1 conf.js

```
// conf.js
exports.config = {
  framework: 'jasmine',
  seleniumAddress: 'http://localhost:4444/wd/hub',
  specs: ['spec.js'],

  capabilities: {
    browserName: 'chrome'
  }
}
```

Abbildung 8: conf.js Datei

In der Konfigurationsdatei des Tests wird das Framework, die Standardadresse des Seleniumservers, der Name der eigentlichen Testdatei und der zu verwendende Browser angegeben.

In diesem Test haben wir mit dem Open-Source Framework „jasmine“ gearbeitet, welches das Testen von JavaScript-Dateien mit Node.js ohne zusätzliche Abhängigkeiten und übersichtlicher Syntax bietet.

5.3.2 spec.js

In dieser Datei werden nun innerhalb der einzelnen „it()“-Aufgaben die Schritte definiert, die ausgeführt werden sollen. Hierbei bekommt jede Funktionalität eine eigene Aufgabe, so dass bei einem Fehler festgelegt werden kann, bei welcher Aufgabe dieser auftritt. Dieses Feature ist für Performancetests nicht so wichtig, da der Test dazu funktionieren muss, allerdings fördert es die Übersichtlichkeit und Lesbarkeit.

```
it('should open front page', function() {
  browser.get('http://localhost:4200');
});
```

Abbildung 9: Öffnen der Startseite

Der Aufbau einer „it()“-Aufgabe besteht aus zwei Teilen: Zum einen aus einer Bezeichnung der Aufgabe und zum anderen aus den eigentlichen Anweisungen. So soll in der ersten Aufgabe die Startseite per direktem Aufruf der Adresse geöffnet werden.

```
it('should open login page', function(){  
  element(by.linkText('Log In')).click();  
});
```

Abbildung 10: Öffnen der Login-Seite durch Buttonclick

In der zweiten Aufgabe soll ein Button geklickt werden, um zur Loginseite zu gelangen. Dies ist wichtig, da wir einen Nutzer simulieren wollen. Das Element des Buttons wird durch den Namen angesprochen und der Klick aktiviert.

```
it('should perform the login and open start page', function(){  
  element(by.id('emailInput')).sendKeys('nathalie.giessler@hs-furtwangen.de');  
  element(by.id('passwordInput')).sendKeys('123456');  
  element(by.css('button[type="submit"]')).click();  
  browser.driver.sleep(2000); //needed to let the site load  
});
```

Abbildung 11: Durchführung des Logins

Die dritte Aufgabe soll die Textfelder anhand ihrer ID finden und mit Daten befüllen. Anschließend wird der Absenden-Button per Klick aktiviert. Bei dieser Aufgabe, muss aufgrund des Uploads, eine Sleep-Funktion eingebaut werden. Protractor wartet nun automatisch bis alle Angular-Funktionen abgeschlossen sind, nicht auf andere Aktionen. Ohne das Warten würden weitere Aufgaben fehlschlagen, da diese auf das Ergebnis der vorherigen Aufgabe aufbauen.

```
it('should click on first picture', function(){  
  element(by.xpath("/html/body/app-root/div/app-front-page/div[1]/div/a/img")).click();  
})
```

Abbildung 12: Anklicken des ersten Bildes

Bei dieser „it“-Aufgabe wird das erste Bild per Xpath angeklickt. Dabei wird anhand der Baumstruktur über Knotenpunkte zum Zielelement navigiert. Die Anweisung „[1]“ spricht hierbei das erste Element des Containers an.

```
it('should insert a comment an click send button', function(){  
  element(by.css("input[id='chatInputId']")).sendKeys("automatic comment");  
  element(by.id("commentButtonId")).click();  
})
```

Abbildung 13: Kommentar einfügen und absenden

Anschließend wird der Kommentar in das Kommentarfeld eingefügt und abgesendet, indem der dementsprechende Button angeklickt wird.

```
it('should click logout button', function(){  
  element(by.linkText("Log Out")).click();  
})
```

Abbildung 14: Logout

Als letzter Schritt wird nun der Logout per Klick auf den Logout-Button der Navigationsleiste durchgeführt.

5.4 Navigation Timing API

Als zweite Testumgebung haben wir die Navigation Timing API eingesetzt, welche mit Java benutzt wird. Dabei kommt ebenfalls ein Seleniumserver zum Einsatz, welcher den Browser simuliert. Die Syntax weicht daher nicht sehr von Protractor ab. Die Zeitmessungen werden mit Hilfe von JavaScripts durchgeführt. Als Imports für Java brauchen wir dabei verschiedene Selenium-Pakete:

- org.openqa.selenium.By
- org.openqa.selenium.JavaScriptExecutor
- org.openqa.selenium.WebDriver
- org.openqa.selenium.chrome.ChromeDriver

Der Aufbau dieses Tests unterscheidet sich kaum von einem Protractortest. Am auffälligsten ist dabei, dass die Aufteilung in verschiedene Aufgaben wegfällt und die Methoden zur Zeitmessung aufgerufen werden müssen.

Zu Beginn muss in der Main-Function allerdings der Treiber für den Browser festgelegt werden und die Objekte für den Webdriver und den JavaScript-Executor erstellt werden.

```
System.setProperty("webdriver.chrome.driver",  
    "C://Users/Nathalie/Desktop/AIN4/Semesterprojekt Angular Universal/Selenium_javafiles/chromedriver.exe");  
WebDriver driver = new ChromeDriver();  
JavascriptExecutor js = (JavascriptExecutor) driver;
```

Abbildung 15: Festlegen des Treibers

Zudem werden die Methoden zur Bestimmung von der Seitenladezeit und der Renderzeit benötigt. Hierbei wird auf die Methode „Window.performance“ aus der Navigation Timing API zurückgegriffen. Diese gibt ein „Performance“-Objekt zurück, welches einen Zeitstempel und Navigationsattribute enthält.¹¹ Letztere geben beispielsweise an, wie viele Weiterleitungen nötig waren, um an eine Ressource heranzukommen.

¹¹ Vgl Performance Interface. In <https://developer.mozilla.org/en-US/docs/Web/API/Performance>, zugegriffen am 12.01.2018

```

static long pageLoadTime(JavascriptExecutor js) {
    long loadEventEnd = (Long) js.executeScript("return window.performance.timing.loadEventEnd;");
    long navigationStart = (Long) js.executeScript("return window.performance.timing.navigationStart;");
    return loadEventEnd - navigationStart;
}

static long pageProcessingTime(JavascriptExecutor js) {
    long domLoading = (Long) js.executeScript("return window.performance.timing.domLoading;");
    long domComplete = (Long) js.executeScript("return window.performance.timing.domComplete;");
    return domComplete - domLoading;
}

```

Abbildung 16: Methoden für Seitenladezeit und Processing-Zeit

Bei der Seitenladezeit prüft der Test die benötigte Dauer für die gesamte Ladezeit der Website. Dies entspricht dem kompletten Verlauf eines Seitenaufrufs, angefangen bei dem Aufruf der Adresse, bis hin zum Laden des letzten Events. Die Processing-Zeit prüft dagegen nur die benötigte Dauer für das Laden der Inhalte der Web Applikation.

Allerdings gibt es noch viele weitere Möglichkeiten der API, wie zum Beispiel das Auslesen der benötigten Zeit des Nachschlagens der Domain. In der folgenden Abbildung sind alle möglichen Zeitpunkte aufgelistet.

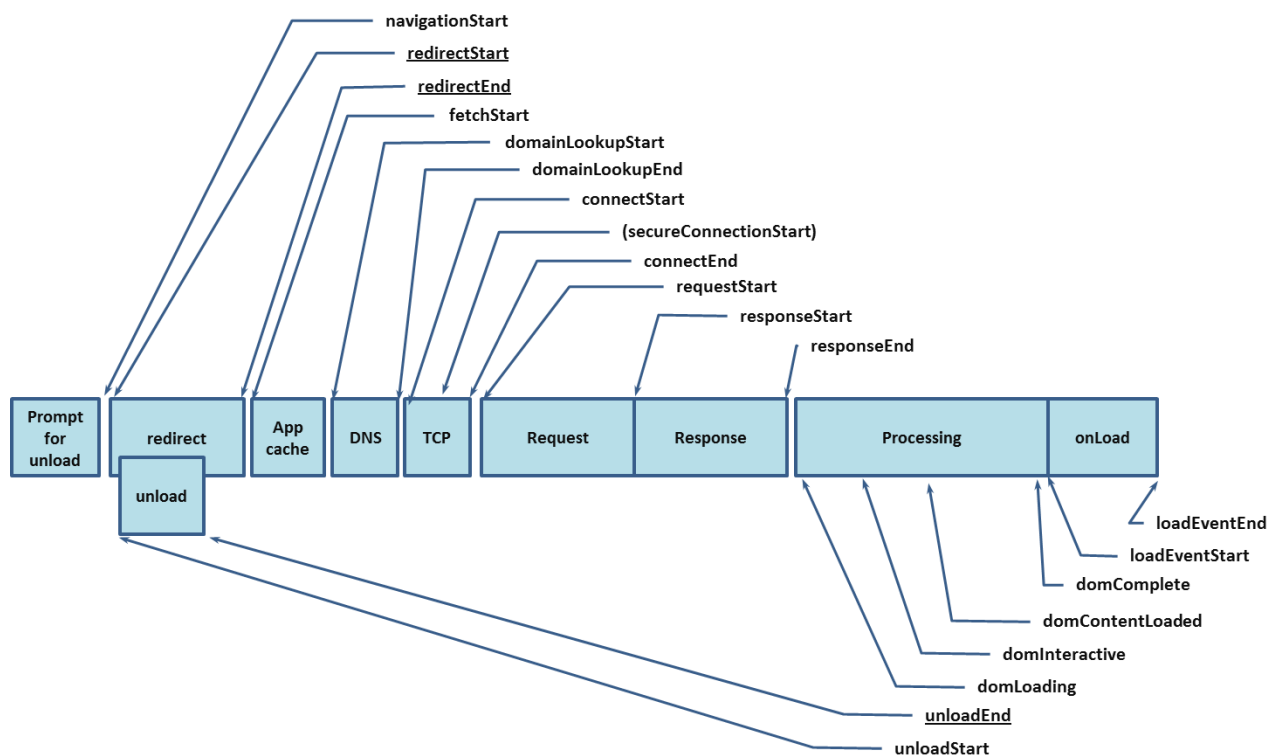


Abbildung 17: Übersicht der Attribute des Timing-Interfaces

Quelle: 5 Process. In <https://www.w3.org/TR/navigation-timing>. Eingesehen am 12.01.2018.

Syntaktisch unterscheidet sich der Aufruf von Elementen bei der Nutzung von Java kaum von Protractor, da beide Varianten mit der grundlegenden Syntax von Selenium arbeiten. Unterschiede bestehen durch das Ansprechen der Elemente über das Treiberobjekt und der Großschreibung von „By“, da dies in Java eine Klasse ist. Das folgende Beispiel zeigt die Unterschiede. Dies ist auf alle Codebeispiele des Protractor-Kapitels 5.3.1. `conf.js` anwendbar und entspricht damit dem in Java geschriebenen Test.

```
driver.findElement(By.LinkText("Log In")).click();
```

Abbildung 18: Syntax der API in Java

```
// click Login button
driver.findElement(By.LinkText("Log In")).click();
timePageLoadTotal += pageLoadTime(js);
timePageProcessingTotal += pageProcessingTime(js);
```

Abbildung 19: Aufruf der Methoden zur Zeitbestimmung

Die Methoden zur Bestimmung der Zeiten werden nach jeder Anweisung aufgerufen und auf den Gesamtzähler für den Testdurchlauf addiert.

5.5 Chrome Developer Tools

Um die *Performance* auch auf mobilen Geräten testen zu können, haben wir die „Chrome Developer Tools“ verwendet. Deren „Remote Debugging“ Funktion ermöglicht das Verbinden eines Android *Smartphones* mit dem Entwickler-Computer; über „port forwarding“ wird dann auf den lokalen NodeJS-Server zugegriffen. Das *Smartphone* wird dann zum gewünschten lokalen Server navigiert (Angular localhost:4200, Universal localhost:4000), wo die Performance-Tests der „Developer Tools“, über den Entwickler-Computer gestartet werden können. Das exakte Vorgehen ist in der Google-Dokumentation einsehbar:

<https://developers.google.com/web/tools/chrome-devtools/remote-debugging/>

(zugegriffen am 24.01.2017)

<https://developers.google.com/web/tools/chrome-devtools/remote-debugging/local-server> (zugegriffen am 24.01.2017)

Die Tests wurden mit einem Moto G4 Plus (Android 7.0) durchgeführt. Es wurden die Ladezeiten folgender Seiten von Angular, als auch von Universal gemessen: Front-Seite, Login-Seite, Upload-Seite, Comment-Seite. Dabei liefern uns die „Developer Tools“ folgende Zeitwerte:

- Loading
- Scripting
- Rendering
- Painting
- Other

Die mobilen Tests wurden nicht automatisiert; aber jeweils 10-mal wiederholt (die dokumentierten Zeiten entsprechen dem Durschnitt).

5.6 Auswertung der Testergebnisse

5.6.1 Navigation Timing API

Um aussagekräftige Ergebnisse zu erhalten, wurden die Tests mit jeweils 100 Iterationen durchgeführt.

```
Page Loading Time:  
Average time per iteration over 100 iterations: 3535ms.  
Page Processing Time:  
Average time per iteration over 100 iterations: 3369ms.
```

Abbildung 20: Ergebnisse von Angular

```
Page Loading Time:  
Average time per iteration over 100 iterations: 1215ms.  
Page Processing Time:  
Average time per iteration over 100 iterations: 971ms.
```

Abbildung 21: Ergebnisse von Angular Universal

Aus diesen Messwerten ist erkennbar, dass die durchschnittliche Loading und Processing Time bei Angular Universal etwa 34% der benötigten Zeit der normalen Angular Applikation entspricht.

Chrome Developer Tools Smartphone

Tabelle 1: Auswertung Developer Tools

Seiten	Loading	Scripting	Rendering	Painting	Other	Summe
Front	61,3A	4442,0A	114,6A	47,1A	4725,3A	9390,3A
	59,3U	1231,6U	132,1U	66,4U	1094,1U	2583,5U
Login	65,4A	3869,3A	24,1A	6,9A	4057,5A	8023,2A
	49,7U	894,0U	34,1U	5,5U	519,4U	1502,7U
Upload	53,0A	4045,5A	22,1A	3,8A	3940,9A	8065,3A
	50,8U	1049,6U	34,3U	6,3U	728,0U	1869,0U
Com- ment	42,8A	4376,3A	54,3A	50,9A	4165,3A	8689,6A
	50,6U	1317,5U	76,3U	31,7U	1025,0U	2501,1U

*A = Angular; U = Universal; Zeitangaben sind in Millisekunden.

Angular hat eine durchschnittliche Gesamtladezeit von 8542,1 Millisekunden. Damit entspricht die Ladezeit von Universal mit 2114,1 Millisekunden etwa 25% der von Angular benötigten Zeit. Die Performance-Gewinne kommen hauptsächlich von der deutlich geringeren Scripting- und Other- Zeit. Die anderen Werte weisen nur geringe Abweichungen auf und sind daher vernachlässigbar. Die Netzwerkbelastung ist für beide Varianten in etwa gleich.

6 Fazit

Angular ist ein vielseitig einsetzbares Framework, dessen Implementierung dank TypeScript sehr intuitiv ist. Gerade Entwickler mit „Java“ Erfahrung, wie es bei unseren Teammitgliedern der Fall ist, finden sich in dem sehr ähnlichen TypeScript schnell zurecht. Über npm stehen eine Vielzahl von Erweiterungen zu Verfügung, wie zum Beispiel die genannten Firebase-Erweiterungen, welche viel Arbeit bei der Implementierung abnehmen. Die Umstellung von Angular 4 auf Universal geht leicht von der Hand und ist in der Angular-Dokumentation ausführlich beschrieben. Es ist zudem bemerkenswert, dass keine Änderungen am eigentlichen Quellcode nötig sind, sondern nur einige Anpassungen an der Konfiguration.

Bis auf Protractor gibt es keine gängigen Testtools, die speziell auf Angular zugeschnitten sind und gleichzeitig automatisierte Tests bieten. Protractor hat allerdings keine Möglichkeit um präzise Performancetests auszuführen und ist daher für dieses Projekt ungeeignet. Für Webanwendungen im Allgemeinen gibt es allerdings einige Open Source Tools, mit denen dies möglich ist. Nach gründlicher Recherche ist die Entscheidung für die Navigation Timing API in Kombination mit Selenium gefallen. Damit konnte ein sehr übersichtlicher und unkomplizierter Test geschrieben werden, welcher zuverlässig funktioniert. Ursprünglich war die Verwendung von den Chrome Dev Tools geplant, welche allerdings aufgrund fehlender Automatisierungsmöglichkeiten verworfen wurde. Daher wurde damit nur der Test für mobile Endgeräte durchgeführt.

Die Erwartung, mit Angular Universal einen Performancegewinn zu erzielen, wurde erfüllt. Die Gesamtladezeit von Universal entspricht etwa einem Drittel der Zeit, welche das normale Angular für den Ladevorgang benötigt. Der Test von Angular und Universal auf mobilen Endgeräten hat ergeben, dass dort ein noch höherer Performancegewinn besteht. Die Ladezeit von Universal entspricht dabei etwa einem Viertel der für das normale Angular benötigten Zeit. Die Performancegewinne sind allerdings auf Kosten von Serverlast, da die Verarbeitung von JavaScript dort zentral abläuft.

Literaturverzeichnis


- 1 Vgl. Angular Universal. In <https://universal.angular.io/>, zugegriffen am 19.01.2018
- 2 Vgl. Architecture Overview. In <https://angular.io/guide/architecture>, zugegriffen am 18.12.2017
- 3 Vgl. Angular Universal: server-side rendering. In <https://angular.io/guide/universal>, zugegriffen am 18.12.2017
- 4 Vgl. Angular Universal: Modify the Client App. In <https://angular.io/guide/universal#transition> zugegriffen am 31.12.2017
- 5 Vgl. Setting up Visual Studio Code. In <https://code.visualstudio.com/docs/setup/setup-overview>, zugegriffen am 18.12.2017.
- 6 Vgl. Hello World. In <https://guides.github.com/activities/hello-world/>, zugegriffen am 18.12.2017
- 7 Vgl. Add Firebase to your JavaScript Project. In <https://firebase.google.com/docs/web/setup>, zugegriffen am 18.12.2017.
- 8 Vgl. Angular Universal Integration. In <https://github.com/angular/angular-cli/wiki/stories-universal-rendering>, zugegriffen am 19.01.2017.
- 9 Vgl. Navigation Timing API. In https://developer.mozilla.org/en-US/docs/Web/API/Navigation_timing_API, zugegriffen am 12.01.2018.
- 10 Vgl. Setup. In <http://www.protractortest.org/#/>. Eingesehen am 18.12.2018.
- 11 Vgl Performance Interface. In <https://developer.mozilla.org/en-US/docs/Web/API/Performance>, zugegriffen am 12.01.2018

Eidesstattliche Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbständig verfasst und hierzu keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Stellen der Arbeit die wörtlich oder sinngemäß aus fremden Quellen entnommen wurden, sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt oder an anderer Stelle veröffentlicht.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.



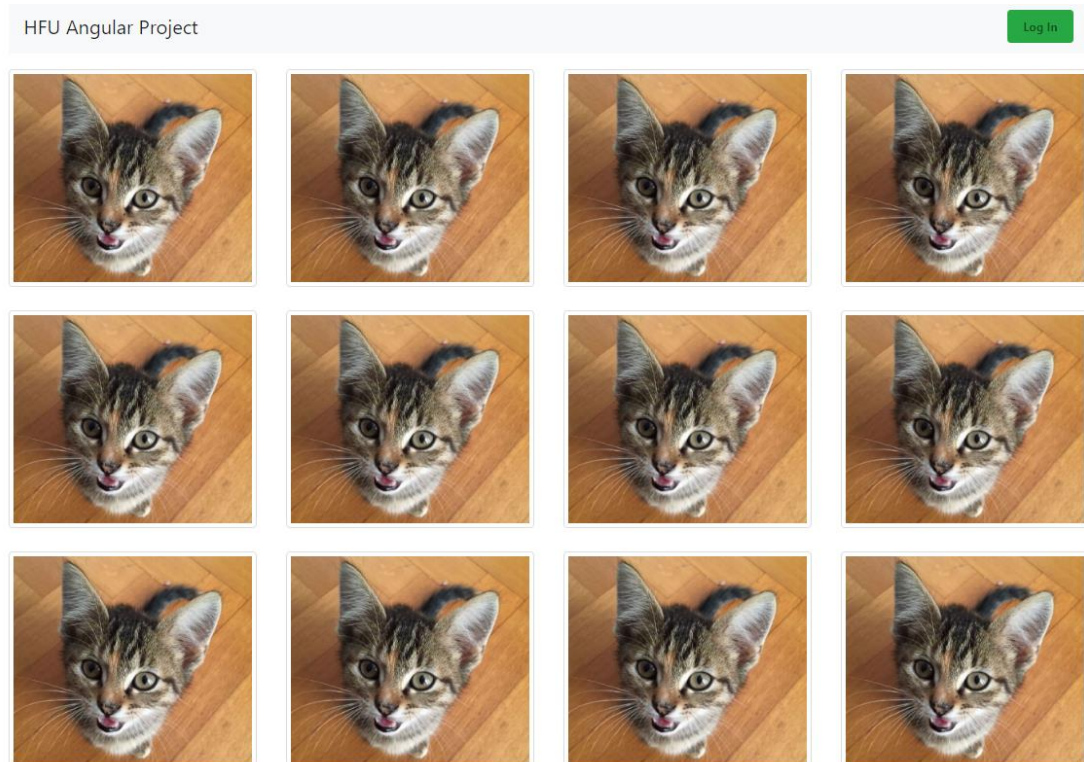
Furtwangen, 26.01.2018, Gießler Nathalie



Furtwangen, 26.01.2018, Keßler Jan

A. [Anhang]

Front Page



Log In

The screenshot shows the 'Log In' form of the 'HFU Angular Project'. The header bar at the top contains the text 'HFU Angular Project' and a green 'Log In' button. Below the header, the form is titled 'Log In' in a light gray box. The form contains two input fields: 'Enter email' and 'Enter password'. Below these fields is a blue 'Go!' button. The form is set against a light gray background.

Upload

HFU Angular Project	User Gallery	Upload	Log Out
---------------------	--------------	--------	---------

Upload New Images

Dateien auswählen Keine ausgewählt

Upload Images

Comments

nathalie.giessler@hs-furtwangen.de
automatic comment nr. 15
2018/1/26 12:22:2

nathalie.giessler@hs-furtwangen.de
automatic comment nr. 16
2018/1/26 12:22:13

nathalie.giessler@hs-furtwangen.de
automatic comment nr. 17
2018/1/26 12:22:23

nathalie.giessler@hs-furtwangen.de
automatic comment nr. 18
2018/1/26 12:22:33