



Fachinformatikerin für Anwendungsentwicklung
Dokumentation zur schulischen Projektarbeit

Entwicklung einer Wetter-App mit Angular und Express.js
unter Nutzung externer APIs

Abgabedatum: 13.03.2025

Schule: BBZ Rendsburg-Eckernförde

Lehrkraft: Dennis Clausen



Inhaltsverzeichnis

1.	<i>Einleitung</i>	3
2.	<i>Anforderungsanalyse</i>	3
3.	<i>Beschreibung des Projekts</i>	4
4.	<i>Dokumentation Frontend</i>	5
4.1	<i>Screenshots und Beschreibungen</i>	5
4.2	<i>Benutzeroberfläche und Funktionalitäten</i>	8
5.	<i>Nutzer-/Bedienungshandbuch</i>	8
6.	<i>Dokumentation Projektablauf</i>	10
7.	<i>Testverfahren und Qualitätssicherungsmaßnahmen</i>	11
8.	<i>Reflexion der Projektarbeit</i>	11
9.	<i>Anhang</i>	12



1. Einleitung

Das Ziel dieses Projekts war die Entwicklung einer Wetter-App, die es Nutzern ermöglicht, aktuelle Wetterdaten für jede Stadt weltweit abzurufen. Die Anwendung wurde mit Angular im Frontend und Express.js im Backend umgesetzt. Durch die Integration der Weather API für Wetterdaten und der Unsplash API für passende Bilder wird den Nutzern zusätzlich eine visuelle Darstellung ihrer eingegebenen Stadt geboten.

Das Projekt fokussiert sich auf eine benutzerfreundliche Oberfläche und eine funktionale, schnelle Datenabfrage.

2. Anforderungsanalyse


Die Wetter-App soll es Nutzern ermöglichen, schnell und einfach das aktuelle Wetter für jede beliebige Stadt abzurufen. Dazu soll eine benutzerfreundliche Oberfläche bereitgestellt werden, die sowohl die Wetterdaten als auch ein zufälliges Bild der Stadt anzeigt.

Funktionale Anforderungen:

- Wetterabfrage:
 - Der Nutzer kann eine Stadt eingeben und das aktuelle Wetter anzeigen lassen.
 - Die App zeigt Temperatur, Wetterbeschreibung und weitere relevante Daten (z. B. Luftfeuchtigkeit).
- Bildanzeige:
 - Nach Eingabe einer Stadt wird ein zufälliges Bild von der Stadt über die Unsplash API angezeigt.
- Benutzeroberfläche:
 - Die Anwendung soll einfach und intuitiv zu bedienen sein.
 - Die Wetterdaten und das Bild sollen in einem klar strukturierten Layout angezeigt werden.

Nicht-funktionale Anforderungen:

- Performance:
 - Die Wetterdaten sollen schnell geladen werden (weniger als 2 Sekunden Ladezeit)

- 
- Kompatibilität:
 - o Die App muss auf allen gängigen Webbrowsern (Chrome, Firefox, Edge, Safari) funktionieren.

Technische Anforderungen:

- Frontend:
 - o Angular für die Entwicklung der Benutzeroberfläche.
- Backend:
 - o Express.js für die API-Anbindung und Kommunikation mit den externen APIs.
- APIs:
 - o Weather API für die Wetterdaten.
 - o Unsplash API für die Bildanzeige der Stadt.

3. Beschreibung des Projekts

Das Projekt besteht in der Entwicklung einer Wetter-App, die es Nutzern ermöglicht, das aktuelle Wetter für jede beliebige Stadt weltweit abzurufen. Zusätzlich wird ein zufälliges Bild der eingegebenen Stadt angezeigt, das durch die **Unsplash API** bereitgestellt wird. Die Anwendung kombiniert Wetterinformationen mit einer visuellen Darstellung der Stadt, um dem Nutzer ein ansprechendes Erlebnis zu bieten.

Die App wurde mit Angular für das Frontend und Express.js für das Backend entwickelt. Das Backend übernimmt die Kommunikation mit zwei externen APIs:

- Unsplash API: Stellt ein zufälliges Bild der gesuchten Stadt zur Verfügung, um die Wetterinformationen visuell zu ergänzen.
- Weather API: Liefert Wetterinformationen wie Temperatur, Wetterbeschreibung, Luftfeuchtigkeit, Windgeschwindigkeit und Luftdruck.

Der Datenabruf erfolgt über das Backend, das die Anfragen der Nutzer verarbeitet, die benötigten Informationen von den APIs abrufen und anschließend an das Frontend übermitteln.

Die Benutzeroberfläche ist bewusst einfach und intuitiv gestaltet, um eine benutzerfreundliche Bedienung zu gewährleisten. Nutzer können eine Stadt in eine Suchleiste eingeben und erhalten nach dem Absenden der Anfrage die aktuellen Wetterdaten sowie ein passendes Bild zur Stadt. Die Suchleiste sowie der Suchbutton verfügen über Mouse Hover- und Click-Effekte, um das Nutzererlebnis zu verbessern.

Durch den Einsatz moderner Webtechnologien bietet die Anwendung eine schnelle und zuverlässige Wetterabfrage mit einer attraktiven visuellen Darstellung.

4. Dokumentation Frontend

Die Wetter-App wurde mit Angular entwickelt und bietet eine benutzerfreundliche Oberfläche. Die Anwendung ermöglicht es dem Nutzer, das aktuelle Wetter für jede beliebige Stadt abzufragen. Die Benutzeroberfläche ist intuitiv und enthält mehrere interaktive Elemente, die das Nutzungserlebnis verbessern

4.1 Screenshots und Beschreibungen

Startseite:

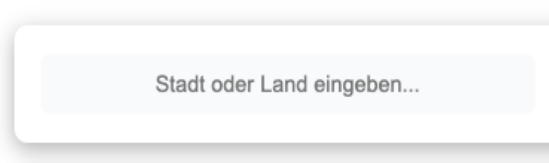
- Die Benutzeroberfläche der Wetter-App ist benutzerfreundlich und interaktiv gestaltet. Die folgenden Elemente sind zentral für die Bedienung:

Suchleiste (Search Bar)

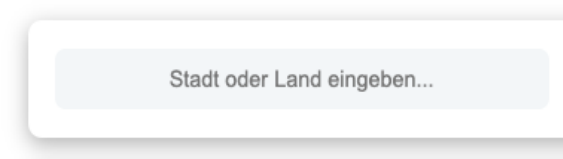
- Der Nutzer gibt den Namen einer Stadt ein.
- Hover-Effekt: Beim Darüberfahren mit der Maus hebt sich die Leiste optisch hervor.
- Click-Effekt: Beim Anklicken wird das Eingabefeld fokussiert.

Screenshot der Suchleiste (z. B.):

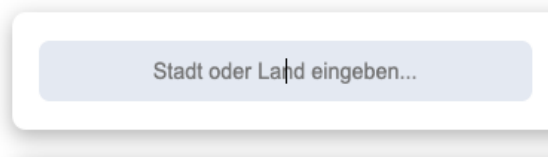
1. kein Hover



2. Hover-Effekt



3. Click-Effekt



Suchbutton:

- Neben der Suchleiste befindet sich der Suchbutton, mit dem die Wetterabfrage gestartet wird.
- Hover-Effekt: Der Button verändert sich leicht, wenn die Maus darüberfährt.

Screenshot des Suchbuttons (z. B.):



Anzeige Der Wetterdaten:

Nach Eingabe der Stadt und Klick auf den Suchbutton erscheinen folgende Wetterdaten:

- Landname
- Temperatur.
- Wetterbeschreibung
- Luftfeuchtigkeit
- Windgeschwindigkeit
- Luftdruck

Screenshot der Wetterdaten (z. B.):

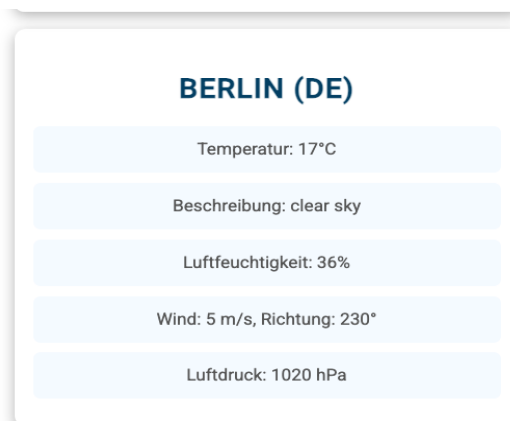


Bild der Stadt:

- Zusätzlich zu den Wetterdaten wird ein Zufallsbild der Stadt von der Unsplash API angezeigt.

Screenshot des Bildes der Stadt (z. B.):



Beispielhafte HTML-Struktur:

```
1 <div class="weather-container">
2   <div class="weather-header">
3     <h1>Weather App</h1>
4     <div class="search-container">
5       <input
6         class="input-field"
7         type="text"
8         [(ngModel)]="city"
9         (keyup.enter)="getWeather()"
10        placeholder="Stadt oder Land eingeben..."
11      />
12     <button class="button" (click)="getWeather()">Suchen</button>
13   </div>
14   <p *ngIf="errorMessage" class="error">{{ errorMessage }}</p>
15 </div>
16
17 <div *ngIf="weatherData" class="info-box">
18   <h3>{{ weatherData.city }} ({{ weatherData.countryCode }})</h3>
19   <p>Temperatur: {{ weatherData.temperature }}°C</p>
20   <p>Beschreibung: {{ weatherData.description }}</p>
21   <p>Luftfeuchtigkeit: {{ weatherData.humidity }}%</p>
22   <p>
23     Wind: {{ weatherData.windSpeed }} m/s, Richtung:
24     {{ weatherData.windDeg }}°
25   </p>
26   <p>Luftdruck: {{ weatherData.pressure }} hPa</p>
27 </div>
28
29 <app-image class="app-image" [city]="weatherData?.city || ''"></app-image>
30 </div>
31
```



4.2 Benutzeroberfläche und Funktionalitäten

Die App besteht aus drei Hauptbereichen:

1. Suchfunktion
 - Suchleiste mit Hover- & Click-Effekt
 - Suchbutton mit Hover-Effekt
2. Wetteranzeige
 - Abruf der Wetterdaten über die **Weather API**
 - Anzeige von Temperatur, Wetterbeschreibung, Luftfeuchtigkeit, Windgeschwindigkeit, Luftdruck
3. Stadtbild & Fehlerbehandlung
 - Zufallsbild der Stadt über die Unsplash API
 - Fehlermeldung, falls die Stadt nicht gefunden wird

5. Nutzer-/Bedienungshandbuch

Dieses Handbuch beschreibt die Nutzung der Wetter-App und erklärt die wichtigsten Funktionen. Die App ermöglicht es, das aktuelle Wetter für eine beliebige Stadt weltweit abzurufen. Zusätzlich wird ein zufälliges Bild der Stadt angezeigt.

Systemanforderungen:

Um die App nutzen zu können, benötigt der Benutzer:


- Einen aktuellen Webbrowser (Google Chrome, Firefox, Edge, Safari)
- Eine aktive Internetverbindung

Start der Anwendung:

1. Öffnen Sie die Wetter-App über den Webbrowser.
2. Auf der Startseite befindet sich eine Suchleiste, in die Sie den Namen einer Stadt eingeben können.
3. Neben der Suchleiste befindet sich der Suchbutton, mit dem Sie die Anfrage starten.

Wetterabfrage durchführen:

1. Stadt eingeben:
 - a. Tippen Sie den Namen einer Stadt in das Eingabefeld ein.

- 
- b. Während Sie mit der Maus über das Eingabefeld fahren, hebt sich dieses visuell hervor (Hover-Effekt).
 - c. Sobald Sie in das Suchfeld klicken, erhält es einen Fokus-Effekt, um die Eingabe zu erleichtern.

2. Suche auslösen:

- a. Drücken Sie die Enter-Taste oder klicken Sie auf den Suchbutton, um die Wetterdaten abzurufen.
- b. Der Suchbutton hat ebenfalls einen Hover-Effekt, der ihn optisch hervorhebt.

3. Anzeige der Ergebnisse:

Nach wenigen Sekunden erscheinen die Wetterdaten und ein Bild der eingegebenen Stadt.

Anzeige der Wetterdaten:

Nach einer erfolgreichen Suche werden folgende Informationen angezeigt:

- Landname: Das Land, in dem sich die Stadt befindet.
- Temperatur: Die aktuelle Temperatur in Grad Celsius.
- Wetterbeschreibung: Eine Kurzbeschreibung des aktuellen Wetters (z. B. „klarer Himmel“, „bewölkt“).
- Luftfeuchtigkeit: Der prozentuale Feuchtigkeitswert der Luft.
- Windgeschwindigkeit: Die Geschwindigkeit des Windes in km/h.
- Luftdruck: Der aktuelle Luftdruck in hPa (Hektopascal).
- Bild der Stadt: Ein zufälliges Foto der Stadt, das von der Unsplash API bereitgestellt wird.

Fehlerbehandlung:

Falls eine Stadt nicht gefunden wird oder ein Fehler auftritt, erscheint eine Fehlermeldung.
Mögliche Gründe:

- Die eingegebene Stadt existiert nicht oder wurde falsch geschrieben.
- Probleme mit der Internetverbindung.
- Die externen APIs sind nicht erreichbar.

Lösung:

- Überprüfen Sie die Schreibweise der Stadt.
- Versuchen Sie es später erneut.
- Prüfen Sie die Internetverbindung



Beenden der Anwendung:

Die Anwendung kann jederzeit durch Schließen des Browserfensters beendet werden.

6. Dokumentation Projektablauf

1. Projektplanung und Anforderungsanalyse

- Definition der Hauptfunktionen (Wetterdaten, Stadtbilder)
- Auswahl der Technologien (Angular, Express.js, Weather API, Unsplash API)
- Entwurf der Benutzeroberfläche

2. Frontend-Entwicklung – Grundstruktur

- Aufbau der Grundstruktur mit Angular
- Implementierung der Suchleiste und Suchbutton mit Hover-Effekten
- Gestaltung der Wetterdatenanzeige (Temperatur, Beschreibung, etc.)

3. Backend-Entwicklung mit Express.js

- Einrichtung des Servers mit Express.js
- API-Integration (Weather API und Unsplash API)
- Abrufen von Wetterdaten und Bildern

4. Frontend- und Backend-Integration

- Verbindung der Angular-App mit dem Express-Backend
- Dynamische Anzeige der Wetter- und Bilddaten
- Fehlerbehandlung und Datenvalidierung

5. Testing und Debugging

- Funktionstests und Behebung von Fehlern

6. Feinschliff und Verbesserungen

- Optimierung der Benutzeroberfläche und Performance
- Verfeinerung der visuellen Effekte
- Code-Refactoring und Performance-Optimierung

7. Abschluss und Dokumentation

- Erstellung der Projektdokumentation
- Vorbereitung der Präsentation
- Bereitstellung des Projekts auf GitHub



7. Testverfahren und Qualitätssicherungsmaßnahmen

Im Rahmen der Entwicklung der Wetter-App wurden verschiedene Testverfahren angewendet, um sicherzustellen, dass die App stabil und fehlerfrei funktioniert. Die Qualitätssicherung wurde durch die Implementierung von Unit-Tests sowie durch manuelles Testing durchgeführt.

Unit-Tests mit Jest:

- Jest wurde als Test-Framework verwendet, um Unit-Tests für die einzelnen Komponenten der App zu schreiben. Ziel war es, die Funktionalität jeder Komponente isoliert zu überprüfen und sicherzustellen, dass sie korrekt funktioniert.
- Testabdeckung:
 - Frontend-Komponenten: Alle wichtigen Angular-Komponenten, wie die Suchleiste, der Suchbutton und die Anzeige der Wetterdaten, wurden mit Unit-Tests abgedeckt.
 - Backend-Routen: Die API-Aufrufe im Backend, die Wetterdaten und Stadtbilder abrufen, wurden ebenfalls mit Unit-Tests getestet.

8. Reflexion der Projektarbeit

Was lief gut?

1. Technologische Umsetzung:
 - Die Wahl der Technologien (Angular für das Frontend und Express.js für das Backend) hat sich als äußerst passend herausgestellt. Die Integration der externen APIs (Weather API und Unsplash API) verlief reibungslos, und das Setup des Backends war relativ einfach zu implementieren.
 - Besonders positiv war die reibungslose Kommunikation zwischen Frontend und Backend. Die Wetterdaten sowie die Bilder von der Unsplash API wurden korrekt und schnell abgerufen und in der Benutzeroberfläche angezeigt.
2. Benutzeroberfläche und Benutzererlebnis:
 - Die App bietet eine benutzerfreundliche und ansprechende Oberfläche. Die Implementierung von Hover-Effekten und die klare Struktur der Wetteranzeige haben das Benutzererlebnis erheblich verbessert.

3. Testing und Qualitätssicherung:

- Durch den Einsatz von Unit-Tests mit Jest konnte ich die einzelnen Komponenten und ihre Funktionalitäten umfassend testen. Dies hat dazu beigetragen, die Fehlerquote zu reduzieren und das Vertrauen in die Stabilität der App zu stärken.
- Das manuelle Testing und die Browserkompatibilitätstests sorgten dafür, dass die App auf verschiedenen Plattformen gut funktionierte.

Was lief schief?

1. Probleme bei der Unsplash API:

- Bei der Integration der Unsplash API traten hin und wieder Probleme mit den API-Limits auf, was in einigen Fällen dazu führte, dass keine Stadtbilder abgerufen werden konnten. Dies wurde durch die Implementierung von Fehlerbehandlungen jedoch schnell abgefangen. In solchen Fällen wurde der Benutzer darüber informiert, dass keine Bilder zur Verfügung standen.

2. Komplexität der Fehlerbehandlung:

- Die Fehlerbehandlung bei der Eingabe ungültiger Städte war nicht immer intuitiv genug. Zu Beginn hatte die App Probleme, Fehlermeldungen präzise anzuzeigen. Nach einer Überarbeitung wurde dieses Problem jedoch gelöst, und die Benutzer wurden klarer über die Art des Fehlers informiert.

9. Anhang

Testergebnisse / Codebeispiele:

- Wetterdaten API Aufruf (Backend - Express.js):

```
1  const express = require('express');
2  const axios = require('axios');
3  const cors = require('cors');
4
5  const app = express();
6  const PORT = process.env.PORT || 3000;
7
8  app.use(cors());
9
10 app.get('/weather', async (req, res) => {
11   const city = req.query.city;
12   const apiKey = 'e1ba42943e7568d0454f6db1418ea4fd';
13   try {
14     const response = await axios.get('http://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}&units=metric');
15     res.json(response.data);
16   } catch (error) {
17     console.error(error);
18     res.status(500).send('Fehler beim Abrufen der Wetterdaten');
19   }
20 });
21
22 app.listen(PORT, () => {
23   console.log('Server läuft auf http://localhost:${PORT}');
24 });
```

- Ein Beispiel für einen Unit-Test, der die erfolgreiche Abfrage von Wetterdaten prüft, lautet wie folgt:

```
1  it('should fetch weather data successfully when city is valid', fakeAsync(() => {
2    component.city = 'Berlin';
3    component.getWeather();
4    tick();
5
6    expect(component.weatherData).toEqual({
7      temperature: 15,
8      feelsLike: 14,
9      description: 'Clear sky',
10     minTemp: 10,
11     maxTemp: 20,
12     pressure: 1012,
13     windSpeed: 5,
14     windDeg: 180,
15     countryCode: 'DE',
16     city: 'Berlin',
17   });
18
19   expect(component.errorMessage).toBe('');
20 });
```

Output:

```
• + backend git:(master) x cd '/Users/amer.shaadounh/Documents/SchulProjekt/weather-app-school'
• + weather-app-school git:(master) x node 'node_modules/.bin/jest' '/Users/amer.shaadounh/Documents/SchulProjekt/weather-app-school/src/app/components/weather/weath
er.component.spec.ts' -t 'WeatherComponent should fetch weather data successfully when city is valid'
PASS src/app/components/weather/weather.component.spec.ts
  WeatherComponent
    ✓ should fetch weather data successfully when city is valid (39 ms)
    ○ skipped should create the component

Test Suites: 1 passed, 1 total
Tests:       1 skipped, 1 passed, 2 total
Snapshots:   0 total
Time:        3.266 s
Ran all test suites matching /\Users\amer.shaadounh\Documents\SchulProjekt\weather-app-school\src\app\components\weather\weather.component.spec.ts/i with
tests matching "WeatherComponent should fetch weather data successfully when city is valid".
❖ + weather-app-school git:(master) x
```