



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 Mühlthal



Ausbildungsbetrieb:
ion2s GmbH
Donnersberggring 16
64295 Darmstadt

Inhaltsverzeichnis

1	Einleitung	3
1.1	Projektbeschreibung	3
1.2	Zeitplan	4
2	Technisches Konzept	6
2.1	Anforderung	6
2.2	Realisierung	6
2.2.1	Client	6
2.2.2	Server	7
2.2.3	Test	8
2.2.4	Kommunikation	8
2.2.5	Versionierung	8
3	Gestalterisches Konzept	9
3.1	Bilderwand-Ansicht	9
3.2	Liste-Ansicht	10
3.3	Weltkarte-Ansicht	10
3.4	Detail-Ansicht	11
4	Durchführung	12
4.1	Die Entwicklungsumgebung	12
4.2	Projektstruktur	12
4.2.1	client	13
4.2.2	docs	14
4.2.3	server	15
4.3	AngularJS	15
4.4	Umsetzung	15

4.4.1	Controller	15
4.4.2	Service	16
4.4.3	Directive	17
4.5	Test	19
4.5.1	Testdurchläufe	20
4.6	Server	21
5	Projektabschluss	22
5.1	Vorgehensweise	22
5.2	Abschließender Zeitplan	24

1 Einleitung

1.1 Projektbeschreibung

Als Unterstützung für die gemeinnützige Stiftung Right Livelihood¹ hat mein Ausbildungsbetrieb ion2s GmbH² 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 im Betrieb zu vertiefen bzw. zu erweitern.

In der Mobilanwendung sollen die Preisträger des Right Livelihood Awards, die sogenannten Laureaten, multimedial präsentiert werden. Um dieses Ziel zu erreichen sollen moderne Webtechnologien wie JavaScript, HTML5 und CSS3 verwendet werden. Der Fokus dieser Mobilanwendung liegt dementsprechend auf dem Client. Serverseitig sollen die statischen Ressourcen bereitgestellt und eine Datenbank angelegt werden, welche die Daten der Preisträger beinhaltet. Der Server soll mit NodeJS³ 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 - Beim ersten Aufruf der Mobilanwendung wird eine Intro-Ansicht gezeigt. Während dieser Ansicht werden die Daten der Laureaten heruntergeladen.
2. Liste-Ansicht - Die Liste wird scrollbar sein. Das Scrollen soll durch CSS einen 3D-Effekt bekommen. 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 ein einzelnes Bild per Klick vergrößern.
4. Weltkarte-Ansicht - Auf einer Weltkarte werden die Laureaten ihrem Herkunftsland entsprechend positioniert. Mit einem Klick auf die Weltkarte kann der Benutzer in dieses Gebiet hereinzoomen.

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. Diese Ansicht ist von den anderen Ansichten erreichbar.

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 Arbeitspakete gegliedert:

1. Struktur

- Verzeichnis anlegen.
- Eine Applikationsstruktur festlegen.
- Routing implementieren.

2. Services

- Umsetzung der REST Schnittstellen.
- Implementierung der lokalen Speicherung.

3. Controllern

- Umsetzung des Intro.
- Umsetzung der Listen-Ansicht.
- Umsetzung der Bilderwand-Ansicht.
- Umsetzung der Weltkarte-Ansicht.
- Umsetzung der Detail-Ansicht.

Die Umsetzung für den Server ist in folgende Unteraufgaben gegliedert:

1. Struktur

- Verzeichnisse anlegen.
- Serverstruktur festlegen.
- Umgang mit Anfragen an den Server.

2. Module

- Umsetzung der REST Schnittstelle.
- Implementierung der Datenbank.

Das Schreiben und Ausführen von Tests ist Bestandteil der Umsetzung. Darüber hinaus 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 umfangreicheres Feedback zu erhalten.

Aus diesen Angaben ergibt sich folgender Zeitplan:

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

2 Technisches Konzept

2.1 Anforderung

Die Mobilanwendung für den Right Livelihood Award soll die Preisträger medial 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- und 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

Aufgrund der genannten Technologien, die zum Einsatz kommen, sollen nur moderne Browser unterstützt werden, die diese Technologien beherrschen. Browser, die nicht unterstützt werden, bekommen nur eine entsprechende Nachricht angezeigt.

Zur Unterstützung bei der Entwicklung soll das JavaScript Framework AngularJS⁴ zum Einsatz kommen. Für eine hohe Qualität der Anwendung soll das

⁴<https://angularjs.org/>

Scaffolding Tool Yeoman⁵ sorgen. Das Tool selbst ist ein Generator, welches ein Grundgerüst für die Anwendung bereitstellt. Mit Yeoman kommt das *JavaScript Task Runner Tool* Grunt⁶ und das *Package Manager Tool* Bower⁷ 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 des HTML5 Features 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
- SQLite⁸

NodeJS ist eine serverseitige JavaScript Plattform. Der Vorteil dieser Technologieauswahl ist, dass sowohl Client als auch Server in der Skriptsprache JavaScript geschrieben werden. Zur Unterstützung bei der Implementierung des Servers, soll das NodeJS Framework ExpressJS⁹ eingesetzt werden. Als Datenbank wird SQLite verwendet, weil an die Datenbank keine größeren Anforderungen gestellt werden, als die Daten der Laureaten zu liefern.

⁵<http://yeoman.io/>

⁶<http://gruntjs.com/>

⁷<http://bower.io/>

⁸<https://sqlite.org/>

⁹<http://expressjs.com/>

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 auch des Servers soll das Framework Jasmine 2.0¹⁰ 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-Format¹¹ 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 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 Git¹³ zum Einsatz kommen. Git ist weit verbreitet und verfügt über viele Plug Ins. Das Projekt soll außerdem auf GitHub¹⁴ hochgeladen werden. So ist es möglich, dass andere Projektmitglieder sich einfach 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/>

¹²http://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 auszusehen hat und zu bedienen ist. Ich hab mich bei der Umsetzung der Mobilanwendung weitestgehend 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

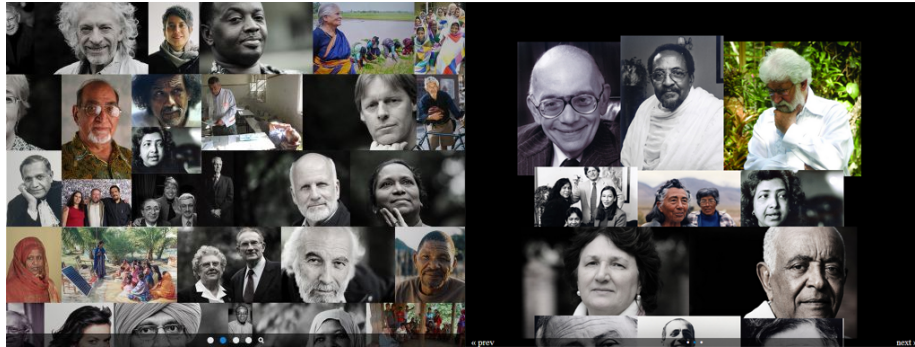


Abbildung 1: Vergleich von Gestaltung (links) und Umsetzung (rechts) der Bilderwand-Ansicht.

Bei der Aneinanderreihung der Bilder sollte kein statischer Eindruck entstehen. Außerdem sollte sich der Benutzer frei durch die Bilderwand bewegen können. Da bei der Umsetzung der Gesten-Steuerung Probleme aufgetreten sind, habe ich mich dazu entschieden, die Bilder vertikal aneinander zu reihen und somit nur ein vertikales Scrollen dem Benutzer zu ermöglichen. Auch ein freies Zoomen konnte nicht umgesetzt werden. Stattdessen kann der Benutzer auf ein Bild klicken das sich vergrößert und so positioniert wird, das der Benutzer das gesamte Bild sehen kann. In dieser Ansicht werden zusätzliche Informationen des Laureaten angezeigt. Ein Klick auf diese Zusatzinformationen führt den Benutzer zur Detail-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 wurde deshalb nicht implementiert.

3.3 Weltkarte-Ansicht

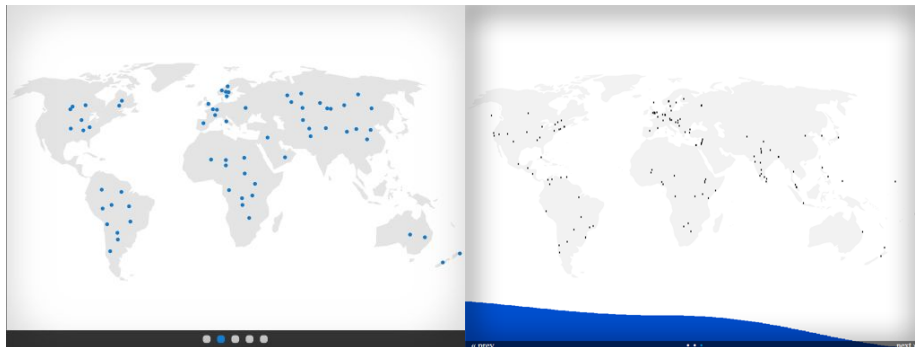


Abbildung 3: Vergleich von Gestaltung (links) und Umsetzung (rechts) der Weltkarte-Ansicht.

Auch bei der Weltkarte sollte ein freies Bewegen und Zoomen möglich sein. Aufgrund ähnlicher Probleme, wie bei der Bilderwand, habe ich mich dazu entschlossen, dass der Benutzer nur in bestimmte Bereiche der Weltkarte zoomen kann. Durch einen Klick auf die Weltkarte wird auf diesen Punkt herangezoomt

und die 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 Detail-Ansicht geführt.

3.4 Detail-Ansicht

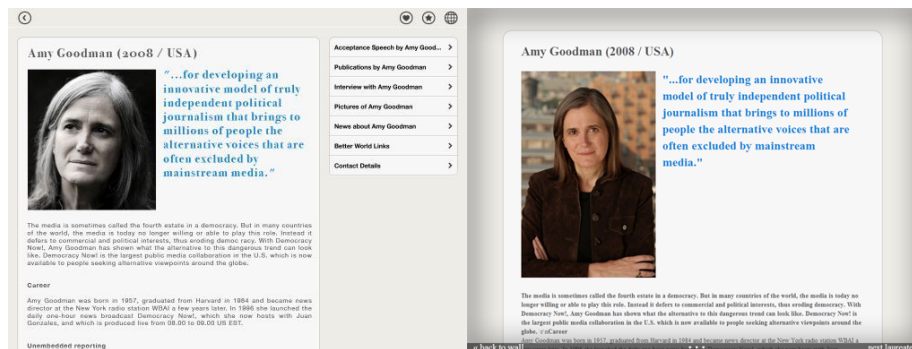


Abbildung 4: Vergleich von Gestaltung (links) und Umsetzung (rechts) der Detail-Ansicht.

Bei der Detail-Ansicht hab ich mich dazu entschieden die weiterführenden Links des Laureaten nach unten zu platzieren, um somit die Lesbarkeit zu verbessern.

4 Durchführung

Das komplette Projekt wurde auf GitHub¹⁵ 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 WebStorm¹⁶ 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.

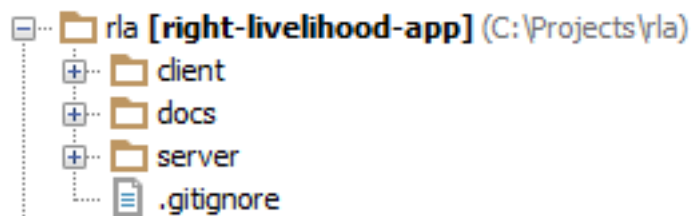


Abbildung 5: Projektstuktur der Mobilanwendung.

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 Verbesserungen 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 Felder *name* und *version*. Diese Felder sind nur erforderlich, wenn das Projekt in anderen Projekt verwendet werden soll. Dies ist hier nicht der Fall. Die *package.json* ist hilfreich 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 Visio¹⁸ 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 Client bereitgestellt 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. *Services* 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: (Detail-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.

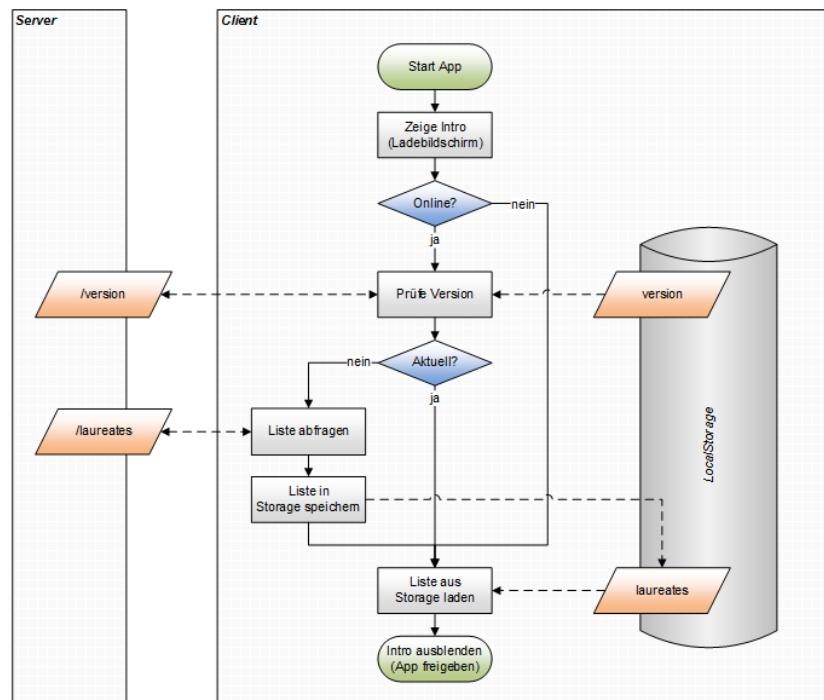


Abbildung 6: Ablaufdiagramm des initialen Ladeprozesses.

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 `isHorizontal(deltaX, deltaY)` wird ein horizontales Wischen registriert. Erfolgt das Wischen innerhalb eines vorgegeben Zeitfenster wird in die entsprechende Richtung navigiert. Welche Ansicht sich 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 zeichnet werden die Callbacks, die mit der Funktion `requestAnimationFrame` registriert 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 Verblässungswert. Die falsche Position, der Skalierungs- und Verblässungswert werden mit 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 die richtige Position weiter bewegt. Dazu wird das Listenelement identifiziert, das sich am nächsten der 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 Verblässungswert:

```
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 sein, 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 dieser Bereich vergrößert und die 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, mit einer Wellenanimation das komplette Wasser im Hintergrund zu befüllen, sobald die Weltkarte angezeigt wird. Eine mögliche Umsetzung einer solchen Animation wurde 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 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 *Controllers*, 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.


```

vent@VENT-PC /c/Projects/rla/client (master)
$ grunt fast
Running "clean:server" (clean) task
Cleaning .tmp...OK

Running "concurrent:test" (concurrent) task

Running "copy:styles" (copy) task
Copied 1 files

Done, without errors.

Running "autoprefixer:dist" (autoprefixer) task
Prefixed file ".tmp/styles/main.css" created.

Running "connect:test" (connect) task
Started connect web server on 127.0.0.1:9001.

Running "karma:unit" (karma) task
[INFO] [karma]: Karma v0.12.9 server started at http://localhost:8080/
[INFO] [launcher]: Starting browser phantomJS
[WARN] [watcher]: Pattern "/c/Projects/rla/client/test/mock/**/*.*.js" does not match any file.
[INFO] [PhantomJS 1.9.7 (Windows 7)]: Connected on socket LP-FDU-Tcm9ELVaCEzqx with id 53481764
[PhantomJS 1.9.7 (Windows 7) controller: MapCtrl should extract coordinates correct FAILED
Expected ["100", "250"] to equal ["100", "250"]
PhantomJS 1.9.7 (Windows 7): Executed 22 of 22 (1 FAILED) (0.12 secs / 0.118 secs)
Warning: Task "karma:unit" failed. Use --force to continue.

Aborted due to warnings.

Execution Time (2014-04-22 20:14:21 UTC)
concurrent:test 2.6s
karma:unit 3.9s
Total 6.6s

```

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 Antwort im JSON-Format zur einfacheren Verarbeitung im Client 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 Datenbankankbindung 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. Mit diesem Programm lassen sich sqlite Datenbanken über eine grafische Benutzeroberfläche bearbeiten.

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.

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 Funktionsgraphen²¹, konnte ich entsprechende 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, dass diese Umsetzung ein horizontales Bewegen in der Mobilanwendung nicht zuließ. 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 nach ihrer Höhe zu kategorisieren und ihnen eine feste Höhe 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.

²¹<http://www.walterzorn.de/grapher/grapher.htm>

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, dass MongoDB einen größeren Funktionsumfang für diese Mobilanwendung benötigen wird und sqlite für die Anforderungen vollkommen ausreichend war. Außerdem kannte ich die SQL-Syntax bereits, wohingegen ich mir die Syntax für MongoDB, hätte erst 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 konform ist. Tools und Skripte, die ich im Internet gefunden habe, haben 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 Qualitätsverluste entstehen. 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 Umsetzung der Weltkarte war, 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 durchgelaufen sind und die Mobilanwendung einen guten Eindruck machte, konnte ich die Umsetzung abschließen.

5.2 Abschließender Zeitplan

Die Datenbank Umsetzung war ursprünglich bei dem Modul 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 habe 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 mussten ausführlichere Tests auf spezifischen mobilen Geräten wegfallen, um den Zeitplan einhalten zu können.

Phase	Beschreibung			Soll-Std	Ist-Std
I.	Technisches Konzept			10	4
II.	Umsetzung			35	66
	Art der Umsetzung	Soll-Std	Ist-Std		
	Struktur	1	1		
	Services	2	10		
	Controllern	25	28		
	Client	28	39		
	Struktur	2	1		
	Module	5	2		
	Datenbank	-	24		
	Server	7	27		
III.	Testdurchführung			15	10
IV.	Dokumentation / Abschluss			10	8
	Gesamt			70	88