Abschlussprüfung Sommer 2014

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Right Livelihood App

Mobilanwendung für den Right Livelihood Award

Prüfungsbewerber:
Thomas Stern
Alte Darmstädter Str. 4
64367 Mühltal

Ausbildungsbetrieb:
ion2s GmbH
Donnersbergring 16
64295 Darmstadt

Inhaltsverzeichnis

1 Einleitung

1.1 Projektbeschreibung

Als Unterstützung für die gemeinnützige Stiftung Right Livelihood[1] hat mein Ausbildungsbetrieb ion2s GmbH[2] eine Webseite erstellt. Die Mobilanwendung soll diese Webseite ergänzen. Die Idee für dieses Projekt kommt von meinem Betrieb. Neben der Unterstützung für Right Livelihood ist das Ziel das Wissen im Bereich der mobilen Geräte intern im Betrieb zu vertiefen bzw. zu erweitern.

In der Mobilanwendung sollen die Preisträger des Right Livelihood Awards, die sogenannten Laureaten, multimedial präsentiert. Um dieses Ziel zu erreichen sollen moderne Webtechnologien wie JavaScript, HTML5 und CSS3 clientseitig verwendet werden. Der Fokus dieser Mobilanwendung liegt dementsprechend auf dem Client. Serverseitig sollen die statischen Resourcen bereitgestellt und eine Datenbank angelegt werden, die die Daten der Preisträger beinhaltet. Der Server soll mit NodeJS[3] realisiert werden. Der Vorteil ist, dass sowohl Client als auch Server in der Programmiersprache JavaScript entwickelt werden.

Ein Konzept für die Gestaltung existiert bereits. Die technische Umsetzung des kompletten Konzepts würde den zeitlichen Rahmen sprengen und deswegen sind folgende fünf Teile der Mobilanwendung geplant:

1. Intro - Das Intro wird ein Ladebildschirm, welches der Mobilanwendung die Zeit gibt die benötigten Daten der Laureaten herunterzuladen. Nachdem Ladevorgang wird zur entsprechenden Darstellungs Ansicht gewechselt.

2. Liste-Ansicht - Die Liste wird scrollbar sein. Jedes Listenelement beinhaltet den Namen des Laureaten und ein Zitat.

3. Bilderwand-Ansicht - In der Bilderwand-Ansicht werden Bilder aller Preisträger zufällig positioniert. Der Benutzer kann die Bilderwand vertikal bewegen und die einzelnen Bilder vergrößern.

4. Weltkarte-Ansicht - Auf einer Weltkarte werden die Laureaten ihrem Herkunftsland entsprechend markiert. Mit einem Klick auf die Weltkarte kann der Benutzer näher ranzoomen.

[1] http://www.rightlivelihood.org/
[2] http://www.ion2s.com/
[3] http://nodejs.org/

5. Detail-Ansicht - Hier werden detaillierte Informationen zu den einzelnen Laureaten dargestellt.

1.2 Zeitplan

Die gesamte Dauer dieses Projektes sollte 70 Stunden betragen. Sobald das technische Konzept steht wird mit der Umsetzung der Mobilanwendung begonnen. Die Umsetzung für den Client ist in folgende Unteraufgaben gegliedert:

1. Struktur
    - Verzeichnisse anlegen
    - Applikation Initiliasierung
    - Routing implementieren
2. Services
    - REST Schnittstelle
    - Lokale Speicherung
3. Controllers
    - Intro
    - Liste
    - Bilderwand
    - Weltkarte
    - Detail

Die Umsetzung für den Server ist in folgende Unteraufgaben gegliedert:

1. Struktur
    - Verzeichnisse anlegen
    - Server Initialisierung
    - Verteilung der Anfragen
2. Module

- REST Schnittstelle
- Datenbankanbindung

Das Schreiben und Ausführen von Tests ist Bestandteil der Umsetzung. Dennoch sollen nach abgeschlossener Umsetzung ausführliche Tests durchgeführt werden. Unter anderem sollen Personen, die nicht direkt am Projekt beteiligt sind, die Anwendung testen um so ein besseres und größeres Feedback zu erhalten.

Aus diesen Angaben ergibt sich dieser Zeitplan:

<table>
<tr><td>Phase</td><td>Beschreibung</td><td>Stunden</td></tr>
<tr><td>I.</td><td>Technisches Konzept</td><td>10</td></tr>
<tr><td>II.</td><td>Umsetzung
<table><tr><th>Art der Umsetzung</th><th>Teilstunden</th></tr><tr><td>Struktur</td><td>1</td></tr><tr><td>Services</td><td>2</td></tr><tr><td>Controllers</td><td>25</td></tr><tr><td>Client</td><td>28</td></tr><tr><td>Struktur</td><td>2</td></tr><tr><td>Module</td><td>5</td></tr><tr><td>Server</td><td>7</td></tr></table></td><td>35</td></tr>
<tr><td>III.</td><td>Testdurchführung</td><td>15</td></tr>
<tr><td>IV.</td><td>Dokumentation / Abschluss</td><td>10</td></tr>
</table>

2 Technisches Konzept

2.1 Anforderung

Die Mobilanwendung für den Right Livelihood Award soll die Preisträger multimedial darstellen. Zu diesem Zweck soll es folgende Darstellungsformate geben:

- Listendarstellung aller Laureaten
- Bilderwand mit Bildern aller Laureaten
- Weltkarte mit Markierungen zu den Herkunftsorten aller Laureaten
- Detailansicht eines einzelnen Laureaten

Weitere Anforderungen sind Funktionsfähigkeit auf mobilen Geräten (iOS- bzw. Android-System) sowie Desktop-Rechnern und der Betrieb im offline Modus.

2.2 Realisierung

2.2.1 Client

Um eine ansprechende Darstellung und ein angenehmes Benutzergefühl realisieren zu können sollen folgende Webtechnologien eingesetzt werden:

- JavaScript
- HTML5
- CSS3

Auf Grund der genannten Technologien, die zum Einsatz kommen, sollen nur moderne Browser unterstützt werden, die diese Technologien beherrschen.

Zur Untersützung bei der Entwicklung soll das JavaScript Framework AngularJS[4] zum Einsatz kommen. Für eine hohe Qualität der Anwendung soll das *Scaffolding Tool* Yeoman[5] sorgen. Das Tool selbst ist ein Generator, welches ein Grundgerüst für die Anwendung bereitstellt. Mit Yeoman kommt das *JavaScript*

[4] https://angularjs.org/
[5] http://yeoman.io/

Task Runner Tool Grunt[6] und das *Package Manager Tool* Bower[7] als Unterstützung. Mit Grunt lassen sich Aufgaben, wie z. B. automatische Ausführung von Tests bei jeder Änderung im Quellcode, beschreiben und ausführen. Bower ist nützlich um externe Bibliotheken in das bestehende Projekt einzubinden, weil es automatisch vorhandene Abhängigkeiten externer Bibliotheken auflöst.

Damit die Mobilanwendung auch im offline Modus funktioniert sollen die Daten der Laureaten mit Hilfe der HTML5 Besonderheit Local Storage lokal auf dem entsprechenden Gerät gespeichert werden. Außerdem müssen Bilder Base64-codiert in den Local Storage abgespeichert werden damit diese auch im offline Modus verfügbar sind. Dies bedeutet einen höheren Aufwand der Implementierung, aber bewirkt neben der Verfügbarkeit im offline Modus einen deutlichen Performancevorteil beim wiederholten Laden.

2.2.2 Server

Der Server soll neben der Bereitstellung der statischen Dateien auch eine aktuelle Versionsnummer und Informationen zu allen Laureaten liefern. Für diese Aufgaben sollen folgende Technologien eingesetzt werden:

- NodeJS
- SQLite[8]

NodeJS ist eine serverseitige JavaScript Plattform. Der Vorteil dieser Entscheidung ist, dass sowohl Client als auch Server in der Skriptsprache JavaScript geschrieben werden. Zur Unterstützung bei der Implementierung des Servers soll das NodeJS Framework ExpressJS[9] eingesetzt werden. Als Datenbank wird SQLite verwendet, weil an die Datenbank keine größeren Anforderungen gestellt werden als die Daten der Laureaten zu liefern.

2.2.3 Test

Es werden Unittests geschrieben, die während der Entwicklung automatisch bei Änderungen im Quellcode ausgeführt werden sollen. Zum Testen des Clients als

[6] http://gruntjs.com/
[7] http://bower.io/
[8] https://sqlite.org/
[9] http://expressjs.com/

auch des Servers soll das Framework Jasmine 2.0[10] verwendet werden. Die beschriebenen Tests behandeln nur die Geschäftslogik der Mobilanwendung. Manipulationen am *Document Object Model* werden vom JavaScript Framework AngularJS übernommen und müssen nicht getestet werden.

2.2.4 Kommunikation

Die Übertragung der Daten soll ausschließlich im JSON[11] Format stattfinden. Als Schnittstelle zwischen Client und Server soll das Programmierparadigma REST (Representational State Transfer)[12] verwendet werden. Diese Entscheidung ermöglicht eine gute Erweiterung der Mobilanwendung zu einem späteren Zeitpunkt. Es sollen nur folgende zwei REST URI implementiert werden:

- /laureates [GET] - Liefert eine Liste von Daten aller Laureaten.
- /version [GET] - Liefert die aktuellste Versionsnummer.

2.2.5 Versionierung

Zur Versionierung der Mobilanwendung soll das Versionsverwaltungssystem Git[13] zum Einsatz kommen. Das Versionsverwaltungssystem ist weit verbreitet und es gibt viele Plug Ins, die die Arbeit mit Git vereinfachen. Das Projekt soll außerdem auf GitHub[14] hochgeladen werden. So ist es möglich das andere Projektmitglieder relativ leicht sich an dem Projekt beteiligen können. Außerdem bietet GitHub mehrere visuelle Darstellung des Projektverlaufes an, so dass es möglich ist den Projektfortschritt einfach zu erkennen.

[10] http://jasmine.github.io/2.0/introduction.html
[11] http://www.json.org/
[12] http://en.wikipedia.org/wiki/Representational_state_transfer
[13] http://git-scm.com/
[14] https://github.com/

3 Gestalterisches Konzept

Das gestalterische Konzept wurde nicht von mir erstellt. Es besteht hauptsächlich aus Abläufen und Bildern, die illustrieren sollen wie die Mobilanwendung aussehen und zu bedienen ist. Ich hab mich bei der Umsetzung der Mobilanwendung mehr oder weniger an diesen Vorgaben orientiert. Eine exakte Umsetzung der Gestaltung war zum Teil aus technischen und zeitlichen Gründen nicht möglich. Außerdem wurden aus zeitlichen Gründen nicht alle Ansichten und Funktionen übernommen.

3.1 Bilderwand-Ansicht

Abbildung 1: Vergleich von Gestaltung (links) und Umsetzung (rechts) der Bilderwand-Ansicht.

Bei der Aneinanderreihung der Bilder sollte kein statischer Eindruck entstehen. Außerdem sollte sich der Benutzer frei durch die Bilderwand bewegen können. Da ich Probleme mit der Umsetzung der Gesten-Steuerung hatte habe ich mich entschieden die Bilder vertikal aneinander zu reihen und somit nur ein vertikales Scrollen dem Benutzer zu ermöglichen. Auch ein freies Zoomen ist nicht umgesetzt. Dagegen kann der Benutzer auf ein Bild klicken das vergrößert und entsprechend positioniert wird das der Benutzer das ganze Bild sehen kann. Bei einem vergrößerten Bild werden zusätzlich ein paar Informationen zu dem Laureaten angezeigt. Ein Klick auf diese Zusatzinformationen führt den Benutzer zur Info-Ansicht des Laureaten.

3.2 Liste-Ansicht

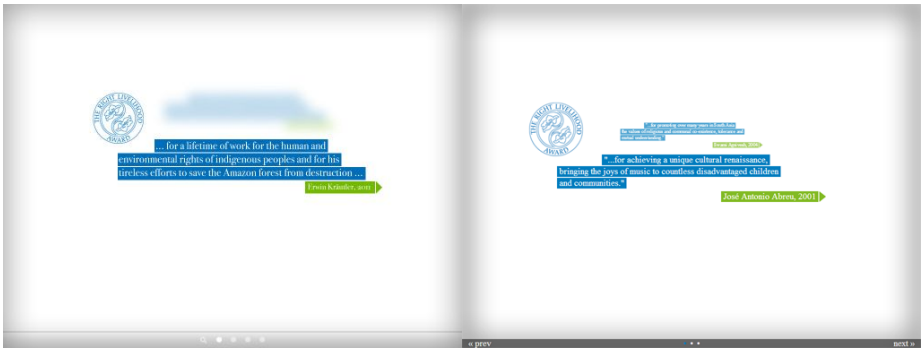

Abbildung 2: Vergleich von Gestaltung (links) und Umsetzung (rechts) der Liste-Ansicht.

In der Gestaltung sollen weiter entfernte Objekte verschwommen dargestellt werden. Dieser Effekt ist technisch schwer umzusetzen und ist in meiner Umsetzung nicht vorhanden.

3.3 Weltkarte-Ansicht

Abbildung 3: Vergleich von Gestaltung (links) und Umsetzung (rechts) der Weltkarte-Ansicht.

Auch bei der Weltkarte sollte ein freies Bewegen und Zoomen möglich sein. Auf Grund ähnlicher Probleme wie bei der Bilderwand habe ich mich entschlossen das der Benutzer nur in bestimmte Bereiche der Weltkarte zoomen kann. Durch

einen Klick auf die Weltkarte wird auf diesen Punkt herangezoomt und Laureaten, die sich in diesem Bereich befinden werden dargestellt. Ein Freies Bewegen der Weltkarte ist nicht möglich. Der Benutzer muss erneut auf die Weltkarte klicken um wieder die komplette Weltkarte zu sehen. Klickt der Benutzer im gezoomten Zustand auf einen Laureaten wird er zur Info-Ansicht geführt.

3.4 Info-Ansicht

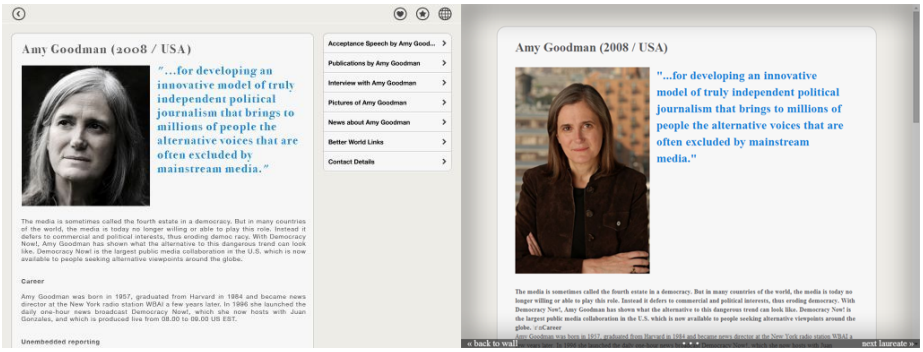

Abbildung 4: Vergleich von Gestaltung (links) und Umsetzung (rechts) der Info-Ansicht.

Bei der Info-Ansicht hab ich mir die Freiheit genommen die weiterführenden Links des Laureaten nach unten zu platzieren.

4 Durchführung

Das komplette Projekt wurde auf GitHub[15] hochgeladen und kann dort eingesehen werden.

4.1 Die Entwicklungsumgebung

Die Mobilanwendung wurde auf einem Rechner mit dem Betriebssystem Windows 7 entwickelt. Als Entwicklungsumgebung wurde das Programm WebStorm[16] verwendet. Damit die Werkzeuge, die zur Entwicklung benötigt werden, funktionieren muss NodeJS auf dem Rechner installiert sein. NodeJS wird zum einen für die client-seitigen Entwicklungswerkzeuge benötigt und zum anderen für die Implementierung des Servers verwendet. Um die entsprechenden Konsolenbefehle auszuführen wurde die Git Bash verwendet.

4.2 Projektstruktur

Es existiert ein Projekt, das drei Unterprojekte beinhaltet. Im Unterprojekt *client* befinden sich alle Dateien, die für die Client-Entwicklung eine Rolle spielen. Dokumentation und Material, wie z. B. Diagramme und Bilder, das für die Dokumentation verwendet werden befinden sich im Unterprojekt *docs*. Das letzte Unterprojekt ist der *server*. Hier liegen die statischen Ordner, die Server JavaScript Datei und die Datenbank ab.

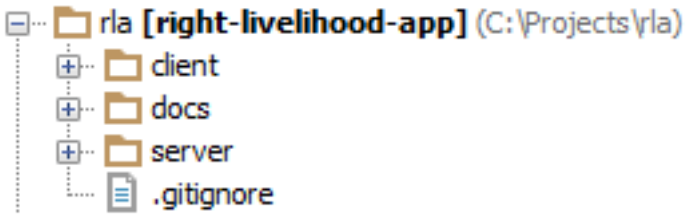

Abbildung 5: Projektstuktur der Mobilanwendung.

[15] https://github.com/tstern/rla
[16] http://www.jetbrains.com/webstorm/

4.2.1 client

Für Grunt werden die beiden Dateien *Gruntfile.js* und *package.json* benötigt. In der *Gruntfile.js* werden Prozesse definiert, die bei der Entwicklung des Projektes als Befehle ausgeführt werden können. Die wichtigsten Befehle für die Ausführung der Prozesse sind *serve*, *test* und *build*.

serve Dieser Befehl startet einen simplen Webserver und öffnet die Webanwendung automatisch im Standardbrowser. Außerdem beobachtet dieser Prozess bestimmte Dateien. Gespeicherte Änderungen an diesen Dateien werden dann erkannt und es werden weitere Befehle ausgeführt. Unter anderem wird der Befehl *test* ausgeführt, das dem Entwickler direkt mitteilt, wenn ein Test fehlschlägt. Wurden alle weiteren Prozesse durchgeführt wird automatisch die Webanwendung im Browser aktualisiert.

test Dieser Befehl führt die selbstgeschriebenen Unit Tests aus. In diesem Projekt wird die Testumgebung *karma* verwendet, die einen vorgegebenen Browser öffnet in dem die Tests schließlich laufen.

build Dieser Befehl soll am Ende eine Version der Anwendung liefern, die im Produktionsmodus zum Einsatz kommt. Dazu werden Veresserungen am Quellcode vorgenommen, wie z. B. Code-Komprimierung und es werden mehrere Dateien in einer einzelnen Datei zusammen gefasst. Das Resultat dieses Prozesses wird in den statischen *public* Ordner des Servers kopiert.

In der *package.json* werden Abhängigkeiten eingetragen, die für die Client-Entwicklung existieren. Da es nur Abhängigkeiten für die Grunt Prozesse gibt stehen alle Abhängigkeiten im Feld *devDependencies*. Zusätzlich gibt es die Fehler *name* und *version*. Diese Felder sind nur sinnvoll, wenn das Projekt in anderen Projekt verwendet werden soll. Dies ist hier nicht der Fall. Die *package.json* ist sinnvoll für neue Projektmitglieder. Nachdem diese das Projekt mit Git auf ihren Rechner kopiert haben müssen diese nur noch den Befehl *npm install* in ihre Konsole eingeben und es werden automatisch alle Abhängigkeiten installiert.

Für Bower existiert die Datei *bower.json*, welche ähnlich aufgebaut ist wie die *package.json*. Der Unterschied ist das in der *bower.json* Abhängigkeiten stehen, die die fertig Anwendung besitzt, z. B. das JavaScript Framework Angular JS.

Für Karma existieren zwei Konfigurationsdateien. Die erste ist *karma.conf.js* und die zweite ist *karma-e2e.conf.js*. In der ersten Dateien stehen Einstellungen für die Umgebung in denen die Unit Tests laufen sollen. So lässt sich z. B. der Browser bestimmen. In diesem Projekt wird *PhantomJS* verwendet. Die zweite Datei wird für Einstellungen benötigt wenn Integrationstests durchgeführt werden sollen. Integrationstests gibt es in diesem Projekt nicht.

Git benutzt zwei Dateien, zum einen *.gitignore* und zum anderen *.gitattributes*. In der ersten können Dateien und Verzeichnisse von der Versionierung ausgeschlossen werden. Die zweite Datei gibt dem Pfad in dem sich die Datei befindet spezielle Attribute. In diesem Unterprojekt gibt es nur das Attribute `* text=auto`, welches bewirkt das Zeilenumbrüche normalisiert werden.

Desweiteren existieren noch verschiedene Ordner die kurz erklärt werden:

.tmp Dieser Ordner dient nur als Zwischenspeicher für verschiedene Gruntprozesse.

app In diesem Ordner befindet sich der Quellcode und die Abhängigkeiten, die in der *bower.json* deklariert sind.

dist Dies ist das Zielverzeichnis des Grunt Prozesses *build*.

node_modules Abhängigkeiten die in der *package.json* angegeben sind werden hier installiert.

scripts Beinhaltet Skriptdateien, die bestimmte Grunt Prozesse in einem eigenem Konsolenfenster starten.

test Testdateien liegen in diesem Ordner. Besitzt die gleiche Verzeichnisstruktur wie *app*.

4.2.2 docs

In diesem Unterprojekt liegt die Dokumentation ab. Außerdem werden hier Bilder und Diagramme abgelegt, die für die Dokumentation verwendet werden. Die Dokumentation ist mit LyX[17] geschrieben. Das Diagramm für den Ladeprozess ist mit Microsoft Visio[18] gemacht.

[17] http://www.lyx.org/
[18] http://office.microsoft.com/de-de/visio/

4.2.3 server

Es existieren zwei wichtige Dateien in diesem Unterprojekt. Erste der beiden Dateien ist die Datenbank *rla.s3db*. Hier werden alle Informationen über die Laureaten abgespeichert. Die zweite Datei ist *server.js*, welche den Server initialisiert. Desweiteren existieren die zwei statischen Ordner *laureates_images* und *public*. Im Ordner *laureates_images* liegen alle Bilder der Laureaten ab. Im zweiten statischen Ordner *public* liegen die Dateien ab, die dem Clien geschickt werden. Im letzten Ordner *node_modules* liegen die Projekte ab, die als Abhängigkeit für den Server deklariert sind.

4.3 AngularJS

AngularJS bietet verschiedene Funktionalitäten. In diesem Projekt werden die Funktionalitäten *Controller*, *Directive* und *Service* verwendet. Ein *Controller* definiert den Bereich für den *$scope*. Ein *$scope* erbt durch das *prototyping* in JavaScript vom *$rootScope* bzw. vom übergeordneten *$scope*. Das *$scope* Objekt dient als Schnittstelle zwischen Geschäftslogik und der Ansicht. Eine *Directive* existiert nur in der Ansicht und erweitert die Funktion und Aufgaben von HTML. So lässt sich das Verhalten von Elementen im *Document Object Model* bestimmen. *Service* sind Singletons, die es ermöglichen Logik auszulagern. In AngularJS gibt es außerdem *Injection*. Damit lassen sich verschiedene AngularJS Funktionalitäten als Parameter in andere AngularJS Funktionalitäten injizieren.

4.4 Umsetzung

4.4.1 Controller

Für jede Ansicht existiert ein *Controller*. Die Aufgabe des *Controller* ist es das Array von Laureaten so anzupassen, das diese in der Ansicht korrekt dargestellt werden.

- ListCtrl: (Liste-Ansicht) Teilt die Zitate der Laureaten in einzelne Zeilen.
- WallCtrl: (Bilderwand-Ansicht) Extrahiert Bilddaten und benötigte Informationen aus dem Laureaten Objekt.

- MapCtrl: (Weltkarte-Ansicht) Liest die Koordinaten der Laureaten aus.
- InfoCtrl: (Info-Ansicht) Sucht anhand einer id den darzustellenden Laureaten.

Außerdem existiert ein *Controller* mit dem Namen IntroCtrl für die Intro-Ansicht und ein *Controller* mit dem Namen NavCtrl für die Navigation und Navigationsleiste.

4.4.2 Service

In diesem Projekt gibt es drei *Services* HelperService, RESTService und ResourceService. Der HelperService ist eine Sammlung von Hilfsfunktionen. Im RESTService werden die beiden Requests an die REST-Schnittstelle des Servers definiert. Diese werden im ResourceService verwendet. Deswegen wird der RESTService in den ResourceService injiziert. Der ResourceService steuert den Ablauf wie die Laureaten geladen werden und mit Hilfe von Events wird der Zustand des Ladeprozesses geteilt.

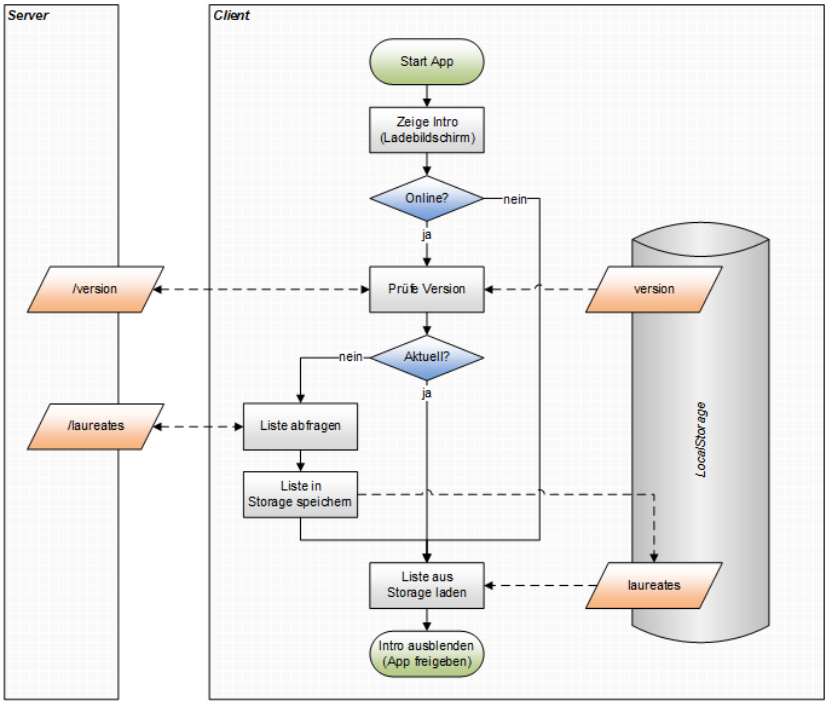

Abbildung 6: Ablaufdiagramm des initialen Ladeprozesses.

4.4.3 Directive

Jede *Directive* hat ts als Prefix. Diese Namenskonvention soll einen Konflikt mit einer anderen *Directive* aus einer externen Quelle verhindern. Folgend wird jede *Directive*, die in diesem Projekt entwickelt wurde, erläutert.

ts-slider Diese *Directive* ist in jeder der vier Ansichten im Einsatz. Mit ihr wird das Navigieren per Wischfunktion realisiert. Mit der Hilfsfunktion `isHorizonzal(deltaX, deltaY)` wird ein horizontales Wischen registriert. Erfolgt das Wischen innerhalb eines vorgegeben Zeitfenster wird in die entsprechende Richtung navigiert. Welche Ansicht rechts bzw. links von der momentanen Ansicht befindet wird im NavCtrl festgelegt. Die Animation, das die momentane Ansicht zur Seite bewegt wird, ist mit JavaScript umgesetzt. Eine Callback-Funktion, die die Berechnungen für die Animation durchführt, wird der Funktion `requestAnimationFrame` als Parameter übergeben. Bevor der Browser einen neuen Frame render (zeichnet) werden die Callbacks, die mit der Funktion `requestAnimationFrame` regristiert wurden, aufgerufen. Die Berechnungen erfolgen somit synchron mit dem Renderer des Browsers. Die Vorteile sind eine bessere Performance der Animation und wenn der Browser im Hintergrund läuft pausiert der Renderer und somit auch die Animation.

ts-scroller Diese *Directive* wird nur in der Liste-Ansicht verwendet. Die Liste wird scrollbar und reagiert auf jede vertikale Touch-Bewegung. Jedes Listenelement besitzt dazu folgende vier Attribute. Eine echte und falsche Position, ein Skalierungs- und ein Verblassungswert. Die falsche Position, der Skalierungs- und Verblassungswert werden durch Algorithmen durch die echte Position, die als Parameter übergeben wird, ermittelt. Diese Berechnungen werden bei jeder Bewegung ausgeführt. Wird die Touch-Bewegung beendet werden die sichtbaren Listenelemente auf bestimmte Position weiter bewegt. Dazu wird das Listenelement identifiziert, das sich am nähesten der echten Position 0 befindet. Die vorhandene Differenz wird mit Hilfe der `requestAnimationFrame` Funktion weiter animiert. Sollte die Geschwindigkeit der Touch-Bewegung einen bestimmten Wert überschreiten wird ein Abklingen simuliert. dazu wird zusätzlich mit jedem Frame die Geschwindigkeit etwas reduziert. Folgend stehen die Algorithmen für die Berechnungen:

Algorithmus für die falsche Position:

```
function calculateY(delta) {
  return (Math.sqrt(-delta + deltaSteps)
    - Math.sqrt(deltaSteps)) * -20;
}
```

Algorithmus für die Skalierung:

```
function calculateScale(delta) {
  var scale = delta / -deltaMinVisible + 1;

  if (scale < 0) {
    scale = 0;
  } else if (scale > 1) {
    scale = scale * scale;
  }

  return scale;
}
```

Algorithmus für den Verblassungswert:

```
function calculateOpacity(delta) {
  var opacity = 1;

  if (delta === 0) {
    return opacity;
  }

  if (delta < 0) {
    opacity = 1 + ((delta + deltaSteps / 2)
      / (deltaSteps * 1.5));
  } else {
    opacity = 1 - (delta / deltaSteps);
  }

  return opacity > 1 ? 1 : opacity < 0 ? 0 : opacity;
}
```

ts-wall-image Diese *Directive* kommt nur in der Bilderwand-Ansicht zum Einsatz und wird jedem Bild-Element hinzugefügt. Als erstes wird anhand der Kategorie, in welches das Bild vorher zugeordnet wurde, eine feste Höhe gegeben. Die Differenz zwischen fester Höhe und tatsächlicher Höhe wird als CSS-Attribute *margin* dem Bild-Element hinzugefügt. Dadurch sind die Bilder perfekt in einer Reihe, aber durch das CSS-Attribut *margin* sind die einzelnen Bilder leicht versetzt. Außerdem wird ein *Click-Event* registriert. Bei einem Klick auf das Bild wird dies hervorgehoben und Informationen zum Laureaten werden eingeblendet. Sollte ein anderes Bild bereits aktiv sind wird dies automatisch zurückgesetzt.

ts-map-zoom Diese *Directive* kommt nur in der Weltkarte-Ansicht zum Einsatz. Hier wird ein *Click-Event* registriert. Sobald ein Benutzer auf die Weltkarte klickt wird in dieser Bereich vergrößert und Laureaten, die sich in diesem Bereich befinden, werden dargestellt. Ein erneuter Klick verkleinert die Ansicht wieder auf die komplette Weltkarte.

ts-wave Auch diese *Directive* kommt nur in der Weltkarte-Ansicht zum Einsatz. Die Aufgabe dieser *Directive* ist es, dass sobald die Weltkarte geöffnet wird, eine Wellenanimation das komplette Wasser im Hintergrund befüllt. Eine mögliche Umsetzung einer solchen Animation habe ich auf http://codepen.io/miyovanstenis/pen/BfGeK gefunden. Die Idee dahinter ist es den Hintergrund in mehrere gleichgroße Streifen zu unterteilen. Jeder Streifen bekommt die gleiche Animation, aber mit immer größerer Verzögerung (Animation-Delay). Dadurch entsteht der Welleneffekt.

4.5 Test

Es wurden nur Unit Tests für die Controller und Services geschrieben. Für den Aufbau der Tests wird das Framework Jasmine verwendet.

Controller Unit Test Es wird hier geprüft ob die *Controller* die ihr übergebenes Array von Laureaten korrekt verarbeiten. Dazu wird der *Controller* mit einem vorgegebenem Array initialisiert. Nach der Initialisierung des *Controller* wird das übergebene Array geprüft.

HelperService Unit Test Die verschiedenen Hilfsfunktionen werden hier geprüft. Dazu werden verschiedene Parameter übergeben und der Rückgabewert der Hilfsfunktionen wird auf Korrektheit geprüft.

RESTService Unit Test Hier wird nur getestet ob die http-Anfrage korrekt aufgebaut wird. Dazu wird das verwendete *$http* Objekt mit dem TestObjekt *$httpBackend* gemockt. Dadurch können http-Anfragen abgefangen werden und definierte Antworten können gesendet werden.

ResourceService Unit Test Durch die asynchrone Funktionsweise dieses *Service* ist der Aufbau der Unit Tests komplizierter als bei den anderen Unit Tests. Es müssen verschiedene Zustände simuliert werden, dass mit Hilfe von Mocks realisiert wird. Außerdem müssen asynchrone Funktionen in speziellen von Jasmine bereitgestellten `run` Funktionen umschlossen werden. Die ebenfalls von Jasmine bereitgestellte `waitsFor` Funktion blockiert den weiteren Testablauf so lange bis ein bestimmtes Ereignis eintrifft oder ein Zeitlimit erreicht wurde. Getestet wird also hier ob sich der *Service* in einer bestimmten Situation entsprechend reagiert. Am Ende sollte aber immer eine Liste von Laureaten existieren oder eine Fehlermeldung abgefangen sein.

4.5.1 Testdurchläufe

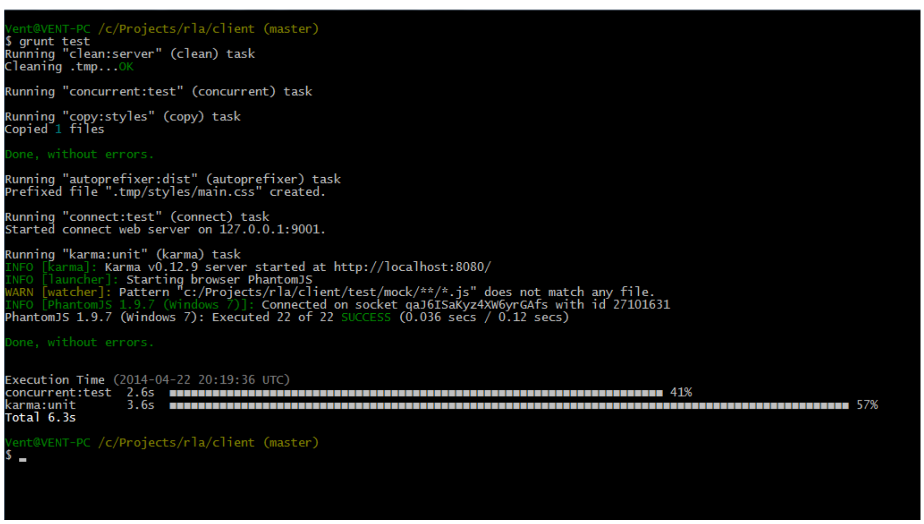

Abbildung 7: Ausgabe eines erfolgreichen Testdurchlauf.

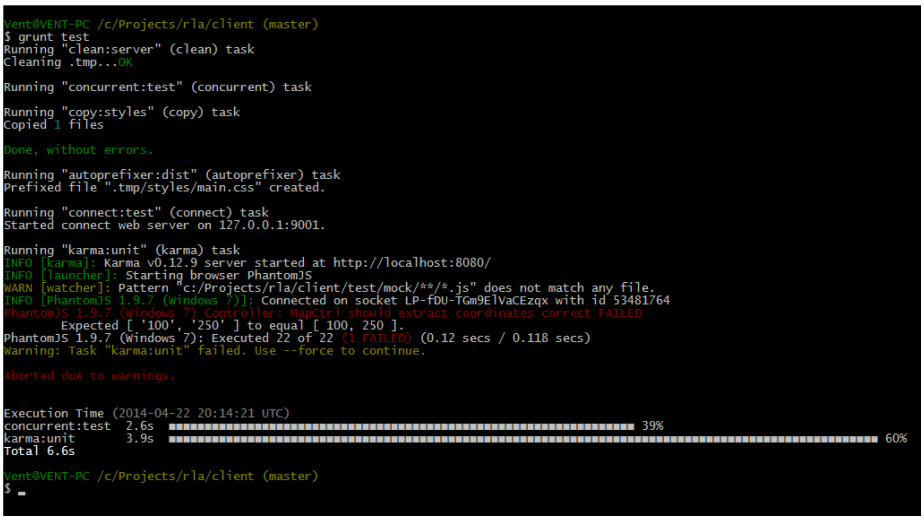

Abbildung 8: Ausgabe eines fehlgeschlagenen Testdurchlauf.

4.6 Server

Der Server ist mit dem Framework ExpressJS realisiert. Es existieren zwei statische Pfade und ein Pfad als REST Schnittstelle. Bei den statischen Pfaden können die entsprechenden Dateien direkt angefragt werden. Bei der REST Schnittstelle wird die URI und die Request Methode geprüft. Sind URI bekannt und die Request Methode erlaubt wird eine entsprechende Antwort gegeben. Sollte die URI unbekannt sein wird als Antwort die Fehlermeldung *"REST URI is unknown."* mit dem Statuscode 404 gesendet. Ist die Request Methode nicht erlaubt wird als Antwort eine Fehlermeldung mit den erlaubten Request Methoden und mit dem Statuscode 405 gesendet.

Die Datenbankanbindung erfolgt mit dem NodeJS Modul sqlite3[19]. Sollte ein REST Request für alle Laureaten beim Server eintreffen werden mit einem Query alle Daten aus der Datenbank geholt. Diese Daten werden im JSON Format als Antwort gesendet.

Die Datenbank ist eine sqlite Datenbank. Die Daten werden mit dem Programm sqliteman[20] eingepflegt.

[19] https://www.npmjs.org/package/sqlite3
[20] http://sqliteman.com/

5 Projektabschluss

5.1 Vorgehensweise

Der Projektanfang verlief sehr gut. Das technische Konzept konnte ich schneller abschließen als geplant. Während der Umsetzung gab es nur geringe Anpassungen am technischen Konzept. Der Ladeprozess wurde abgeändert. Die REST URI /rest/laureates/:id wurde entfernt und es wurde anstatt MongoDB die Datenbank sqlite verwendet.

Der nächste Schritt war die Ordnerstruktur festzulegen und die Projektstruktur durch Yeoman generieren zu lassen. Zur generierten Projektstruktur fügte ich noch ein paar nützliche JavaScript Bibliotheken hinzu. Schließlich definierte ich die vier Bereiche, die es in der Mobilanwendung geben wird.

Nachdem die Projektstruktur des Clients fertig war hab ich die Projektstruktur für den Server per Hand angelegt. Zuerst gab es nur einen statischen Ordner für die Client-Daten und eine JavaScript Datei, die den Server initialisiert. Später folgten die Datenbank und ein weiterer statischer Ordner für die Bilder der Laureaten. Ich habe mich dazu entschieden die Bilder dem Server zuzuordnen, weil ich wollte das die Bilder dort sind wo die Datenbank sich befindet. Die Bilder direkt in der Datenbank zu speichern wurde mir von Arbeitskollegen abgeraten.

Die erste Ansicht, die ich umgesetzt habe, war die Liste-Ansicht. Problematisch bei der Umsetzung war es einen guten Algorithmus zu finden, der die entsprechenden Werte lieferte. Mit etwas rumprobieren und der Verwendung von Funktionsgraphen konnte ich die entsprechenden Algorithmen entwickeln.

Bevor ich die nächste Ansicht anging habe ich den Slide-Effekt für die Navigation implementiert. Die Umsetzung klappte gut und lieferte direkt ein gutes Ergebnis. Dennoch stellte sich heraus, das diese Umsetzung ein horizontales Bewegen in der Mobilanwendung nicht zu lies. Aus diesem Grund habe ich mich dazu entschieden die Bilderwand-Ansicht vertikal aufzubauen und die Weltkarte nur per Klick zoombar zu machen.

Die Umsetzung der Bilderwand verlief gut. Meine Idee war es die Bilder ihren Höhen zu kategorisieren und feste Höhen zuzuweisen. Die Differenz der Höhen wird dann als CSS-Attribute margin gesetzt. Mit einem Klick auf ein Bild wird

dieses hervorgehoben und Information werden eingeblendet. Bis zu diesem Zeitpunkt lag ich gut in der Zeit.

Ich wollte jetzt die Datenbank erzeugen. Zuerst war als Datenbank MongoDB geplant. Nach einem Gespräch mit einem Arbeitskollegen habe ich mich dann entschieden sqlite als Datenbank zu nehmen. Gründe dafür waren MongoDB mehr bietet als ich für diese Mobilanwendung benötige und sqlite dagegen vollkommen ausreichend war. Außerdem kannte ich die SQL-Syntax wohingegen ich mir dir Syntax für MongoDB hätte aneignen müssen. Mein Betrieb besitzt für die Right Livelihood Webseite bereits eine Datenbank mit allen Laureaten. Diese habe ich als SQL Dump exportiert. Bis dahin gab es soweit keine Probleme. Problematisch war dann die Konvertierung des SQL Dump in eine Syntax, die sqlite versteht. Tools und Skripte, die ich im Internet gefunden habe, haben bei mir nicht richtig funktioniert. Ich habe dann per Hand die Syntax angepasst um die Rohdaten in meine sqlite Datenbank zu importieren. Dieses Vorgehen hat ziemlich viel Zeit gekostet.

Das nächste Problem waren die Bilder. Ich konnte diese nicht direkt aus dem SQL Dump übernehmen, weil zum Teil das Bild nicht für meine Bilderwand geeignet war. Deswegen habe ich alle Bilder vom Server, auf dem die Right Livelihood Webseite liegt, heruntergeladen und per Hand zwei passende Bilder ausgesucht. Dabei habe ich die Bildergrößen angepasst und die Bilder entsprechend unbenannt. Bei über 140 Laureaten hat diese Prozedur enorm viel Zeit gekostet und war auch in der Zeitplanung nicht vorgesehen. Entsprechend lag ich zu diesem Zeitpunkt nicht mehr in der Zeit. Glücklicherweise war dann das Ergebnis sehr zufriedenstellend, so dass ich mit der nächsten Ansicht fortfahren konnte.

Die Umsetzung der Weltkarte war zu Beginn kein Problem. Als erstes suchte ich eine passende Weltkarte im SVG Format aus, damit beim Zoomen keine Qualität verloren geht. Danach implementierte ich die Wellenanimation, die ein befüllen der Weltmeere darstellen sollte. Schließlich baute ich eine Zoomfunktion ein, die in den geklickten Bereich hereinzoomt.

Das Problem bei der Weltkarte waren die Koordinaten der Laureaten herauszufinden. Ich hatte die GPS-Koordinaten der Laureaten. Diese musste ich in Pixel-Koordinaten konvertieren, die auf meine Weltkarte passen. Mit ein paar Arbeitskollegen versuchte ich einen Algorithmus für diese Aufgabe zu finden. Wir fanden ein paar Algorithmen, aber bei der Implementierung lieferten diese

kein befriedigendes Ergebnis. Schließlich habe ich mich dazu entschieden wieder per Hand die Koordinaten in die Datenbank einzupflegen. Dazu schaute ich auf Google Maps die Positionen der Laureaten an und klickte entsprechend auf meine Weltkarte und lies mir die Pixel-Koordinaten ausgeben. Dieses Vorgehen hat wieder viel Zeit gekostet und war so nicht im Zeitplan vorgesehen.

Zum Schluss wurde ein Refactoring der Controller und Tests durchgeführt. Nachdem alle Tests erfolgreich durchliefen und die Mobilanwendung einen guten Eindruck machte habe ich mich dazu entschieden die Umsetzung abzuschließen.

5.2 Abschließender Zeitplan

Die Datenbank Umsetzung war eigentlich bei der Umsetzung der Module für den Server eingeplant. Da ich mit der Datenbank und den Daten größere Probleme hatte habe ich eine neue Zeile für die Datenbank hinzugefügt. Beim Client hab ich die Umsetzung der Services unterschätzt. Das liegt daran das ich Logik, die eigentlich für den Controller geplant war, sehr oft in die Services verlegt hab um diese Funktionen besser testen zu können. Das Problem mit der Datenbank war ein Planungsfehler von mir. Ich hätte mich besser mit den Daten, die mir zur Verfügung standen, auseinander setzen müssen. Durch diesen Zeitverlust fielen ausführlichere Tests auf anderen mobilen Geräten weg.

<table>
<tr><th>Phase</th><th>Beschreibung</th><th>Soll-Std</th><th>Ist-Std</th></tr>
<tr><td>I.</td><td>Technisches Konzept</td><td>10</td><td>4</td></tr>
<tr><td rowspan="11">II.</td><td colspan="2">Umsetzung</td><td rowspan="11">35</td><td rowspan="11">66</td></tr>
<tr><td>Art der Umsetzung</td><td>Soll-Std</td><td>Ist-Std</td></tr>
<tr><td>Struktur</td><td>1</td><td>1</td></tr>
<tr><td>Services</td><td>2</td><td>10</td></tr>
<tr><td>Controllers</td><td>25</td><td>28</td></tr>
<tr><td>Client</td><td>28</td><td>39</td></tr>
<tr><td>Struktur</td><td>2</td><td>1</td></tr>
<tr><td>Module</td><td>5</td><td>2</td></tr>
<tr><td>Datenbank</td><td>-</td><td>24</td></tr>
<tr><td>Server</td><td>7</td><td>27</td></tr>
<tr><td>III.</td><td>Testdurchführung</td><td>15</td><td>10</td></tr>
<tr><td>IV.</td><td>Dokumentation / Abschluss</td><td>10</td><td>8</td></tr>
<tr><td></td><td>Gesamt</td><td>70</td><td>88</td></tr>
</table>