



Klima-Viewer

Abschlussdokumentation

Projektteam KlimaViewer

Ivo Kozina
Ohran Mujkic

Version 1.0
Bern, 19. Januar 2019

Inhaltsverzeichnis

1	ZWECK DES DOKUMENTS	3
2	PROJEKTKONTEXT	3
3	RAHMENBEDINGUNGEN, ORGANISATORISCHER KONTEXT	3
4	BEWERTUNG DER FUNKTIONALEN ANFORDERUNGEN	4
5	FRONTEND	5
5.1	LIVEWEATHER – ANSICHT	5
5.2	HISTORICAL – ANSICHT	6
5.3	MATERIAL	7
6	BACKEND	8
6.1	GESAMTÜBERSICHT ALLER KOMPONENTEN	8
6.2	LIVEWEATHER – STANDORTE ANZEIGEN	9
6.3	LIVEWEATHER – STANDORT HINZUFÜGEN	9
6.4	HISTORICAL – DATEN ANZEIGEN	10
6.5	HISTORICAL – DATENBESCHAFFUNG	10
7	DATENMODELL	11
7.1	DATENBANKMODELL	11
7.2	KLASSENDIAGRAMM	12
8	TESTING – PROTRACTOR	13
8.1	SETUP	13
8.2	RESULTATE	13
9	HERAUSFORDERUNGEN	14
9.1	WORLDCLIM	14
9.2	OPENWEATHERMAP	14
10	PROJEKTORGANISATION	15
10.1	PROJEKTSTRUKTUR	15
10.2	VERSIONISIERUNG	16
10.3	NAMENSKONVENTIONEN	16
11	AUSBlick	17
12	GLOSSAR	18
13	ABBILDUNGSVERZEICHNIS	19
14	ANHANG	20
14.1	SCREENSHOTS DER APPLIKATION	20
14.2	PROJEKTBESCHREIBUNG	22
15	VERSIONSKONTROLLE	23

1 Zweck des Dokuments

Der Hauptzweck dieses Dokument ist es, die erreichten Ziele sowie die Anforderungen für das Projekt „KlimaViewer“ aufzuführen und zu vergleichen, sowie die verschiedenen Arbeitsschritte und Entscheidungen näher zu bringen.

2 Projektkontext

Mit dem Projekt wird eine Webapplikation entwickelt, die es möglich macht Klimaveränderungen über einen bestimmten Zeitraum für die Nutzer verständlich, informativ sowie attraktiv darzustellen. Dabei soll der Verlauf der Klimaveränderung in grafischen Darstellungen erfolgen. Die grafischen Darstellungen können ohne bestimmte Vorkenntnisse interpretiert werden. Die Nutzer dieser Webapplikation können mit echten Klimadaten verschiedene Auswertungen und Trends erstellen. Zudem ermöglicht die Webapplikation das Aufrufen von aktuellen Wetterdaten. Somit wird ein neues Tool entwickelt, welches auf den bisherigen gesammelten Daten aufbaut. Das Tool soll als Prototyp dienen und keine produktive und vollendete Software darstellen.

3 Rahmenbedingungen, organisatorischer Kontext

Die Realisierung für dieses Projekt erfolgt im Rahmen des Moduls „Projekt 1“. Die Umsetzung erfolgte hauptsächlich mit der Technologie JavaScript, da wir uns auf das MEAN Stack stützten. Die Lösung ist auf einer virtuellen Maschine im BFH Netz angesiedelt und steht über eine URL zur Nutzung.

4 Bewertung der Funktionalen Anforderungen

Hier werden alle funktionalen Anforderungen angezeigt, um aus diesen ein Fazit zu ziehen.

Attribute:

ID: eindeutige Identifikation

Status: Entwurf / Geprüft / Freigegeben

Priorität: Muss / Optional P1, P2, P3 / Wunsch (Nice to have)

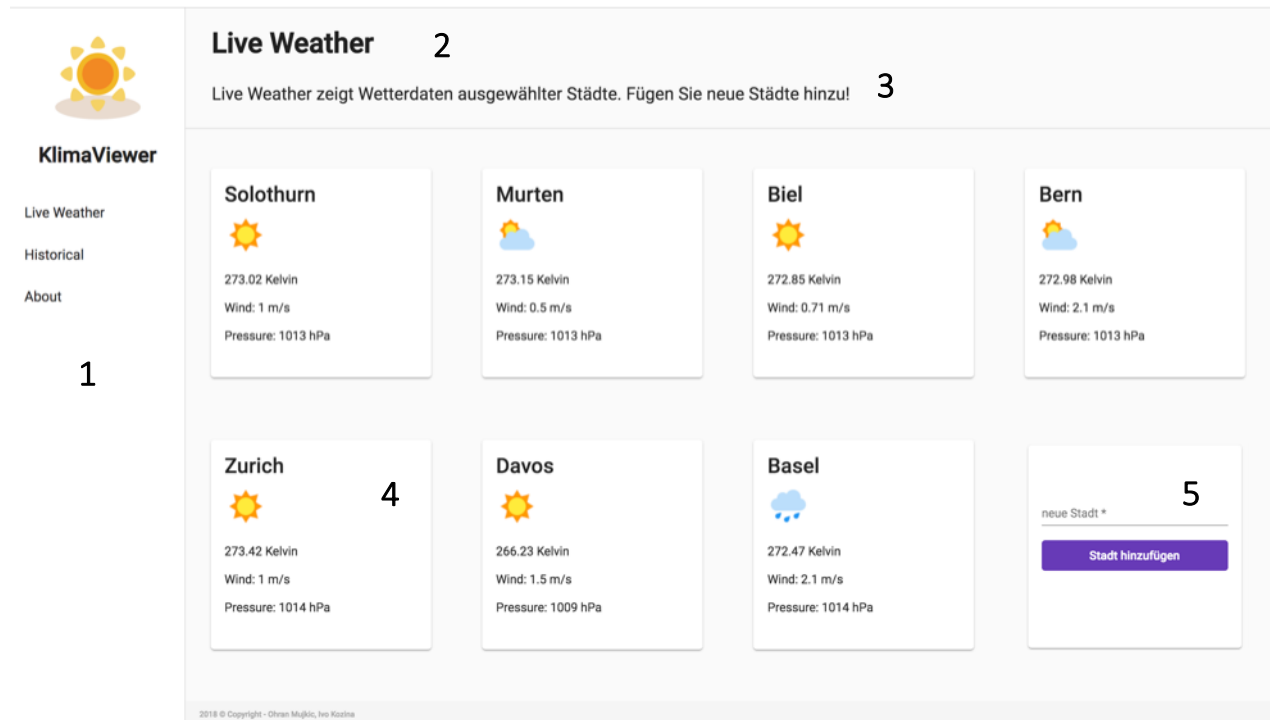
ID	Status	Priorität	Beschreibung
F1	Freigegeben	M	Der Benutzer kann die Applikation in einem Webbrowser öffnen.
F2	Freigegeben	M	Der Benutzer kann für verschiedene Kennzahlen (Temperatur, Windgeschwindigkeit, Niederschlag sowie Luftdruck) über einen bestimmten Zeitraum in der Vergangenheit eine graphische Darstellung erhalten, welche sich hauptsächlich auf die Veränderung fokussiert. Diese Daten kann er auch als CSV-Datei exportieren.
F3	Freigegeben	M	Der Benutzer kann für verschiedene Städte die aktuellen Wetterdaten (Temperatur, Windgeschwindigkeit, Niederschlag sowie Luftdruck) erfahren.
F4	Freigegeben	M	Der Benutzer kann eine neue Stadt hinzufügen und diese wird in der Webapplikation hinterlegt und zusätzlich in der Datenbank weiter getrackt.

ID	Fazit
F1	Da die Applikation auf einer VM im BFH Netz läuft und mit einem Webserver aufgebaut wurde, ist es dem Benutzer möglich, die Applikation in einem Webbrowser aufzurufen. Dieser muss sich jedoch im BFH Netz befinden.
F2	Diese Anforderung wurde im Teil „Historical“ der Applikation umgesetzt. Die nötigen Daten dazu kann man bei OpenWeatherMap herunterladen. Da dies jedoch kostenpflichtig ist, haben wir auf unserer VM einen Cronjob erstellt, welcher diese Daten über die OpenWeatherMap API holt und in unserer mongoDB speichert. In unserer Applikation rufen wir dann diese Daten auf und verwenden sie für den graphischen Chart.
F3	Diese Anforderung wurde im Teil „Live Weather“ der Applikation umgesetzt. In der Übersicht Live Weather ist es möglich die Wetterinformationen von selbst hinzugefügten Städten anzuzeigen. Die Städte sind auf die Schweiz begrenzt.
F4	Unabhängig vom Benutzer kann jeder eine Schweizer Stadt hinzufügen und diese wird danach in der Datenbank hinterlegt und durch den Cronjob getrackt.

5 Frontend

Eine Anforderung war, dass die Applikation ansprechend und leicht zu bedienen ist. Deshalb werden hier kurz die Gedanken beim Erstellen der Designs der beiden Hauptseiten **LiveWeather** und **Historical** beschrieben.

5.1 LiveWeather – Ansicht



Nr.	Beschreibung
1	Als Navigation war im Mockup ein Menu gewählt, auf der Seite oben und zentriert. Beim Umsetzen merkten wir jedoch, dass sich eine Sidebar an der Seite viel mehr eignet. Dadurch wurde die Seite in zwei Teile geteilt was dazu führte, dass man stets den Überblick über die anderen Links hatte und zum anderen konnte dadurch der Inhalt übersichtlicher dargestellt werden.
2	Auf jeder Seite steht oben, wo man sich gerade befindet.
3	Unter dem Titel der Seite wird auch die Funktion dieser kurz beschrieben.
4	Die gewählten Städte befinden sich in Kartenähnlichen Boxen. Diese wurden so programmiert, dass sie skalierbar sind und sich an die Seite anpassen.
5	Neue Standorte können hinzugefügt werden.

5.2 Historical – Ansicht



Nr. Beschreibung

- | | |
|---|--|
| 1 | Der Benutzer kann seine gewünschten Eingaben machen, um den dazugehörigen Graphen zu erhalten. |
| 2 | Anhand der Eingaben des Benutzers wird ein Graph erstellt, welcher die Wetterinformationen im gewählten Zeitabstand darstellt. |
| 3 | Von den ausgewählten Daten kann auch ein Export erstellt werden, welcher beispielsweise im Excel geöffnet werden kann. |

5.3 Material

Für die Frontend-Umsetzung stützen wir uns auf das Material Framework Version 7.1.1. Es bietet viele vorgegebene Elemente, welche auch zum Teil ein vordefiniertes Aussehen besitzen. Dies haben wir uns zu Nutze gemacht und an unser Projekt angepasst.

```
<mat-grid-tile class="myCard"
  *ngFor="let location of locations"
  [colspan]="1"
  [rowspan]="1"
  [style.background]="transparent">
  <mat-card>
    <mat-card-title>{{ location.name }}</mat-card-title>
    <mat-card-content>
      <img [src]="location.iconPath">
    </mat-card-content>
    <mat-card-content *ngIf="location.response">
      {{location.response.main.temp}} Kelvin
    </mat-card-content>
    <mat-card-content *ngIf="location.response">
      Wind: {{location.response.wind.speed}} m/s
    </mat-card-content>
    <mat-card-content *ngIf="location.response">
      Pressure: {{location.response.main.pressure}} hPa
    </mat-card-content>
    <mat-card-content *ngIf="location.response.precipitation">
      Precipitation: {{location.response.main.pressure}} mm
    </mat-card-content>
  </mat-card>
</mat-grid-tile>
```

Als Beispiel für Material wird oben die Darstellung der Standorte angezeigt. Elemente die von Material gestyled werden fangen in der Regel mit ‚mat‘ an. So auch <mat-card>, welches eine Kartenähnliche Box darstellt. Mithilfe von Angular kann eine For-Schleife mit *ngFor aufgerufen werden und somit alle gewünschten Städte als Karten anzeigen.

6 Backend

Das Backend und natürlich die ganze Applikation besteht aus mehreren Komponenten, welche hier dargestellt und beschrieben werden.

6.1 Gesamtübersicht aller Komponenten

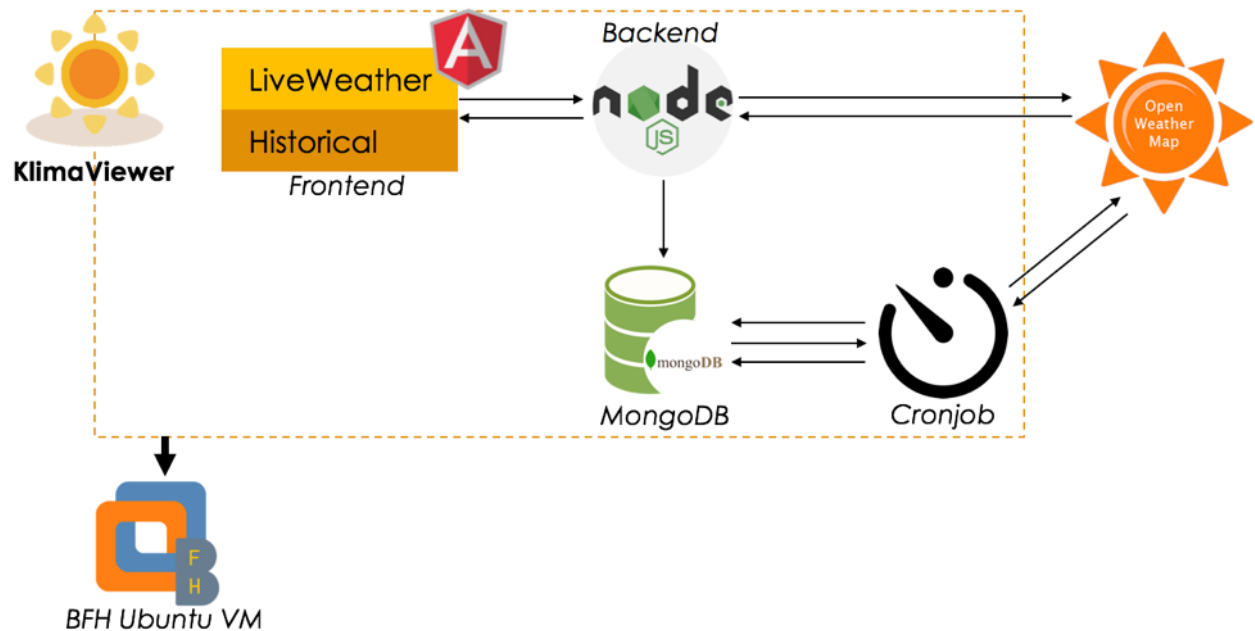









Abbildung 1: Gesamtübersicht aller Komponenten

Icon	Bezeichnung, Funktion	Version
	KlimaViewer, stellt die ganze Applikation dar.	-
	Angular, Frontend, wird mit ng serve gestartet	7.1.0
	NodeJS, Backend, wird mit nodemon server.js gestartet	8.9.4
	MongoDB, Datenbank, wird mit mongod gestartet	7.1.1
	Cronjob, Datensammlung	8.9.4
	VM, virtuelle Maschine als Host http://147.87.116.18:4200 (innerhalb des BFH-Netzes)	Ubuntu
	OpenWeatherMap, externe Datenquelle	-

6.2 LiveWeather – Standorte anzeigen

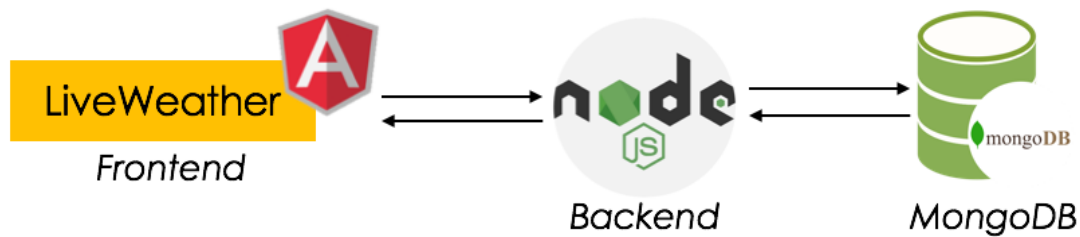


Abbildung 2: LiveWeather - Standorte anzeigen

In der Ansicht LiveWeather werden alle vom Benutzer einmal gewählten Standorte mit den aktuellen Wetterinformationen angezeigt. Das Backend holt dazu den aktuellsten Datensatz aus der Datenbank und gibt sie dem Frontend zurück.

6.3 LiveWeather – Standort hinzufügen

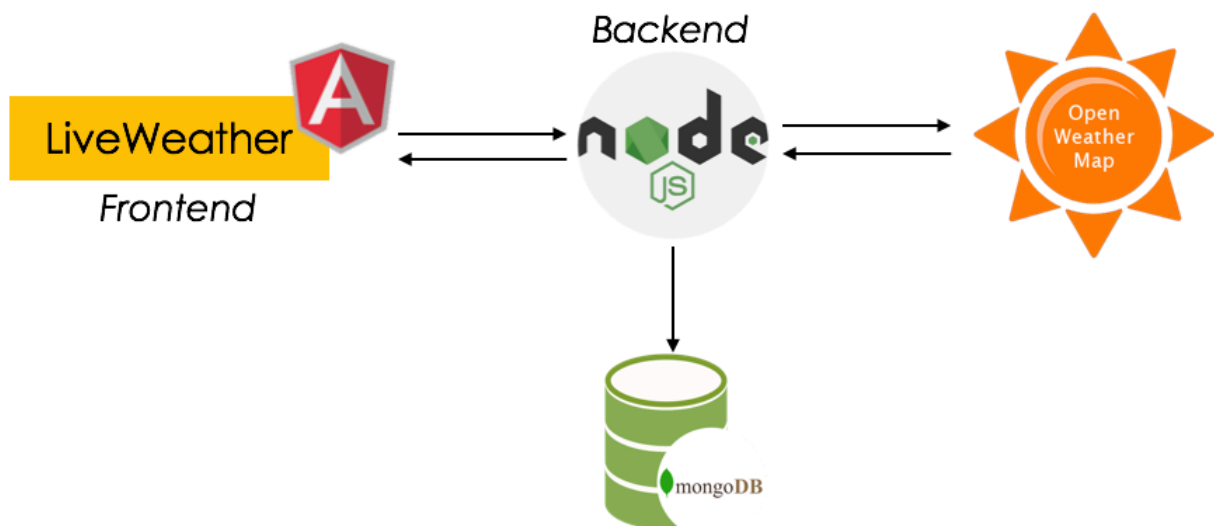


Abbildung 3: LiveWeather - Standort hinzufügen

Falls der User einen neuen Standort hinzufügt, holt das Backend einen aktuellen Datensatz von der OpenWeatherMap API, speichert diesen in die Datenbank und gibt den aktuellen Stand dem Frontend zurück. Diese Stadt ist nun auch im Cronjob hinterlegt, somit werden in Zukunft weiter alle drei Stunden Wetterinformationen gesammelt.

6.4 Historical – Daten anzeigen

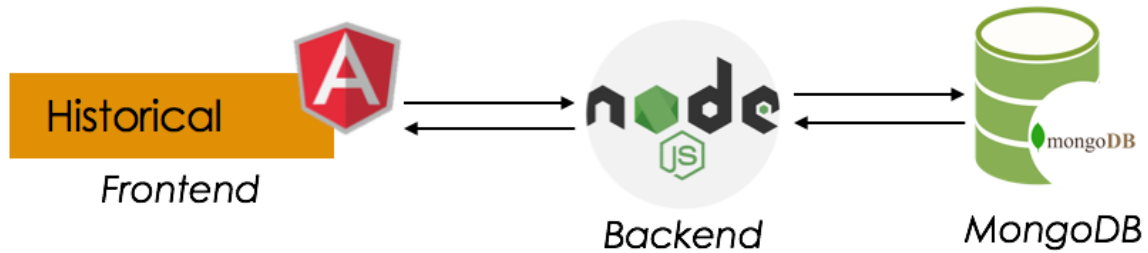


Abbildung 4: Historical - Daten anzeigen

Im Historical werden die gesammelten Daten in der Datenbank graphisch angezeigt. Dazu holt das Backend die vom Benutzer gewählten Informationen und gibt sie dem Frontend zurück.

6.5 Historical – Datenbeschaffung

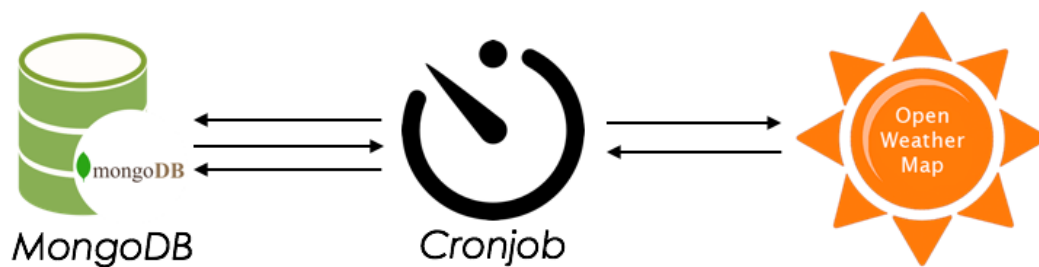


Abbildung 5: Historical - Datenbeschaffung

Um Daten sammeln zu können, fragt der Cronjob nach in der Datenbank nach den gewählten Standorten und holt sich dann alle drei Stunden Wetterinformationen über die OpenWeatherMap. Die Datensätze speichert er in die Datenbank, damit sie für die Historical Ansicht gebraucht werden können.

7 Datenmodell

7.1 Datenbankmodell

In der Datenbank werden die gesammelten Daten vom Cronjob oder bei neuen Standorten vom Backend gespeichert. Ein Standort (=Location) besitzt mehrere Datensätze, nämlich die welche alle drei Stunden vom Cronjob geholt werden. Diese Datensätze beinhalten die gewünschten Wetterinformationen wie Temperatur, Windgeschwindigkeit oder Luftdruck. Die einzige Veränderung ist die neue Kolonne «response», die die komplette Antwort von openWeatherMap speichert, damit man bei Erweiterung des Projekts mit weiteren Kennzahlen schon auf getrackten Daten zurück greifen kann.

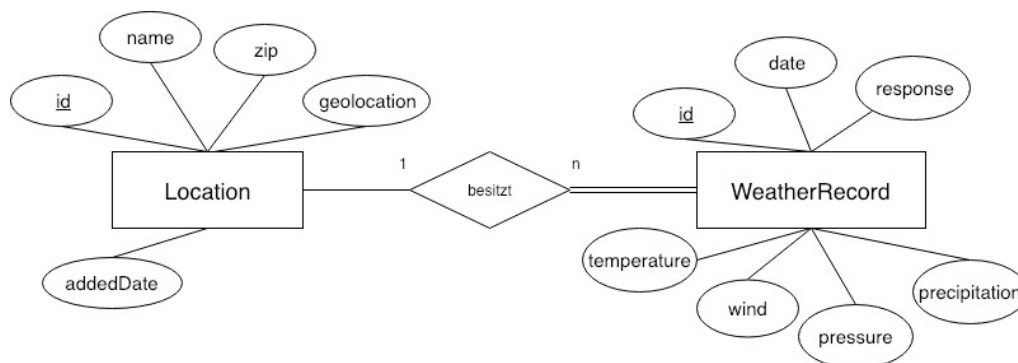


Abbildung 6: ERD

```
function sendRequest(name) {

request(`https://api.openweathermap.org/data/2.5/weather?q=${name},ch&APPID=c5ff046ef910a43225f16e306180c09`, function (error, response, body) {
    if (!error && response.statusCode == 200) {
        saveData(response);
    } else {
        console.log(error);
    }
});
}

function saveWeatherRecord(weatherRecord) {
    MongoClient.connect(url, function (err, db) {
        if (err) throw err;
        dbConn = db.db("ClimaViewer");
        dbConn.collection("WeatherRecord").insertOne(weatherRecord, function (err, res)
        {
            if (err) throw err;
        });
        db.close();
    });
}
```

Im Obigen Codeschnipsel kann man in der Funktion **sendRequest** sehen, wie der Cronjob einen API-Aufruf macht und mit der Funktion **saveWeatherRecord**, die vom Request erhaltenen Informationen in die Datenbank speicher.

7.2 Klassendiagramm

Hier ist das Klassendiagramm der Applikation sichtbar. Es wurde für das Frontend und das Backend eine einzelne Ansicht erstellt.

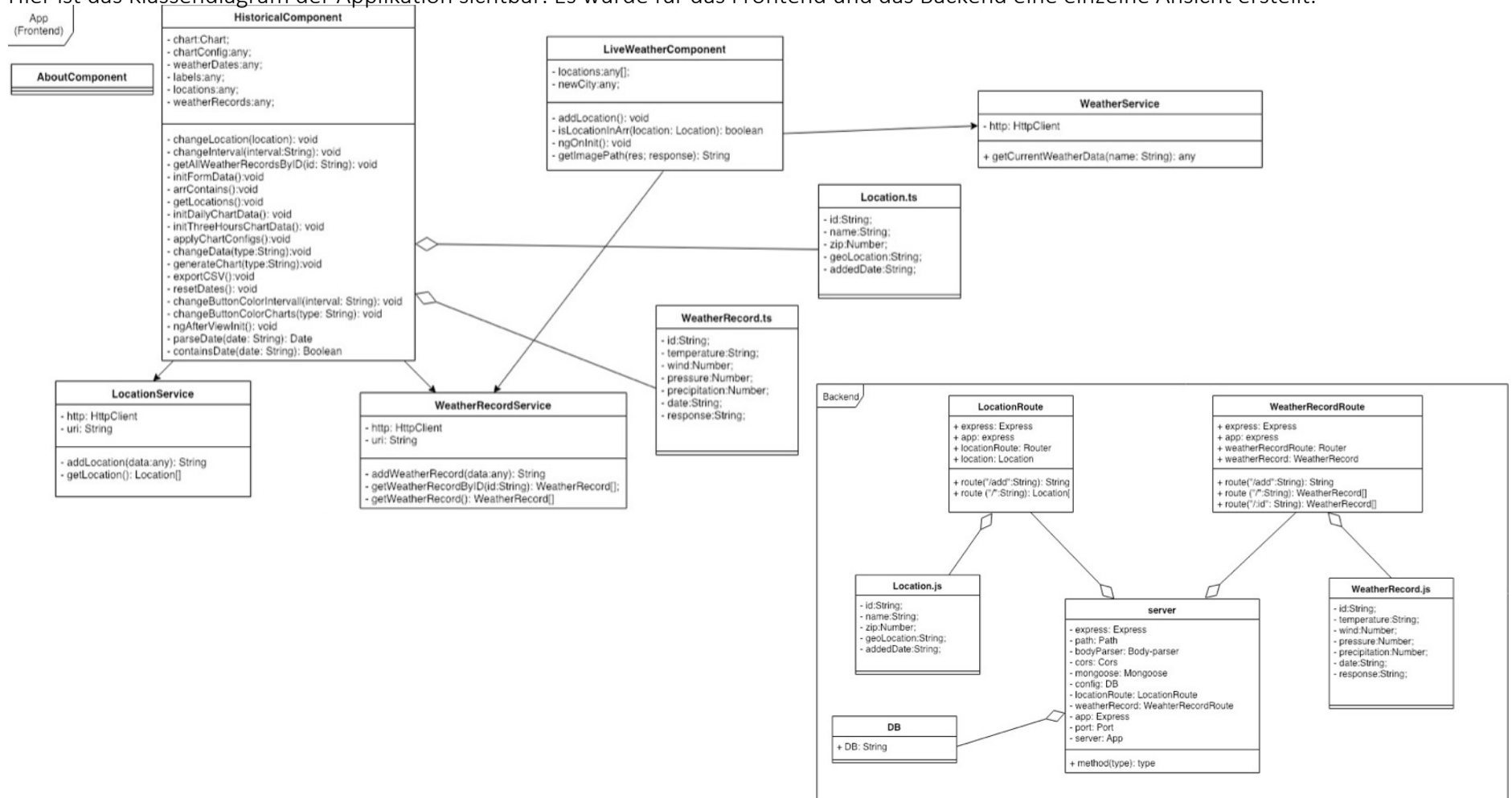


Abbildung 7: Klassendiagramm

Das Klassendiagramm wurde stark angepasst. Der Grund für diese Anpassung sind die mangelnden Kenntnisse im Framework. Die starke Abstraktion im vorherigen Entwurf mussten in der Umsetzung nicht erfolgen. Durch das simple MVC-Pattern und diversen Helper-Funktionen, welche Javascript bieten, konnte auf vielen Klassen und vor allem selbstgebaute Helper-Klassen verzichtet werden.

8 Testing – Protractor



“Protractor is an end-to-end test framework for Angular and AngularJS applications. Protractor runs tests against your application running in a real browser, interacting with it as a user would.”¹

Für End-to-end-Test verwenden wir Protractor. Damit können wir Nutzerverhalten simulieren. Dabei wird durch automatisiertes Skript ein Internetbrowser geöffnet und unsere Tests werden durchgeführt.

8.1 Setup

```
npm install -g protractor
webdriver-manager update
webdriver-manager start
protractor conf.js #run tests
```

8.2 Resultate

Mit Protractor konnten der eigene Tests erfolgreich durchgeführt werden. Unten sind die zu durchführenden Tests ersichtlich.

```
chrome-response.xml https://chromedriver.storage.googleapis.com/
[wdm]: Compiled successfully.
[14:10:17] I/update - chromedriver: file exists /Users/ohranmujkic/Desktop/Studium/Klima-Viewer/climaViewer/node_modules/protractor/node_modules/webdriver-manager/selenium/chromedriver_2.45.zip
[14:10:17] I/update - chromedriver: unzipping chromedriver_2.45.zip
[14:10:17] I/update - chromedriver: setting permissions to 0755 for /Users/ohranmujkic/Desktop/Studium/Klima-Viewer/climaViewer/node_modules/protractor/node_modules/webdriver-manager/selenium/chromedriver_2.45
[14:10:17] I/update - chromedriver: chromedriver_2.45 up to date
[14:10:18] I/launcher - Running 1 instances of WebDriver
[14:10:18] I/direct - Using ChromeDriver directly...
Jasmine started

workspace-project App
  ✓ should display welcome message Live Weather
  ✓ should display description Live Weather
  ✓ should display welcome message Historical
  ✓ should display description Historical

Executed 4 of 4 specs SUCCESS in 5 secs.
[14:10:26] I/launcher - 0 instance(s) of WebDriver still running
[14:10:26] I/launcher - chrome #01 passed
Ohrans-MacBook-Pro:climaViewer ohranmujkic$
```

Abbildung 8: Protractor Resultate

¹ Protractor. URL: <http://www.protractortest.org/>. Stand 20. Januar 2019

9 Herausforderungen

9.1 WorldClim

WorldClim war die Empfehlung im Projektbeschrieb als Datenlieferant. Leider stellte sich heraus, dass WorldClim nur .tif Dateien zum Download anbietet. Diese müssen jedoch vorher von einem anderen Programm geöffnet und bearbeitet werden. Wir kamen schnell zum Entschluss, dass dies Probleme bei der Automatisierung der Datenverarbeitung darstellen würde und haben nach Alternativen gesucht.

9.2 OpenWeatherMap

Als Alternative konnten wir OpenWeatherMap finden. Jedoch sind bei OpenWeatherMap historische Daten (Daten in der Vergangenheit) kostenpflichtig. Da dieses Projekt aber nur zur Vorzeige dient, konnten wir uns einig werden, dass wir Daten aus einer kürzeren Zeitperiode selber speichern. Dazu wurde ein Cronjob erstellt, welcher Wetterinformationen von OpenWeatherMap sammelte und in die Datenbank speicherte. Diese Daten wurden dann als „historische“ Daten verwendet.

10 Projektorganisation

10.1 Projektstruktur

- Api
 - node_modules : Speicherort ganzen Library-Package
 - DB.js : URL zur Datenbank
 - package.json : Projekt-Konfiguration
 - server.js : Backend, das eine Verbindung mit der Datenbank herstellt und die Abfragen erhält
- App
 - node_modules : Speicherort ganzen Library-Package
 - package.json : Projekt-Konfiguration
 - e2e : End-to-End Testing
 - models : Enthält die Models, welche eine Abbildung der Tabellen in der Datenbank sind
 - routes : Der API-Service, welcher die Abfragen empfängt und weiter bearbeitet. Aufgeliedert nach Tabellen
 - src : Source Code der Webapp
 - app
 - services : Services beinhalten verschiedene Funktionen, die immer wieder verwendet werden.
 - Live-weather : Logik und View von der LiveWeather-Seite
 - Historical : Logik und View von der Historical-Seite
 - About: Logik und View von der About-Seite
 - Location.ts : Interface für Location-Objekt
 - WeatherRecord.ts : Interface für WeatherRecord-Objekt
 - App-routing.module.ts : Routing der Web-App
 - App.component.ts : Logik Appübergreifend
 - App.component.html : View Appübergreifend
 - App.module.ts : Alle importierten Libraries
 - App.component.scss : Stylesheet Appübergreifend
 - App.component.spec.ts : Unit-Test Appübergreifend
 - Assets : Ablage für Bilder
 - Environments : Konfiguration für verschiedene Environments
 - Tests.ts : Tests-Config

10.2 Versionisierung

Das Projekt wurde mithilfe von Github realisiert. Somit ist jeder Commit und damit jede Veränderung ersichtlich. Es wurden auch Issues erstellt, um Teilaufgaben zuteilen zu können und somit die offenen Tasks im Auge behalten zu können.

Git-Repository: <https://github.com/mujko1/Klima-Viewer>

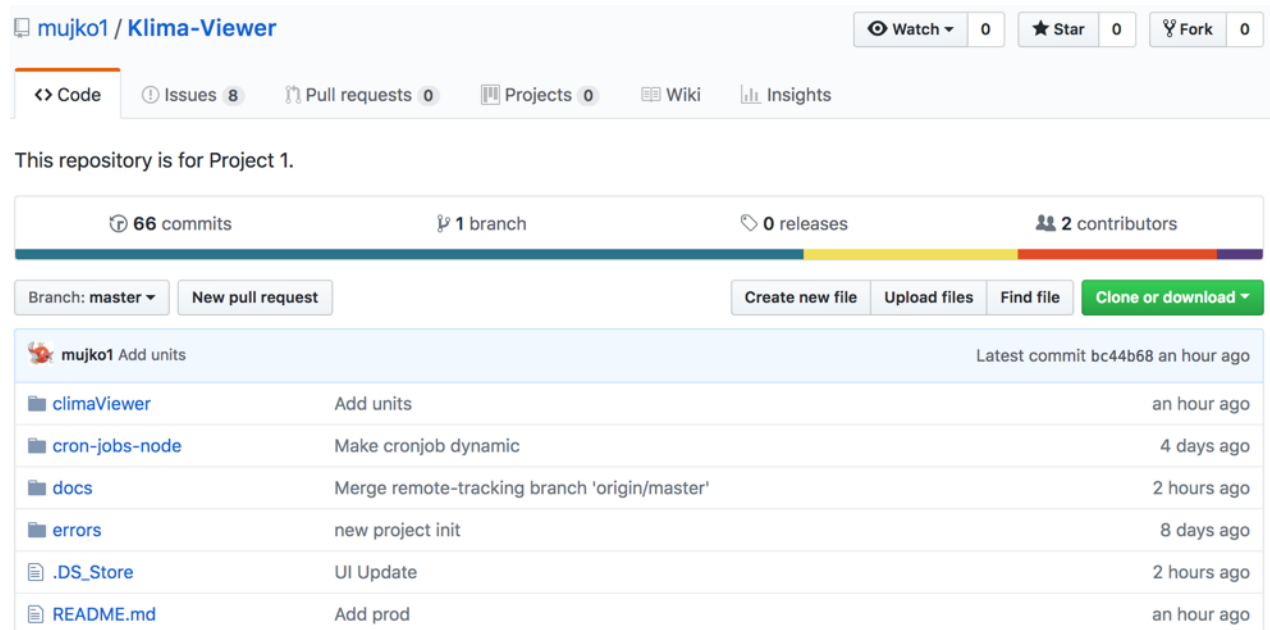


Abbildung 9: Auszug aus Github

10.3 Namenskonventionen

Wir haben uns in der Namenskonvention an den üblichen Konventionen des Frameworks gehalten. Dazu haben wir die Styling Guides befolgt, welche von Angular zu Verfügung stehen. <https://angular.io/guide/styleguide>

11 Ausblick

- Die Applikation könnte mit einem Userlogin erweitert werden. Somit hätte jeder seine persönliche Ansicht der eigenen Städte.
- Es könnten Anhand der Wetterdaten auch Wetterprognosen erstellt werden.
- Historische Daten könnten komplett verwendet werden, damit auch in weiter Vergangenheit Charts erstellt werden können.
- Die Applikation könnte flexibler umgebaut werden, damit auch die Datennutzung von anderen und verschiedenen Anbietern gleichzeitig möglich ist.
- Das Datenmodell wurde so erweitert, dass alle Informationen von OpenWeatherMap gespeichert werden. Somit können dies evtl. in zukünftigen Projekten verwendet werden.
- Die mögliche Auswahl der Städte könnte auf weltweit erweitert werden.

12 Glossar

Begriffe	Erklärung
Prototyp	Muster
Browser	Programm um nach Webseiten zu suchen, lesen oder verwalten.
UI	User Interface, Benutzeroberfläche
WorldClim	Plattform, welche Wetterdaten aus der ganzen Welt als Datei zu Verfügung stellt.
OpenWeatherMap	Plattform, welche Wetterdaten aus der ganzen Welt über einen API zu Verfügung stellt.
API	Programmierschnittstelle
REST	Representational state transfer – Software-Architekturstyl meistens für Webservices.
JSON	JavaScript Object Notation – Dateiformat meistens im Gebrauch für Webservices.
DB	Datenbank – Sammlung und Ablage von Daten.
Mean Stack	Kostenloser, Open-Source JavaScript Library zum Erstellen von dynamischen Webseiten.
URL	Adressierung einer Webseite
Entity-Relationship-Model	Zeigt anhand von einem Diagramm das Grobkonzept einer Datenbank. Hierbei spielen die Tabellen sowie deren Beziehung zueinander die Schlüsselrolle.
Cronjob	Ein Skript, welches im Hintergrund regelmässig zu einer vordefinierten Zeit, immer sich ausführt.
MongoDB	Das ist eine Datenbankprogramm.
CSV	Comma-separated Values – Ist ein Dateityp, dass sich besonders gut für den Export von Daten eignet.

13 Abbildungsverzeichnis

Abbildung 1: Gesamtübersicht aller Komponenten	8
Abbildung 2: LiveWeather - Standorte anzeigen	9
Abbildung 3: LiveWeather - Standort hinzufügen	9
Abbildung 4: Historical - Daten anzeigen	10
Abbildung 5: Historical - Datenbeschaffung	10
Abbildung 6: ERD	11
Abbildung 7: Klassendiagramm	12
Abbildung 8: Protractor Resultate.....	13
Abbildung 9: Auszug aus Github.....	16
Abbildung 10: LiveWeather Page	20
Abbildung 11: Historical Page.....	20
Abbildung 12: About Page	21

14 Anhang

14.1 Screenshots der Applikation

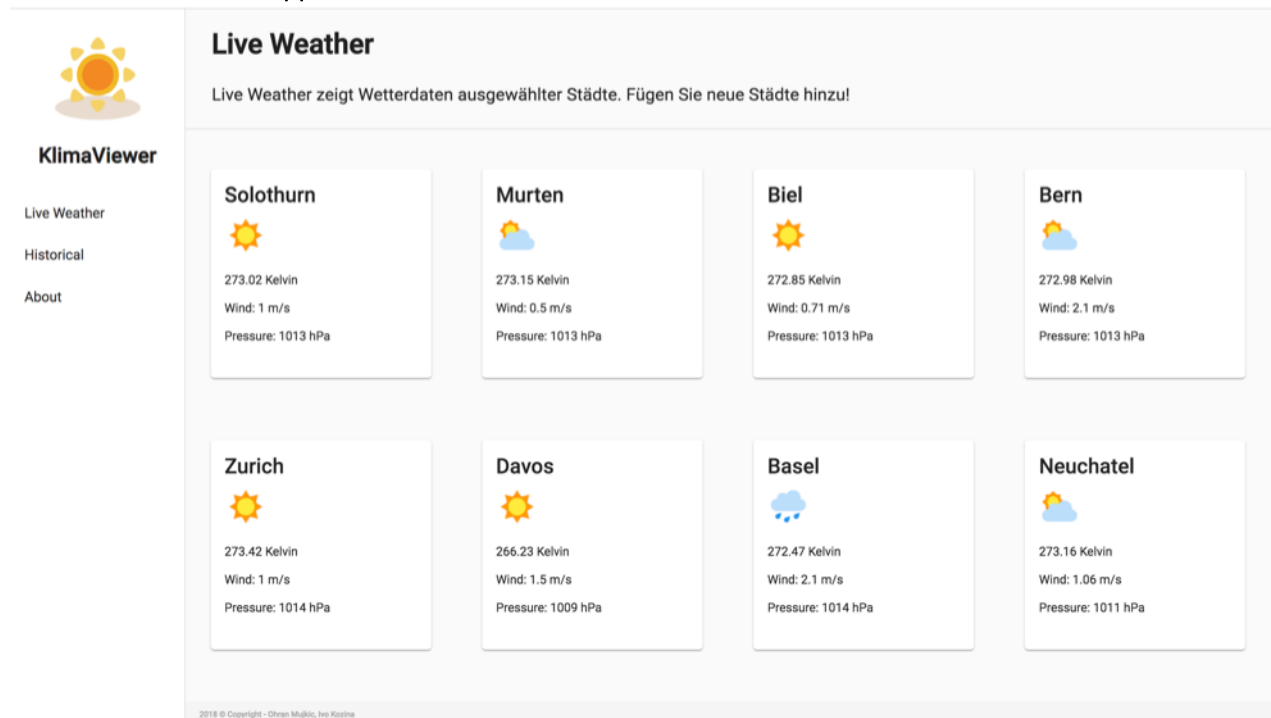


Abbildung 10: LiveWeather Page

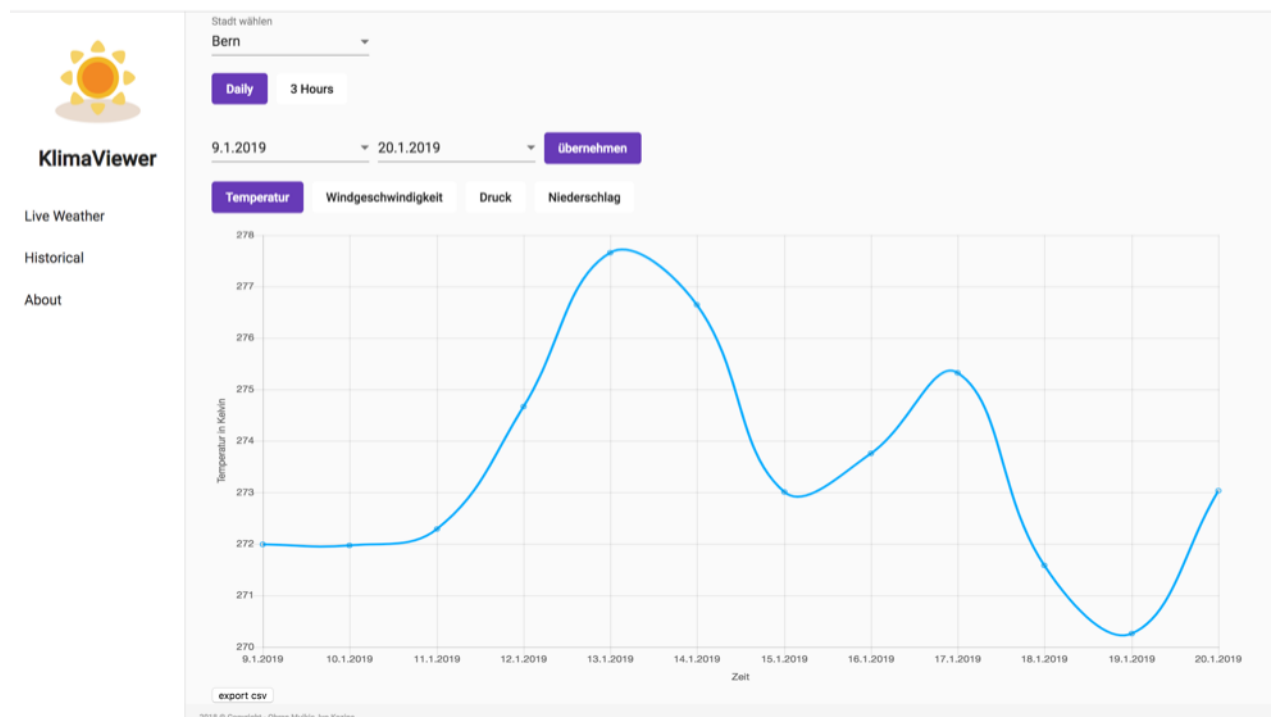


Abbildung 11: Historical Page

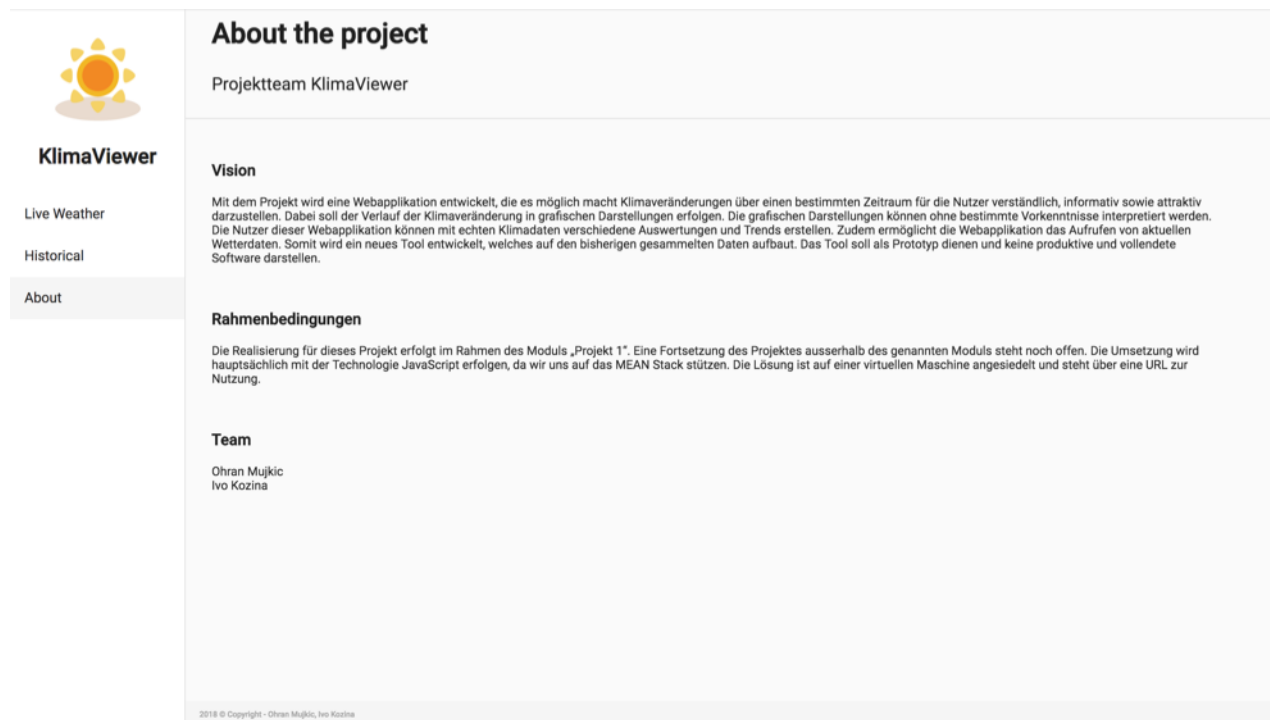


Abbildung 12: About Page

14.2 Projektbeschreibung

Modul BT17301

Projekt 1

Herbstsemester 2018

Projektname:	Klima Viewer	
Firma:		
Studierende:		
Betreuer:	Firma:	BFH: J. Wolfgang Kaltz
Kurzbeschreibung:	<p>In diesem Projekt soll eine Applikation „Klima Viewer“ entwickelt werden, die Klimaveränderungen über die letzten Jahre sichtbar macht. Der Benutzer soll für verschiedene Kennzahlen (Temperatur, Windgeschwindigkeit usw.) eine graphische Darstellung über einen bestimmten Zeitraum erhalten, samt Veränderungen über diesen Zeitraum.</p> <p>Als Datenbasis kann z.B. WorldClim (http://worldclim.org) verwendet werden.</p>	
Technologie:	Java, Web Applikation, oder App	

15 Versionskontrolle

Version	Datum	Beschreibung	Autor
X0.1	15.12.2018	Dokument erstellt	Ohran Mujkic
X0.2	16.12.2018	Zweck des Dokuments, Projektkontext und Rahmenbedingungen erstellt	Ivo Kozina
X0.3	18.12.2018	Bewertung der Funktionalen Anforderungen	Ohran Mujkic
X0.4	19.12.2018	Design	Ivo Kozina
X0.5	20.12.2018	Datenmodell	Ohran Mujkic
V1.0	19.01.2019	Abschlussarbeiten am ganzen Dokument und Freigabe	O. Mujkic und I. Kozina