TheCaliCampTeaches  
Dominic Hehli

Inhaltverzeichnis

[2 Vorwort 6](#_Toc503044640)

[3 Ausgangsituation 7](#_Toc503044641)

[4 Projektinformationen 8](#_Toc503044642)

[4.1 Projektstrukturplan 9](#_Toc503044643)

[4.2 Projektorganisation 10](#_Toc503044644)

[5 Installationsanleitung 11](#_Toc503044645)

[5.1 Entwicklungsumgebung 11](#_Toc503044646)

[5.2 Node.js 11](#_Toc503044647)

[5.3 RethinkDB 11](#_Toc503044648)

[5.4 Angular-CLI 12](#_Toc503044649)

[5.5 Github 12](#_Toc503044650)

[5.6 Einfügen der Config-Dateien 12](#_Toc503044651)

[5.7 Installation Skripts ausführen 12](#_Toc503044652)

[5.8 Testen 12](#_Toc503044653)

[5.9 Build 12](#_Toc503044654)

[6 Budget / Kosten 13](#_Toc503044655)

[6.1 Soll 13](#_Toc503044656)

[6.2 Meilensteine 13](#_Toc503044657)

[6.3 IST 14](#_Toc503044658)

[6.4 Fazit 14](#_Toc503044659)

[7 Projektziele 15](#_Toc503044660)

[7.1 SOLL 15](#_Toc503044661)

[7.1.1 Gesamtziel 15](#_Toc503044662)

[7.1.2 Teilziele 15](#_Toc503044663)

[7.1.3 Kann-Ziele 15](#_Toc503044664)

[7.2 IST 16](#_Toc503044665)

[7.3 Fazit 16](#_Toc503044666)

[8 Technologieentscheid 17](#_Toc503044667)

[8.1 Datenbank 17](#_Toc503044668)

[8.1.1 MYSQL 17](#_Toc503044669)

[8.1.2 MongoDB 17](#_Toc503044670)

[8.1.3 RethinkDB 18](#_Toc503044671)

[8.1.4 Entscheidung 18](#_Toc503044672)

[8.2 Webserver 18](#_Toc503044673)

[8.2.1 Apache 18](#_Toc503044674)

[8.2.2 NGINX 19](#_Toc503044675)

[8.2.3 Node.js 19](#_Toc503044676)

[8.2.4 Entscheidung 19](#_Toc503044677)

[8.3 Framework 19](#_Toc503044678)

[8.3.1 Angular 19](#_Toc503044679)

[8.3.2 React 20](#_Toc503044680)

[8.3.3 Vue.js 20](#_Toc503044681)

[8.3.4 Entscheidung 21](#_Toc503044682)

[8.4 Frontend 21](#_Toc503044683)

[8.5 Versionierung 21](#_Toc503044684)

[8.5.1 Entscheidung 22](#_Toc503044685)

[9 Projektaufbau 23](#_Toc503044686)

[9.1 Config 23](#_Toc503044687)

[9.2 Dist 23](#_Toc503044688)

[9.3 Documentation 23](#_Toc503044689)

[9.3.1 Nodemodule 23](#_Toc503044690)

[9.4 Server 23](#_Toc503044691)

[9.4.1 Connection 23](#_Toc503044692)

[9.4.2 Helpers 24](#_Toc503044693)

[9.4.3 Models 24](#_Toc503044694)

[9.4.4 Setup 24](#_Toc503044695)

[9.5 SRC 24](#_Toc503044696)

[9.5.1 Components 24](#_Toc503044697)

[9.5.2 Guard 24](#_Toc503044698)

[9.5.3 Shared 24](#_Toc503044699)

[9.5.4 Routing 24](#_Toc503044700)

[9.5.5 App-Module 25](#_Toc503044701)

[9.5.6 Assets 25](#_Toc503044702)

[9.5.7 Index.html 25](#_Toc503044703)

[9.6 Test 25](#_Toc503044704)

[9.7 Uploads 26](#_Toc503044705)

[9.8 Package.json 26](#_Toc503044706)

[10 Analyse und Implementierung Design 27](#_Toc503044707)

[10.1 Designvorlage 27](#_Toc503044708)

[10.2 Icons 27](#_Toc503044709)

[11 Umsetzung HTML 28](#_Toc503044710)

[11.1 Gridsystem 28](#_Toc503044711)

[11.1.1 Spalten in der Desktopansicht 28](#_Toc503044712)

[11.1.2 Spalten in der Mobileansicht 28](#_Toc503044713)

[11.2 Formulare 29](#_Toc503044714)

[11.3 Tabellen 29](#_Toc503044715)

[11.4 JavaScript 30](#_Toc503044716)

[11.4.1 Navigation in der Desktopansicht 30](#_Toc503044717)

[11.4.2 Navigation in der Mobileansicht 30](#_Toc503044718)

[11.5 Breakpoints 30](#_Toc503044719)

[11.6 Sass 31](#_Toc503044720)

[11.6.1 Abstracts 31](#_Toc503044721)

[11.6.2 Base 31](#_Toc503044722)

[11.6.3 Bootstrap 31](#_Toc503044723)

[11.6.4 Components 31](#_Toc503044724)

[11.6.5 Layout 31](#_Toc503044725)

[11.7 Mixins 32](#_Toc503044726)

[11.7.1 Beispiel CSS anhand von border-radius 32](#_Toc503044727)

[12 Angular Frontend 33](#_Toc503044728)

[12.1 Aufbau 33](#_Toc503044729)

[12.1.1 Klasse 33](#_Toc503044730)

[12.1.2 HTML 33](#_Toc503044731)

[12.1.3 Typescript 34](#_Toc503044732)

[12.1.4 Service 35](#_Toc503044733)

[13 Datenbank 36](#_Toc503044734)

[13.1 Einleitung 36](#_Toc503044735)

[13.2 Datenbankmodell 37](#_Toc503044736)

[13.3 Fazit 37](#_Toc503044737)

[14 Express Backend 38](#_Toc503044738)

[14.1 Einführung 38](#_Toc503044739)

[14.2 Packages 39](#_Toc503044740)

[14.2.1 Multer 39](#_Toc503044741)

[14.2.2 Cloudinary 39](#_Toc503044742)

[14.2.3 Mailgun 40](#_Toc503044743)

[14.2.4 Express Validator 41](#_Toc503044744)

[14.2.5 Registrierung & Login 41](#_Toc503044745)

[15 Testen 44](#_Toc503044746)

[15.1 Setup 44](#_Toc503044747)

[15.2 Unit-Test 44](#_Toc503044748)

[15.3 Assertation 45](#_Toc503044749)

[16 Risiko- und Erfolgsfaktoren 46](#_Toc503044750)

[16.1 Risiken 46](#_Toc503044751)

[16.2 Erfolgsfaktoren 46](#_Toc503044752)

[17 Summary 47](#_Toc503044753)

[18 Glossar 48](#_Toc503044754)

[19 Quellenverzeichnis 51](#_Toc503044755)

[20 Abbildungsverzeichnis 52](#_Toc503044756)

# Vorwort

Die Firma Netlive IT AG gründete im Jahr 2013 ein Surf Camp in Kaliforniern namens TheCaliCamp. TheCaliCamp hat sich in den letzten Jahren zu einer beliebten Feriendestination etabliert. Jährlich besuchen Hunderte von Touristen das Surf Camp. Während ihrer Aufenthaltszeit werden die Gäste von den angestellten Surfguides an verschiedene Strände in Südkalifornien gebracht, um zu surfen.

Während der Wintermonate (November bis Mai), sind die Surfguides gezwungen einen zweiten Job anzunehmen, da sie über diese Zeit nicht voll ausgelastet werden können. Oftmals müssen die schlecht ausgebildeten Guides als Kellner in einem Restaurant arbeiten, da Kellner in Amerika gerade mal den Mindestlohn erhalten, sind sie auf Trinkgeld angewiesen.

Hier kommt TheCaliCampTeaches in Spiel. Es soll eine neue Website erstellt werden, über die ehemalige Gäste wie auch auswärtige Surfer ihre Videos über einen Memberbereich hochladen. Die hochgeladenen Videos werden anschliessend von unseren professionellen Surfguides analysiert. Zu einem Preis von 90$ erhalten die User der Website Feedback zu ihren Surfvideos. Stehen sie richtig auf dem Surfboard? Paddeln sie die Welle zum richtigen Zeitpunkt an? Surfen Sie während der richtigen Tageszeit? Usw.

Dies hat den Sinn die Surfguides während der Wintermonate in Kalifornien finanziell zu unterstützen. Zudem stellt dies einen aussergewöhnlichen Service für unsere Surfgäste dar. Gäste, die bereits in unserem Surfcamp waren, erhalten die Analyse ihrer Surfvideos zu einem reduzierten Preis über einen Gutscheincode.

Ob und mit welcher Kadenz die Website genutzt wird, hängt in erster Linie von der Vermarktung ab. In dieser Arbeit soll in erster Linie das technische Grundgerüst für eine solide Plattform erstellt werden.

# Ausgangsituation

Für TheCaliCampTeaches soll ein neues Webprojekt erstellt werden. Der Kunde hat bereits vorgängig ein simples Mockup geliefert (siehe Anhang s. 1-11), es soll einen ersten Eindruck vermitteln, was die Webseite alles beinhalten kann.

Über die TheCaliCampTeaches Plattform sollen Videos in allen gängigen Formaten hochgeladen werden können. Dies führt zwangsweise zu grossen Datenmengen, die auf einen Server hochgeladen werden müssen. Hier wird eventuell ein externer Dienst geprüft.

Aus diesem Mockup muss das Datenbankmodell erstellt werden, um die vom Kunden gewünschten Anforderungen zu erfüllen.

Der Kunde ist grundsätzlich lösungsneutral, mit welcher Technologie der Auftrag umgesetzt wird, ist für ihn nicht von Interesse. In der Anfangsphase meines Projektes werde ich daher eine Analyse von geeigneten Technologien erstellen. Hierbei ist wichtig, dass die eingesetzte Technologie zum einen den Zweck bestmöglich erfüllt und der Implementierungsaufwand in einem vernünftigen Rahmen liegt.

Im Rahmen dieser Projektarbeit können nicht alle seine Wünsche erfüllt werden, dies aufgrund begrenzter zeitlicher Ressourcen.

Der Kunde wird ein Design bei einem externen Dienstleister in Auftrag geben. Die Konzeption der öffentlichen Website mit deren Inhalt und Struktur wird grundsätzlich dem Kunden überlassen.

# Projektinformationen

|  |  |
| --- | --- |
| Projektbezeichnung | TheCaliCampTeaches |
| Diplomand | Dominic Hehli  Dorfstr 77  9204 Andwil  079 832 22 77  dh@netlive.ch |
| Lehrgang | HF 14-17 A Softwareentwicklung |
| Betreuer | Raphael Looser  Steinwichslenstrasse 2a  9052 Niederteufen [+41 78 775 57 80](tel:+41%2078%20775%2057%2080)  [raphael.looser@lump.ch](mailto:raphael.looser@lump.ch) |
| Experte | Michael Nägeli |
| Auftraggeber | TheCaliCamp San Clemente, California [dave@thecalicamp.com](mailto:dave@thecalicamp.com) +1 (949) 289-8274 |

## Projektstrukturplan

Abbildung 1: Projektplan

## Projektorganisation

Abbildung 2: Projekorganisation

# Installationsanleitung

Erfahren Sie in diesem Kapitel, wie das Projekt «TheCaliCampTeaches» in Betrieb genommen wird.

## Entwicklungsumgebung

Generell kann die Entwicklungsumgebung frei gewählt werden. Es stehen diverse Umgebungen zur Verfügung, mit denen moderne Node-Applikationen realisiert werden können. Für mein Projekt habe ich mich für die IDE Atom entschieden. Atom wird von GitHub als Open-Source-Editor zur Verfügung gestellt. Durch die ständige Entwicklung und Weiterentwicklung von Plug-ins steht eine breite Palette von nützlichen Instrumenten für den Atom-Editor zur Verfügung.

Weitere IDE’s:

* Visual Studio
* Intellij idea
* Sublime

## Node.js

Das Grundgerüst der Applikation bildet Node.js mit dem integrierten Package Manager. Installieren Sie Node.js auf Ihrem Computer, achten Sie darauf eine Version >= 8.9.3 zu installieren. Zudem muss der Package Manager mitinstalliert werden, dieser ist standardmässig angewählt.

Als Ergebnis können nun auf Ihrem Computer Node.js Applikationen gestartet werden sowie neue Packages mit dem Befehl npm installiert werden. Installierte Packages werden nun im Ordner node\_modules abgelegt.

## RethinkDB

Installieren Sie nun die RethinkDB auf Ihrem Computer (Applikation getestet mit Version 2.3). Die ausführbare Datei können Sie an einem beliebigen Ort auf ihrem Computer ablegen. Wechseln Sie nun per Terminal an den Ort an dem Sie die ausführbare Datei abgelegt haben und geben Sie den Befehl rethinkdb ein.

Auf einem Windows Computer sollten Sie nun etwa folgendes sehen:

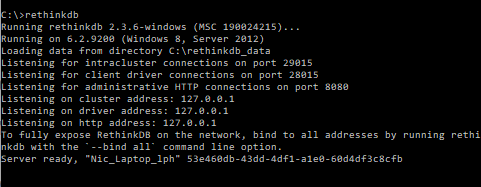


Abbildung 3: RethinkDB Terminal

Achten Sie darauf, diesen Terminal während der kompletten Betriebszeit der Applikation nicht zu schliessen.

## Angular-CLI

Da Sie nun über den Befehl npm neue Packages installieren können, müssen Sie das Angular-CLI installieren. Angular-CLI stellt diverse Befehle zur Verfügung, die bei der Erstellung von Angular-Applikationen benötigt werden.

Eine Auswahl der Befehle:

* Ng serve kompiliert die Applikation und startet den Entwicklungsserver
* Ng generate component erstellt eine neue Angularkomponente

## Github

Klonen Sie nun das Github Projekt «https://github.com/dhehli/thecalicampteaches». Dies können Sie wahlweise mittels Kommandozeile, Ihrem Editor oder per Download des ZIP-Files machen.

## Einfügen der Config-Dateien

Fügen Sie nun die separat mitgelieferten Config-Dateien test.json und default.json in den Ordner config ein. Diese Dateien konnten aus sicherheitstechnischen Gründen nicht direkt auf Github geladen werden (siehe Anhang s. 34).

## Installation Skripts ausführen

Wechseln Sie nun mittels Terminal zum Ordner «thecalicampteaches», welcher Sie im vorherigen Schritt von Github heruntergeladen haben.

1. Achten Sie darauf, dass die RethinkDB noch immer in einem separaten Terminal läuft
2. Führen Sie den Befehl npm install aus, dies installiert alle benötigten Node Packages in den Ordner node\_modules
3. Führen Sie den Befehl npm run setup aus, dieser Befehl erstellt alle benötigten Datenbanken sowie Tabellen
4. Mittels npm run server wird das Backend auf dem Port <http://localhost:3000> gestartet
5. Öffnen Sie ein weiteres Terminal für das Frontend und geben den Befehl ng serve ein. Dieser Befehl startet das Angular-Frontend auf dem Port <http://localhost:4200>

Die komplette Applikation sollte nun laufen.

## Testen

Um den Testdurchlauf zu starten, muss der Entwicklungsserver aus vorherigem Abschnitt Punkt 4 gestoppt werden.Anschliessend können Sie den Befehl npm run test ausführen. Dieser Befehl durchläuft alle Testfälle, die im Verzeichnis «test" liegen (siehe Kap. 15).

## Build

Mittels des Befehls ng build wird der Dist-Ordner erstellt. Er wird benötigt, um die Applikation live zu schalten. Das Backend liefert anschliessend das generierte index.html im Dist-Folder aus.

# Budget / Kosten

Die Zeiterfassung wurde mittels des Tools Toggl gemacht. Es bietet eine einfache Möglichkeit Zeiten zu erfassen, dies kann mittels Stoppuhr oder per manueller Eingabe erfolgen. Ein Projekt kann in beliebig viele Teilaufgaben zerlegt werden, die anschliessend mit einer geschätzten Zeit sowie einem Stundenansatz versehen werden können.

## Soll

Bei TheCaliCamp handelt es sich um eine Tochtergesellschaft von Netlive. Ihnen wird daher ein reduzierter Stundenansatz von 70.- pro Stunde verrechnet.

Die Tabelle 1 zeigt den geschätzten Aufwand vor Beginn des Projektes.

|  |  |  |
| --- | --- | --- |
| **Tätigkeit** | **Geschätzter Aufwand in h** | **Total in CHF** |
| Vorstudie | 16 | 1120 |
| Datenbankmodell | 8 | 560 |
| Prüfung von Frameworks | 2 | 140 |
| Umsetzung HTML/CSS | 30 | 2100 |
| REST API mit Anbindung and DB | 40 | 2800 |
| Userbereiche erstellen | 4 | 280 |
| Anmeldung/Login von Usern | 10 | 700 |
| Backendprogrammierung | 60 | 4200 |
| Testing | 10 | 700 |
| Dokumentation | 40 | 2800 |
| **Total** | **220** | **15400** |

Tabelle 1: Kosten

## Meilensteine

|  |  |  |
| --- | --- | --- |
| **ID** | **Datum** | **Meilenstein** |
| 1 | 01.09.2017 | Einreichung Projektantrag |
| 2 | 15.10.2017 | Vorstudie abgeschlossen |
| 3 | 22.10.2017 | Datenbankmodell erstellt |
| 4 | 07.11.2017 | HTML/CSS Umsetzung |
| 5 | 01.12.2017 | Backend |
| 6 | 01.01.2018 | Dokumentation |
| 7 | 08.01.2018 | Abgabe Diplomarbeit |

Tabelle 2: Meilensteine

## IST

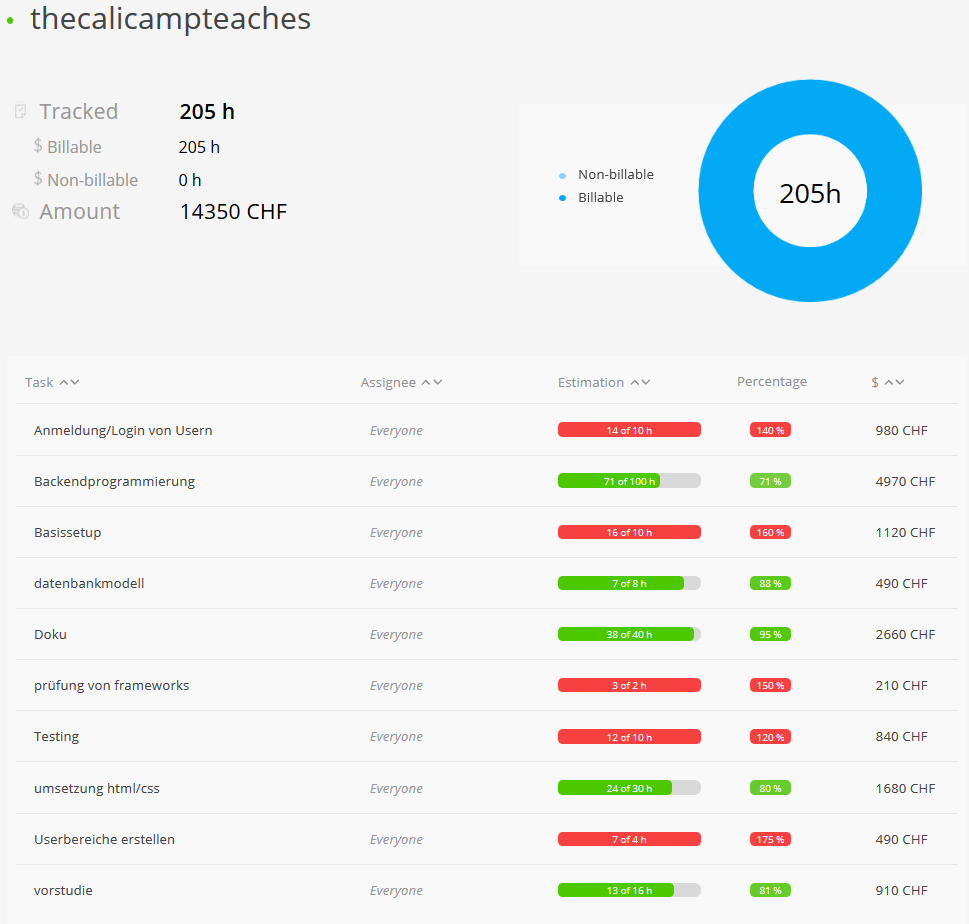


Abbildung 4: Zeiterfassung

## Fazit

Bei näherer Betrachtung des Diagramms kann festgestellt werden, dass es einige Zeitüberschreitungen gab, diese aber vorrangig bei zeitlich kurzen Aufgaben. Hier sticht vor allem das Basissetup ins Auge. Der Grund hierfür ist, dass das Projekt nicht wie anfänglich geplant mit React umgesetzt wurde, sondern mit Angular (siehe Kap. 12). Da ich die Technologie während des Projektes erlernen musste, habe ich vor allem beim Aufsetzen mehr Zeit gebraucht, als ursprünglich geplant.

Positiv ist, dass die mit React verlorene Zeit mit der HTML/CSS Umsetzung wettgemacht werden konnte. Ebenfalls konnten bei der Backendprogrammierung 30h eingespart werden. Nach einem sauberen Ablauf konnte hier vieles wiederverwendet oder mit leichten Anpassungen erneut verwendet werden.Die Meilensteine wurden zu meiner eigenen Überraschung alle eingehalten und somit Termingericht fertiggestellt.

Das Projekt konnte unter der erwarteten Zeit und somit unter den geschätzten Kosten realisiert werden. Was für ein Erfolg!

# Projektziele

## SOLL

### Gesamtziel

Es soll eine Website entstehen, die in drei verschiedene Bereiche gegliedert ist.

* Öffentliche Website
* Memberbereich
* Adminbereich

### Teilziele

* Analyse der Technologien, die eingesetzt werden
* Responsive Umsetzung des Designs mittels HTML/CSS für die öffentliche Website sowie den Memberbereich
* Valides HTML/CSS
* Erstellung des Datenbankmodells
* REST API für das erstellte Datenbankmodell
* Implementierung Useranmeldung/Login/Passwort vergessen für User
* User können Videos in den Memberbereich hochladen
* Adminbereich für Surfguides mit einem Login für alle
* Liste mit hochgeladenen Videos innerhalb des Memberbereiches
* Surfguides können Kommentare zu einem Video schreiben
* Testing mittels Framework

### Kann-Ziele

* Zahlungsabwicklung
* User können Kommentare von Surfguides bewerten
* Mail an User, sobald eine neue Bewertung für ihn erstellt wurde
* Mail an Admin, sobald ein neues Video von einem User hochgeladen wurde
* Gutscheincodes einlösen
* Separates Login für jeden Surfguide

## IST

|  |  |
| --- | --- |
| **Ziel** | **Erreichung** |
| Gesamtziele |  |
| Öffentliche Website |  |
| Memberbereich |  |
| Adminbereich |  |
| Teilziele |  |
| Responsive Umsetzung des Designs mittels HTML/CSS für die öffentliche Website sowie den Memberbereich |  |
| Valides HTML/CSS |  |
| Erstellung des Datenbankmodells |  |
| REST API für das erstellte Datenbankmodell |  |
| Implementierung Useranmeldung/Login/Passwort vergessen für User |  |
| User können Videos in den Memberbereich hochladen |  |
| Adminbereich für Surfguides mit einem Login für alle |  |
| Liste mit hochgeladenen Videos innerhalb des Memberbereichs |  |
| Surfguides können Kommentare zu einem Video schreiben |  |
| Testing mittels Framework |  |
| Kann-Ziele |  |
| Zahlungsabwicklung |  |
| User können Kommentare von Surfguides bewerten |  |
| Mail an Admin, sobald ein neues Video von einem User hochgeladen wurde |  |
| Gutscheincodes einlösen |  |
| Separates Login für jeden Surfguide |  |

Tabelle 3: Erreichte Ziele

## Fazit

Meine Teilziele habe ich vollumfänglich erreicht, sowie ein Kann-Ziel. Für mein nächstes Projekt würde ich allerdings den Projektumfang etwas kleiner wählen. Dies würde mir mehr Zeit für die Implementierung gewisser Funktionen bieten.

# Technologieentscheid

## Datenbank

### MYSQL

MYSQL gehört zu den weitverbreitetsten relationalen Datenbanksystemen. Da MYSQL Open-Source ist, erlangte es vor allem in Kombination mit der Programmiersprache PHP eine hohe Verbreitung. Typischerweise werden mehrere Tabellen angelegt, welche anschliessend über DQL mit Join’s verbunden werden. Mittels Transaktionen wird sichergestellt, dass verschiedene Datenbankmodifikationen nur dann erfolgreich sind, wenn mehrere gemeinsame Änderungen erfolgreich sind.

Vorteile:

* Stabil
* Hohe Sicherheit
* Weite Verbreitung
* Transaktionen

Nachteile:

* Langsam im Vergleich zu dokumentorientierten Datenbanken
* Schlechte Skalierbarkeit

### MongoDB

MongoDB ist wie MYSQL eine Open-Source Datenbank. Der wesentliche Unterschied besteht darin, dass MongoDB ein dokumentorientiertes Datenmodell verwendet. Anders als eine relationale Datenbank, welche Tabellen besitzt, werden bei der MongoDB sogenannten Collections gespeichert. Daten werden im BSON-Format gespeichert, also eine binäre Form vom weitverbreiteten JSON-Format. Den wesentlichen Vorteil offenbart die Datenbank beim Umgang mit sehr grossen Datenmengen, sie ist um ein Vielfaches schneller als eine relationale Datenbank. Zudem kann der Datenbestand sowie die Arbeitslast auf mehrere Server verteilt werden.

Vorteile:

* Sehr schnell bei grossen Datenmengen
* Skalierbarkeit
* Syntax lehnt sich stark an JavaScript an

Nachteile:

* Keine Transaktionen
* Daten müssen nicht immer aktuell sein

### RethinkDB

RethinkDB ist wie MongoDB eine dokumentorientierte Datenbank. Den Haupteinsatzbereich findet RethinkDB bei der Bereitstellung von Echtzeitdaten, mussten bei bisherigen Echtzeitsystemen über das sogenannte Polling Daten abgefragt werden, bietet diese Datenbank die Möglichkeit den Client über allfällige Änderungen zu informieren. Im Gegensatz zu MongoDB legt diese Datenbank mehr Wert auf die Datenkonsistenz und bietet die Möglichkeit an Join’s zu machen. Kurz gesagt sie verbindet das Beste aus der SQL und NoSQL Welt.

Vorteile:

* Pushen von Änderungen auf der DB zum Client
* Join’s mit REQL

Nachteile:

* Nicht weit Verbreitet
* Langsamer wie MongoDB

### Entscheidung

Ich finde, RethinkDB bietet das Beste aus beiden Welten. Es vereinfacht die Konzeption einer Echtzeitanwendung und kommt mit seiner eigenen Query-Language (REQL) sehr nahe an die Möglichkeiten von DQL. Der Syntax lehnt sich sehr stark an den von JavaScript an und ist somit für einen JavaScript-Entwickler einfach zu verstehen. Bei der MongoDB fehlte mir persönlich die Möglichkeit, Daten welche in verschiedenen Collections liegen, wieder sinnvoll zu kombinieren. Natürlich gibt es diese Möglichkeit auch auf MongoDB, über einen sogenannten lookup, jedoch scheint mir diese Möglichkeit unnötig kompliziert und nicht sehr intuitiv. Die positiven Erfahrungen bei bisherigen Projekten mit RethinkDB tragen ebenfalls zu diesem Entscheid bei.

## Webserver

### Apache

Apache ist der weitverbreitetste Webserver auf der Welt. Der Webserver wird von der Apache-Community weiterentwickelt. Er kann auf allen gängigen Betriebssystemen installiert und betrieben werden. Der grosse Vorteil dieses Webservers ist die Erweiterung durch diverse Zusatzmodule, welche frei zugänglich sind. Er findet sich vor allem bei Skriptsprachen wie PHP wieder, kann aber mit Tomcat auch für Java eingesetzt werden.

Vorteile:

* Hohe Verbreitung
* Ausführliche Dokumentation
* Modulsystem

Nachteile:

* Stösst an seine Grenze bei hoher Last

### NGINX

NGINX ist nach Apache einer der weitverbreitetsten Webserver auf dem Markt. Wie Apache ist auch NGINX modular aufgebaut und bietet somit diverse Erweiterungsmöglichkeiten. Im Gegensatz zu Apache ist NGINX jedoch viel schlanker gehalten und ist dafür bekannt, auch unter hohen Lasten eine nahezu konstante Zugriffszeit zu bieten. Dies geschieht im Inneren des Webservers mit einer unterschiedlichen Handhabung von Threads.

Vorteile:

* Benötigt weniger Memory als Apache
* Ist eventbasiert und damit schneller als Apache

Nachteile:

* Schwacher Community Support
* Nicht so viele Module wie Apache

### Node.js

In den letzten Jahren ist die gesamte Nodeumgebung stark gewachsen. Wie andere gängige Webserver beruht Node.js auf einem Modulsystem, das beliebig erweitert werden kann. Node.js nutzt JavaScript für die Serverprogrammierung, somit können Backend und Frontend komplett mit einer Programmiersprache geschrieben werden. Node.js ist sehr ressourcensparend und bietet eine grosse Anzahl von möglichen Netzwerkverbindungen, was der Asynchronität von JavaScript zugeschrieben werden kann. Dieses asynchrone Konzept hat gegenüber klassischen Threads den Vorteil, dass keine Blockierungen entstehen können und so massiv besser skalierbar ist.

Vorteile:

* Sehr effizient im Speicherumgang
* Keine Threads
* Diverse Packages über NPM (Node Package Manager)
* Gleiche Sprache für Backend & Frontend

Nachteile:

* Relativ neue Technologie

### Entscheidung

Ich entschied mich für Node.js. Aus meiner Sicht liegt der grösste Vorteil in der universellen Verwendung von JavaScript für das Backend & Frontend. Apache offenbart seine Vorteile in der Verwendung mit PHP. Zudem habe ich schon einige Erfahrungen mit Node.js und dem hervorragenden Modulsystem namens NPM.

## Framework

### Angular

Angular ist ein Front-End-Webapplikationsframework. Es wird als Open-Source Software angeboten und durch Google sowie die Community weiterentwickelt. Bei Angular 2 wird der Einsatz von Typescript empfohlen, was klassenbasierte objektorientierte Programmierung, statische Typisierung sowie Generics erlaubt. Dies ermöglicht im JavaScript Umfeld eine sonst atypische Typisierung von Variablen und führt somit zu einer stabileren Gesamtarchitektur. Angular ist eine sogenannte Single Page Application d.h. es wird über JavaScript definiert, welche Module der User zu sehen bekommt, für den Browser ist es jedoch eine einzige HTML-Website.

Vorteile:

* Sehr schnelle Ladezeiten
* Sehr gut geeignet für Mobilgeräte (Lazy Loading)
* Automatisiertes Testen
* Agile Entwicklung durch modularen Aufbau

Nachteile:

* Wird von alten Browsern nicht unterstützt
* Kompliziert bei der Erlernung

### React

React ist wie Angular eine JavaScript Library zum Erstellen von Frontends. Im Kern erstellt React ein sogenanntes virtuelles DOM. Wer bereits mit dem herkömmlichen DOM gearbeitet hat, stellte bei grossen Manipulationen eine schlechte Performanz fest, einfach gesagt es wird langsam. React schafft hier Abhilfe in dem es ein virtuelles DOM hält und nur dann auf das normale DOM zugreift, wenn dies erwünscht ist. Das modulare System bietet auch hier die Möglichkeit gut nachvollziehbaren Code zu schreiben. React verwendet nicht mehr reines JavaScript oder HTML, sondern JSX. JSX kann als JavaScript Erweiterung verstanden werden, aus der direkt HTML geschrieben wird. Wie Angular 2 erlaubt es das serverseitige Rendern von Komponenten.

Vorteile:

* Wiederverwendbare Komponenten
* Sehr schnell durch virtuelles DOM
* Eventhandler standardisiert
* Server Rendering

Nachteile:

* Klassische JavaScript-Plug-ins müssen wegen eigenen Eventhandlern und virtuellem DOM umgeschrieben werden.
* Wenig Built-In Funktionen

### Vue.js

Vue.js ist ebenfalls ein JavaScript-Framework, welches in die Kategorie der Single Page Applications einzuordnen ist. Wie React und Angular 2 erlaubt es das serverseitige Rendern von Websites. Vue.js zeichnet sich dadurch aus, dass viele Entwickler es für einfach zu erlernen halten. Es kann ebenso einfach für bereits bestehende Projekte eingesetzt werden. Wie auch React setzt Vue.js auf ein virtuelles DOM und besitzt vor allem Kernfunktionalitäten. Das Routing wird durch zusätzliche Libraries zur Verfügung gestellt, um den Kern möglichst schlank zu halten. Wie auch React wird das sogenannte JSX Format unterstützt.

Vorteile:

* Einfach zu erlernen
* Sehr schnell durch virtuelles DOM
* Server Rendering

Nachteile:

* Noch eher unbekannt

### Entscheidung

Beim Frontend-Framework entscheide ich mich für Angular. Anfänglich habe ich auf React tendiert, da ich mit dieser Technologie bereits Erfahrungen gemacht habe. Wie sich allerdings beim Projektstart herausstellte, gab es diverse Änderungen in React. Diese Änderungen waren so gravierend, dass ich mich für Angular entschieden habe. Die Änderungen bezogen sich hauptsächlich auf das empfohlene Package Redux, dessen Konzept mir komplett unverständlich war (siehe Kap. 19).

## Frontend

SASS ist de facto ein Standard, mit dem heutzutage das Erstellen von CSS vereinfacht wird. Es erweitert das CSS um nützliche Features wie Variablen, sogenannte Mixins mit denen man komplette Codeabschnitte wiederverwendbar machen kann und es ist sehr einfach zu erlernen. Beim Sass wird ein sogenannter Präprozessor eingesetzt, dieser verwandelt die SASS-Files wieder in normale CSS-Files. Sass hat mir in der Vergangenheit schon viele Male das Leben vereinfacht.

Vorteile:

* Variablen innerhalb von SASS
* Mixins für wiederverwendbaren Code
* Einfach zu erlenen
* Klarere Strukturen innerhalb von SASS

Nachteile:

* Präprozessor wird benötigt

## Versionierung

Bei der Versionierung gibt es zwei sehr bekannte Produkte, das eine ist GIT und das andere SVN. Der wesentliche Unterschied zwischen den beiden Systemen ist, dass bei SVN ein zentrales Repository auf dem Server zur Verfügung steht. Bei GIT hingegen gibt ein zentrales Repository auf dem Server, jedoch besitzt jeder User zudem noch eine lokale Kopie. Das zentrale Repository bei SVN hat den Nachteil, dass Änderungen nur mittels Internetverbindung hochgeladen werden können, bei GIT hingegen können Änderungen auf das lokale Repository hinzugefügt werden und zu einem späteren Zeitpunkt auf den Server. Dieser Vorteil ist zugleich der Nachteil von GIT, es ist komplizierter zu erlernen. Fast jeder Beginner hat so seine Schwierigkeiten mit dem Verständnis was jetzt das lokale Repository und was das Remote-Repository ist.

### Entscheidung

Ich entscheide mich beim Versionensystem für GIT, da es vor allem in der Webwelt stärker vertreten ist und ich mich bereits an das lokale und Remote Repository-System gewöhnt habe.

# Projektaufbau

Im Nachfolgenden wird der grobe Aufbau des Projektes dargestellt. Es soll einen Überblick über die gewählte Ordnerstruktur und deren Zweck geben (siehe Anhang s.12).

## Config

Im Config-Ordner befinden sich drei verschiedene JSON-Dateien, welche sämtliche Konfigurationen für das Projekt beinhalten. Das NPM-Package config ist dafür verantwortlich, welches der Files geladen wird.

Das public.json enthält alle Schlüsselwerte-Paare, die für das Projekt benötigt werden. Die Werte wurden in dieser Datei jedoch leer gelassen, um sie auf das öffentliche GIT-Repository zu laden, ohne einen API-Key zu veröffentlichen. Somit erhält der Entwickler trotzdem Einblick in die benötigten Keys.

Das default.json befindet sich im .gitignore d.h. es wird nicht auf das öffentliche Repository hochgeladen. Jeder Entwickler kann hier seine privaten API-Keys einfügen, welche benötigt werden, um das Projekt zu starten.

Das test.json beinhaltet die Konfiguration, welche für das Testen benötigt wird. Konkret bedeutet das, dass während des Testdurchlaufs auf dieses JSON umgeschaltet wird und die Datenbank gewechselt wird. Der Datenbankwechsel gewährleistet das Daten zu Testzwecken geschrieben, geändert und gelöscht werden können, ohne die Live-Datenbank zu modifizieren.

## Dist

Angular hat standardmässig einen Entwicklungsserver beigelegt, der es erlaubt sämtliche Änderungen im Quellcode zu erkennen. Bei einer Änderung muss somit die Ausgabe der Applikation nicht von Hand neu geladen werden, sondern es passiert automatisch. Möchte man allerdings die Applikation liveschalten, braucht es diesen Entwicklungsserver nicht mehr. Mittels des Konsolenkommandos «ng build» wird der Dist-Ordner erstellt. Konkret bedeutet dies, dass sämtliche Angular-Files «gebundlet» werden und in diesen Ordner geschrieben werden.

## Documentation

Dieser Ordner enthält sämtliche Dateien, die für die Dokumentation benötigt werden. Dazu gehört das Datenbankmodell, Diagramme, Anhang, Bilder sowie die Dokumentation selbst.

### Nodemodule

Der Nodemodule-Ordner beinhaltet sämtliche Dependencies, die im package.json definiert wurden.

## Server

In diesem Ordner befinden sich sämtliche Files, die für das Betreiben des Backend’s benötigt werden (siehe Kap. 14).

### Connection

In diesem File wird die Datenbankverbindung aufgebaut, dies geschieht mittels eines Packages, dass die Verwaltung der Verbindungen übernimmt.

### Helpers

Die Helpers werden benötigt, um externe Dienste, die an verschiedenen Orten wiederverwendet werden, zentral zu konfigurieren. Unter anderem wird hier der das Multer-Package konfiguriert, welches das Uploaden von multipart-formdata ermöglicht, sowie das Cloudinary-Package welche Dateien auf einen externen Grafik- und Videodienst hochlädt (siehe Kap. 14.2.2).

### Models

Im Model-Ordner wird der eigentliche Zugriff auf die Datenbank mittels REST-Schnittstelle gewährleistet. Für jedes Model werden die benötigten CRUD-Funktionen zur Verfügung gestellt (siehe Anhang s. 19).

### Setup

Im Setup werden vor der Erstellung des Projektes sämtliche Datenbanken und Tabellen erstellt, welche für das Projekt benötigt werden. Dies soll dem Entwickler ermöglichen, ohne grossen Aufwand das Projekt in Betrieb zu nehmen.

## SRC

In src-Ordner befindet sich das eigentliche Angular-Frontend (siehe Kap. 12).

### Components

Unter Components werden in Angular Komponenten verstanden, die typischerweise aus einer HTML-Datei sowie einer Typescript-Datei bestehen. Das HTML stellt den statischen Teil dar, während Typescript die Funktionalität für die jeweilige Komponente zur Verfügung stellt. Bei der Gliederung der Komponenten ist der Entwickler grundsätzlich frei. Ich habe mich für eine Gliederung nach den verschiedenen Bereichen (Admin, Member, Public) entschieden.

### Guard

Unter einem Guard wird in Angular ein Wächter verstanden, der entscheidet, ob die vom User aufgerufene Route z.B. /member aufrufbar ist. Hat der Benutzer die entsprechende Bemächtigung, wird er auf die angefragte Route weitergleitet. Sollte dies nicht der Fall sein, wird er auf eine vordefinierte Route zurückgeworden, dies kann z.B. die Route /login sein.

### Shared

Im Shared-Ordner befinden sich sämtliche Komponenten, die sich über das gesamte Projekt nicht unterscheiden und somit von allen Bereichen geteilt werden. In diesem Projekt gehören der Footer sowie die Logoutkomponente dazu.

### Routing

In der Routing-Komponente befinden sich sämtliche aufrufbare URL’s für dieses Projekt. Die Routes sind wie die Komponenten in die Bereiche public, admin und member aufgeteilt.

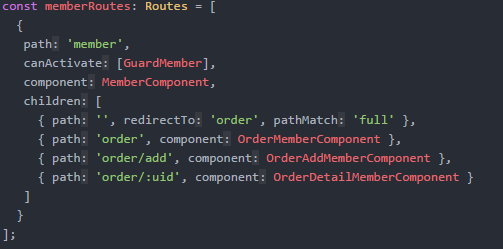


Abbildung 5: Angular-Router

Der Path gibt den Routenamen an, über welchen die Komponente erreichbar sein soll. CanActivate bezeichnet den Guard (siehe oben), welcher für die aktuelle Route zutreffen muss, damit sie erreichbar ist.

Component beschreibt die Komponente, die beim Aufruf dieser Route geladen werden soll. Children sind die Kindelemente, welche unter der Hauptkomponente zur Verfügung stehen z.B. member/order.

### App-Module

Das App-Module ist das Herzstück von Angular, hier werden sämtliche Komponenten importiert, welche für das Projekt zur Verfügung stehen sollen. Ebenfalls werden hier sämtliche Provider instanziiert, welche den HTTP-Zugriff auf das Backend gewährleisten.

### Assets

Innerhalb des Assets Ordners befinden sich sämtliche statischen Dateien von Angular. Unter statischen Dateien werden z.B. Bilder verstanden. In meiner Projektstruktur sind dies Schriften, Bilder und das SASS.

### Index.html

Das index.html bildet den Eintrittspunkt des Angular-Frontend’s. Es beinhaltet das kommerzielle HTML-Grundgerüst mit Head und Body. Als einziges Element befindet sich der Angular app-root innerhalb des Body-Tags, was wiederum die Hauptkomponente von Angular darstellt. Mit Webpack werden sämtliche JavaScript-Files, CSS, Fonts und Images gebündelt und eingehängt. Die nähere Ausführung der Funktionsweise von Webpack würde den Rahmen dieses Projektes allerdings sprengen.

## Test

Im Testordner befinden sich sämtliche Dateien, welche zum Testen der REST-API benötigt werden. In diesem File wird beim Starten des Tests die NODE\_ENV Variable auf «test» gesetzt, dies führt dazu das, dass oben erwähnte test.json aus der config geladen wird und die Datenbank umgeschaltet wird. Zusätzlich beinhaltet dieser Ordner ein JPG sowie eine mp4-Datei, welche für das Simulieren des Fileuploads benötigt werden.

## Uploads

Der Upload-Ordner wird benötigt, um Dateien, welche vom Admin- oder Memberbereich hochgeladen werden, zwischen zu speichern. Sämtliche Fileuploads werden vom externen Dienst «cloudinary» verwaltet (siehe Kap. 14.2.2). Die Files werden somit kurz im lokalen Ordner abgelegt und bei erfolgreichem Hochladen auf den externen Dienst lokal wieder gelöscht.

Funktioniert alles richtig, ist der Ordner Uploads die meiste Zeit leer. GIT würde diesen Ordner deshalb ignorieren, was bei einer Neuinstallation zu einem Fehler führt. Darum gibt es das File .gitkeep, welches GIT sagt, dass dieser leere Ordner erfasst werden muss.

## Package.json

Im package.json werden sämtliche installierte Node-Packages verwaltet. Generell wird hier zwischen Dependencies und Development-Dependencies unterschieden. Die regulären Dependencies werden für die Live-Umgebung der Applikation verwendet, wohingegen die Development-Dependencies nur für die Entwicklung benötigt werden. Packages, welche installiert werden sollten immer in diesem File eingetragen werden. Sie dienen anderen Entwicklern zur schnellen Übersicht und zudem wird der meist sehr grosse node\_modules Ordner nicht auf das Repository geladen. Mittels des Befehls npm install werden alle Packages, die in diesem File vorkommen, auf der lokalen Maschine installiert.

Das package.json beinhaltet zudem alle Skripts, die über die Konsole ausgeführt werden können.

Die verfügbaren Skripts sind:

* Ng serve zum Starten des Angulars
* Ng build zum Erstellen des Dist-Ordners
* Npm run test zum Starten des Tests
* Npm run server zum Starten des Backend’s
* Npm run setup zum Erstellen der Datenbank und Tabellen

# Analyse und Implementierung Design

<http://shibbythemes.com/psd-freebies/surfersco-psd-template/>

## Designvorlage

Die Designvorlage wurde nicht, wie ursprünglich angedacht, von unserer Designerin Maru Cruz geliefert. Aus zeittechnischen Gründen mussten wir auf eine Alternative zurückgreifen. Unser Partner in Kalifornien fand im Internet eine Photoshopvorlage, die ihm sehr zusagte (siehe Anhang s.13-14).

Dieses Design konnte sehr gut auf unsere Bedürfnisse angepasst werden und lieferte gleichzeitig bereits die wichtigsten Elemente.

Das komplette Design wurde von Luisa Costa entwickelt und für den Bootstrap-Grid gelayert und organisiert, was konkret bedeutet, dass sämtliche Elemente wunderbar in den Bootstrap-Grid passen. Das Design wurde anschliessend im Photoshop mittels Hilfslinien so eingeteilt, wie anschliessend das HTML entwickelt wird (siehe Anhang s.15).

## Icons

Die Icons, welche im Layout vorhanden waren, wurden zeitgemäss nicht mehr als Grafiken exportiert, sondern mittels Icomoon generiert. Icomoon ist eine Webplattform, die diverse vorgefertigte Icons zur Verfügung stellt. Über diese Plattform können alle gewünschten Icons selektiert werden und anschliessend als Web-Font exportiert werden. Sämtliche Icons werden als Font im CSS implementiert.

Dies bietet einige Vorteile:

* Icons können über CSS in der Farbe angepasst werden
* Icons können über die Schriftgrösse auf die gewünschte Grösse skaliert werden
* Es muss nur einmal die Schrift geladen werden und nicht wie bei z.B. PNG-Grafiken jede einzeln

Die Grafiken werden anschliessend mittels Klasse ins HTML eingebunden:

.icon-arrow-right2:before { content: "\ea3c"; }

.icon-facebook:before { content: "\ea90"; }

.icon-pinterest2:before { content: "\ead2"; }

# Umsetzung HTML

Bei der Umsetzung des HTML’s habe ich mich für die Library Bootstrap 4 entschieden. Sie bietet diverse Möglichkeiten, um die Entwicklung zu vereinfachen. Bootstrap ist im Endeffekt eine Gestaltungsvorlage für Entwickler.

## Gridsystem

Das Gridsystem bietet eine einfache Möglichkeit Spalten für responsive Layouts zu organisieren. Eine Zeile wird in zwölf Einheiten unterteilt. Je nach gewünschter Spaltenbreite kann eine CSS-Klasse gesetzt werden. Das Gridystem vereinfacht insbesondere die Darstellung auf mobilen Endgeräten, passen zwei Spalten nicht mehr auf den Viewport, werden diese untereinander dargestellt. In der neusten Version wurde zudem das komplette Gridsystem auf das neuere Boxmodell «flex-box» umgestellt. Das Flexboxmodell bietet diverse Verbesserungen zum alten Boxmodell.

Vorteile:

* Müssen HTML-Elemente entgegen ihrer Anordnung im Quellcode dargestellt werden, kann der sogenannte «order» Parameter gesetzt werden.
* Float’s werden überflüssig
* Nebeneinander angeordnete HTML-Elemente haben standardmässig dieselbe Höhe, was bei Float’s nicht möglich war
* Mittels «flex-grow» bzw. flex-shrink kann angegeben werden, wie weit ein Element wachsen oder schrumpfen darf.

Nachteile:

* Wird von älteren Browsern nicht oder nur begrenzt unterstützt
* Hat wie fast jede neue Technologie noch einige Bugs

### Spalten in der Desktopansicht



Abbildung 6: Spalten auf Desktop

### Spalten in der Mobileansicht



Abbildung 7: Spalten auf Mobile

## Formulare

Wer schon einmal von Grund auf ein Formular mittels CSS gestaltet hat, kennt die Problematik mit verschiedenen Browser. Jedes Browser-CSS rendert die Formularelemente wieder auf seine eigene Art und Weise. Bootstrap schafft auch hier Abhilfe und hat alle gängigen Formularelemente schon für uns gestaltet.

Für den Entwickler ist es anschliessend leicht das standardisierte Formular seinen Wünschen entsprechend anzupassen.

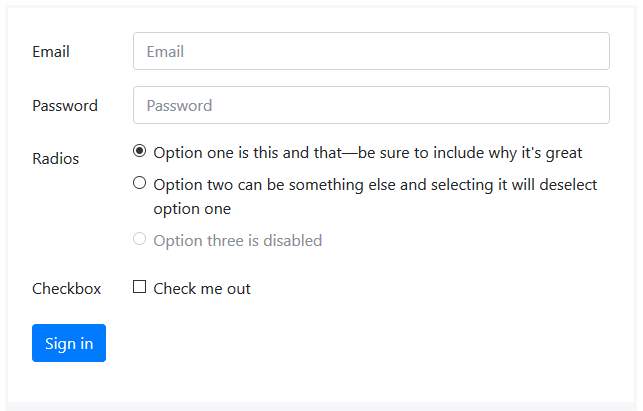


Abbildung 8: Formular

## Tabellen

Wie bei den Formularen hat Bootstrap auch Tabellen in diversen Variationen gestaltet. Der massgebende Vorteil dieser Tabellen ist, dass mittels einer Klasse «responsive-table» die Tabellen auf der Mobileansicht eine horizontalen Scrollbalken erhalten.

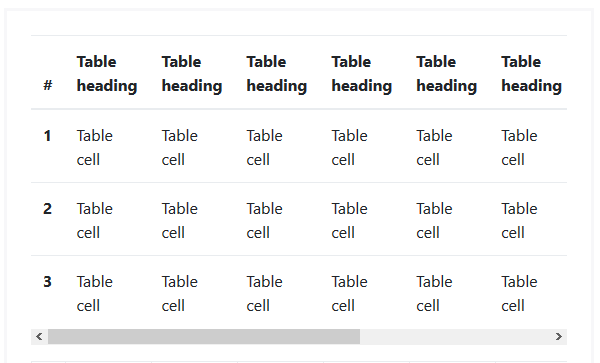


Abbildung 9: Responsive Tabelle

## JavaScript

Im Bereich JavaScript bietet Bootstrap ebenfalls einige spannende Komponenten. Zu der Meistgenutzten dürfte hier mit hoher Wahrscheinlichkeit die Mobile-Navigation gehören. Mittels Breakpoint (siehe weiter unten) wird die Navigation versteckt und ein Icon eingeblendet, welches die Navigation per Knopfdruck öffnet.

### Navigation in der Desktopansicht



Abbildung 10: Navigation Desktop

### Navigation in der Mobileansicht

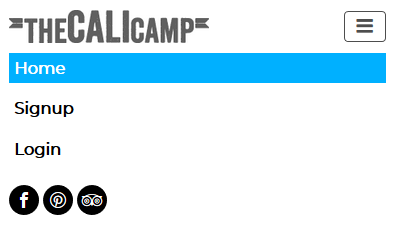


Abbildung 11: Navigation Mobile

## Breakpoints

Unter Breakpoints versteht man eine Anweisung im CSS, die ab einer bestimmten Grösse neue CSS-Befehle laden kann. Somit kann sichergestellt werden, dass jedes Endgerät, egal mit welcher Auflösung, optimal bedient werden kann. Bootstrap hat für ihr eigenes Gridsystem ebenfalls diese Breakpoints eingesetzt.

// Small devices (landscape phones, 576px and up)

@media (min-width: 576px) { ... }

// Medium devices (tablets, 768px and up)

@media (min-width: 768px) { ... }

// Large devices (desktops, 992px and up)

@media (min-width: 992px) { ... }

// Extra large devices (large desktops, 1200px and up)

@media (min-width: 1200px) { ... }

## Sass

Nach dem Aufbau des HTML’s wurde mittels Sass das Frontend gestaltet. SASS-Dateien können von Browsern jedoch nicht direkt interpretiert werden, dies macht einen sogenannten Präprozessor notwendig, der SASS-Dateien wieder in natives CSS übersetzt. Angular bietet bereits einen solchen Präprozessor in Verbindung mit Webpack an.

Im ersten Schritt wurde die Ordnerstruktur nach SASS-Guidline angelegt (siehe Kap.19). Folgende Ordnerstruktur ist vorgesehen (siehe Anhang s. 16).

### Abstracts

Im Abstract-Ordner werden alle Tools und Helper definiert, welche über das gesamte Projekt geteilt werden.

Dazu gehören:

* Variables (Farben, Schriften, Default Sizes)
* Mixins
* Functions

### Base

Im Base-Ordner werden grundsätzliche Stylings gemacht.

Dazu gehören:

* Reset (Zurücksetzen von Browserstylesheets)
* Typography (Definieren von Schriftarten, Schrittgrösse, Margin)
* Media (Breakpoint Funktionen)
* Fonts (Schriftimport)

### Bootstrap

Hier werden Bootstrap spezifische Styles überschrieben. Dies geschieht in diesem File, damit Bootstrap bedenkenlos auf eine neuere Version upgedatet werden kann, ohne dass die Styles überschrieben werden.

### Components

In Components werden kleinere Komponenten eingebunden, die an verschiedensten Orten auf der Website wiederverwendet werden.

Typische Komponenten:

* Buttons
* Slider
* Forms

### Layout

In diesem Ordner werden die klassischen Layoutsektionen abgebildet. Hier entscheidet jeder Entwickler selbst, in welche Teile er sein Layout zerlegen möchte. Ich habe mich hier für eine sehr modulare Lösung entscheiden, mit dem Vorteil, dass Files klein und damit einfach überschaubar bleiben.

Gliederung:

* Nav
* Intro
* Content
* About Us
* Register
* News
* Team
* Footer

## Mixins

Mittels Mixins können Codefragmente erstellt werden, welche an verschiedensten Orten wiederverwendet werden. Eine klassische Fliessarbeit, mit der sich jeder Frontendentwickler sicher bereits konfrontiert sah, ist das Präfixen von CSS-Attributen. Die Präfixe sind notwendig, um älteren Browsern CSS3 beizubringen.

### Beispiel CSS anhand von border-radius

Klassisches CSS

.header{  
 -webkit-border-radius: 20px;  
 -moz-border-radius: 20px;  
 border-radius: 20px;  
}

SASS

@mixin borderradius($value) {  
 -webkit-border-radius: $value;  
 -moz-border-radius: $value;  
 border-radius: $value;  
}

.#header{  
 @include borderradius(20px);  
}

Wie im obigen Bespiel erkennbar ist, kann das Mixin für eine beliebige Klasse inkludiert werden. Dies erspart Zeit und mühsame Tipparbeit.

# Angular Frontend

Für die Entwicklung des Frontend’s wurde Angular in der Version 5.0 verwendet. Angular empfiehlt die Verwendung von Typescript. Typescript ist ein Superset von JavaScript d.h. es erweitert die Variablentypen von JavaScript in der Anlehnung an hohe Programmiersprachen wie z.B. Java. Der grosse Vorteil solcher Typisierungen ist zum einen ein stabilerer Code, da schon bei der Kompilierung auf gewisse Fehler aufmerksam gemacht werden kann und zum anderen sind sich viele Programmierer an die Typisierung gewöhnt. Für die Entwicklung mittels Typescript wurde eine neue Dateiendung .ts eingeführt. Nachfolgend wird anhand einer Komponente der grundsätzliche Aufbau von Angular erklärt. Es wird bewusst nur eine Abstraktion gezeigt, um die Komplexität gering zu halten.

## Aufbau

### Klasse

Den Grundbaustein stellt typischerweise die Klasse selbst dar:

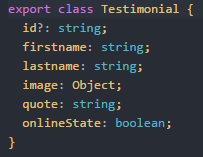


Abbildung 12: Klasse

In der obigen Abbildung ist zu sehen, dass eine Klasse «Testimonial» definiert wird. Innerhalb dieser Klasse werden die Member definiert. Die Klasse wird anschliessend exportiert, um sie an verschiedensten Orten wiederverwenden zu können.

### HTML

Für die Testimonials wurde ein Formular benötigt, um diese in die Datenbank einzutragen. Dieses wurde in der Datei «testimonial-add.component.html» erstellt. Das Formular verwendet die aus Bootstrap bekannten Elemente für den Grundaufbau. Neben dem klassischen HTML können mittels Angular diverse erweiternde Elemente eingefügt werden. Dazu gehören z.B. bedingte Anweisungen, Event-Handler oder Links aus dem Angularrouter.

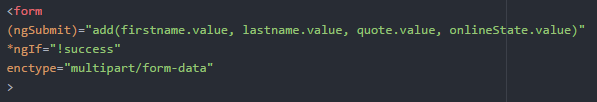


Abbildung 13: Formsubmit

Die Abbildung zeigt wie mittels des Event-Handlers (ngSubmit) das Formular abgeschickt wird. Wird der Submit-Button geklickt, wird die Funktion add() in der dazugehörenden Typescript-Komponente mit den Parametern (Vorname, Nachname, Quote und Onlinestatus) aufgerufen. Mit dem \*ngIf wird ermittelt, ob das Formular bereits abgeschickt wurde, ist dies der Fall, wird das Formular ausgeblendet.

### Typescript

Wie im obigen Kapitel erwähnt wird die Funktion add() aufgerufen. Diese Funktion befindet sich in der Datei «testimonial-add.component.ts» (siehe Anhang s. 17).

Erläuterung:

|  |  |
| --- | --- |
| Zeile | Erklärung |
| 4 | Import der Testimonialklasse (siehe Kap.12.1.1) |
| 5 | Import der Serviceklasse (siehe Kap. 12.1.4) |
| 12 | Error-Member um Fehler im Formular darzustellen |
| 13 | Status um zu ermitteln, ob das Formular erfolgreich abgeschickt wurde |
| 14 | Status um während der Übermittlung zum Server ein Ladeicon anzuzeigen |
| 17 | Initialisierung des Testimonialservices |
| 24 | Add-Funktion |
| 25 | Status des Ladeicons ändern |
| 26 | Erzeugung der Formulardaten, die an den Server geschickt werden |
| 28 | File aus dem Formular auslesen |
| 30-34 | Felder aus dem Formular, dem aus Zeile 26 erstellten Objekt, anhängen |
| 37 | Übermittlung der Formulardaten an den Testimonialservice |
| 39 | Status Ladeicon wieder ändern |
| 40 | Erstellung eines Arrays für allfällige Fehler, welche vom Server zurückgeschickt werden |
| 43 | Sind Fehler vom Server gesendet worden, werden diese in, das in Zeile 12 erstelle Objekt, geschrieben. |
| 45 | Sind keine Fehler entstanden, wird der Status aus Zeile 13 auf true gesetzt und die Abschlussseite angezeigt |

### Service

Der Testimonialservice stellt die eigentliche Verbindung von Angular zum Express-Backend her. Angular stellt das Package @angular/common/http zu Verfügung mit dessen Hilfe HTTP-Aufrufe realisiert werden können.



Abbildung 14: API-URL

Für jeden Service wird die URL zum Express-Backend definiert. Im obigen Beispiel zeigt diese auf die Testimonial-API.

Anschliessend werden die benötigten Funktionen implementiert, um auf das Backend zuzugreifen.

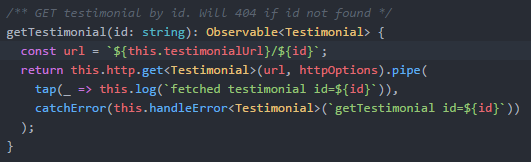


Abbildung 15: Service GET

Wie zu erkennen ist, wird hier mittels der URL und der übergebenen ID die gewünschte URL zusammengestellt. Anschliessend wird mittels HTTP-Get das Testimonial auf dem Server angefragt und zum Client zurückgeschickt.

# Datenbank

## Einleitung

RethinkDB ist wie schon erwähnt eine NoSQL Datenbank. Sie bietet aber dennoch die Möglichkeit Daten mittels Query-Language, wie aus MySQL bekannt, zu joinen. Wird die RethinkDB über die Konsole gestartet, kann über den Port <http://localhost:8080> das Adminpanel aufgerufen werden. Dieses bietet einige nette Features:

* Connection Management
* Perfomance Monitor
* Logfiles
* Dataexplorer

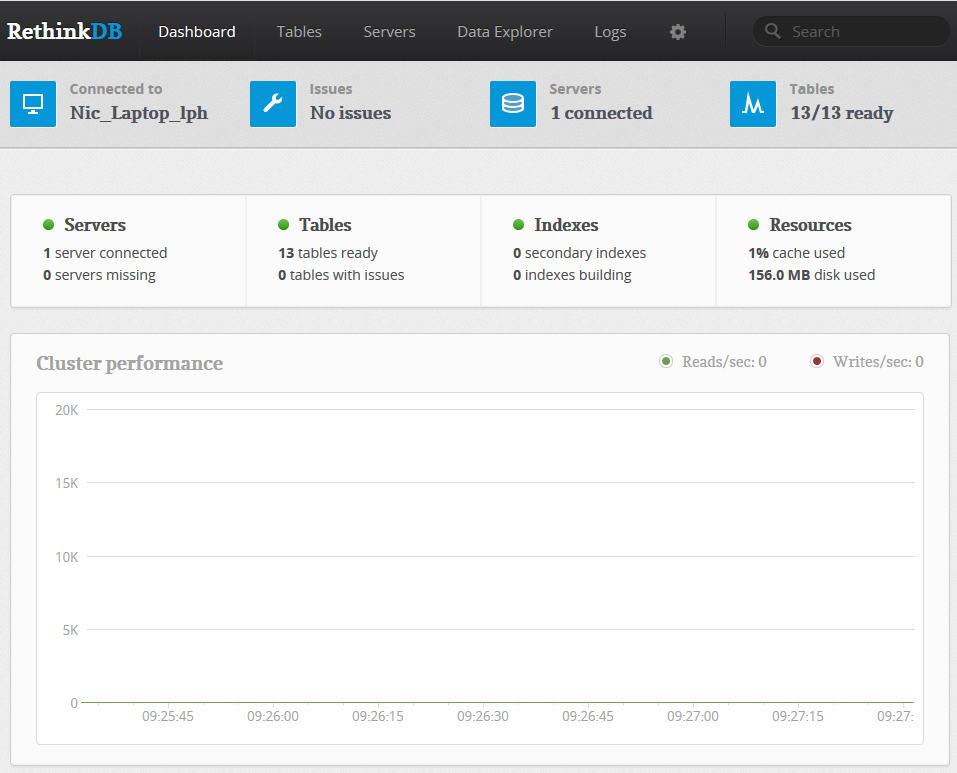
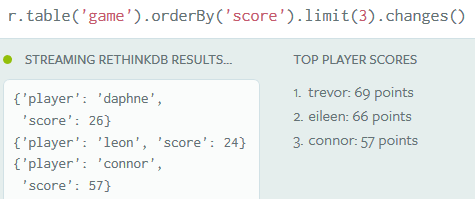


Abbildung 16: Adminpanel RethinkDB

Der grosse Vorteil zeigt RethinkDB allerdings bei Echtzeitanwendungen. Bei herkömmlichen Datenbanken musste jeweils mittels «polling» die Datenbank abgefragt werden, ob eine Änderungen passiert ist. RethinkDB stellt hier eine Funktion zur Verfügung die sich «changes» nennt, mittels dieser Funktion werden Daten zur Echtzeit von der Datenbank an den Client geschickt.



Quelle: https://www.rethinkdb.com/ 1

## Datenbankmodell

Da RethinkDB eine NoSQL Datenbank ist, habe ich mich auch bei der Modellierung des Datenbank Modells auf eine json-artige Dokumentation gestützt (weitere Modelle siehe Anhang s. 18).

Abbildung 17: DB Modell

Auch bei der Modellierung des Datenbankmodells ist mir aufgefallen, dass ich die Query-Sprache von RethinkDB nicht genutzt habe. Ich habe mich komplett für eine flache Hierarchie entschieden.

## Fazit

Was ich anfänglich als grossen Vorteil der RethinkDB evaluiert hatte, stellte sich für mein Projekt als wenig sinnvoll heraus. Zum einen hat das Projekt einfach keinen nützlichen Anwendungsfall für eine Echtzeitumgebung, zum anderen wäre der Einsatz von socket.io notwendig gewesen, was einen grossen Mehraufwand mit sich gebracht hätte, ohne wirklich viel Nutzen zu bringen. In meinen nächsten Projekten, die nicht explizit Echtzeitanforderungen aufweisen, würde ich MongoDB vorziehen. Sie ist weit verbreitet und bietet eine bessere Abstraktion mittels Node-Packages.

# Express Backend

## Einführung

Das Express Package für Node.js ist ein einfach zu benutzendes Framework, zum Erstellen von Webanwendungen. Es bietet diverse Funktionen und Features für Mobile- sowie Webanwendungen. In der folgenden Grafik kann ein einfaches «Hello world» Beispiel betrachtet werden:



Quelle: http://expressjs.com/de/starter/ 1

Wie zu erkennen ist, ist dieser Dienst wirklich sehr einfach zu verstehen. Zuerst wird das Express Package von den Nodemodulen importiert und anschliessend initialisiert. Auf die App-Variable können nun die gängigen CRUD-Funktionen ausgeführt werden (in obigen Bespiel «GET»). Die Funktionsparameter Request und Response beinhalten diverse nützliche Informationen über den Aufruf und die Antwort des Webservers. Mittels res.send() wird eine Antwort an den Client gesendet, in diesem Fall in Plain-Text. Anschliessend wird mittels listen der Port angegeben, auf welche der Webserver operieren soll. Eines der mächtigsten Funktionen von Express ist das Schreiben sogenannter Middlewares. Mit Middlewares können eigene Funktionen geschrieben werden, die beim Aufruf einer beliebigen Route ausgeführt werden. Diverse Packages, die für Express verfügbar sind, basieren auf diesem Konzept. In der nachfolgenden Grafik ein Beispiel:



Quelle: http://expressjs.com/de/guide/writing-mi 1

Im obigen Beispiel ist zu erkennen, wie die Funktion «myLogger» bei jedem Aufruf einer Route ausgeführt wird, mittels app.use() wird sie an Express gebunden.

## Packages

### Multer

Multer ist eine Middleware für Node.js, die es erlaubt sogenannte multipart/form-data zu behandeln. Der Haupteinsatzzweck besteht darin, mittels Formularen Dateien auf den Server zu laden. Im Anwendungsbereich dieses Projektes sind es Bilder und Videos, die auf den Server geladen werden. Die Multer-Konfiguration erlaubt es den Uploadordner sowie den Filenamen zu bestimmen und vieles mehr.

Konfigurationsbeispiel:

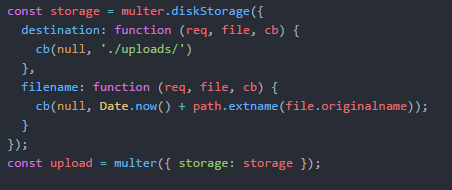
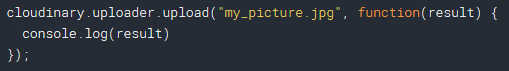


Abbildung 18: Multer

Im obigen Beispiel ist erkennbar, dass der Zielordner für Uploads angegeben wird. Der ursprüngliche Dateiname erhält das aktuelle Datum als Präfix. So wird sichergestellt, dass es keine Konflikte beim hochladen gleicher Dateinamen gibt. Das File wird nur temporär auf dem Server gespeichert (siehe Kap. 14.2.2).

### Cloudinary

Cloudinary ist eine Plattform, mit deren Hilfe Bilder und Videos hochgeladen werden und beim Ausliefern bearbeitet werden können. Cloudinary bietet Programmierschnittstellen in diversen Programmiersprachen an. Die API ist sehr einfach verständlich und einfach zu integrieren. Ein weiterer Vorteil ist, dass Bilder nicht mehr auf dem eigenen Server gehostet werden müssen, sondern direkt auf Cloudinary. Nach dem der API-Key eingetragen wurde, können Bilder wie folgt auf den Server geladen werden:



Quelle: https://cloudinary.com/documentation/nod 1

Das Resultat beinhaltet, nach erfolgreichem Upload, die URL zum Bild sowie weitere nützliche Zusatzinformationen wie Höhe, Breite, Dateigrösse usw.

Der grösste Vorteil von Cloudinary zeigt sich allerdings beim Ausliefern einer Bild- oder Videodatei. Mittels dem Package cloudinary/angular oder direkt per URL-Parameter können nun Bild- und Videodateien transformiert werden.

Transformation Beispiel:



Quelle: https://cloudinary.com/documentation/ima 1

Das obige Beispiel wurde mit folgender Bildtransformation versehen:

<https://res.cloudinary.com/demo/image/upload/w_400,h_400,c_crop,g_face,r_max/w_200/lady.jpg>

w\_400 für die Breite  
h\_400 für die Höhe  
g\_face um das Gesicht zu zentrieren  
usw.

Diese Auswahl an Transformationsmöglichkeiten stellt nur eine kleine Auswahl, an zu Verfügung stehenden Optionen dar, die auf Bilder und Videos anwendbar sind.

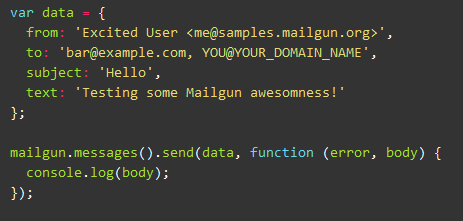
In der Angular Applikation sieht es wie folgt aus:



Abbildung 19: Cloudinary Angular

### Mailgun

Mailgun ist ein Maildienst, mit dem 10'000 Mails pro Monat gratis verschickt werden können. Wie Cloudinary ist auch dieser Dienst sehr einfach zu verwenden, man verifiziert lediglich seine Domain und schon können Mails per API verschickt werden. Zusätzliche nützliche Features sind die Analyse der versendeten E-Mails und die Verifizierung von Mailadressen. Unter der Verifizierung versteht man einen Dienst, mit dem ermittelt werden kann, ob eine E-Mailadresse überhaupt existiert. Ein kleines Bespiel:



Quelle: https://documentation.mailgun.com/en/lat 1

### Express Validator

Der Express Validator ist eine Middleware, die verwendet werden kann, um den Inhalt zu prüfen, der mittels REST-Aufrufs vom Client an den Server geschickt wird. Oft sieht man sich mit der Anforderung konfrontiert, dass ein Formularfeld vom Server validiert werden muss. HTML5 bietet diese Möglichkeit ebenfalls an, aber eben nur auf der Clientseite. Da bekanntermassen die Clientseite manipuliert werden kann, werden die Formularfelder auf der Serverseite nochmals geprüft. Diese Prüfungen können auf Dauer sehr mühsam werden, wenn z.B. E-Mailadressen usw. geprüft werden müssen.

Beispiel:



Abbildung 20 : Express-Validator

Im obigen Beispiel ist zu erkennen, wie das Formularfeld mit dem Name-Attribut «email» geprüft wird. Sollte das Feld keine E-Mailadresse beinhalten, wird die Nachricht «No email» an den Client zurückgeschickt. Selbiges passiert mit dem Passwortfeld hier wird geprüft, ob das Feld nach entfernen von Leerschlägen am Anfang und Ende nicht leer ist.

Sollten Fehler vorhanden sein, werden sie an den Clint zurückgeschickt:

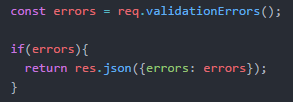


Abbildung 21: Express Fehler

### Registrierung & Login

#### Registrierung

Bei der Registrierung habe ich mich dafür entschieden den kompletten Prozess selbst zu programmieren (siehe Anhang s. 33). Natürlich hätte es hier Libraries gegeben, die einem viele Dinge abgenommen hätten, doch den kompletten Prozess einmal selbst zu schreiben hatte auch seinen Reiz. Nachfolgend werden alle Abläufe per Happy Path aufgezeigt.

Die Registrierung erfolgt mittels Formular (siehe Anhang s.20).

Der Ablauf der Registrierung ist wie folgt:

1. Prüfe, ob alle Felder des Formulars ausgefüllt sind
2. Prüfe, ob E-Mailadresse schon vorhanden ist
3. Passwort verschlüsseln
4. Schreibe User in Tabelle

Mit dem Package «bcrypt» wird das Passwort verschlüsselt in der Datenbank abgelegt, so wird sichergestellt, dass bei einem allfälligen Hackerangriff auf die Datenbank das Passwort nicht «Plain Text» ausgelesen werden kann. Die Übermittlung des Passworts vom Client zum Server geschieht momentan noch über eine unverschlüsselte Verbindung. Bei der Liveschaltung der Website muss die Übertragung mittels HTTPS geschehen, dies war aber aus zeitlichen Gründen nicht Inhalt dieses Projektes.

#### Login

Nach der erfolgreichen Registrierung des Users wird er per Link an die Loginpage weitergeleitet (siehe Anhang s. 21).

Der Ablauf des Logins ist wie folgt:

1. Prüfe, ob alle Felder des Formulars ausgefüllt sind
2. Prüfe, ob die E-Mailadresse in der Datenbank existiert
3. Prüfe, ob das Passwort stimmt
4. Prüfe, ob es sich um eine Member oder Admin handelt
5. Weiterleitung in den richtigen Bereich

Auch hier wird mit dem Package «bcrypt» überprüft, ob das Passwort mit dem eingetragenen und verschlüsselten Passwort in der Datenbank übereinstimmt. Das Package bietet dafür folgende Methode an:

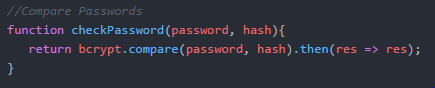


Abbildung 22: bcrypt

Wie zu erkennen ist, wird das «Plain Text Passwort» mit dem verschlüsselten Passwort, hier hash genannt, überprüft. Stimmen die beiden Passwörter überein, wird true zurückgegeben andernfalls false. Ist das Login erfolgreich, wird ein Cookie gesetzt das den User als solchen auf dem Server identifiziert. Das Cookie wird mit dem Package «express-session» auf dem Client gespeichert. Bei jeder Anfrage vom Client an den Server wird das Cookie vom Browser mitgeschickt, sofern eines vorhanden ist.

#### Routes

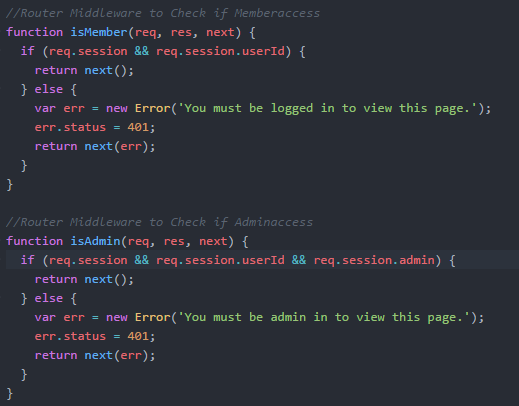
Die Routes auf dem Server sind in drei Bereiche aufgeteilt. Die öffentlichen Routes, die für jeden zugänglich sind, den Member- und den Adminbereich. Versucht nun ein User eine Route aufzurufen die im Member- oder Adminbereich liegt, wird mittels folgender Middleware geprüft, ob das Cookie vom Browser mitgeschickt wurde: 

Abbildung 23: Express Middleware

Wie in der Grafik zu erkennen ist, gibt es eine Funktion für den Admin- und eine für den Memberbereich. Hat der User ein gültiges Cookie, wird er auf die angefragte Route mittels «next» weitergeleitet, ansonsten wird ein Fehler geworfen.

Diese Funktionen werden anschliessend für alle gewünschten Routes eingefügt.

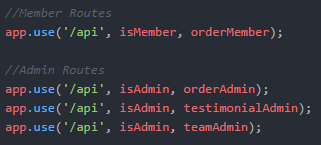


Abbildung 24: Middleware in Router

#### Passwort vergessen

Mit der Passwortvergessen-Funktion können sich die Benutzer selbst ein neues Passwort setzten (siehe Anhang s. 22). Die E-Mailadresse kann derzeit noch nicht geändert werden.

Der Ablauf der Passwortvergessen-Funktion ist wie folgt:

1. Prüfe, ob die E-Maildresse in der Datenbank vorhanden ist
2. Schreibe Eintrag in die Tabelle «forgotpassword» mit den Feldern E-Mail, Hash und Aktivierungsstatus
3. Sende Mail an User mit Aktivierungslink (siehe Anhang s. 23)
4. User kann mittels Link ein neues Passwort setzen (siehe Anhang s. 23)
5. Wird der Link aktiviert und ein neues Passwort gesetzt, verfällt der Link

Wichtig zu erwähnen ist, dass das momentane System noch eine Sicherheitslücke aufweist. Derzeit wird keine Überprüfung mittels Token bei der REST-API durchgeführt. Aus zeitlichen Gründen konnte dieses Feature noch nicht implementiert werden. Dieses muss jedoch vor der Liveschaltung programmiert werden, da ansonsten Daten von einem beliebigen Client verändert werden können.

# Testen

Angular bietet von Haus aus eine integrierte Testumgebung an. Ich habe mich dafür entschieden nicht den von Haus aus integrierte Unit-Test «Karma-Jasmine» zu verwenden, sondern das Package «Mocha-Chai». Mocha ist das eigentliche Testframework, wohingegen Chai für Assertions eingesetzt wird. Grundsätzlich bieten beide Packages nahezu den gleichen Funktionsumfang, es ist lediglich eine Frage des Geschmacks.

Aus zeitlichen Gründen legte ich das Hauptaugenmerk des Tests vor allem auf die REST-API. Das Angular-Frontend wurde nicht getestet. Natürlich würde es Sinn machen dieses auch zu testen, was allerdings im Rahmen dieses Projekt nicht realisiert werden konnte.

## Setup

Zuerst werden die benötigten Packages (Chai, Chai-http, mocha und Supertest) als Entwicklungs-Dependencies installiert, da sie nur für die Entwicklung benötigt werden.

Im nächsten Schritt wird ein Skript in das package.json eingefügt, das den Test startet.

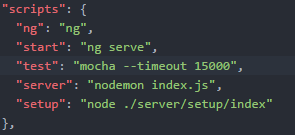


Abbildung 25: Start Skripts

Nun kann der Befehl npm run test in der Konsole ausgeführt werden, welcher sämtliche Unit-Tests durchläuft. Um Komplikationen mit langen Antwortzeiten seitens des Servers zu vermeiden, wird ein timeout von 15 Sekunden gesetzt. Mocha sucht nun automatisch nach einer test.js Datei (kann auch anders konfiguriert werden).

In der test.js Datei wird nun die Node-Umgebungsvariable auf «test» gesetzt:

process.env.NODE\_ENV = 'test';

Im Config-Ordner wir das test.json geladen, welches wiederum die Datenbank auf die Testdatenbank umschaltet. Mit der Testdatenbank können bedenkenlos Operationen ausgeführt werden, ohne die Datenintegrität zu gefährden.

## Unit-Test

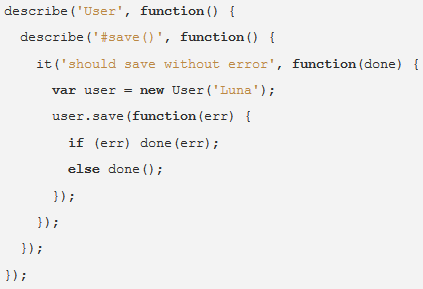
Der Mocha Unit-Test stellt alle gängigen Funktionen zur Verfügung, die man aus den üblichen Testframeworks kennt. Dazu gehören z.B. die Hooks:

* Before
* After
* Before Each
* After Each

Diese Hooks können dazu verwendet werden, um vor dem Test etwas in die Datenbank zu schreiben bzw. nach dem Test die Datenbank wieder zu leeren.

In der JavaScript-Welt sieht man sich sehr oft mit asynchronen Aufrufen konfrontiert. Auch hier hat Mocha ein passendes Instrument bereitgestellt. Mittels der Funktion «it» wird dem Test beigebracht, dass hier ein asynchroner Aufruf stattfindet. Anschliessend wird die Funktion «done» aufgerufen, um den Test abzuschliessen.

Ein Beispiel:



Quelle: https://mochajs.org/#hooks 1

Wie zu erkennen ist, wird hier ein asynchroner Aufruf gemacht, der einen User speichert. Anschliessend wir die Funktion «done» aufgerufen um den Test abzuschliessen.

## Assertation

Das Package Chai stellt diverse Funktionen zur Verfügung, mit denen Werte verglichen werden können. Mit Chai-HTTP können HTTP-Aufrufe simuliert werden.

Beispiel:

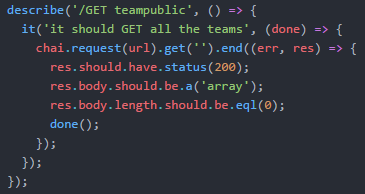
  
Wie zu erkennen ist, wird mittels chai.request() ein HTTP-Aufruf simuliert. Anschliessend werden mit should die Werte, die vom Aufruf erwartet werden, auf ihre Korrektheit überprüft.

Abbildung 26: Test

# Risiko- und Erfolgsfaktoren

## Risiken

Je nach gewählter Technologie besteht das Risiko, vor allem bei neuen Technologien, dass noch sehr wenig Dokumentation vorhanden ist.

Es muss sichergestellt werden, dass der Anmelde- und Registrierungsprozess richtig funktioniert und keine Daten an Dritte gelangen.

Die Datenbank muss gegen SQL-Injection’s gesichert werden, um ebenfalls Manipulation durch Dritte zu verhindern.

Eingesetzte Packages sollen in vernünftigen Abständen Updates erhalten, um die Qualität der Website nicht unnötig zu gefährden.

Ein Risiko, das für TheCaliCamp besteht, ist die Nutzung der Plattform. Sie müssen sicherstellen, dass die Leute überhaupt von der Plattform erfahren und diese auch Nutzen.

## Erfolgsfaktoren

Die Website lässt sich durch ihr responsive Design auf gängigen Endgeräten gut bedienen.

Durch die neue Website sollen sich Surfguides über die Wintermonate etwas dazu verdienen können. Sie haben somit eine zweite Einnahmequelle über den Winter.

TheCaliCamp Teaches soll ein aussergewöhnliches Angebot für Surfer aus der ganzen Welt sein. Der Erfolg der Website ist massgeblich durch die Qualität, der Kommentare von unseren Surfguides abhängig. Die Plattform soll lediglich den Rahmen bieten.

Das Endprodukt soll für User und Admins einfach verständlich sein. Das heisst es soll für den User selbsterklärend sein, wie er Videos auf die Plattform hochladen kann.

# Summary

Grundsätzlich konnte die Arbeit im gegebenen Zeitrahmen fertiggestellt werden, was für den Erfolg des Projektes spricht. Was ich jedoch festgestellt habe, ist, dass der Projektumfang bereits im Projektantrag sehr gross war. In einem zukünftigen Projekt würde ich den Umfang etwas kleiner ansetzen, um mehr Zeit für die einzelnen Funktionen zu haben.

Bei der Implementierung der einzelnen Funktionen kristallisierten sich so einige Schwierigkeiten heraus. Das grösste Problem war, dass ich mich bei einer schnelllebigen Technologie darauf verlassen habe, dass keine grossen Änderungen stattfanden: Konkret React. Hier wurde für das Backend ein völlig neues Konzept eingeführt, dass für mich schlicht und einfach nicht verständlich war (Redux). Glücklicherweise war Angular 5 für mich relativ einfach erlernbar und ich konnte es gut auf mein Projekt adaptieren. Die Umsetzung des HTML und CSS mittels Bootstrap und SASS ging mir sehr leicht von der Hand. Bei Bootstrap wurde in der Version 4.0 Beta so einiges positiv verändert, was dem Entwickler zugutekommt. Mit der schon erwähnten Einführung von Flexbox ist das Positionieren von Elementen viel einfacher und dynamischer geworden. Mit SASS habe ich ebenfalls in der Vergangenheit schon sehr gute Erfahrungen gemacht.

Bei der Datenbank habe ich mich für die RethinkDB entschieden. Bei der Technologieevaluation erschien mir die Funktion, dass die Datenbank Änderungen an den Client schicken kann, sehr spannend. Wie sich aber im Verlauf des Projektes herausstellte, wäre die Nutzung dieser Funktion zum einen sehr zeitaufwendig gewesen und zum anderen machte sie für meine Applikation schlicht keinen Sinn. Bei einem Chat oder Onlinespiel wäre dieses Feature sicherlich besser nutzbar gewesen. In Zukunft würde ich mich also eher für eine Lösung mit MongoDB oder MySQL entscheiden. Diese beiden Technologien sind um einiges besser dokumentiert. Ebenso bestehen für die erwähnten Datenbankanbindungen diverse Packages, welche die Komplexität besser abstrahieren.

Beim Backend konnte ich viel von meinen vergangenen Erfahrungen mit einbringen. Die Auswahl von geeigneten Packages beruhte Grossteiles aus positiven Erfahrungen bei früheren Projekten. Das Einbinden von Librariers und das Verwenden von externen Diensten wie cloudinary, mailgun oder multer waren verhältnismässig einfach.

Für den kompletten Registrations- und Loginprozess habe ich mich aus Neugier dafür entschieden, alles selbst zu programmieren. Den kompletten Ablauf selbst zu programmieren war sehr spannend, allerdings würde ich mich in der Zukunft dagegen entscheiden. Es gibt diverse Packages wie z.B. passport.js, die einem das Leben erheblich vereinfachen. Sie bieten diverse Funktionen bereits an wie z.B. die Verschlüsselung des Passwortes oder die Passwortvergessen-Funktion. Zudem können sehr einfach weitere Authentifizierungsdienste wie z.B. Facebook-Login eingefügt werden. Ein weiterer Aspekt ist, dass einem solche Libraries einen Grossteil sichertechnischer Aspekte abnimmt.Beim Testen habe ich mich dafür entschieden nur das REST-API zu testen. Wie schon erwähnt, würde es natürlich ebenfalls Sinn machen das Angular-Frontend zu testen, da auch dieses eine potenzielle Fehlerquelle ist.

Alles in allem machte die Umsetzung des Projektes aber sehr viel Spass. Ich habe für mich persönlich einiges an Knowhow für meine weiteren Projekte mitgenommen.

# Glossar

|  |  |
| --- | --- |
| Admin | Administrator |
| Agile | beweglich |
| API | Programmierschnittstelle |
| Assertation | Vergleich |
| Asynchron | nicht gleichzeitig |
| Backend | Unterbau eines Systems |
| Breakpoint | Umbruch |
| Browser | Programm zur Darstellung von Webseiten |
| BSON | Binäres Dateiformat |
| Bug | Fehler |
| Build | Erstellungsprozess |
| Built-In | Eingebaut |
| CLI/Terminal | Kommandozeile |
| Client | Rechner oder Software die auf Daten zugreift |
| Collection | Sammlung von Daten |
| Community | Gemeinschaft |
| Config | Konfiguration |
| Connection | Verbindung |
| Cookie | Enthält Daten von besuchten Webseiten |
| CRUD | Create Read Update Delete |
| CSS | Cascading Style Sheets |
| DB | Datenbank |
| Default | Standard |
| Dependencie | Abhängigkeit |
| Development | Entwicklung |
| Dist | Ausgabeordner des Erstellungsprozesses |
| DOM | Document Object Model |
| DQL | Data Query Language |
| Error | Fehler |
| Evaluation | fachgerechte Untersuchung |
| Event-Handler | Auslöser einer Aktion |
| Feature | Eigenschaft |
| Float | CSS-Anweisung zur Ausrichtung von Elementen |
| Fonts | Schrift |
| Framework | Programmiergerüst |
| Frontend | Überbau eines Systems |
| Gebundelt | Gebündelt/Zusammengefasst |
| Gelayert | aufgeteilt |
| Generics | generische Elemente |
| GIT/SVN | Versionensystem |
| GitHub | Versionensystem |
| Happy Path | Optimaler Ablauf |
| Hash | Verschlüsselter Wert |
| Hook | Einschubmethode |
| HTML | Hypertext Markup Language |
| HTTP | Hypertxt Transfer Protocol |
| IDE/Atom/Editor | Entwicklungsumgebung |
| Intervall | Zwischenraum zwischen zwei Vorgängen |
| Join | Verbindung |
| JSON | Datenformat |
| JSX | Dateiformat von React |
| Key | Schlüssel |
| Konsistent | In Zusammenhang stehend |
| Lazy Loading | Datenobjekte werden erst bei konkreter Anfrage geladen |
| Member | Mitglied |
| Memory | Speicher |
| Middleware | Schnittstelle in Express |
| Mixin | Wiederverwendbares Element |
| Mockup | Attrappe |
| Multipart/formdata | Formdaten mit Dateien |
| Ng | Konsolenbefehl für Angular |
| Node.js | serverseitige Plattform in der Softwareentwicklung zum Betrieb von Netzwerkanwendungen |
| Node\_modules | Ordner für Node Packages |
| NoSQL | Not only SQL |
| Npm | Konsolenbefehl für Package Manager |
| Open-Source | Freie Software |
| Package | Erweiterung |
| Package Manager | System zur Verwaltung von Erweiterungen |
| Panel | Umgebung |
| Path | Pfad |
| Performance | Leistung |
| PHP | Programmiersprache |
| Plain Text | Nicht verschlüsselt |
| Plug-in | Zusatzmodul |
| Polling | Zeitlich gesteuerte Anfrage von Daten |
| Port | Teil einer Netzwerk-Adresse |
| Pushen | Weitergeben |
| React/Angular/Vue.js | Grundgerüst für die Ausgabe von User-Interface-Komponenten |
| Relational | Tabellarischer Verbindung |
| Repository | Depot auf Server |
| REQL | Rethink Query Language |
| Responsive | Passt sich dem Endgerät an |
| REST | Representational State Transfer |
| RethinkDB/MySQL/MongoDB | Datenbank |
| Routing | Wegfindung in Netzwerk |
| Sass/SCSS | Syntactically Awesome Stylesheets |
| Skript | Ausführbare Datei |
| Socket.io | Librarie für Echtzeitanwendungen |
| SPA | Single Page Application |
| Superset | Erweiterung |
| Syntax | Schlüsselwörter einer Programmiersprache |
| Threads | Prozess |
| Toggl | Programm zur Zeiterfassung |
| Typescript | Typisierung innerhalb von Javascript |
| URL | Uniform Resource Locator |
| User | Benutzer |
| Valide | Gültig |
| Viewport | Ausschnitt eines Bildes |
| Webpack | Bündelt Dateien |

# Quellenverzeichnis

|  |  |
| --- | --- |
| <https://angular.io/> |  |
| <https://cloudinary.com/> |  |
| <https://db-engines.com/de/system/MongoDB%3BRethinkDB> | |
| <https://de.wikipedia.org/wiki/Angular> | |
| <https://de.wikipedia.org/wiki/Apache_HTTP_Server> | |
| <https://de.wikipedia.org/wiki/Apache_Subversion> | |
| <https://de.wikipedia.org/wiki/GitHub> | |
| <https://de.wikipedia.org/wiki/Integrierte_Entwicklungsumgebung> | |
| <https://de.wikipedia.org/wiki/Nginx> | |
| <https://de.wikipedia.org/wiki/React> | |
| <https://de.wikipedia.org/wiki/RethinkDB> | |
| <https://github.com/angular/angular-cli> | |
| <https://github.com/cloudinary/cloudinary_angular> | |
| <https://github.com/expressjs/multer> | |
| <https://github.com/webpack/webpack> | |
| <https://nodejs.org/en/about/> | |
| <https://reactjs.org/> |  |
| <https://redux.js.org/docs/introduction/> | |
| <https://sass-guidelin.es/#main-file> | |
| <https://socket.io/> |  |
| <https://vuejs.org/> |  |
| <https://www.mongodb.com/de> | |
| <https://www.mysql.com/de/> | |
| <https://www.npmjs.com/> | |
| <https://www.npmjs.com/package/config> | |
| <https://www.rethinkdb.com/> | |

# Abbildungsverzeichnis

[Abbildung 1: Projektplan 9](#_Toc502948704)

[Abbildung 2: Projekorganisation 10](#_Toc502948705)

[Abbildung 3: RethinkDB Terminal 11](#_Toc502948706)

[Abbildung 4: Zeiterfassung 14](#_Toc502948707)

[Abbildung 5: Angular-Router 25](#_Toc502948708)

[Abbildung 6: Spalten auf Desktop 28](#_Toc502948709)

[Abbildung 7: Spalten auf Mobile 28](#_Toc502948710)

[Abbildung 8: Formular 29](#_Toc502948711)

[Abbildung 9: Responsive Tabelle 29](#_Toc502948712)

[Abbildung 10: Navigation Desktop 30](#_Toc502948713)

[Abbildung 11: Navigation Mobile 30](#_Toc502948714)

[Abbildung 12: Klasse 33](#_Toc502948715)

[Abbildung 13: Formsubmit 33](#_Toc502948716)

[Abbildung 14: API-URL 34](#_Toc502948717)

[Abbildung 15: Service GET 35](#_Toc502948718)

[Abbildung 16: Adminpanel RethinkDB 36](#_Toc502948719)

[Abbildung 17: DB Modell 37](#_Toc502948720)

[Abbildung 18: Multer 39](#_Toc502948721)

[Abbildung 19: Cloudinary Angular 40](#_Toc502948722)

[Abbildung 20 : Express-Validator 41](#_Toc502948723)

[Abbildung 21: Express Fehler 41](#_Toc502948724)

[Abbildung 22: bcrypt 42](#_Toc502948725)

[Abbildung 23: Express Middleware 42](#_Toc502948726)

[Abbildung 24: Middleware in Router 43](#_Toc502948727)

[Abbildung 25: Start Skripts 44](#_Toc502948728)

[Abbildung 26: Test 45](file:///C:\workspace\angular-workspace\thecalicampteaches\documentation\Doku.docx#_Toc502948729)