# Hochschule für Technik Stuttgart

## Konzeption und Umsetzung eines Klima-/Wetterinformations-Portals

**Bachelor-Thesis**

Als PL nach der SPO-2024
Ausgeführt für die Bachelorprüfung am Ende des Sommersemesters

Betreuung:
Prof. Dr. -Ing. Franz-Josef Behr
Prof. Dr. -Ing. Michael Bach

Von: Bora Aslan
Matrikelnummer: 1002700
E-Mail: 02asbo1bil@hft-stuttgart.de
Abgabetermin: 30.06.2025

# Kurzfassung

Bei dieser Arbeit handelt es sich um die Konzeption und Umsetzung eines Klima- und Wetterinformationsportals unter Verwendung von Open Data und Open-Source-Software. Ziel ist es, eine Internetplattform zu entwickeln, die Bürgerinnen und Bürgern[1] den Zugang zu meteorologischen Messdaten ermöglicht und sie in die Lage versetzt, sich selbstständig über den Klimawandel zu informieren. Der Schwerpunkt liegt dabei auf der Nutzung von Daten, die der Deutsche Wetterdienst (DWD) über sein Open-Data-Portal zur Verfügung stellt. Die Arbeit baut auf früheren Untersuchungen auf, die sich mit Java-Programmierung, Datenerfassung und Datenbankintegration auseinandersetzen. Dabei wird methodisch ein systematischer Ansatz verfolgt, der die Analyse bestehender Lösungen, den Entwurf der Zielvisualisierung und die Programmierung der Darstellung umfasst. Am Ende soll ein benutzerfreundliches Webportal entstehen, das Klimadaten der letzten Jahrzehnte für jeden Ort in Deutschland zugänglich und verständlich macht.

This thesis focuses on the conception and implementation of a climate and weather information portal using open data and open-source software. The aim is to develop an online platform that enables citizens to access meteorological measurement data and empowers them to independently inform themselves about climate change. The emphasis lies on utilising data provided by the German Weather Service (in German: *Deutscher Wetterdienst*, abbreviated as DWD) through its open data portal. The thesis builds upon previous studies that explored Java programming, data collection, and database integration. A systematic methodological approach is applied, encompassing the analysis of existing solutions, the design of the target visualisation, and the programming of the presentation. The final result is intended to be a user-friendly web portal that makes climate data from the past decades accessible and comprehensible for any location in Germany.

# Schlüsselbegriffe

Klimawandel, Klimainformationsportal, Klimadatenanalyse, Wetterdaten, Meteorologische Messdaten, Programmierung, Datenaufbereitung, Datenvisualisierung, Datenbank, Open Data, Open-Source.

[1] Gender-Hinweis: Zur besseren Lesbarkeit werden in der Bachelorarbeit häufig entweder das generische Maskulinum oder Partizipien verwendet. Daher beziehen sich die in dieser Arbeit verwendeten Personenbezeichnungen – sofern nicht anders kenntlich gemacht – auf alle Geschlechter (männlich, weiblich und divers).

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| API | Application Programming Interface |
| CDC | Climate Data Center |
| CORS | Cross-Origin Resource Sharing |
| CSV | Comma-Separated Values |
| DWD | Deutscher Wetterdienst |
| FAIR | Findable, Accessible, Interoperable, Reusable |
| GUI | Graphical User Interface |
| HTTP | Hypertext Transfer Protocol |
| IPCC | Intergovernmental Panel on Climate Change |
| JSON | JavaScript Object Notation |
| KL | Klimadaten |
| OGC | Open Geospatial Consortium |
| ORM | Object-Relational Mapping |
| REST | Representational State Transfer |
| UI | User Interface |
| UKCRP | UK Climate Resilience Programme |
| UML | Unified Modeling Language |
| WCAG | Web Content Accessibility Guidelines |
| ZIP | Komprimiertes Archivformat |

# 1 Einleitung

Die Auswirkungen des Klimawandels auf Gesellschaft und Umwelt zählen zu den größten Herausforderungen unserer Zeit. Organisationen wie der Deutsche Wetterdienst (kurz: DWD) bemühen sich, die Öffentlichkeit mit meteorologischen Daten zu informieren, die jedoch aufgrund ihrer Komplexität für viele Menschen schwer verständlich sind. Um sicherzustellen, dass alle Teile der Gesellschaft fundierte Entscheidungen treffen können, ist es erforderlich, diese Daten nicht nur bereitzustellen, sondern sie auch in einer benutzerfreundlichen und verständlichen Weise aufzubereiten. Auch die Europäische Kommission erkennt die Relevanz dieses Anliegens an und versucht im Rahmen des Projekts „HARMONIA“, nicht nur die wissenschaftlichen Auswirkungen, sondern auch die gesellschaftlichen Folgen des Klimawandels zu vermitteln.[2]

Das Wetter ist ein unvermeidbarer und bedeutender Faktor in unserem Alltag. Unabhängig von Beruf oder sozialem Status kann die meteorologische Lage unterschiedlichste Lebenssituationen beeinflussen. In den vergangenen Jahrzehnten hat die Bedeutung des Klimawandels zugenommen, insbesondere durch die Zunahme von Temperaturextremen, den stetig steigenden Meeresspiegel sowie durch vermehrt auftretende Extremniederschläge. Seit der Gründung des DWD im Jahr 1952 besteht dessen Aufgabe darin, die Bevölkerung über die meteorologische Lage in Deutschland zu informieren. Im Jahr 2024 wurden etwa 200.000 Wetterwarnungen manuell erstellt; darüber hinaus wurden rund 5.200 Unwetter- und Extremwetterwarnungen versendet.[3] Im Vergleich dazu lag die Zahl der Wetterwarnungen im Jahr 2022 bei rund 177.000, dies entspricht einem Anstieg von ca. 13 % gestiegen.[4]

Der DWD stellt seine erhobenen meteorologischen Daten über ein Open-Data-Portal frei zur Verfügung. In Kombination mit zahlreichen Open-Source-Softwarelösungen ergibt sich daraus eine tragfähige Grundlage zur Entwicklung eines Klima- und Wetterinformationsportals. Bereits heute bieten Unternehmen wie die meteoblue AG entsprechende Portale mit umfangreichen meteorologischen Daten und Visualisierungen an. Trotz dieser Vielfalt bestehen weiterhin Lücken, die im Rahmen dieser Arbeit näher untersucht werden. Hierzu zählt unter anderem das Fehlen erweiterter Analysemöglichkeiten. Zwar sind historische Daten in den Portalen enthalten, doch fehlt es vielfach an Funktionen zur Trend- oder Vergleichsanalyse. Mit der vorliegenden Arbeit soll ein Beitrag zur gesellschaftlichen Auseinandersetzung mit dem Klimawandel geleistet werden, indem dieser durch eine geeignete Aufbereitung der Daten für einen breiteren Teil der Bevölkerung verständlich und zugänglich gemacht wird.

[2] vgl. Europäische Kommission (2021, online)

[3] vgl. Deutscher Wetterdienst (2025, online)

[4] vgl. Deutscher Wetterdienst (2023, online)

## 1.1 Motivation

Der Wunsch zur Umsetzung eines Klima- und Wetterinformationsportals entstand durch die aktuellen Debatten über den Klimawandel und dessen Auswirkungen auf die Politik und die Gesellschaft. Durch das politische Spektrum hört man unterschiedlichste Meinungen über den Klimawandel. Wissenschaftler geben objektive Aussagen über den Klimawandel, die jedoch oft global formuliert sind und für den durchschnittlichen Bürger in Deutschland nicht greifbar sind. Es stellt sich die Frage, wie man den komplexen Sachverhalt um den Klimawandel näher und verständlicher an die Gesellschaft bringt. Diese Bachelorarbeit verfolgt den Ansatz, dies mit mittels eines Online-Portals zu bewerkstelligen, welches als Fundament öffentliche Daten und Webtechnologien nutzt. Die Herausforderung bei der Umsetzung des Portals besteht darin, die Daten von der Quelle abzugreifen, ordnungsgemäß zu verarbeiten, zugänglich zu machen und visuell verständlich darzustellen.

## 1.2 Problemstellung

Laut einer Umfrage des Umweltbundesamtes im Jahr 2020 sind 47 % der Befragten „ziemlich“ interessiert an dem Thema Klimawandel und Klimaschutz. 25 % der Befragten sind sogar „sehr“ interessiert an dem Thema. Zu dem Thema Klimawandel und Klimaschutz fühlen sich laut der Umfrage 60 % der Befragten „gut“ oder „sehr gut“ informiert und 40 % fühlten sich nur „etwas“ oder „gar nicht“ informiert. Zudem wird der Klimawandel von 80 % der Befragten als Bedrohung für die Lebensgrundlage in Deutschland gesehen. Eine allgemeine Übereinstimmung besteht auch hinsichtlich der Dringlichkeit und der Maßnahmen zur Bekämpfung des Klimawandels. Über 90 % der Befragten sind der Ansicht, dass „dringend Maßnahmen zur Anpassung an die Folgen“ erforderlich sind. Zur Frage nach der Ursache des Klimawandels sind sich die Befragten ebenfalls weitgehend einig – 77 % sind überzeugt, dass der Klimawandel überwiegend oder allein vom Menschen verursacht ist.[5]

In einer weiteren Umfrage des Umweltbundesamtes im Jahr 2022 berichteten 85 % der Befragten, dass sie die Folgen des Klimawandels bereits spüren – etwa durch Dürren, Hochwasser oder Hitze.[6]

Zusammenfassend zeigt die Umfrage, dass ein grundlegendes Verständnis des Klimawandels vorhanden ist. Dieses Verständnis spiegelt sich jedoch nicht im Handeln wider, denn 44 % der Befragten gaben an, dass es ihnen an Möglichkeiten fehlt, persönlich etwas für den Klimaschutz zu tun. Wiederum 19 % meinen, dass der Klimaschutz sie überfordert. Die Bereitschaft zur Beschränkung des alltäglichen Konsums in Bezug auf

[5] vgl. Umweltbundesamt (2022, online)

[6] vgl. Umweltbundesamt (2023, online)

die Umwelt lag unter den Befragten insgesamt nur bei 28 % mit der Antwort entweder „ja, auf jeden Fall“ oder „wird bereits gemacht“.[7]

Weiterhin zeigt eine internationale Umfrage der Ipsos GmbH aus dem Jahr 2025, dass das Interesse der Deutschen am Klimaschutz im Vergleich zur Umfrage aus dem Jahr 2021 deutlich gesunken ist. Nur noch 53 % der befragten Deutschen fühlen sich in der Verantwortung für den Klimaschutz – im Jahr 2021 waren es noch 69 %. Anhand der Befragungen ist eine mögliche Interpretation, dass es innerhalb der Bevölkerung eine Wissenslücke zum lokalen Bezug des Klimawandels und deren lokalen Auswirkungen gibt. Demnach nimmt ein Großteil der Bevölkerung den Klimawandel als globales Phänomen wahr und stellt keinen direkten Bezug bzw. Auswirkungen des Klimawandels zu seiner Lokalität (z.B. Wohnort und Umgebung) her. [8]

Das Angebot frei zugänglicher wissenschaftlicher Klimadaten besteht weiterhin, jedoch sind diese Daten für Laien schwer verständlich. Dienste wie der DWD bieten Klimadaten frei zur Verfügung an, jedoch richten sich diese Daten primär an Fachleute und sind dadurch schwer nutzbar für Menschen ohne technisches Vorwissen.

## 1.3 Forschungsfrage

Zur Umsetzung des Portals werden folgende Hauptfragen untersucht:

- Wurden schon ähnliche Portale umgesetzt und veröffentlicht?
- Welche Open Source-Technologien eigenen sich für eine Umsetzung?
- Welche Open Data-Datenanbieter können benutzt werden, die sowohl historische als auch aktuelle meteorologische Daten liefert?
- Welche Anforderungen müssen erfüllt werden, damit das Portal möglichst für einen breiten Teil der Gesellschaft anwendbar und benutzerfreundlich ist?

## 1.4 Ziel der Arbeit

Ziel dieser Arbeit ist es ein Portal zu konzipieren und umzusetzen, das Nutzern ermöglicht, die meteorologischen Änderungen ihrer persönlichen Lokalität einzusehen. Im Näheren soll dem Nutzer die Möglichkeit zur Verfügung gestellt werden, historische Daten im Portal einzusehen. Somit ergibt sich für den Nutzer das Potenzial sich eigenständig über das Thema Klimawandel zu informieren. Dabei soll das Portal die Klimadaten in einer verständlichen und benutzerfreundlichen Darstellung wiedergeben. Das Portal wird anhand Open Data und Open-Source-Software konzipiert und entwickelt. Als Werkzeuge dienen visuelle und interaktive Aufbereitungen von historischen Wetter- und Klimadaten im Bezug zur spezifischen Lokalität des Nutzers.

[7] vgl. Umweltbundesamt (2022, online)

[8] vgl. Ipsos GmbH (2025, online)

## 1.5 Abgrenzung

Der in dieser Bachelorarbeit vorliegende Entwurf des Portals sieht im ersten Schritt eine Begrenzung auf den Standort Deutschland vor. Daher wird das Portal ausschließlich mit historischen sowie gegenwärtigen Wetter- und Klimadaten Deutschlands versorgt. Aufgrund dessen ist die Nutzersprache des Portals Deutsch. Eine Prognose für die zukünftige Entwicklung des Klimas ist im Rahmen dieser Bachelorarbeit nicht vorgesehen. Weiterhin dient als einzige Open Data Quelle die Daten des DWD.

# 2 Grundlagen

In diesem Kapitel werden dem Leser für eine bessere Verständlichkeit der vorliegenden Arbeit die Grundlagen zum Klimawandel, Open Data, Open Source, API, dem DWD sowie weitere technologische Definitionen nähergebracht.

## 2.1 Klimawandel

Der Klimawandel stellt durch tiefgreifende Auswirkungen auf die Gesellschaft, Wirtschaft und Politik einer der größten Herausforderungen des 21. Jahrhunderts dar. Als Folgen des Klimawandels werden unter anderem der globale Temperaturanstieg, die Zunahme von extremen Wetterereignissen und der Anstieg des Meeresspiegels genannt.[9]

Auch in Mitteleuropa und Deutschland sind die Folgen durch Zunahme von Hitzewellen, Starkniederschlägen und Trockenperioden zu beobachten. Diese Ereignisse schränken nicht nur den Alltag, sondern sowohl die Infrastruktur als auch die Landwirtschaft ein. Im Sommer 2022 wurden Temperaturen von über 40 °C in Deutschland gemessen[10].

Ein weiteres Beispiel für die Folgen des Klimawandels ist die Starkregenkatastrophe im Juli 2021. Wie in Abbildung 1 zu sehen ist, waren die Bundesländer Nordrhein-Westfalen, Rheinland-Pfalz, Bayern und Sachsen massiv von Überschwemmungen betroffen. Die Katastrophe brachte über 180 Todesopfer, zerstörte Häuser sowie Brücken mit sich und verursachte finanzielle Schäden in Höhe von mehreren Milliarden Euro.[11]

[9] vgl. IPCC (2021, online)

[10] vgl. Deutscher Wetterdienst (2022, online)

[11] vgl. Bundesregierung (2021, online)

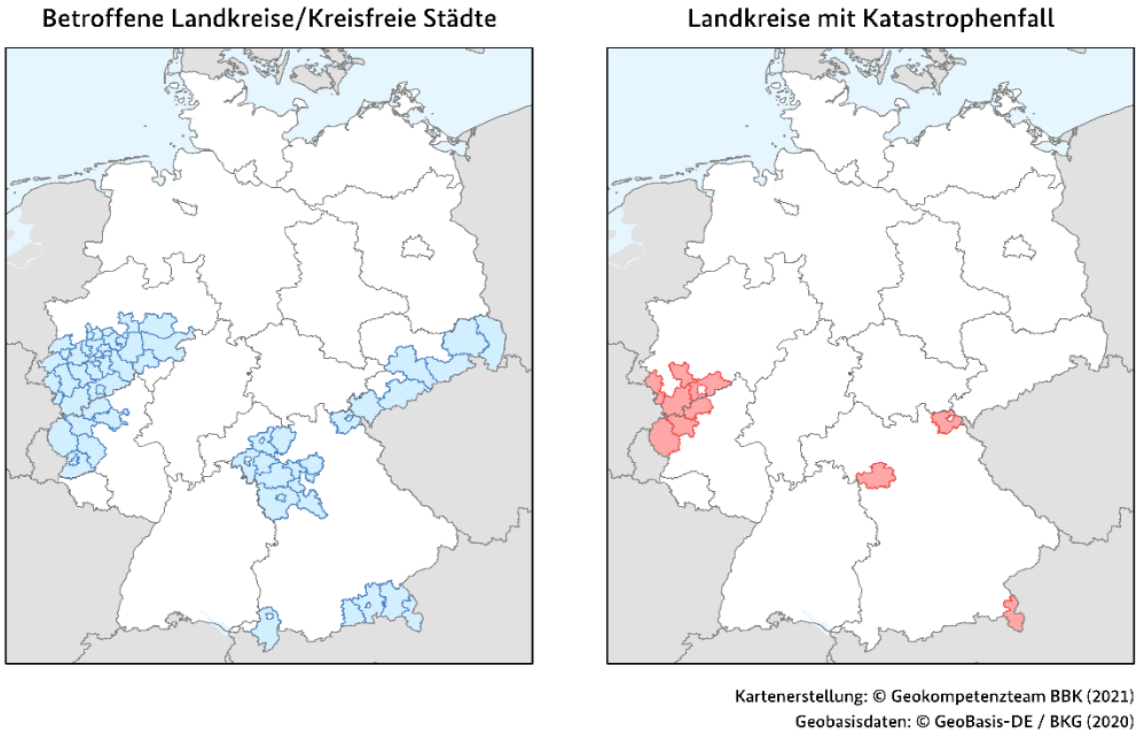



Abbildung 1: Betroffene Regionen der Starkregenkatastrophe im Juli 2021 (Bundesregierung (2021, online))

Diese Ereignisse verdeutlichen, dass es sich beim Klimawandel nicht nur um ein globales Phänomen handelt, sondern auch spürbare Auswirkungen auf Regionen Deutschlands hat. Daraus ergibt sich ein Bedarf an verständlichen und lokalen Klimadaten, welches die Gesellschaft über den Klimawandel informiert.

Wie jedoch die Umfrage des Umweltbundesamtes im Jahr 2020 zeigt, fühlt sich ein signifikanter Teil (40 %) der Bevölkerung nicht genügend informiert über den Klimawandel.[12] Ein wichtiger Grund dafür ist die Komplexität der Daten, die meist im globalen Kontext betrachtet und technisch anspruchsvoll aufbereitet werden.

## 2.2 Open Data – Open Source – API

Open Data bezeichnet Daten, die frei verwendet, weiterverarbeitet und geteilt werden dürfen. Das Konzept freier Daten soll Transparenz, Innovation und Nachvollziehbarkeit fördern – insbesondere bei der Umsetzung digitaler Informationsportale im öffentlichen Interesse, wie im Rahmen dieser Arbeit. Die Basis der Definition liegt auf den Grundsätzen der Open Definition der Open Knowledge Foundation. Diese fasst Open Data wie folgt zusammen:

[12] vgl. Umweltbundesamt (2022, online)

*Wissen ist offen, wenn jeder darauf frei zugreifen, es nutzen, verändern und teilen kann – eingeschränkt höchstens durch Maßnahmen, die Ursprung und Offenheit des Wissens bewahren.*[13]

Der Begriff Open Source bezieht sich auf Software, deren Quellcode öffentlich zugänglich ist und die von jedermann genutzt, verändert und weiterverbreitet werden kann. Die Open Source Initiative definiert Open Source anhand von zehn Kriterien. Dazu zählen unter anderem die freie Weitergabe der Software sowie der uneingeschränkte Zugang zum Quelltext.[14]

Im Kontext dieser Arbeit ist auch der Begriff API (Application Programming Interface) von zentraler Bedeutung. Eine API ist eine Programmierschnittstelle, die es verschiedenen Softwarekomponenten ermöglicht, miteinander zu kommunizieren. In Webanwendungen handelt es sich dabei meist um sogenannte REST-APIs, über die Daten durch standardisierte HTTP-Anfragen (wie GET, POST oder DELETE) bereitgestellt oder abgerufen werden können.[15] APIs sind insbesondere dann essenziell, wenn externe Open Data-Quellen automatisiert integriert oder eigene Daten für Nutzer oder andere Systeme verfügbar gemacht werden sollen.[16]

### 2.2.1 Verfügbarkeit von Open Data

Für die Konzeption und Umsetzung des Portals werden Open-Data-Datenquellen benötigt, die frei zugänglich sind und von vertrauenswürdigen Stellen bereitgestellt werden. In Deutschland ist der DWD die zentrale öffentliche Institution für meteorologische Daten. Der DWD stellt sowohl historische als auch Echtzeitdaten über das Climate Data Center (CDC) zur freien Verfügung. Die Untersuchung der angebotenen Datensätze zeigen, dass die Klimadaten bereits ab dem späten 18. Jahrhundert aufgezeichnet wurden.[17]

[13] Open Knowledge Foundation (Jahr unbekannt, online)

[14] vgl. Open Source Initiative (2024, online)

[15] REST = Representational State Transfer; GET = Daten lesen; POST = Daten schreiben; DELETE = Daten löschen.

[16] vgl. Mozilla (Jahr unbekannt, online)

[17] https://opendata.dwd.de/climate_environment/CDC/observations_germany/climate/daily/kl/BESCHREIBUNG_obsgermany-climate-daily-kl_de.pdf

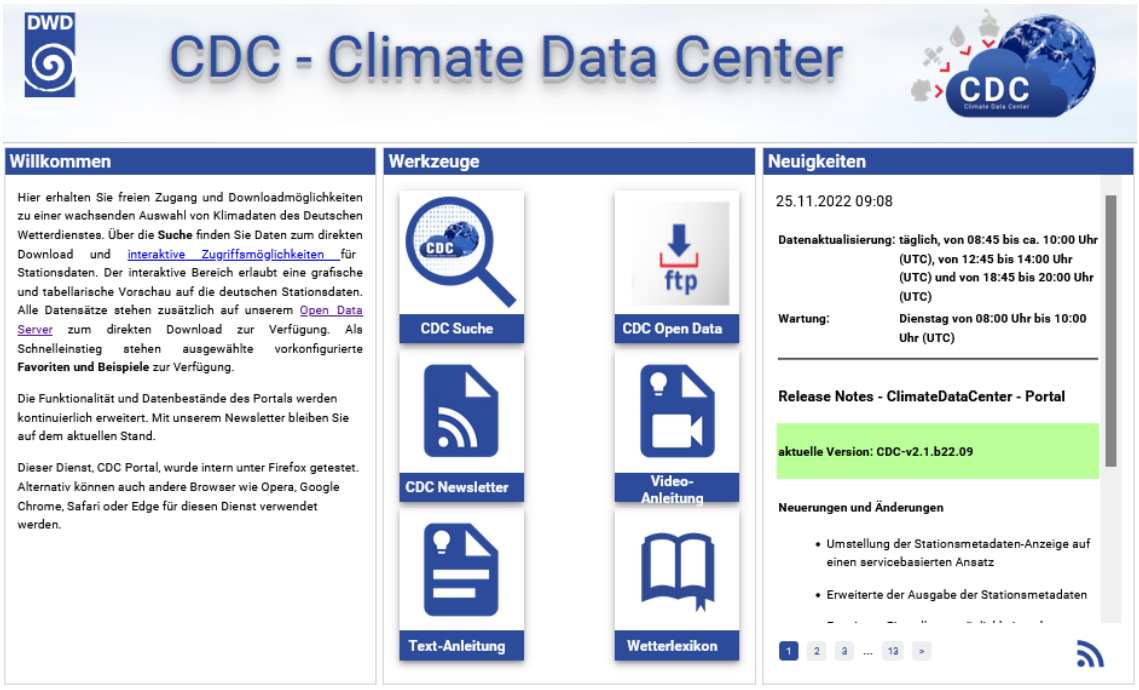



Abbildung 2: Climate Data Center (https://cdc.dwd.de/portal/)

Die Daten des DWD unterliegen einem kontinuierlichen Qualitätssicherungsprozess. Seit Ende der 1970er-Jahre werden sie durch ein standardisiertes Prüfverfahren systematisch kontrolliert. Für Messreihen vor diesem Zeitraum wird seit dem Jahr 2007 sukzessive eine nachträgliche Qualitätsprüfung durchgeführt.[18]

Während den Messungen werden unter anderem folgende meteorologische Parameter erfasst:

- Station ID,
- Windspitze und -geschwindigkeit,
- Niederschlagshöhe und -form,
- Sonnenscheindauer,
- Schneehöhe,
- Bedeckungsgrad,
- Dampf- und Luftdruck,
- Lufttemperatur (Mittel, Minimum, Maximum),
- Relative Feuchte.

[18] vgl. Deutscher Wetterdienst (Jahr unbekannt, online)

Diese Daten liegen in einem maschinenlesbaren Format (CSV)[19] vor, wo die einzelnen Werte nach Semikolon aufgeteilt sind. Für den Zugang der Daten wird vom DWD keine Authentifizierung benötigt.

### 2.2.2 Open-Source-Technologien

Für webbasierte Portale stehen zahlreiche Open-Source-Technologien zur Verfügung. In den letzten Jahren haben sich Open-Source-Lösungen als fester Bestandteil moderner Webentwicklung etabliert. Besonders im Kontext von Open-Data-Portalen ermöglichen sie eine kosteneffiziente, rechtlich unproblematische und community-getragene Umsetzung.

Auch im öffentlichen Sektor wird die Nutzung von Open Data und Open-Source aktiv gefördert, etwa durch das Bundesministerium des Innern[20] oder die Europäische Kommission im Rahmen der Digital Europe Strategy[21].

Im Laufe der Arbeit werden Open-Source-Komponenten und Technologien vorgestellt und im Anwendungskontext analysiert.

## 2.3 Der Deutsche Wetterdienst

Der Deutsche Wetterdienst ist die nationale meteorologische Behörde der Bundesrepublik Deutschland und untersteht dem Bundesministerium für Digitales und Verkehr. Als Bundesoberbehörde mit Sitz in Offenbach am Main wurde der DWD im Jahr 1952 gegründet und hat unter anderem die gesetzliche Aufgabe, die Bevölkerung, Behörden und Einrichtungen des Bundes mit meteorologischen Informationen zu versorgen (§ 4 DWD-Gesetz)[22]. Neben der Wettervorhersage und Warnung vor Unwetterlagen ist der DWD auch für die Klimabeobachtung und Datenarchivierung zuständig.

Durch die klare Strukturierung, rechtliche Offenheit und die hohe fachliche Qualität ist der DWD eine äußerst geeignete Datenquelle für datenbasierte Wetter- und Klimaportale. Im Rahmen dieser Arbeit bilden die Daten des DWD die Grundlage für die Konzeption und Umsetzung des Portals.

[19] CSV (Comma-Separated Values) ist ein einfaches Textformat zum Speichern tabellarischer Daten, bei dem die einzelnen Werte durch Kommata (oder andere Trennzeichen) getrennt sind.

[20] vgl. Bundesregierung (2021, online) (2)

[21] vgl. Europäische Kommission (Jahr unbekannt, online)

[22] Vgl. Bundesamt für Justiz (Jahr unbekannt, online)

## 2.4 Technologische Grundlagen

Für die Konzeption und Umsetzung des Klima- und Wetterinformationsportals ist der Einsatz von moderner Webtechnologien, Datenbanksysteme sowie automatisierter Datenverarbeitungsverfahren notwendig. In diesem Kapitel werden die zentralen Rollen beschrieben, die für die Implementierung der Anwendung verwendet werden.

### 2.4.1 Webarchitektur und Systemdesign

Die Architektur des Portals basiert auf dem Modell einer mehrschichtigen Client-Server-Architektur. Die Anwendung besteht aus Folgendem:

- Backend: Ruft und verarbeitet die Daten vom DWD und stellt sie über eine REST-API zu Verfügung,
- Datenbank: Relationale Datenbank speichert die Daten vom DWD.
- Frontend: Stellt Benutzeroberfläche zur Verfügung,

Die Trennung folgt dem Prinzip der Entkopplung von Darstellung, Logik und Datenhaltung. Somit ist die Anwendung und die damit zusammenhängende Komponenten modular, erweiterbar und wartungsfreundlich.

### 2.4.2 Backend

Als zentrales Kommunikationssystem zwischen den verschiedenen Komponenten dient das Backend. Es stellt die Verbindung zum DWD, zur Datenbank und zum Frontend über REST-API her. Die vom DWD abgerufenen Wetterdaten werden im Backend verarbeitet und in der Datenbank gespeichert.

Der Datenabruf erfolgt automatisch und ohne Benutzereingabe. Für jede Stations-ID wird pro Tag genau ein Datensatz erfasst. Falls beim Import zwei Datensätze mit demselben Datum für eine Stations-ID vorliegen, wird der ältere Datensatz durch den neueren ersetzt, um Duplikate zu vermeiden. Diese Vorgehensweise ist notwendig, da nachträgliche Korrekturen durch den DWD in neueren Dateien enthalten sein können.

Weitere Funktionen des Backends umfassen den Zugriff über REST-API-Endpunkte sowie den optionalen Export der Daten als CSV-Datei. Über die API-Endpunkte kann die Datenbank über das Backend angesprochen werden und gibt daraufhin eine JSON[23]-Antwort mit den angeforderten Parametern zurück.

Ein API-Call könnte folgendermaßen aussehen:

- GET-Anfrage: /api/weather/filter?city=berlin&startDate=2000-01-01

[23] JSON ist ein textbasiertes Datenformat zur strukturierten Darstellung von Daten, das häufig für den Austausch zwischen Server und Client verwendet wird.

### 2.4.3 Datenbank

Die Speicherung der verarbeiteten Wetterdaten erfolgt in einer relationalen Datenbank. Relationale Datenbanken organisieren Daten in Tabellen und ermöglichen mittels SQL[24]-Abfragen das Einfügen, Ändern und Auslesen der Informationen.

Die Datenbankstruktur sollte so konzipiert werden, dass sowohl historische als auch aktuelle Datensätze effizient gespeichert und abgefragt werden können. Die Kommunikation zwischen dem Frontend und der Datenbank, soll über die Backend anhand API-Endpunkte erfolgen.

### 2.4.4 Frontend

Das Frontend sollte dem Nutzer die Daten, die von der Backend verarbeitet und in der Datenbank gespeichert werden, dem Nutzer auf einer möglichst verständlichen Art und Weise angezeigt werden. Die Daten werden mittels Karten und Diagramme visualisiert und dem Nutzer wiedergegeben. Interaktivität und schnelle Reaktionszeiten der Frontend werden gewährleistet, um eine zufriedenstellendes Nutzererlebnis zu ermöglichen.

[24] SQL (Structured Query Language) ist eine standardisierte Sprache zur Verwaltung und Abfrage relationaler Datenbanken, mit der Daten eingefügt, geändert, gelöscht und ausgewertet werden können.

# 3 Stand der Technik

Im dritten Kapitel dieser Bachelorarbeit wird der Stand der Technik untersucht. Hierbei wird auf bestehende Wetter- und Klimainformationsportale, verwandte Untersuchungen, relevante Standards sowie auf Lücken und Verbesserungspotenzial von bereits bestehenden Portalen eingegangen.

## 3.1 Bestehende Wetter- und Klimainformationsportale

Zur Entwicklung eines benutzerfreundlichen Klima- und Wetterinformationsportals ist die Analyse bestehender Portale unerlässlich. Es existieren diverse nationale und internationale Angebote, die meteorologische Daten öffentlich zugänglich machen. Diese unterscheiden sich jedoch stark hinsichtlich Benutzerfreundlichkeit, Funktionsumfang und Zielgruppenorientierung. Im Folgenden werden die drei ausgewählten Portale Meteostat, WeatherSpark und LoKlim vorgestellt und hinsichtlich ihrer Stärken und Schwächen verglichen. Die Anzahl der Portale wird auf drei eingegrenzt, um den Rahmen der Bachelorarbeit nicht zu sprengen.

### 3.1.1 Meteostat

Meteostat stellt frei verfügbare Wetter- und Klimadaten aus internationalen Quellen bereit, darunter auch Daten des DWD. Die Plattform bietet einen verlässlichen und schnellen Zugriff auf Langzeit-Zeitreihendaten, insbesondere für Temperatur, Niederschlag und Windgeschwindigkeit, an. Ein Vorteil ist die API, mit der Entwickler auf strukturierte JSON-Daten zugreifen können.[25]

Auf der Meteostat Startseite (Abbildung 3) hat man die Möglichkeit, entweder nach einem Ort zu suchen oder über die Funktion „Mein Standort“ die nächstlegende Station in seiner Umgebung automatisch abzurufen. Nachdem eine Station aufgerufen wird (Abbildung 4), können die meteorologischen Daten in Diagrammen visualisiert werden. Die Visualisierungen sind funktional, jedoch technisch anspruchsvoll dargestellt. Eine detaillierte Untersuchung mit den Daten, z. B. Filterung nach größeren Zeiträumen oder grafische Vergleiche, ist nur eingeschränkt möglich.

Der Quellcode ist über Github[26] offen verfügbar und legt dar, welche Technologien für die Entwicklung eingesetzt worden sind.[27] Das Backend wird mit der Programmierspra-

[25] vgl. Meteostat (Jahr unbekannt, online)

[26] GitHub ist eine webbasierte Plattform zur Versionsverwaltung von Quellcode und Funktionen wie Zusammenarbeit, Code-Hosting, Issue-Tracking und Projektmanagement bietet (https://github.com/).

che Python entwickelt und die Frontend besteht aus Vue.js, einem JavaScript Framework[28] für Webentwicklung.

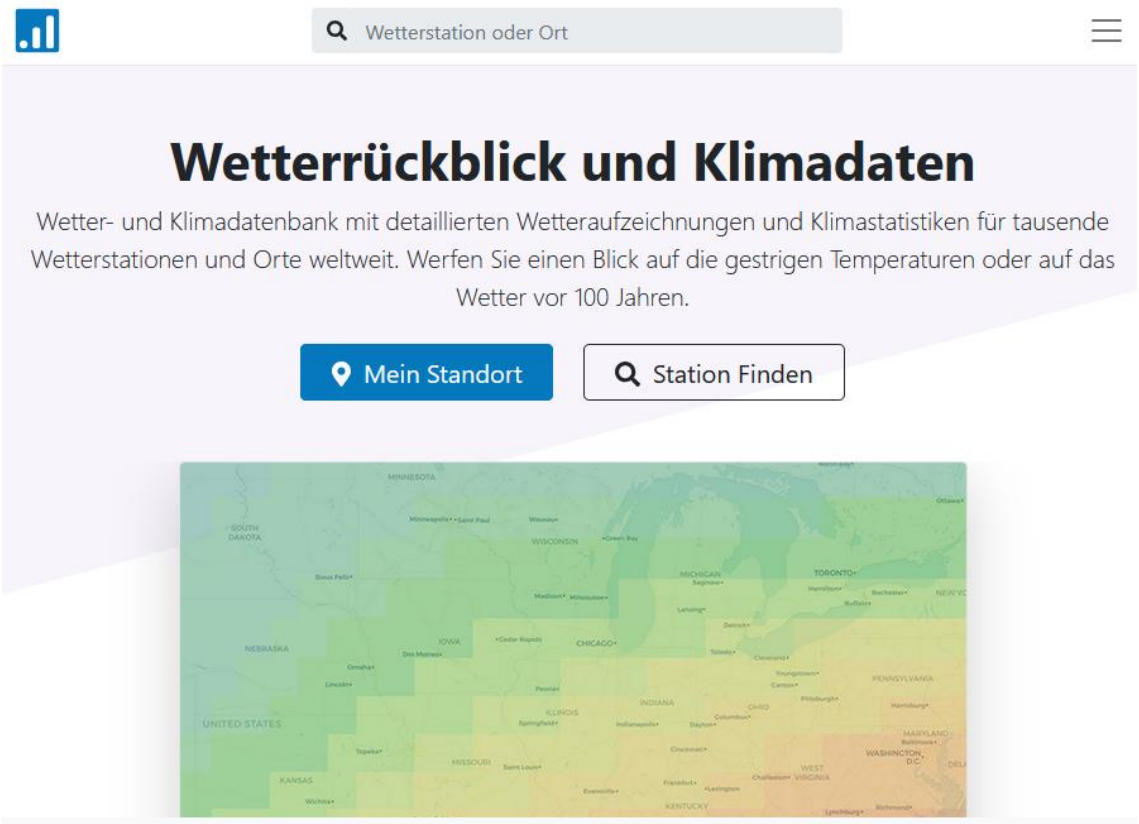



Abbildung 3: Meteostat Startseite (https://meteostat.net/de/)

[27] https://github.com/meteostat

[28] Frameworks sind wiederverwendbare Vorlagen, die eine Grundlage für die Erstellung von Softwareanwendungen bilden.

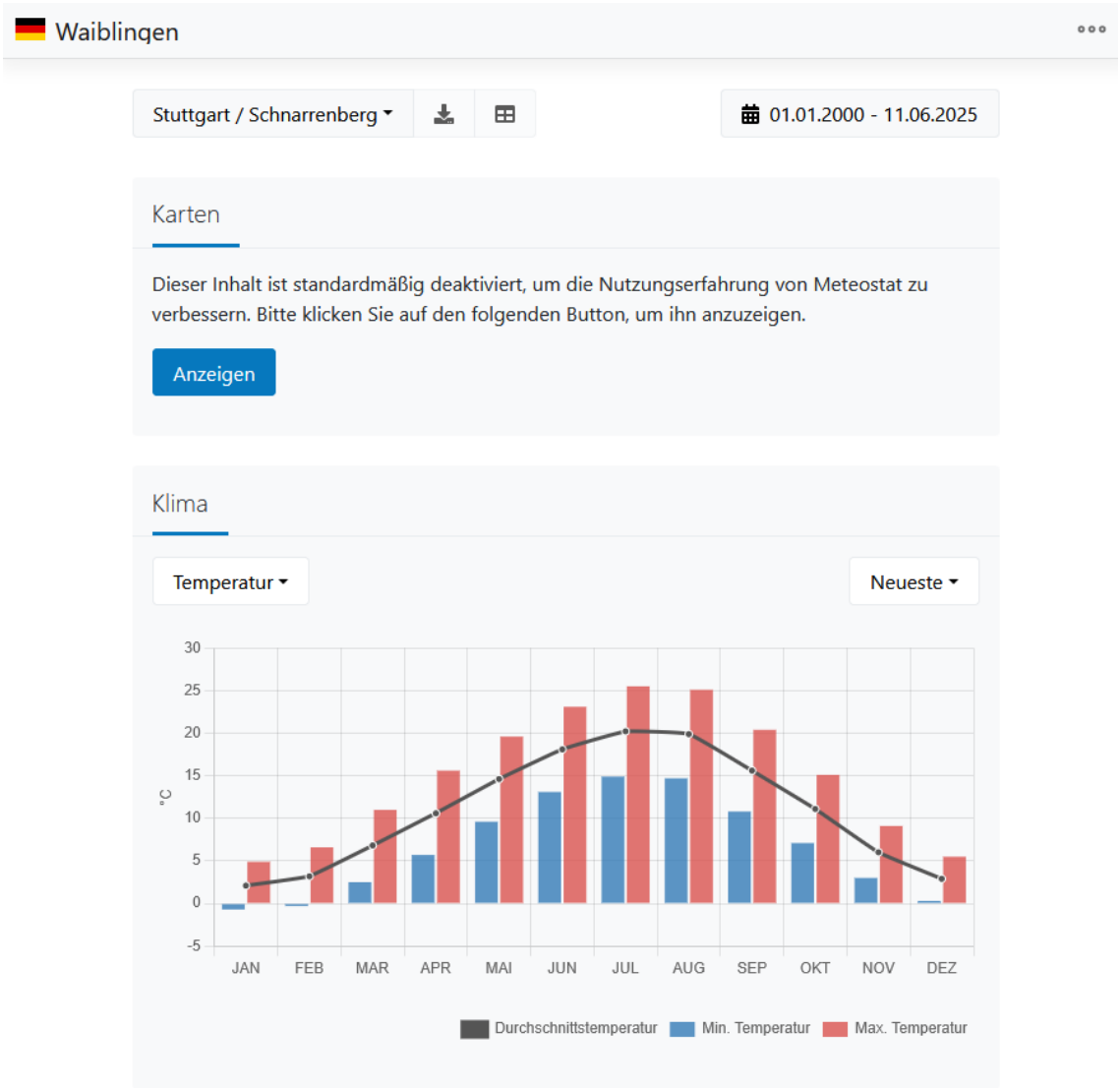



Abbildung 4: Meteostat Waiblingen
(https://meteostat.net/de/place/de/waiblingen?s=10739&t=2000-01-01/2025-06-11)

### 3.1.2 WeatherSpark

WeatherSpark legt den Fokus auf eine visuelle Darstellung durchschnittlicher Wetterdaten. Die meteorologischen Rohdaten werden für kostenpflichtige Abonnements angeboten. Eine API wird dabei nicht angeboten. Die Preise für das Abonnements belaufen sich von 89,99€ bis 224,00€ (Abbildung 7).[29]

Die Startseite von WeatherSpark begrüßt den Nutzer mit einem Suchfeld für die spezifische Ortssuche. Zusätzlich werden mehrere Grafiken von internationalen Großstädten wie New York City, Tokyo oder Seoul angezeigt (Abbildung 5). Sobald eine spezifische Suche durchgeführt wird (Abbildung 6), werden die meteorologischen Daten mit interaktiven Diagrammen dargestellt. Die interaktiven Diagramme zeigen Temperaturverläufe, Sonnenscheindauer, Wind und Niederschlag auf Monats- und Tagesbasis an. In textbasierter Form werden die Diagramme erläutert und ein Fazit über das allgemeine Klima in der Region ausgegeben. Eine Darstellungen über mehrere Jahre hinweg existierten nicht.

Das Fazit für den Standort Waiblingen sieht wie folgt aus: „In Waiblingen sind die Sommer angenehm, die Winter sind sehr kalt und windig, und es ist das ganze Jahr über teilweise bewölkt. Im Verlauf des Jahres bewegt sich die Temperatur in der Regel zwischen -2 °C und 25 °C und liegt selten unter -9 °C oder über 32 °C. Aufgrund der Tourismus-Bewertung ist von Mitte Juni bis Anfang September die beste Zeit des Jahres für einen Besuch in Waiblingen für Aktivitäten bei warmem Wetter.“[30]

Das Portal bietet seinen Quellcode nicht öffentlich an. Eine Analyse welche Technologien benutzt wird, ist nicht möglich. Auch bei der Untersuchung des HTML-Codes, ist nicht festzustellen, welche Technologien eingesetzt werden.

[29] vgl. WeatherSpark (Jahr unbekannt, online)

[30] WeatherSpark (Jahr unbekannt, online) (2)


Abbildung 5: WeatherSpark Startseite (https://de.weatherspark.com)

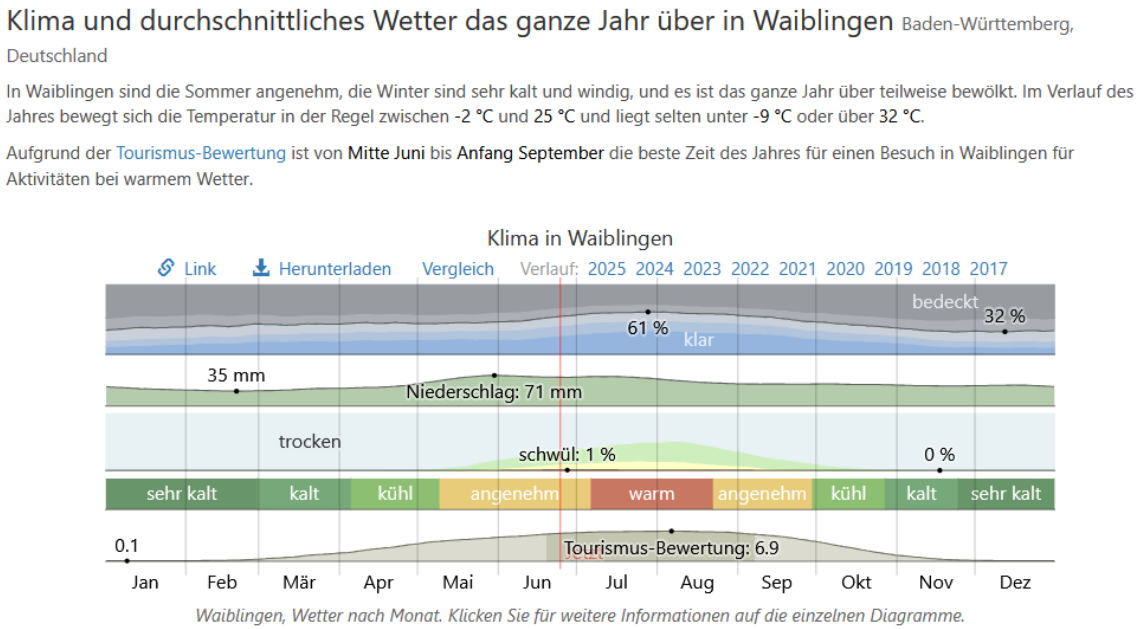



Abbildung 6: WeatherSpark Waiblingen (https://de.weatherspark.com/y/63769/Durchschnittswetter-in-Waiblingen-Baden-W%C3%BCrttemberg-Deutschland-das-ganze-Jahr-%C3%BCber#)

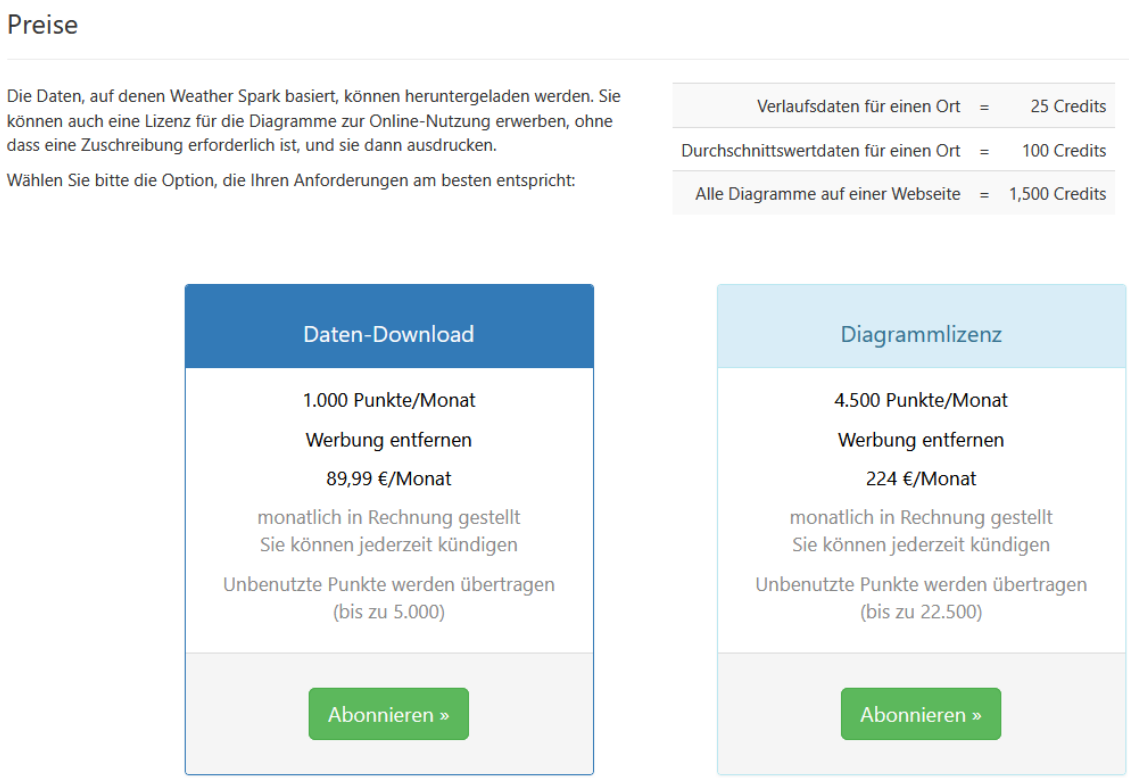



Abbildung 7: WeatherSpark Pricing (https://de.weatherspark.com/pricing)

### 3.1.3 LoKlim – Lokale Klimaanpassung

Dieses vom Umweltbundesamt unterstützte Portal fokussiert sich auf die Klimaanpassung auf kommunaler Ebene. Es bietet Hintergrundwissen und einen Klimasteckbrief für jeden Landkreis im Bundesland Baden-Württemberg an. Eine API oder ein Zugriff auf die meteorologischen Daten ist nicht möglich.[31]

Im Gegensatz zu den zuvor beschriebenen Portalen liegt bei LoKlim die Besonderheit vor, dass auf der Startseite das Forschungsprojekt beschrieben wird (Abbildung 8). Dadurch bietet sich dem Nutzer die Möglichkeit an, weiteres Hintergrundwissen über das Portal zu erlangen. Mit dem Klick auf „Lokales Klimaportal“ wird dem Nutzer eine Karte bereitgestellt, welche meteorologische Daten auf kommunaler Ebene visualisiert (Abbildung 9). In Abbildung 9 ist die Karte vom Bundesland Baden-Württemberg zu sehen. Dabei werden sowohl historische Klimadaten als auch mögliche prognostizierte Klimadaten für die Zukunft visualisiert. Einzelne Tageswerte sind aus dem Portal nicht zu entnehmen. Es besteht auch die Möglichkeit einen Klimasteckbrief für die einzelnen Kommunen herunterzuladen, welches die Daten in Textform wiedergibt (Abbildung 10).

Auch bei diesem Portal ist ein Einblick in den Quellcode und die benutzten Technologien nicht möglich. Die einzige Technologie, die zu sehen ist, ist die Leaflet[32]-Karte die für die Kartenvisualisierung benutzt wird (Abbildung 9).

[31] vgl. LoKlim (Jahr unbekannt, online)

[32] Leaflet ist eine JavaScript-Bibliothek zur Darstellung interaktiver Karten im Web.

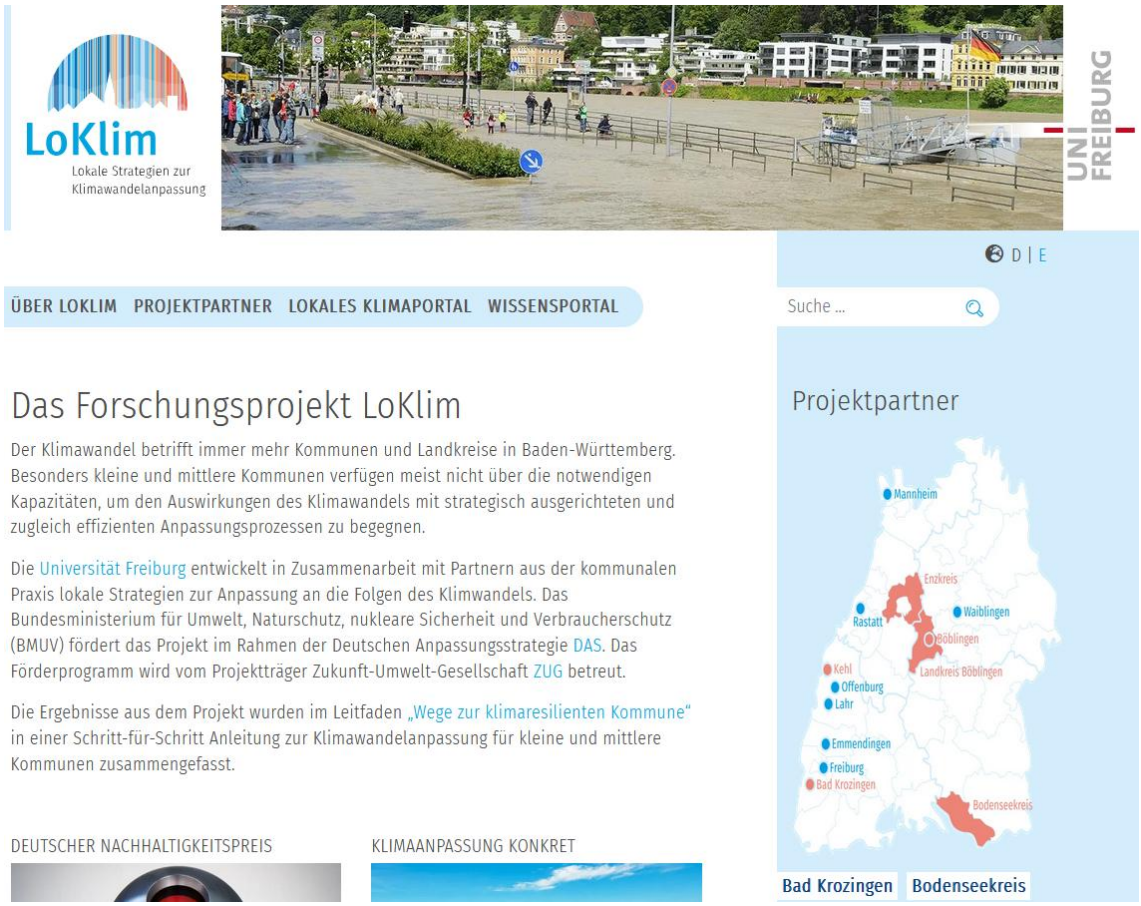



Abbildung 8: LoKlim Startseite (https://lokale-klimaanpassung.de/)

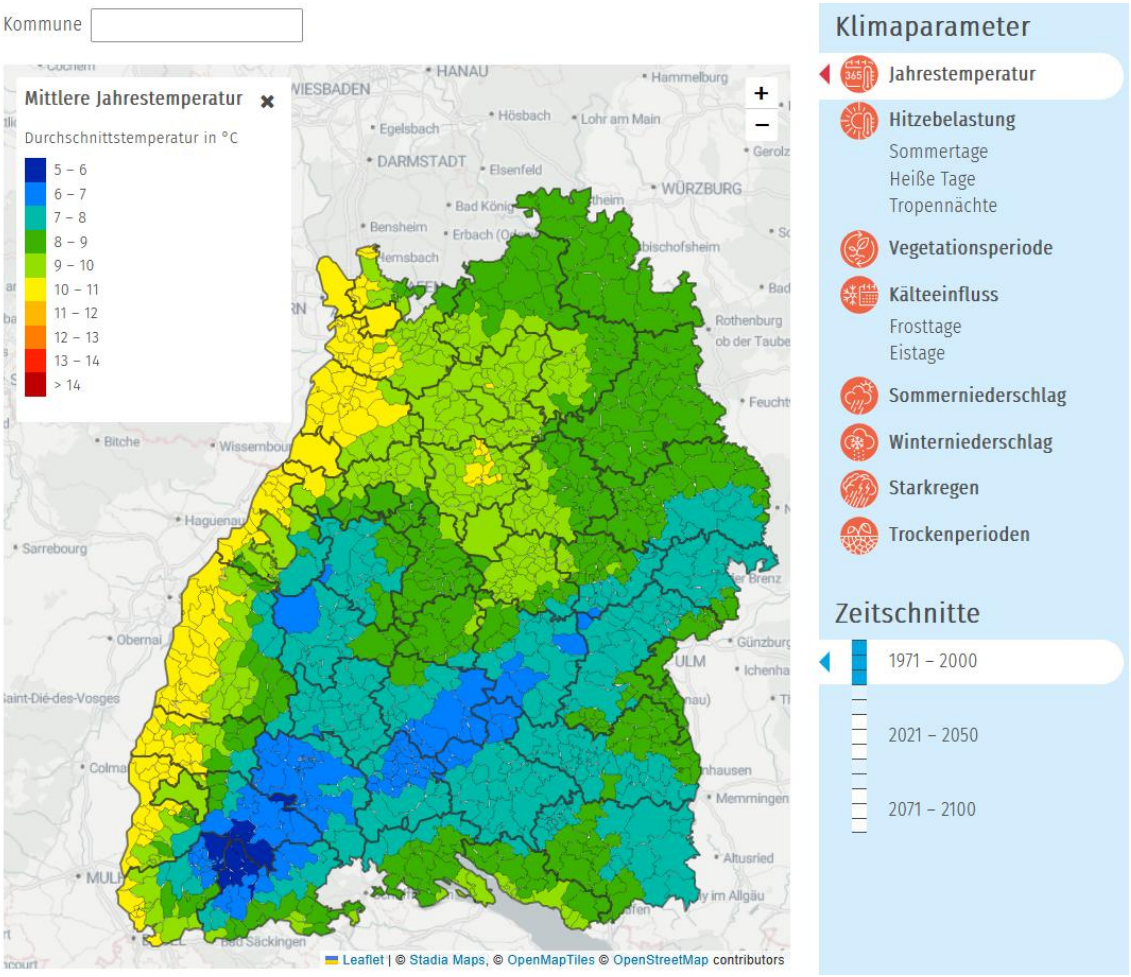



Abbildung 9: LoKlim Klimaportal (https://lokale-klimaanpassung.de/lokales-klimaportal/)

Waiblingen

Klimasteckbrief

Interpretationshilfe auf der nächsten Seite

| | | 1971-2000 | Nahe Zukunft 2021 - 2050 | Ferne Zukunft 2071 - 2100 |
|---|---|---|---|---|
| Mittlere Jahrestemperatur [°C] | | 9,5 | 10,9 (10,3 - 11,1) | 13,2 (12,4 - 13,8) |
| Sommertage [Tag] | Anzahl der Tage mit Tmax > 25°C | 42 | 54 (50 - 67) | 84 (51 - 98) |
| Heiße Tage [Tag] | Anzahl der Tage mit Tmax ≥ 30°C | 8 | 15 (13 - 22) | 36 (21 - 45) |
| Tropennächte [Tag] | Anzahl der Tage mit Tmin > 20°C | 0 | 2 (0 - 3) | 14 (9 - 25) |
| Vegetationsperiode [Tag] | Anzahl der Tage zwischen der ersten Phase mit mindestens 6 Tagen Tmean > 5°C und erster Phase nach dem 1.6. mit mindestens 6 Tagen Tmean < 5°C | 260 | 281 (274 - 286) | 321 (312 - 329) |
| Frosttage [Tag] | Anzahl der Tage mit Tmin < 0°C | 79 | 58 (44 - 71) | 33 (15 - 41) |
| Eistage [Tag] | Anzahl der Tage mit Tmax < 0°C | 14 | 7 (4 - 12) | 2 (1 - 3) |
| Winterniederschlag [mm] | Niederschlagssumme (Dec, Jan, Feb) | 166 | 179 (162 - 200) | 197 (179 - 222) |
| Sommerniederschlag [mm] | Niederschlagssumme (Jun, Jul, Aug) | 248 | 246 (221 - 271) | 217 (199 - 276) |
| Starkniederschlag [Tag] | Anzahl der Tage mit Niederschlag > 20mm | 4 | 5 (4 - 6) | 6 (5 - 7) |
| Trockenperioden [Periode] | Anzahl der Perioden mit mind. 4 aufeinanderfolgenden Trockentagen (Niederschlag < 1mm) | 34 | 33 (28 - 47) | 33 (30 - 78) |

Abbildung 10: LoKlim Waiblingen Klimasteckbrief (https://lokale-klimaanpassung.de/wp-content/uploads/2022/11/08119079_Waiblingen_steckbrief.pdf)

### 3.1.4 Fazit des Vergleichs

Bestehende Wetterportale adressieren entweder technische Zielgruppen oder bieten stark eingeschränkte Visualisierungen für Endnutzer an. Eine Plattform, die sowohl technisch valide Daten liefert als auch intuitiv visuell zugänglich aufbereitet, insbesondere mit lokalem Bezug und interaktiven Filtern, ist aktuell nicht vorhanden.

Diese Lücke bildet die Ausgangsbasis für die Konzeption des im Rahmen dieser Arbeit entwickelten Portals, das historische Daten visuell aufbereitet, lokalisiert darstellt und auch für Nicht-Fachleute verständlich nutzbar macht.

## 3.2 Verwandte Untersuchungen

Im folgenden Abschnitt werden ausgewählte Studien und Initiativen vorgestellt, die sich mit Klimakommunikation, Datenzugänglichkeit sowie der Entwicklung von Informationssystemen im Kontext des Klimawandels befassen. Ziel ist es, diese Arbeit in den bestehenden Forschungs- und Entwicklungsstand einzuordnen und Anschlussmöglichkeiten aufzuzeigen.

### 3.2.1 Kommunikationsprinzipien für den Klimawandel

Effektive Klimakommunikation erfordert mehr als Vermittlung wissenschaftlicher Fakten. Climate Outreach fassen den sozialwissenschaftlichen Forschungsstand dahingehend zusammen, dass Menschen Klimainformationen nicht primär durch nüchterne Abwägung von Fakten aufnehmen, sondern dass die Ansprache auf der Ebene von Werten und Emotionen erfolgen muss. Die Autoren präsentieren zehn evidenzbasierte Prinzipien für wirkungsvolle Klimakommunikation, die als Leitlinien dienen können (Abbildung 11).


Abbildung 11: Prinzipien für Klimakommunikation (Climate Outreach 2022, online)

Es wird unter anderem empfohlen, Anknüpfungspunkte an die Werte der Zielgruppe zu finden und vertrauenswürdige Vermittler einzusetzen, statt allein auf abstrakte Expertendaten zu setzen. Darüber hinaus betonen Climate Outreach, dass Geschichten und Bilder einprägsamere Vermittlungsformen darstellen als reine Zahlenreihen. Kraftvolle Narrative und visuelle Darstellungen können komplexe Klimafakten greifbarer machen. Wichtig ist laut den Autoren auch, dass Informationen zwar korrekt und wissenschaftlich fundiert sein müssen, Unsicherheiten aber transparent und vorsichtig kommuni-

ziert werden sollten, um Missverständnisse zu vermeiden. Insgesamt zielen die Kommunikationsprinzipien darauf ab, Menschen von der bloßen Besorgnis zum aktiven Handeln zu führen.[33]

Das Projekt Climate Visuals von Climate Outreach hat auf Basis empirischer Studien sieben evidenzbasierte Prinzipien für effektive visuelle Klimakommunikation formuliert. Diese lauten:[34]

1. Zeige echte Menschen: Bilder mit menschlichem Bezug erzeugen stärkere emotionale Resonanz
2. Erzähle neue Geschichten: Vermeide stereotypische Darstellungen wie schmelzende Gletscher, setze stattdessen auf visuelle Vielfalt.
3. Zeige Auswirkungen des Klimawandels, nicht nur Ursachen: Betone das „Hier und Jetzt".
4. Nutze Bilder mit lokalem Bezug: Menschen reagieren stärker auf Inhalte aus ihrer Umgebung.
5. Zeige Lösungen und Zusammenhalt: Visualisierungen sollten Hoffnung und Handlungsfähigkeit vermitteln.
6. Achte auf Authentizität und Glaubwürdigkeit: Keine überdramatisierten Darstellungen.
7. Teste deine Bilder mit der Zielgruppe: Visuelle Kommunikation sollte evaluiert werden

Für das geplante Webportal bedeutet dies, dass Visualisierungen (z. B. Temperaturverläufe, Niederschlagsänderungen, etc.) bewusst gestaltet und mit lokalem Bezug und handlungsorientierter Aussagekraft versehen werden sollten. Diese Prinzipien stimmen auch mit den Empfehlungen von Climate Outreach überein, die visuelle Klimakommunikation als Schlüssel zur Aktivierung von Zielgruppen sehen.

Auch die Vereinten Nationen unterstreichen den handlungsorientierten Charakter der Klimawandel-Kommunikation. Klimakommunikation soll demnach Menschen aufklären und mobilisieren, um dem Klimawandel aktiv zu begegnen. Jeder Einzelne könne dazu beitragen, sei es durch das Erheben der eigenen Stimme, das Teilen von Lösungen oder das Eintreten für Veränderungen. Diese Perspektive der UN deckt sich auch mit den Erkenntnissen von Climate Outreach, die besagen den Schwerpunkt auf breite Öffentlichkeitsbeteiligung und emotionaler Ansprache zu legen. Hierbei ist die Harmonie von wissenschaftlicher Genauigkeit bei bestehender Transparenz zu berücksichtigen. Das UN-Leitbild betont, dass faktenbasierte Inhalte so aufbereitet werden müssen, sodass sie zum Handeln ermutigen, anstatt zu Angst zu erzeugen.[35]

[33] vgl. Climate Outreach (2022, online)

[34] vgl. Climate Outreach (Jahr unbekannt, online)

[35] vgl. United Nations (Jahr unbekannt, online)

### 3.2.2 Einbindung der Öffentlichkeit in Klimarisikokommunikation

Das Briefing-Paper „ResilRisk“ formuliert sieben zentrale Empfehlungen für Kommunikatoren, um die Öffentlichkeit über Klimarisiken und Anpassungsmaßnahmen wirksam zu informieren. Die Kernaussagen lauten:[36]

1. Hohe Besorgnis nutzen: Die öffentliche Sorge über Klimawandel ist groß und parteiübergreifend, was ein guter Ausgangspunkt für Engagement ist.
2. Zunehmende Erfahrung mit Extremereignissen als Anlass nehmen: Die Wahrnehmung von Hitzewellen und Wetterextremen steigt, was neue Gesprächsanlässe schafft.
3. Klimarisiken lokal und wertebezogen kommunizieren: Risiken sollen mit geteilten Werten verbunden und durch konstruktive Lösungen ergänzt werden.
4. Milderung und Anpassung nicht trennen: Beide Aspekte sind kommunikativ miteinander verbunden.
5. Gesundheit und Wohlbefinden betonen: Dies sind zentrale Ankerpunkte für breite Akzeptanz.
6. Jenseits von Emissionszielen kommunizieren: Auch Themen wie sozial gerechte Transformation gehören dazu.
7. Von Besorgnis zu Engagement: Der Weg zu langfristiger Beteiligung ist das Hauptziel.

Diese Empfehlungen unterstützen den Ansatz, ein Informationsportal nicht nur als Datenplattform, sondern auch als Kommunikationsinstrument zu gestalten. Die Integration von lokal relevanten Risiken kann dazu beitragen, Bürgerinnen und Bürger aktiv einzubinden, was eines der Ziele dieser Arbeit ist.

### 3.2.3 Nutzung offener Daten in Klimainformationsportalen

Offene Klimadaten bilden das Fundament für Transparenz und Nachvollziehbarkeit in Klimainformationsportalen. Das geplante Portal soll maßgeblich auf Open Data, insbesondere den frei verfügbaren Daten des DWD, basieren.

Dieser offene Datenansatz stimmt mit Entwicklungen in der Klimadienst-Community überein. So verweist der gemeinsame Wissenschaftsplan des UK Climate Resilience Programme (UKCRP) darauf, dass moderne Klima-Services verstärkt auf öffentlich zugängliche Datengrundlagen setzen. Ein Beispiel sind die UK Climate Projections 2018, die als frei verfügbare Klimaprojektionen bereitgestellt wird und die Entwicklung neuartiger Klimadienste ermöglicht haben. Die britischen Klimawissenschaftler waren führend an der Erstellung solcher Datensätze beteiligt, was die Bedeutung von offenen und qualitativ hochwertigen Klima- und Wetterdaten für Forschung und Praxis unter-

[36] vgl. Climate Outreach (2020, online)

streicht. Open Data-Ansätze ermöglichen es unterschiedlichsten Beteiligten (z.B. Politik, Wissenschaft, Wirtschaft, Gesellschaft, etc.) auf die gleichen verlässlichen Informationen zuzugreifen.[37]

Für ein Bürgerportal bedeutet das, dass die Nutzer Schnittstellen zu vertrauenswürdigen Quellen vorfinden. Der DWD beispielsweise stellt über sein Climate Data Center umfangreiche historische Wetter- und Klimareihen, aber auch aktuelle Messdaten und Projektionen kostenlos bereit. Solche amtlichen Datensätze verleihen dem Portal Glaubwürdigkeit, da sie wissenschaftlich validiert sind und regelmäßig aktualisiert werden. Eine Herausforderung bleibt allerdings, die Fülle an offenen Daten benutzerfreundlich aufzubereiten. Rohdaten in Form von Tabellen oder FTP-Servern[38] (wie sie oft von nationalen Wetterdiensten bereitgestellt werden) sind für einen Großteil der Bevölkerung schwer zugänglich. Das UKCRP betont im Science Plan die Notwendigkeit, Forschungsergebnisse und Daten so aufzubereiten, dass sie für Endanwender nützlich und nutzbar sind.[39]

Für das Portal, welches im Rahmen dieser Bachelorarbeit konzipiert und umgesetzt wird, bedeutet es, dass die offenen DWD-Daten in verständlicher Form anzubieten sind. Beispielsweise kann dies durch interaktive Abfragen, Karten und Grafiken ermöglicht werden, anstatt Nutzer mit unstrukturierten Datenmengen allein zu lassen. Weiterhin ist zu beachten, dass der offene Zugang zu Daten einerseits Transparenz schafft, andererseits aber die Gefahr von Fehlinterpretationen mit sich bringt.

### 3.2.4 Technische Umsetzung und Open-Source-Software

Ein Blick auf verwandte technische Untersuchungen verdeutlicht zudem, dass Interoperabilität[40] und Standardisierung zentrale Anforderungen für das Portal sind. So untersuchen Carreras-Coch et al. in ihrem Review Kommunikationslösungen für Notfallsituationen und leiten daraus eine verschiedenartige Kommunikationsarchitektur ab. Auch wenn deren Kontext der Krisenfall (z.B. Waldbrände oder Überschwemmungen) ist, lassen sich Prinzipien auf das Vorhaben dieser Arbeit übertragen. Die vorgeschlagene Architektur vernetzt verschiedene Sensoren, Geräte und Akteure in einem großflächigen Ubiquitous Sensor Network[41], um in Echtzeit Daten zu sammeln und auszutauschen. Carreras-Coch et al. betonen zudem, dass in ihrem Untersuchungsfeld viele Teillösungen sehr szenariospezifisch waren und dass ein Bedarf an standardisier-

[37] vgl. Met Office (2019, online)

[38] FTP-Server (File Transfer Protocol) sind Server, die das Übertragen, Speichern und Abrufen von Dateien über das FTP-Protokoll ermöglichen – häufig genutzt für den Austausch großer Datenmengen über Netzwerke.

[39] vgl. Met Office (2019, online)

[40] Interoperabilität bezeichnet die Fähigkeit unterschiedlicher Systeme nahtlos zusammenzuarbeiten und Daten oder Dienste effizient auszutauschen – trotz technischer oder struktureller Unterschiede.

[41] Ein Ubiquitous Sensor Network ist ein Netzwerk verteilter, miteinander verbundener Sensoren, das kontinuierlich Daten aus der Umgebung erfasst und nahezu überall verfügbar ist.

ten Architekturen besteht, um Interoperabilität und Wiederverwendbarkeit zu gewährleisten.[42]

Für das eigene Portal bedeutet dies, auf etablierte Web-Standards und offene Schnittstellen zu setzen, sodass beispielsweise Daten des DWD nahtlos integriert oder von externen Anwendungen weitergenutzt werden können.

### 3.2.5 Visualisierung von Klima- und Wetterdaten

Die Visualisierung spielt eine entscheidende Rolle, um Klimadaten für die Bevölkerung verständlich und zugänglich zu machen. Viele Menschen können abstrakte Zahlen oder Tabellen kaum einordnen, wohingegen anschauliche Grafiken und Karten komplexe Sachverhalte schnell vermitteln. Die Literatur zur Klimakommunikation hebt deshalb die Bedeutung von durchdachten Visualisierungsstrategien hervor. So gehört zu den zehn Prinzipien von Climate Outreach explizit die Empfehlung, „mächtige Geschichten zu erzählen und effektive Bilder zu nutzen“. Einprägsame visuelle Darstellungen, seien es Infografiken, Diagramme oder interaktive Karten, können Emotionen ansprechen und Muster sichtbar machen, die sonst verborgen bleiben.[43]

Solche Visualisierungen haben in der Öffentlichkeit starke Resonanz gefunden, weil sie wissenschaftliche Daten in ein leicht verständliches, intuitives Format übersetzen. Für das geplante Portal bedeutet dies, dass verschiedene Darstellungsmöglichkeiten geprüft und umgesetzt werden sollten: Zeitreihen-Grafiken könnten historische Trends (z. B. steigende Jahresmitteltemperaturen) zeigen, Karten könnten regionale Unterschiede oder Veränderungen (etwa bei Niederschlägen) illustrieren, und Diagramme könnten Zusammenhänge (z. B. Treibhausgasemissionen und Temperaturwirkung) verdeutlichen. Wichtig ist dabei eine klare Gestaltung, denn die Grafiken sollten Kernaussagen hervorheben und mit Legenden sowie Erläuterungen versehen sein, damit auch Nutzer ohne Vorwissen sie korrekt interpretieren.

## 3.3 Relevante Standards

Bei der Konzeption und Umsetzung des Klima- und Wetterinformationsportal ist die Orientierung an etablierten Standards entscheidend für die Weiternutzbarkeit und Nachhaltigkeit der entwickelten Lösung. Insbesondere im Kontext von Open Data spielen die OGC API Standards (Kapitel 3.3.1) sowie die sogenannten FAIR Data Principles, welche in Kapitel 3.3.2 näher erläutert werden, eine wichtige Rolle.

[42] vgl. Carreras-Coch et al. (2022, online)

[43] vgl. Climate Outreach (2022, online)

### 3.3.1 OGC API – Open Geospatial Consortium

Das Open Geospatial Consortium (OGC) ist eine internationale Organisation zur Standardisierung geodatenbezogener Schnittstellen. Die OGC API-Standards definieren moderne, REST-basierte Schnittstellen zur Bereitstellung von Geodaten und Umweltdaten im Web. Sie sind modular aufgebaut und ermöglichen eine ressourcenorientierte, leicht nutzbare Bereitstellung von Daten, etwa im JSON- oder GeoJSON[44]-Format.[45]

Die wichtigsten Eigenschaften der OGC API sind:

- REST-Schnittstellen: Moderne URL-Struktur, die leicht integrierbar ist in Webanwendungen,
- Geodatenstandardisierung: Punkte und Polygone,
- Kompatibilität mit Open Data und Open-Source,
- Modularität: Erweiterbar um Funktionen wie Filterung.

Obwohl der DWD derzeit keine API bereitstellt, ist die Ausrichtung der eigenen REST-Schnittstelle an diesen Standards langfristig sinnvoll.[46]

### 3.3.2 FAIR Data Principles

Die FAIR-Prinzipien wurden 2016 formuliert, um sicherzustellen, dass Forschungsdaten den Anforderungen moderner Wissenschaftspraxis entsprechen. Die vier Prinzipien lauten:[47]

- Findable – Auffindbar: Daten müssen eindeutig identifizierbar und mit Metadaten versehen sein,
- Accessible – Zugänglich: Daten müssen über standardisierte Protokolle verfügbar sein,
- Interoperable – Kompatibel: Daten müssen maschinenlesbar und mit Datenquellen kombinierbar sein,
- Reusable – Wiederverwendbar: Daten sollen klar lizensiert und wissenschaftlich dokumentiert sein.

Die Open-Data-Angebote des DWD orientieren sich bereits weitgehend an diesen Prinzipien, insbesondere hinsichtlich Zugänglichkeit, Standardisierung und Langfristarchivierung.

[44] GeoJSON ist ein textbasiertes Format zur Darstellung geografischer Daten wie Punkte, Linien oder Flächen, basierend auf JSON und häufig verwendet in Webkarten.

[45] vgl. Open Geospatial Consortium (2022, online)

[46] vgl. Deutscher Wetterdienst (Jahr unbekannt, online) (3)

[47] vgl. GO FAIR International Support and Coordination Office (Jahr unbekannt, online)

### 3.3.3 Bezug auf diese Arbeit

Die Orientierung an OGC- und FAIR-Prinzipien ist insbesondere für die langfristige Erweiterbarkeit des Portals entscheidend. Auch wenn im Rahmen dieser Bachelorarbeit keine vollständige OGC-konforme API umgesetzt wird, wird bei der Konzeption der REST-Endpunkte auf strukturelle Ähnlichkeit (z. B. Filterlogik, JSON-Ausgabe) geachtet. Zudem wird bei der Datenhaltung auf maschinenlesbare Formate, zeitliche Eindeutigkeit und klare Metadaten Wert gelegt. Dadurch wird sichergestellt, dass das entwickelte Portal nicht nur technisch funktional ist, sondern auch zukünftige Anforderungen an Open Science[48] und Datenpublikation erfüllen kann.

## 3.4 Lücken und Verbesserungspotenzial

Die Analyse bestehender Portale und die Betrachtung relevanter Standards zeigen, dass es trotz zahlreicher Angebote zur Darstellung meteorologischer Daten nach wie vor funktionale und nutzerseitige Lücken gibt. Diese Lücken bieten ein klares Potenzial für die Entwicklung eines alternativen, benutzerorientierten Informationsportals.

### 3.4.1 Fehlende Kombination: Datenqualität und Usability

Während Plattformen wie das DWD Climate Data Center qualitativ hochwertige, langfristige Wetterdaten zur Verfügung stellen, fehlt es an benutzerfreundlichen, interaktiven Weboberflächen, die diese Daten allgemeinverständlich visualisieren. Technische Vorkenntnisse, wie sie zum Verständnis und zur Nutzung von CSV-Dateien oder Datenordnerstrukturen erforderlich sind, stellen für viele Nutzer eine Barriere dar.

### 3.4.2 Geringe Analysefunktionen

Viele bestehende Portale bieten keine oder nur sehr eingeschränkte Filter-, Export- oder Analysefunktionen. Nutzer können oft nur statische Diagramme einsehen oder punktuelle Werte abrufen.

### 3.4.3 Fehlende offene API

Nur wenige Portale stellen offene APIs bereit, über die externen Anwendungen auf die Daten zugreifen könnten. Damit bleiben die Daten schwer integrierbar in andere Projekte, Analysen oder Bildungskontexte.

[48] Open Science bezeichnet eine wissenschaftliche Praxis, bei der Forschungsergebnisse, Daten, Methoden und Publikationen für alle offen zugänglich, nachvollziehbar und wiederverwendbar gemacht werden.

### 3.4.4 Fazit und Zielableitung

Die identifizierten Lücken machen deutlich, dass eine Kombination aus:

- offenen, qualitätsgeprüften Daten (z. B. vom DWD),
- visueller, interaktiver und verständlicher Darstellung und,
- technischer Offenheit (API, CSV-Export) nicht ausreichend umgesetzt ist.

Genau hier setzt die vorliegende Arbeit an. Ziel ist die Entwicklung eines benutzerfreundlichen, modularen Portals, das diese Lücken schließt, Nutzer einen verständlichen Zugang zu Klimadaten ermöglicht und dabei moderne Webstandards berücksichtigt.

# 4 Konzeption des Portals

Dieses Kapitel widmet sich der Zielgruppenanalyse, den funktionalen sowie nicht-funktionalen Anforderungen, der Systemarchitektur und der Datenmodellierung.

## 4.1 Zielgruppenanalyse

Die Entwicklung eines Klima- und Wetterinformationsportals erfordert eine zielgerichtete Ausrichtung auf die Bedürfnisse der potenziellen Nutzergruppen. Ziel der Zielgruppenanalyse ist es, Anforderungen und Erwartungen systematisch zu erfassen und daraus Gestaltungsprinzipien für die technische und visuelle Umsetzung abzuleiten.

Folgende Zielgruppen werden ausgewählt: Bürger ohne Fachwissen, Lehrkräfte und Bildungseinrichtungen sowie technikaffine Nutzer. Diese Nutzergruppen werden im Rahmen dieser Arbeit als die am geeignetsten Zielgruppen betrachtet.

### 4.1.1 Bürger ohne Fachwissen

Diese Zielgruppe umfasst Personen aus der allgemeinen Bevölkerung, die sich für Wetter, Klima und Umweltfolgen interessieren, jedoch keine fachlichen oder technischen Vorkenntnisse mitbringen. Laut den Statistiken, die in Kapitel 1.2 untersucht werden, fühlen sich viele Bürger zwar vom Thema Klimawandel betroffen, empfinden jedoch Unsicherheit beim Zugang zu verlässlichen und verständlich aufbereiteten Informationen.

Die möglichen Anforderungen dieser Gruppe an das Portal sind:

- Leichter Zugang ohne Vorkenntnisse über das Portal,
- Verlässliche Datenquellen,
- Einfache und visuelle gestützte Darstellung der Klimadaten,
- Lokaler Bezug (z.B. eigene Stadt oder Region).

### 4.1.2 Lehrkräfte und Bildungseinrichtungen

Im schulischen oder anderen Bildungsbereich besteht ein wachsender Bedarf an konkret aufbereitetem Klima- und Umweltwissen, das methodisch nutzbar ist. Bildungseinrichtungen suchen häufig nach aktuellen und historischen Datengrundlagen, um das Thema Klimawandel greifbar zu vermitteln.

Zusätzlich zu den Anforderungen aus der Zielgruppe „Bürger ohne Fachwissen“ müssen noch folgende Voraussetzungen für das Portal erfüllt werden:

- Zitierfähige Datenquelle,
- Exportierfunktion (z.B. CSV für Analyse).

### 4.1.3 Technikaffine Nutzer

Diese Gruppe umfasst Entwickler oder wissenschaftlich interessierte Personen, die mit strukturierten Daten arbeiten. Ihr Interesse liegt vor allem an der programmatischen Nutzung, Analyse oder Integration der Daten in eigene Projekte.

Die möglichen Anforderungen dieser Gruppe wären:

- Anforderungen aus den Zielgruppen „Bürger ohne Fachwissen“ und „Lehrkräfte und Bildungseinrichtungen“,
- Maschinenlesbarer Datenzugang über REST-API.

### 4.1.4 Relevanz für die Konzeption und Umsetzung

Die Zielgruppenanalyse zeigt, dass das geplante Portal eine verschiedenartige Nutzerbasis anspricht. Um diesen unterschiedlichen Ansprüchen gerecht zu werden, wird in der Umsetzung besonderes Augenmerk auf folgende Kriterien gelegt:

- Leichte verständliche Benutzeroberfläche,
- Klare Visualisierung,
- Experten-Funktionen wie CSV-Export und REST-API.

Die Konzeption folgt damit dem Prinzip der Mehrschichtigkeit: einfache Nutzung für die breite Öffentlichkeit, erweiterte Funktionen für spezifische Nutzergruppen.

## 4.2 Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben, was das Portal technisch leisten soll, also die konkret umzusetzenden Funktionen aus Nutzersicht. Sie ergeben sich aus der Zielsetzung des Projekts (Kapitel 1.4), der Zielgruppenanalyse (Kapitel 4.1) sowie aus der Analyse bestehender Lösungen (Kapitel 3.1). Im Folgenden werden die funktionalen Anforderungen in Gruppen gegliedert.

### 4.2.1 Datenzugriff und Filterung

Nutzer sollen Wetter- und Klimadaten nach verschiedenen Kriterien filtern können, darunter:

- Zeitraum,
- Wetterstationen, Stadt oder Region,
- Messgrößen (Temperatur, Niederschlag, Sonnenscheindauer, etc.).

Die Filterung soll sowohl im Frontend als auch über URL-Parameter (REST-API) möglich sein.

### 4.2.2 Datenvisualisierung

Die gefilterten Daten sollen in grafischer Form dargestellt werden, unter anderem:

- Diagramme mit Verläufen (z.B. Temperaturverläufe),
- Kartenansicht zur Anzeige der Wetterstation.

Die Visualisierungen der Klimadaten reagieren dynamisch, beispielsweise beim Wechsel zwischen verschiedenen Stationen.

### 4.2.3 Lokalisierter Datenbezug

Nutzer sollen sich gezielt Daten für ihren Standort oder eine Stadt anzeigen lassen. Bei Auswahl einer Station durch Mausklick auf der Landkarte wird automatisch die nächstgelegene Wetterstation relativ zur Klickposition ermittelt.

### 4.2.4 Datenexport

Export der gefilterten Daten als CSV-Datei (z. B. für eigene Analysen). Optionale Ausgabe als JSON für technisch affine Nutzer.

### 4.2.5 Automatisierter Datenimport

Das Backendsystem vom Portal wird bei Start nach aktuellen Daten vom DWD suchen und diese bei Bedarf abrufen. Dabei sollen doppelte Datensätze erkannt und durch aktuellere ersetzt werden.

### 4.2.6 Programmierschnittstelle (REST-API)

Bereitstellung einer dokumentierten REST-API mit folgenden Endpunkten:

- Abruf gefilterter Wetterdaten,
- Abruf aggregierter Werte wie Durchschnittstemperatur pro Monat.

### 4.2.7 Benutzererlebnis und Interaktion

Nutzer sollen über ein intuitives, klickbasiertes Interface mit dem Portal interagieren können. Es sollen Hilfstexte und Legenden eingebunden werden, um Messgrößen und Diagramme zu erklären.

## 4.3 Nicht-Funktionale Anforderungen

Nicht-funktionale Anforderungen beschreiben die Qualitätsmerkmale des geplanten Portals sowie technische, rechtliche und organisatorische Rahmenbedingungen, die im Entwicklungsprozess zu berücksichtigen sind. Sie beeinflussen maßgeblich die Nutzerakzeptanz, Wartbarkeit und Nachhaltigkeit der Anwendung, auch wenn sie nicht direkt als sichtbare Funktionen in Erscheinung treten. Im Folgenden werden die zentralen nicht-funktionalen Anforderungen für das Projekt systematisch dargestellt.

### 4.3.1 Benutzerfreundlichkeit

Die Anwendung wird eine intuitive, leicht verständliche Benutzeroberfläche bieten. Nutzer ohne technisches Vorwissen sollen problemlos Daten finden, filtern und interpretieren können. Die Visualisierung ist klar strukturiert, selbsterklärend und responsiv gestaltet.

### 4.3.2 Performance und Skalierbarkeit

Das Portal muss auch bei größeren Datenmengen (z. B. vollständige Jahresdaten) schnell reagieren. Filter- und Exportoperationen müssen auch bei mehreren gleichzeitigen Zugriffen performant bleiben. Die Konzeption und Umsetzung sind so gestaltet, dass eine zukünftige Skalierung mit minimalem Aufwand möglich ist.

### 4.3.3 Wartbarkeit und Erweiterbarkeit

Der Quellcode ist modular, dokumentiert und gut strukturiert, um spätere Erweiterungen (z. B. weitere Filter, neue Diagrammtypen) zu erleichtern. Verwendete Technologien sind weit verbreitet, offen dokumentiert und langfristig unterstützbar.

### 4.3.4 Kompatibilität und Zugänglichkeit

Die Anwendung wird auf aktuellen Desktop- und Mobilbrowsern (Chrome, Firefox, Safari, Edge) funktionieren. Die grafische Darstellung soll sich, wenn möglich, responsiv an verschiedene Bildschirmgrößen anpassen.

### 4.3.5 Offenheit und Lizenzierung

Die Software soll auf Basis von Open-Source-Komponenten entwickelt werden. Die verwendeten Daten (vom DWD) müssen unter der offenen Creative Commons BY 4.0 Lizenz korrekt referenziert werden[49].

## 4.4 Systemarchitektur

Die Systemarchitektur des entwickelten Klima- und Wetterinformationsportals orientiert sich an dem Prinzip einer modularen, mehrschichtigen Webanwendung. Ziel ist eine klare Trennung von Verantwortlichkeiten, hohe Wartbarkeit und die Möglichkeit zur einfachen Erweiterung. Die Architektur umfasst drei Hauptkomponenten: Frontend, Backend und Datenhaltung. Zusätzlich ist im Backend ein automatisierter Importdienst für Open-Data-Quellen vom DWD integriert. Die Architektur ist als Client-Server-Modell mit REST-basierter Kommunikation realisiert.

### 4.4.1 Komponentenübersicht

Die Anwendung besteht aus folgenden Systemteilen (Abbildung 12):

- Frontend (Client): Webanwendung für die Interaktion mit dem Nutzer. Verantwortlich für UI, Visualisierungen und Benutzeraktionen,
- Backend (Server): Zuständig für Datenverarbeitung, API-Endpunkte, Importlogik und Geschäftslogik,
- Datenbank: Relationale Datenbank zur Speicherung historischer und aktueller Wetterdaten,
- Externe Datenquelle (DWD): Open-Data-Server des DWD, insbesondere die Verzeichnisse des Climate Data Center.

[49] vgl. Deutscher Wetterdienst (Jahr unbekannt, online) (2)

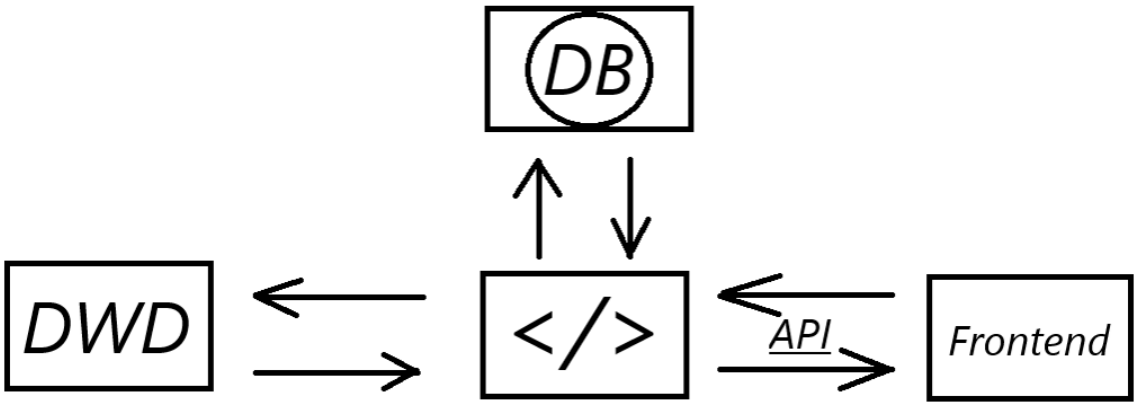



Abbildung 12: Architekturskizze (Aslan, 2025)

### 4.4.2 Technologieauswahl

Die Technologieauswahl für die Backend wird auf die Programmiersprache Java mit dem Spring Boot Framework gelegt. Das Spring Boot ist ein Framework, das die Entwicklung eigenständiger, produktionsreifer Webanwendungen vereinfacht, indem es Konfigurationen automatisiert. Das Spring Boot basiert auf dem Spring-Ökosystem[50]. Folgende Kriterien waren für die Auswahl des Spring Boot Frameworks entscheidend:

- Vertrautheit mit dem Java-Ökosystem,
- Weit verbreitet und etabliert,
- Hohe Stabilität und Wartbarkeit,
- Eignung für datenlastige Anwendungen,
- Integrierte REST-API Komponente.

Die Frontend wird mit dem React.js Framework umgesetzt. React.js ist eine JavaScript-Bibliothek zur Entwicklung dynamischer Benutzeroberflächen, die eine komponentenbasierte Architektur und effiziente Aktualisierung durch den virtuellen DOM[51] nutzt. Die Gründe für die Auswahl basiert auf Folgendem:

- Komponentenbasiertes Design,
- Hohe Performance durch virtuellen DOM,
- Große Community und Ökosystem.

[50] Das Spring-Ökosystem umfasst eine Sammlung modularer Java-Frameworks rund um das Kernframework Spring, die die Entwicklung moderner, skalierbarer und wartbarer Anwendungen erleichtern, z. B. durch Module wie Spring Boot.

[51] Virtuelle DOM ist eine leichtgewichtige Kopie des echten DOM (die strukturierte, hierarchische Darstellung eines HTML- oder XML-Dokuments), die React nutzt, um Änderungen effizient zu berechnen und nur die tatsächlich geänderten Teile im Browser neu zu rendern.

Dabei wird die Frontend mit Node.js als Server zum Laufen gebracht. Node.js ist eine serverseitige Laufzeitumgebung für JavaScript. Sie ermöglicht es, JavaScript nicht nur im Browser, sondern auch auf dem Server oder zur Ausführung von Entwicklungswerkzeugen (wie z. B. das React.js Framework) zu verwenden.

Zum Einsatz der Datenbank wird PostgreSQL als relationale Datenbank eingesetzt und ergab sich aus einer Kombination von Leistungsfähigkeit und Offenheit. Aufgrund der großen Datenmenge der meteorologischen Daten, ist die Performance der Datenbank essenziell für das Nutzererlebnis, da eine nicht ausreichend performante Webseite Nutzer abschrecken könnte.

### 4.4.3 Datenflussbeschreibung

Der Datenfluss innerhalb des Systems folgt typischerweise diesen Schritten:

1. Der Datenimportdienst lädt automatisiert neue Wetterdaten vom DWD herunter.
2. Die Daten werden im Backend verarbeitet, geprüft (z. B. auf Duplikate oder Korrekturen) und in der PostgreSQL-Datenbank gespeichert.
3. Nutzer greifen über das Frontend auf das Portal zu, wählen Filteroptionen und erhalten Daten oder Visualisierungen.
4. Das Frontend sendet entsprechende Anfragen an die REST-API des Backends.
5. Die Antwortdaten werden im Frontend in Tabellen, Diagrammen oder Kartenansichten visualisiert.
6. Optional können Daten als CSV exportiert werden.

### 4.4.4 Kommunikation und Schnittstellen

Die Kommunikation zwischen Frontend und Backend erfolgt über eine REST-API im JSON-Format. Das Backend kommuniziert mit der Datenbank über eine objektrelationale Abbildungsschicht[52], die durch das Framework Hibernate[53] realisiert wird. Dadurch können Datenbankzugriffe deklarativ über Java-Objekte erfolgen, ohne SQL direkt schreiben zu müssen. Der Datenimportdienst nutzt standardisierte HTTP-Abfragen und Dateiparsing für den DWD-Datenzugriff.

[52] Die objektrelationale Abbildungsschicht ist eine Programmierschnittstelle, die Objekte im Code automatisch mit relationalen Datenbanktabellen verknüpft, so kann auf Datenbankinhalte zugegriffen werden, ohne SQL direkt zu schreiben.

[53] Hibernate ist ein Java-Framework zur objektrelationalen Abbildung, dass die automatische Verbindung zwischen Java-Klassen und relationalen Datenbanktabellen herstellt und so den Datenzugriff vereinfacht.

## 4.5 Datenmodellierung

Die Datenmodellierung bildet das Datenfundament des Klima- und Wetterinformationsportals. Ziel ist es, meteorologische Daten des DWD in einer logisch konsistenten, normalisierten und performant abfragbaren Struktur innerhalb der relationalen PostgreSQL-Datenbank zu speichern. Im Zentrum steht die Speicherung von Wettermesswerten in Bezug zu einer bestimmten Wetterstation und einem Messzeitpunkt.

### 4.5.1 Datenquelle und Struktur

Die zugrundeliegenden Datensätze stammen vom Climate Data Center (CDC) des DWD. Jeder Datensatz ist eindeutig durch die Kombination von Stations-ID und Datum identifizierbar, jedoch enthalten sie keine einzigartige ID. Die Datensätze enthalten unter anderem tägliche Werte folgender Messgrößen: Temperatur (Durchschnitt, Maximum, Minimum), Niederschlag (Höhe, Form), Sonnenscheindauer, Windgeschwindigkeit, etc.

### 4.5.2 Tabellenübersicht und Beziehungen

Zur strukturierten Speicherung wird ein relationales Modell mit zwei Haupttabellen entwickelt. Alle Felder erlauben NULL-Werte, falls Messdaten entweder Fehler oder ungültig sind (z.B. -999 im Originaldatensatz):

#### 4.5.2.1 Tabelle weather_station

Die Tabelle weather_station besteht aus vier Parametern. Dabei ist die Station ID der Primärschlüssel (Primary Key)[54] der Tabelle und wird als Fremdschlüssel (Foreign Key)[55] in der weather_data Tabelle wiederverwendet.

| Spalte | Datentyp | Beschreibung |
|---|---|---|
| station_id (Primary key) | INT | Stations ID laut DWD |
| name | VARCHAR | Bezeichnung der Station (z.B. Berlin) |
| lat | DOUBLE | Breitengrad der Station |
| lon | DOUBLE | Längengrad der Station |

Tabelle 1: Datenbanktabellen Aufbau weather_station (Aslan, 2025)

[54] Ein Primärschlüssel ist ein Attribut oder eine Attributkombination in einer Datenbanktabelle, das jeden Datensatz eindeutig identifiziert und keine Nullwerte zulässt.

[55] Ein Fremdschlüssel ist ein Attribut in einer Datenbanktabelle, das auf den Primärschlüssel einer anderen Tabelle verweist und so eine referenzielle Beziehung zwischen den beiden Tabellen herstellt.

Die Informationen über die einzelnen Wetterstationen sind aufrufbar unter https://opendata.dwd.de/climate_environment/CDC/observations_germany/climate/daily/kl/historical/KL_Tageswerte_Beschreibung_Stationen.txt.

#### 4.5.2.2 Tabelle weather_data

Die Haupttabelle weather_data beinhaltet 13 verschiedene meteorologischen Parameter. Zusätzliche Parameter wie „qualityi“ oder „qualityii“ beschreiben das Qualitätsniveau anhand verschiedenen Werten.

| Spalte | Datentyp | Beschreibung |
|---|---|---|
| id (Primary key) | BIGINT | Automatisch vergebene ID |
| stations_id (Foreign key) | INT | Stations ID laut DWD |
| date | DATE | Messdatum (YYYYMMDD) |
| qualityi | DOUBLE | Qualitätsniveau Spalten |
| wind_max | DOUBLE | Tagesmaximum Windgeschwindigkeit (m/s) |
| wind_speed | DOUBLE | Tagesmittel Windgeschwindigkeit (m/s) |
| qualityii | DOUBLE | Qualitätsniveau Spalten |
| precipitation | DOUBLE | Tägliche Niederschlagshöhe (mm) |
| prec_form | DOUBLE | Niederschlagsform |
| snow | DOUBLE | Tageswert Schneehöhe (cm) |
| covering | DOUBLE | Tagesmittel des Bedeckungsgrades (1/8) |
| steam_pressure | DOUBLE | Tagesmittel Dampfdruck (hPa) |
| air_pressure | DOUBLE | Tagesmittel Luftdruck (hPa) |
| temperature | DOUBLE | Tagesmittel Lufttemperatur in 2m Höhe (°C) |
| humidity | DOUBLE | Tagesmittel Relativen Feuchte (%) |
| temp_max | DOUBLE | Tagesmaximum Lufttemperatur in 2m Höhe (°C) |
| temp_min | DOUBLE | Tagesminimum Lufttemperatur in 2m Höhe (°C) |
| temp_min5cm | DOUBLE | Minimum Lufttemperatur am Erdboden in 5 cm Höhe (°C) |

Tabelle 2: Datenbanktabellen Aufbau weather_data (Aslan, 2025)

### 4.5.3 Datenintegrität und Normalisierung

Die Datenbankstruktur ist in der 3. Normalform[56] gehalten, um Redundanz zu vermeiden. Die Wetterdaten enthalten nur Referenzen auf die zugehörige Station, keine mehrfach gespeicherten Ortsnamen oder Koordinaten. Die Kombination aus station_id und date ist in weather_data eindeutig und wird zusätzlich durch einen Unique Constraint[57] abgesichert.

### 4.5.4 Zusammenfassung

Die gewählte Datenmodellierung erlaubt eine skalierbare Speicherung von Zeitreihendaten und unterstützt sowohl einfache Nutzungsfälle (z. B. grafische Darstellung) als auch komplexe Datenabfragen (z. B. statistische Vergleiche). Gleichzeitig sichert die Normalisierung die Konsistenz und Erweiterbarkeit der Anwendung, etwa für weitere Klimagrößen oder zusätzliche Stationstypen.

[56] Eine Normalform bezeichnet in der Datenbanktheorie einen strukturellen Zustand relationaler Tabellen, der bestimmte Regeln erfüllt, um Redundanzen zu vermeiden und die Datenintegrität zu sichern. Die 3. Normalform ist eine Stufe der Datenbanknormalisierung, bei der alle Nicht-Schlüsselattribute nur vom Primärschlüssel abhängen und keine transitiven Abhängigkeiten zwischen Nicht-Schlüsselattributen bestehen.

[57] Ein Unique Constraint ist eine Datenbankregel, die sicherstellt, dass ein bestimmter Wert in einer Spalte (oder Kombination von Spalten) nur einmal vorkommen darf, zur Vermeidung von Duplikaten.

# 5 Umsetzung des Portals

In Kapitel 5 „Umsetzung des Portals“ wird die Backend- und Frontend-Entwicklung sowie der Datenimport und die Datenverarbeitung behandelt.

## 5.1 Backend-Entwicklung

Das Backend des Klima- und Wetterinformationsportals wird mit dem Spring Boot Framework in der Programmiersprache Java umgesetzt. Ziel war es, eine robuste, modular erweiterbare und performante Schnittstelle zu entwickeln, die sowohl den automatisierten Import meteorologischer Daten als auch deren Filterung und Bereitstellung über eine REST-API ermöglicht.

### 5.1.1 Entwicklungsumgebung

Die Entwicklung der Backend wird mit der sogenannten „IntelliJ IDEA (Ultimate Edition)“ (Version: 2024.3.5) Entwicklungsumgebung von JetBrains umgesetzt.[58] Aufgrund der integrierten Spring Boot Unterstützung (inklusiver Konfigurationserkennung), der schnellen und hilfreichen Fehlererkennung, den verfügbaren Werkzeugen für Datenbanken und dem kostenlosen Zugang für Studenten war „IntelliJ IDEA (Ultimate Edition)“ die bevorzugte Entwicklungsumgebung für dieses Portal.

### 5.1.2 Architektur und Komponentenstruktur

Das Backend des Wetterinformationsportals basiert auf dem Java Spring Boot Framework und ist modular aufgebaut. Auf Grund der Lesbarkeit wird kein Quellcode zur Beschreibung dargestellt. In Anhang A befindet sich die Dokumentation für den Quellcode. Das Backend besteht aus mehreren Komponenten, die jeweils klar abgegrenzte Verantwortlichkeiten besitzen:

- *WetterPortalApplication.java*: Diese Klasse ist der Einstiegspunkt der gesamten Anwendung. Sie initialisiert die Spring Boot Applikation und führt beim Start automatisiert die Importlogik aus, indem sie den *WetterDWDImporter* mit dem CommandLineRunner aufruft.
- *WetterDWDImporter.java*: Diese zentrale Komponente ist für das Abrufen, Entpacken, Verarbeiten und Speichern der Wetterdaten vom DWD verantwortlich.

[58] https://www.jetbrains.com/idea/

Sie verarbeitet sowohl historische als auch aktuelle („recent“) ZIP-Dateien[59] vom Open-Data-Portal des DWD. Die Importlogik prüft bereits importierte Dateien mittels *ImportedFileRepository* und aktualisiert bestehende Einträge nur bei inhaltlichen Änderungen.

- *WetterDbEntity.java*: Diese Klasse stellt die Hauptdatenstruktur für Wetterdaten dar und ist mit der Datenbanktabelle weather_data verknüpft. Sie enthält Felder wie stationId, temperature, precipitation, sunshine usw. Jeder Eintrag ist zudem über einen Fremdschlüssel mit einer Wetterstation (*WeatherStation*) verbunden.
- *WeatherStation.java*: Repräsentiert die Metadaten der Wetterstationen mittels Name und geografischen Koordinaten. Diese Entität ist mit der Tabelle weather_station verknüpft und bildet die Beziehung zur *WetterDbEntity.*
- *WetterDbRep.java*: Dieses Repository[60]-Interface erbt von JpaRepository und JpaSpecificationExecutor. Es ermöglicht standardisierte Datenbankzugriffe und stellt benutzerdefinierte Query-Methoden zur Verfügung. Es wird u.a. für die Filterung von Wetterdaten und Aggregationen (z. B. Durchschnittswerte) verwendet.
- *WetterDbSpecs.java*: Diese Klasse enthält spezialisierte JPA Specifications[61], mit denen komplexe Filterkriterien dynamisch zusammengesetzt werden können (z. B. min-max-Temperatur, Datumsbereich, Kombination mehrerer Parameter)
- *ImportedFile.java*: Eine einfache Entität, die alle bereits verarbeiteten Dateinamen speichert. Dadurch wird verhindert, dass dieselbe Datei mehrmals importiert wird.
- *ImportedFileRepository.java*: Repository für die *ImportedFile*-Entität. Es dient ausschließlich zur Überprüfung, ob eine bestimmte Datei bereits importiert wird.
- *WetterDataController.java*: REST-Controller zur Bereitstellung der Wetterdaten über die API. Bietet verschiedene Endpunkte zum Filtern von Wetterdaten, zum Abrufen aggregierter Werte sowie zum CSV-Export an.
- *CorsConfig.java*: Konfiguriert die CORS-Einstellungen, sodass auch Frontends, die auf anderen Domains oder Ports laufen, auf die API zugreifen können.

[59] ZIP ist ein weit verbreitetes Archivformat zur verlustfreien Komprimierung und Bündelung mehrerer Dateien in einer einzigen Datei, um Speicherplatz zu sparen und den Datentransfer zu erleichtern.

[60] Ein Repository ist in der Softwareentwicklung eine Abstraktionsschicht, die den Zugriff auf Datenquellen kapselt und typischerweise CRUD-Operationen (Create, Read, Update, Delete) bereitstellt.

[61] JPA Specifications sind eine Technik in Java, mit der man sehr gezielt nach Daten in einer Datenbank suchen kann, dabei stehen einzelne Suchregeln wie Bausteine zusammen, ohne komplizierte Datenbankabfragen schreiben zu müssen.

### 5.1.3 Import und Verarbeitung von Wetterdaten

Die Klasse *WetterDWDImporter* ist verantwortlich für den Abruf und die Verarbeitung von Wetterdaten vom Open Data-Server des DWD. Dabei werden:

- ZIP-Dateien automatisch heruntergeladen und entpackt,
- Die enthaltene CSV-Datei geparst und,
- Die Dateien in eine relationale PostgreSQL-Datenbank geschrieben.

Zur Vermeidung von Duplikaten enthält das System eine Prüflogik importZipFile():

Für jede Kombination aus stationId und date wird geprüft, ob bereits ein Eintrag existiert. Falls ja, wird dieser nur überschrieben, wenn sich der neue Datensatz unterscheidet. Dadurch können auch nachträgliche Korrekturen des DWD berücksichtigt werden. Zur Kontrolle, welche Dateien bereits verarbeitet wurden, wird die Klasse *ImportedFile* verwendet. Sie speichert den Dateinamen jeder importierten Datei in einer eigenen Tabelle und verhindert damit unnötige Wiederholungen des Imports.

### 5.1.4 Datenpersistenz und -struktur

Die persistierten Wetterdaten werden über die JPA-Entity[62] *WetterDbEntity* in der Datenbank gespeichert. Diese Entity enthält Felder für jeden Parameter wie z.B.:

- stationID, date,
- Temperatur, Niederschlag, Sonnenscheindauer, Windgeschwindigkeit, etc.,
- Fremdschlüsselreferenz auf die „weather_station“-Tabelle.

Die Entitäten werden über das Repository *WetterDbRep* verwaltet. Es erweitert JpaRepository und JpaSpecificationExecutor, womit komplexe und dynamisch filterbare Datenbankabfragen möglich sind.

### 5.1.5 Dynamisches Filtern von Wetterdaten

Für die REST-API wird die Klasse *WetterDataController* genutzt. Über den Endpunkt /api/weather/filter können Nutzer mit Query-Parametern wie stationId, startDate, temperatureMin, temperatureMax usw. ihre Suche verfeinern.

Die Filterung wird durch die Klasse *WetterDbSpecs* ermöglicht. Sie stellt dynamische JPA-Spezifikationen bereit, die zur Laufzeit auf die Datenbankabfragen angewendet werden, ohne dass SQL manuell geschrieben werden muss.

[62] JPA-Entity ist eine Java-Klasse das mit einer Datenbanktabelle verbunden ist. Jeder Datensatz in der Tabelle entspricht dabei genau einem Objekt im Programm, so kann man Daten speichern und abfragen.

Beispiel für eine GET-Anfrage: GET /api/weather/filter?startDate=2024-01-01&endDate=2024-12-31&city=Stuttgart

- Antwort gibt nur Datensätze zurück die zwischen dem Zeitraum 01.01.2024 und 31.12.2024 für die Stationsnamen „Stuttgart“ gemessen wird. In diesem Fall sind mehrere Stationen von Stuttgart eingeschlossen wie unter anderem Stuttgart-Schnarrenberg und Stuttgart-Stadt.

### 5.1.6 CSV-Export

Die bereitgestellten Daten werden standardmäßig als JSON ausgeliefert. Alternativ ist es möglich, dieselben Daten auch im CSV-Format zu exportieren, indem das Query „format=csv“ an die Anfrage angehängt wird.

### 5.1.7 CORS – Cross Origin Resource Sharing

Mittels *CorsConfig* wird ein globales CORS-Setup implementiert, das Cross-Origin-Anfragen vom Frontend (z. B. localhost:5173) erlaubt oder blockiert. Dadurch kann das React-Frontend ohne Einschränkungen auf die API zugreifen.

### 5.1.8 UML-Diagramm

Zur Veranschaulichung der Klassen und Abhängigkeit wird ein UML-Diagramm erstellt (Abbildung 13). Aus dem Diagramm ist zu erkennen, wie die Datenbankstruktur sich bei der Klassenstruktur widerspiegelt.

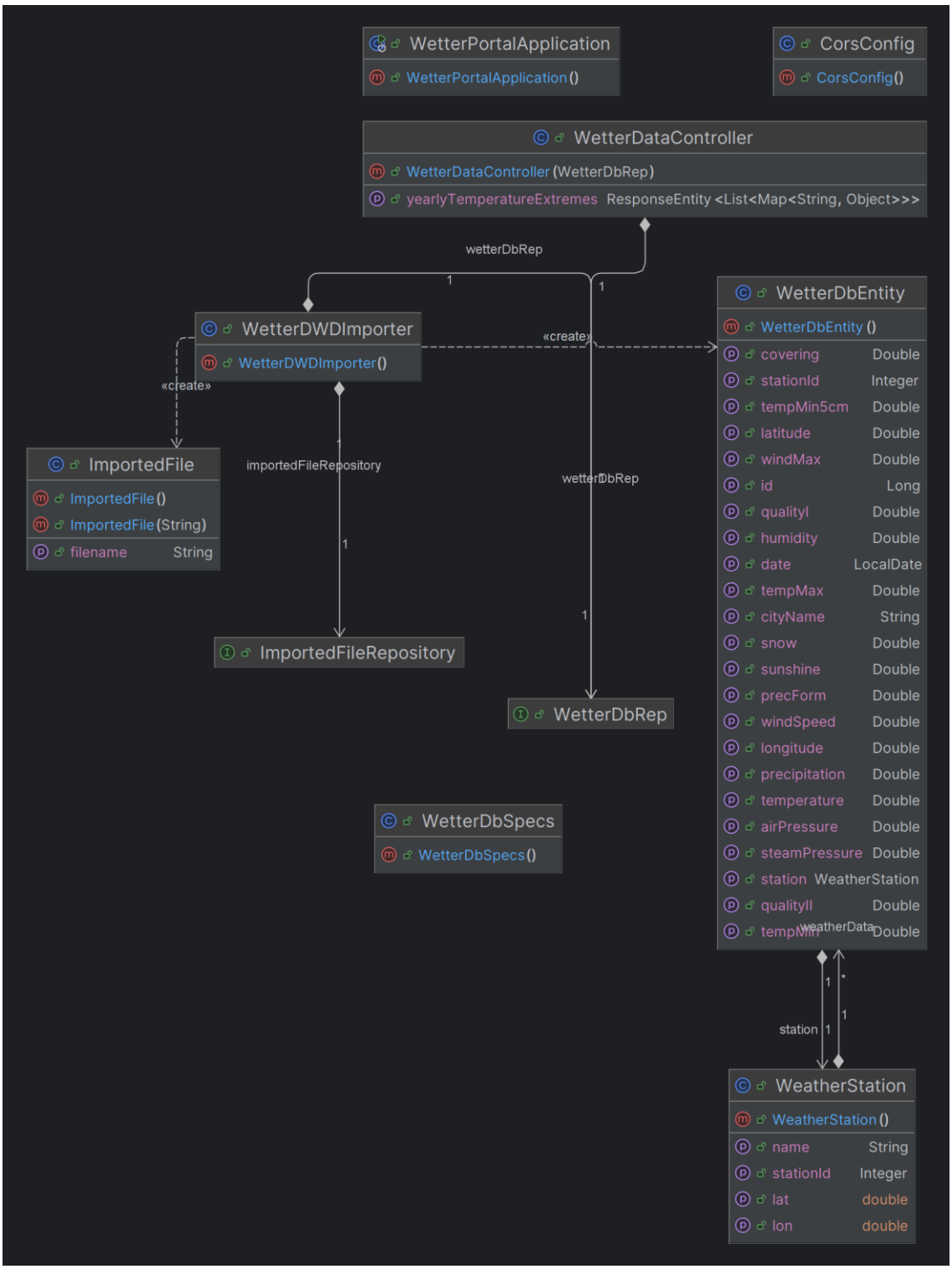



Abbildung 13: UML-Diagramm vom Backend (Aslan, 2025)

## 5.2 Frontend-Entwicklung

Das Frontend des Klima- und Wetterinformationsportals wird vollständig mit React.js entwickelt. Dabei handelt es sich um ein komponentenbasiertes JavaScript-Framework, das durch seinen virtuellen DOM und seine hohe Performance insbesondere für interaktive Webanwendungen geeignet ist, denn das Ziel ist es, meteorologische Daten benutzerfreundlich, interaktiv und visuell zugänglich zu machen.

Als Laufzeitumgebung wird Node.js eingesetzt. Eine Laufzeitumgebung ist eine technische Umgebung, in der ein Programm ausgeführt wird. In diesem Fall wird diese Node.js als Laufzeitumgebung benötigt, damit das React.js Framework regelgerecht funktioniert. Aufgrund der großen Paketbibliothek zum Installieren von verschiedenen Bibliotheken (wie z.B. Leaflet) ist Node.js eine geeignete Wahl für die vorliegende Arbeit. Zusätzlich bietet Node.js eine aktive Community zum Austausch von Problemen sowie Lösungen und ermöglicht die schnelle und einfache Aufsetzung von Projekten.

### 5.2.1 Entwicklungsumgebung

Die Entwicklung der Frontend wird mittels der Entwicklungsumgebung „Visual Studio Code“ (Version: 1.101.1) von Microsoft umgesetzt.[63] Aufgrund der Unterstützung von vielen Skript-Sprachen und ressourceneffizienter Ausführbarkeit ist „Visual Studio Code“ die präferierte Auswahl für Frontend Entwicklung.

### 5.2.2 Architektur und Komponentenstruktur

Die Anwendung ist in modulare Komponenten untergliedert, darunter:

- *App.jsx*: Hauptkomponente, steuert das Dialogverhalten und enthält die Kartenansicht.
- *MapView.jsx*: Zeigt eine Leaflet-Karte mit Wetterstationen, berechnet die nächstgelegene Station per Mausklick des Nutzers.
- *WeatherDialog.jsx*: Ein verschiebbares Fenster mit Wetterinformationen zur ausgewählten Station.
- *WeatherDetails.jsx*: Lädt und visualisiert Wetterdaten in Diagrammform (Recharts[64]).
- *WeatherTable.jsx*: Zeigt tabellarisch alle Messwerte zu einer Station an.
- *ManualQueryForm.jsx*: Eingabemaske zur Filterung von Daten nach Station und Zeitraum.

[63] https://code.visualstudio.com/

[64] Recharts ist eine auf React basierende Bibliothek zur Erstellung von Diagrammen und Visualisierungen, die speziell für die Integration in webbasierte Benutzeroberflächen entwickelt wurde.

Die Komponenten werden im Stil funktionaler React-Komponenten mit Hooks (useState, useEffect)[65] umgesetzt.

### 5.2.3 Datenvisualisierung

Die Wetterdaten einer Station werden in *WeatherDetails.jsx* sowohl tabellarisch (*WeatherTable.jsx*) als auch grafisch dargestellt. Für die Diagramme wird Recharts genutzt. Die enthaltenen Funktionen des Dialoges sind folgende:

- Liniendiagramm zur Darstellung von Temperaturverläufen über ein Jahr,
- Dynamische Gruppierung von Daten nach Jahr,
- Interaktive Tooltips, Achsenbeschriftungen,
- Tabellarische Anzeige der einzelnen Datensätze.

Die nachfolgende Abbildung 14 zeigt anhand des Beispiels der Station Görlitz, wie eine Visualisierung von Wetterdaten über einen Zeitraum von mehr als 150 Jahren aussieht. Für den Anwender sind sowohl der jährliche Temperaturverlauf als auch die einzelnen Datensätze in tabellarischer Form ersichtlich.


Abbildung 14: Langzeit-Datenvisualisierung der Station Görlitz (Aslan, 2025)

### 5.2.4 Benutzerführung und Usability

Das Frontend wird mit dem Fokus konzipiert, dem Nutzer einfache Bedienbarkeit und interaktive Zugänglichkeit zu gewährleisten. Nutzer können über eine „Zoom-In“-

[65] Hooks sind Funktionen in React, mit denen man in Komponenten z. B. Daten speichern (useState) oder auf Ereignisse wie Datenänderung reagieren kann (useEffect).

Funktion ihre erwünschte Region in der Karte anzeigen lassen, Stationen visuell erkennen und erhalten unmittelbar grafische Auswertungen, welche keine spezifischen Vorkenntnisse erfordern. Über einen Experten Modus ist eine erweiterte Filtermaske verfügbar. Diese bietet dem Nutzer die Möglichkeit nach einzelnen oder mehreren Stationsnamen und individuell festgelegtem Zeitraum Klimadaten abzurufen, zu visualisieren und als CSV-Datei zu exportieren.

## 5.3 Datenimport und -verarbeitung

Ein zentraler Bestandteil des Portals ist der automatisierte Import meteorologischer Daten vom DWD. Dieser Prozess umfasst die Erkennung neuer Dateien sowie deren Verarbeitung und Integration in die relationale Datenbank.

### 5.3.1 Datenquelle und Struktur

Die importierten Daten stammen vom Open-Data-Angebot des DWD basierend aus dem täglichen Online-Verzeichnis[66]. Die dort angebotenen ZIP-Archive enthalten tägliche Klimadaten einzelner Wetterstationen, welche im CSV-Format codiert sind. Die relevanten Spalten beinhalten unter anderem:

- Station ID,
- Datum,
- Lufttemperatur,
- Niederschlagshöhe,
- Sonnenscheindauer,
- Windgeschwindigkeit,
- Schnee, etc.

Die Werte werden beim Import in den jeweils richtigen Datentyp konvertiert. Dabei werden fehlerhafte oder fehlende Messungen (z. B. -999) in NULL umgewandelt.

### 5.3.2 Importlogik

Der Importprozess wird von der Klasse *WetterDWDImporter* gesteuert und unterscheidet zwischen dem Import historischer („historical“) und aktueller („recent“) Wetterdaten, da sich die Verfügbarkeit und Verarbeitungsanforderungen unterscheiden. Diese Datenkategorien werden in Kapitel 5.3.3 veranschaulicht.

Der Verlauf bei historischen Daten sieht folgendermaßen aus (siehe Abbildung 15):

[66] https://opendata.dwd.de/climate_environment/CDC/observations_germany/climate/daily/kl/

1. Verzeichnisscan: Mithilfe von Jsoup[67] wird die HTML-Verzeichnisstruktur der DWD-Webseite analysiert und alle verfügbaren ZIP-Dateien aufgelistet.
2. Duplikatsüberprüfung: Über die Tabelle imported_file wird geprüft, ob eine Datei bereits verarbeitet wurde.
3. ZIP-Download und Entpackung: Noch nicht importierte Dateien werden heruntergeladen, entpackt und vorbereitet.
4. CSV-Verarbeitung: Die entpackte CSV-Datei wird zeilenweise gelesen, relevante Felder extrahiert und interpretiert (z. B. -999 wird als null gesetzt).
5. Validierung und Speicherung: Vor dem Einfügen in die Datenbank wird geprüft, ob für die Kombination aus stationId und date bereits ein Eintrag existiert. Nur wenn sich die Werte unterscheiden oder keine übereinstimmende Kombinationen ergeben, wird entweder ein Update oder ein Eintrag durchgeführt.

[67] Jsoup ist eine Java-Bibliothek zum Parsen und Auslesen von HTML-Dokumenten.

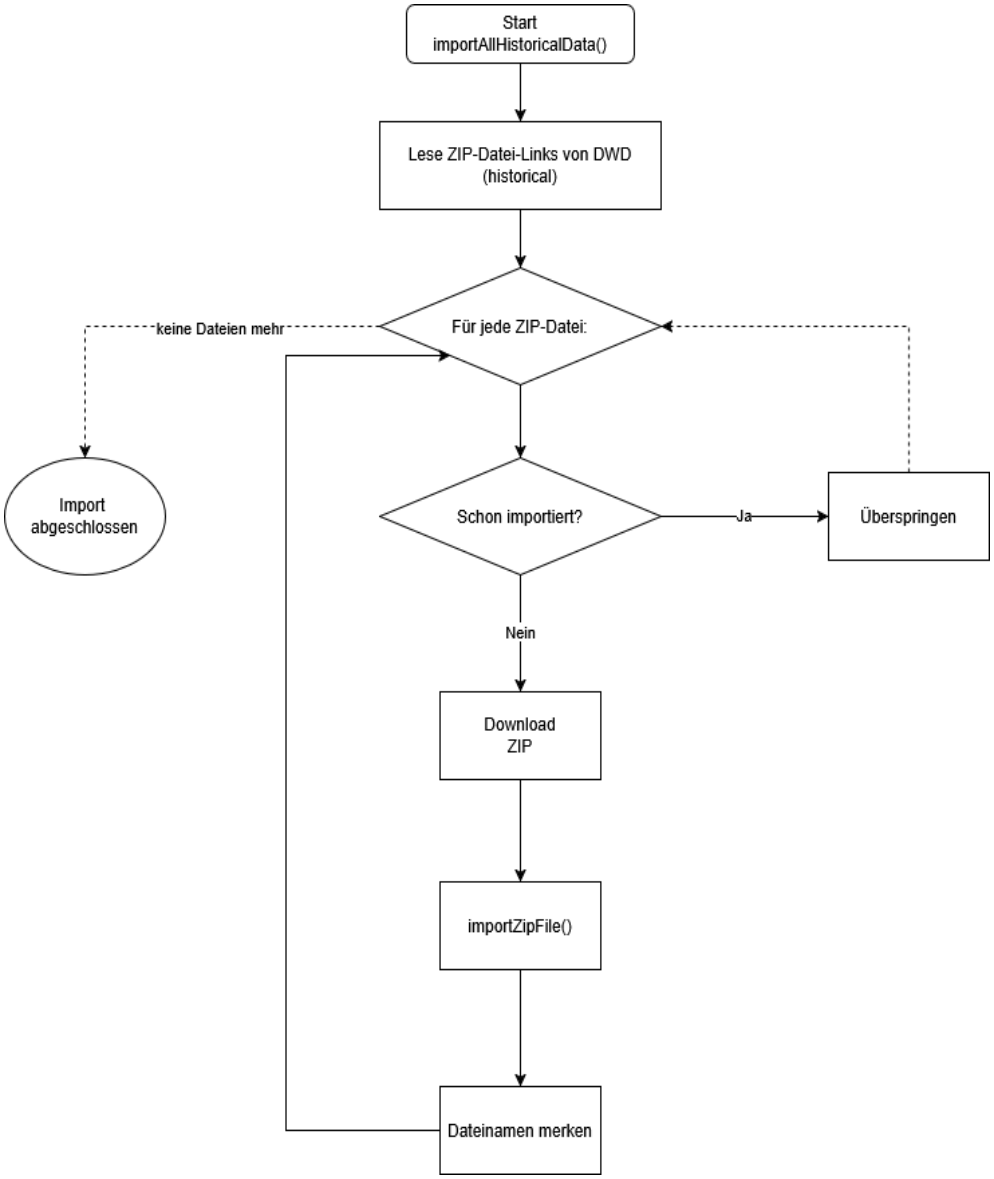



Abbildung 15: Flowchart Historische Datenimport (Aslan, 2025)

Der Verlauf bei aktuellen Wetterdaten verhält sich anders, da eine Kontrolle über Dateinamen nicht möglich ist. Abbildung 16 stellt den Flowchart für aktuellen Datenimport dar:

1. Untersuchen der Datei *recent_time.txt*: Diese Datei dokumentiert, wann zuletzt der recent-Import erfolgte. Falls das aktuelle Datum mit dem Datum von dieser Datei übereinstimmt, wird der Import der aktuellen Daten übersprungen.
2. ZIP-Download der aktuellen Datei: Die aktuelle ZIP-Datei wird heruntergeladen und entpackt.
3. CSV-Verarbeitung: Die entpackte CSV-Datei wird zeilenweise gelesen, relevante Felder werden extrahiert und interpretiert (z. B. -999 wird als NULL gesetzt).

4. Validierung und Speicherung: Vor dem Einfügen in die Datenbank wird geprüft, ob für die Kombination aus stationId und date bereits ein Eintrag existiert. Nur wenn sich die Werte unterscheiden, wird ein Update durchgeführt.
5. Aktualisierung von *recent_time.txt*: Nach erfolgreichem Import wird der Zeitstempel aktualisiert, um doppelte Verarbeitungen zu vermeiden.

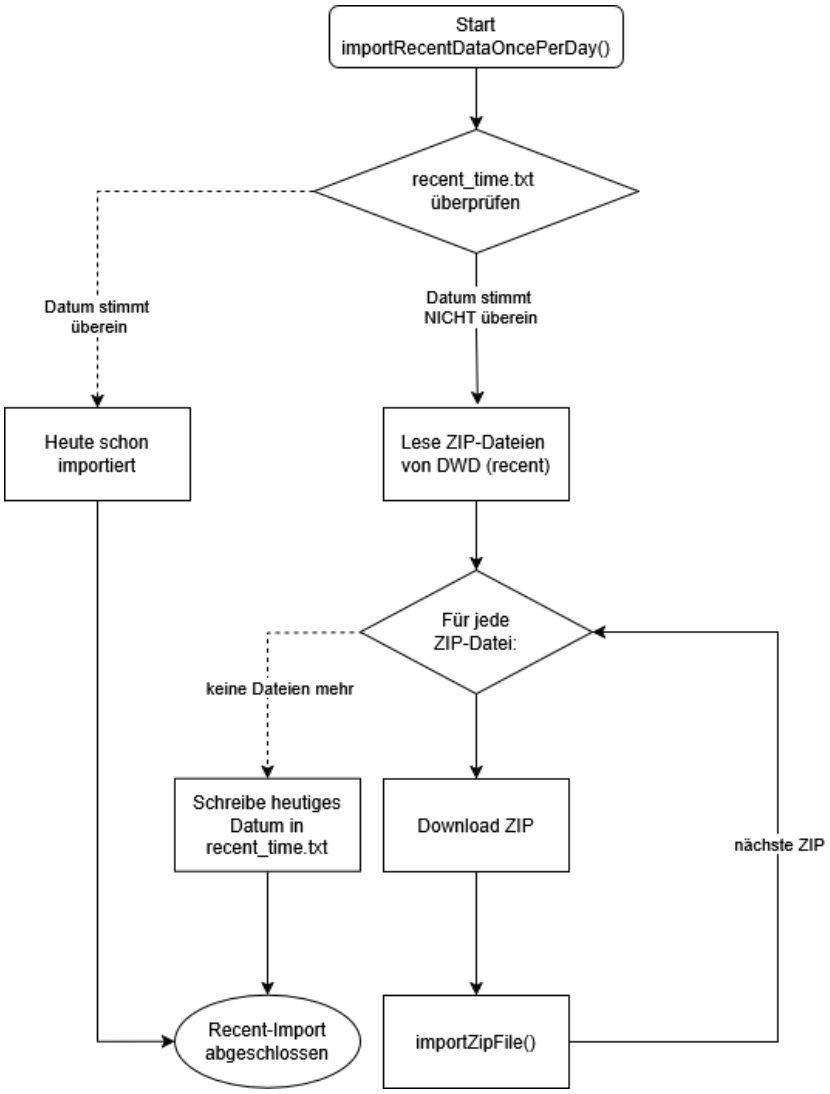



Abbildung 16: Flowchart Aktuelle Datenimport (Aslan, 2025)

### 5.3.3 Historische Daten vs. Aktuelle Daten

Im vorherigen Unterkapitel werden die beiden Datenkategorien historische Daten sowie aktuelle Daten angeführt. Nachfolgend werden die Datenkategorien näher betrachtet und gemäß der Norm definiert. Für wenige Stationen gelten abweichende Definitionen, die nicht näher erläutert werden. Die beiden Datenkategorien sind:

- Historische Daten („historical“): Diese werden nur einmalig importiert (nach Dateinamen) und bilden den Zeitraum seit Anbeginn der Aufzeichnung der Wetterdaten bis einschließlich das jeweils letzte Kalenderjahr ab. Für das aktu-

elle Kalenderjahr 2025 bedeutet dies, dass die historischen Daten den Zeitraum bis einschließlich Kalenderjahr 2024 wiedergeben.

  - Beispiel: tageswerte_KL_01001_19400101_19860630_hist.zip

- Aktuelle Daten („recent“): Diese werden täglich importiert, auch wenn der Dateiname sich nicht ändert. Zur Prüfung dienen hier stationId und date sowie Vergleiche der Messwerte. Aktuelle Daten beinhalten die Klimadaten des aktuellen Kalenderjahrs sowie der letzten zwei vorherigen Kalenderjahre. Beispielsweise bedeutet dies für das aktuelle Kalenderjahr 2025, dass aktuelle Daten auf die Kalenderjahre 2023, 2024 sowie 2025 basieren.
  - Beispiel: tageswerte_KL_01001_akt.zip

Dies ist notwendig, da aktuelle Datensätze vom DWD rückwirkend angepasst werden können. Zwischen den aktuellen Datensätzen und den historischen Datensätzen gibt es in jedem aktuellen Kalenderjahr einen Überschneidungszeitraum. Dieser setzt sich zusammen aus den Daten der letzten zwei Kalenderjahre zuvor. Wie in Abbildung 17 zu sehen ist, ist der Überschneidungszeitraum für das aktuelle Kalenderjahr 2025 die Kalenderjahre 2023 und 2024. Die Relevanz der Berücksichtigung des Überschneidungszeitraums ist hoch, da ansonsten doppelte Datensätze in der Datenbank stehen würde.

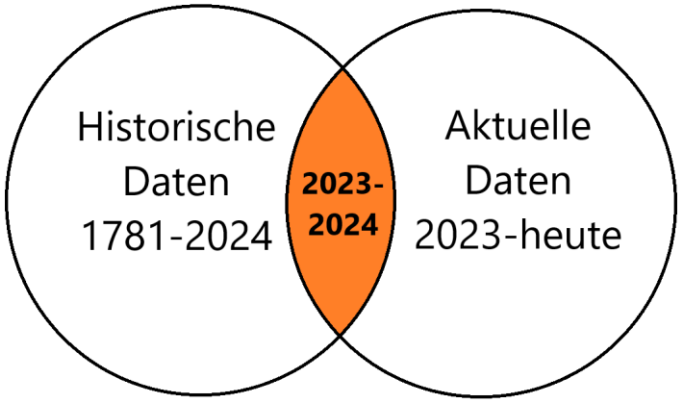



Abbildung 17: Überschneidung der aktuellen und historischen Daten für das Kalenderjahr 2025 (Aslan, 2025)

### 5.3.4 Fehlerbehandlung und Robustheit

Um zu gewährleisten, dass die Anwendung robust gegenüber einzelnen Dateifehler sowie fehlertolerant ist, sind Maßnahmen notwendig. Liegen fehlerhafte oder unvollständige Dateien vor, wird wie folgt vorgegangen: Jede Datei wird in einem separaten try-catch-Block. Ein try-catch-Block ist eine Programmierkonstruktion, mit dem potenziellen Fehler abgefangen und behandelt werden können, ohne dass das Programm unerwartet abstürzt. Die Robustheit des Backendsystems wird dadurch ergänzt, dass Fehler beim Parsing oder bei Netzwerkverbindungen nicht zum Abbruch des gesamten Prozesses führen.

### 5.3.5 Erweiterbarkeit des Datenimports

Bevor in Kapitel 6 die Evaluation des Portals erfolgt, wird im letzten Unterkapitel des Kapitels 5 darauf hingewiesen, dass eine potenzielle Erweiterbarkeit der konzipierten Importlogik besteht.

Dies ist durch den modularen Aufbau der Importlogik möglich. Erweiterungen am System wie der Import anderer Messgrößen (z. B. stündliche Daten) sind ohne strukturelle Änderungen am System realisierbar. Detaillierter ist dies in Kapitel 7.1 beschrieben.

# 6 Evaluation

In diesem Kapitel werden die Funktion, die Usability, die Zugänglichkeit, das Systemverhalten sowie die Leistungsfähigkeit des Portals analysiert.

## 6.1 Funktionale Evaluation

Ziel der funktionalen Evaluation ist es zu überprüfen, inwieweit die im Rahmen der Konzeption definierten Anforderungen (siehe Kapitel 4.2 und 4.3) erfüllt werden. Im Fokus stehen dabei die zentralen Funktionen des Portals:

- Der Import und die Verarbeitung meteorologischer Daten,
- Abfrage und Filterung der importierten Klimadaten,
- Darstellung über das Web-Frontend.

### 6.1.1 Datenimport und -speicherung

Die automatisierte Importlogik funktioniert wie folgt: ZIP-Archive des DWD werden erfolgreich erkannt, entpackt und verarbeitet. Zur Gewährleistung der Datenintegrität sorgen die Validierung von Duplikaten (über die Station ID und Datum) sowie die Möglichkeit, rückwirkende Änderungen in bereits gespeicherten Datensätzen zu berücksichtigen.

Die Speicherung in der PostgreSQL-Datenbank erfolgt in normalisierter Form über die Tabellen weather_data und weather_station. Die Datenstruktur erlaubt sowohl historische als auch aktuelle Einträge.

### 6.1.2 Backend-Logik und API

Die Implementierung der REST-API mit Spring Boot erlaubt die gezielte Abfrage von Wetterdaten über flexible Parameter:

- Station ID oder Ortsname,
- Datumsbereich (Start- und Enddatum),
- Filter für Temperatur, Niederschlag, Windgeschwindigkeit, etc.

Alle API-Endpunkte sind abrufbar, reagieren zuverlässig und liefern, je nach gewähltem Format, strukturierte JSON- oder CSV-Daten. Auch bei hoher Last oder komplexeren Abfragen bleibt die Antwortzeit auf einem zufriedenstellenden Niveau. Nähere Details zur Antwortzeit sind in Kapitel 6.3.2 aufgeführt.

### 6.1.3 Frontend-Funktionalität

Das React-Frontend erlaubt Interaktionen mit den Daten mittels einer Leaflet-Karte mit Klickfunktion. Der Nutzer kann gemäß der Position des Mausklicks die nächstgelegene Wetterstation abrufen. Das Portal ist selbst bei umfangreichen Datensätzen ausreichend leistungsfähig, da durch die Implementierung der „react-window“-Funktion nur sichtbare Datenbereiche gerendert werden.

Zu den erfolgreich umgesetzten Funktionen zählen:

- Auf der Startseite des Portals wird es dem Nutzer ermöglicht, zu entscheiden, welche Stationen auf der Karte visualisiert werden sollen. Entweder kann sich der Nutzer alle Stationen auf der Karte anzeigen lassen oder aber nur solche Stationen, die bis einschließlich Kalenderjahr 2024 aktiv waren und seit mindestens 50 Jahren im Einsatz sind bzw. waren, auswählen. Weiterhin steht ein Experten Modus zur Verfügung, welcher unmittelbar unter der Karte platziert ist. Dieser gewährt Zugriff auf die Filtermaske. Abbildung 18 zeigt die Startseite des Portals.
- Beim Auswählen der Option „Stationen anzeigen“ werden dem Nutzer auf der Landkarte alle Wetterstationen vom DWD angezeigt (Abbildung 19).
- Beim Auswählen der Option „nur Stationen mit Lebensdauer > 50 Jahre anzeigen und aktiv bis mindestens Ende 2024“ werden dem Nutzer nur die Stationen angezeigt, die länger als 50 Jahre aktiv sind bzw. waren sowie bis mindestens Ende 2024 noch Daten ausgegeben haben (Abbildung 20).
- Über die „Zoom-In“-Funktion wird die erwünschte Region des Nutzers vergrößert und genauer betrachtet (Abbildung 21).
- Durch Mausklick auf einen beliebigen Ort auf der Landkarte, wird dem Nutzer ein Dialogfenster angezeigt. Das Dialogfenster beinhaltet sowohl visualisierte Daten (wie z.B. Temperaturverlauf über die Jahre) als auch die einzelnen Datensätze der Region in tabellarischer Form (Abbildung 22).
- Wird der Experten Modus unter der Landkarte ausgewählt, so hat der Nutzer Zugriff auf die Filtermaske. Über diese Filtermaske kann der Nutzer im Suchfeld den von ihm gewünschten Ort (z.B. Stuttgart) eingeben, sowie einen individuellen Zeitraum für die Messwerte festlegen. Mit dem Mausklick auf den „Filtern“-Knopf wird die erwünschte Suche gestartet (Abbildung 23).
- Der Experten Modus ruft die erwünschten Daten ab und verarbeitet sie so, dass sie dem Nutzer veranschaulicht werden. Dafür werden die Daten sowohl in tabellarischer Form (Abbildung 24) wiedergegeben als auch visuell dargestellt. Die Abbildung 25 zeigt eine beispielhafte Grafik, die dem Nutzer bereitgestellt wird. Weiterhin bietet sich unterhalb des Diagramms die zusätzliche Funktion, die abgerufenen Daten als CSV-Datei zu exportieren.

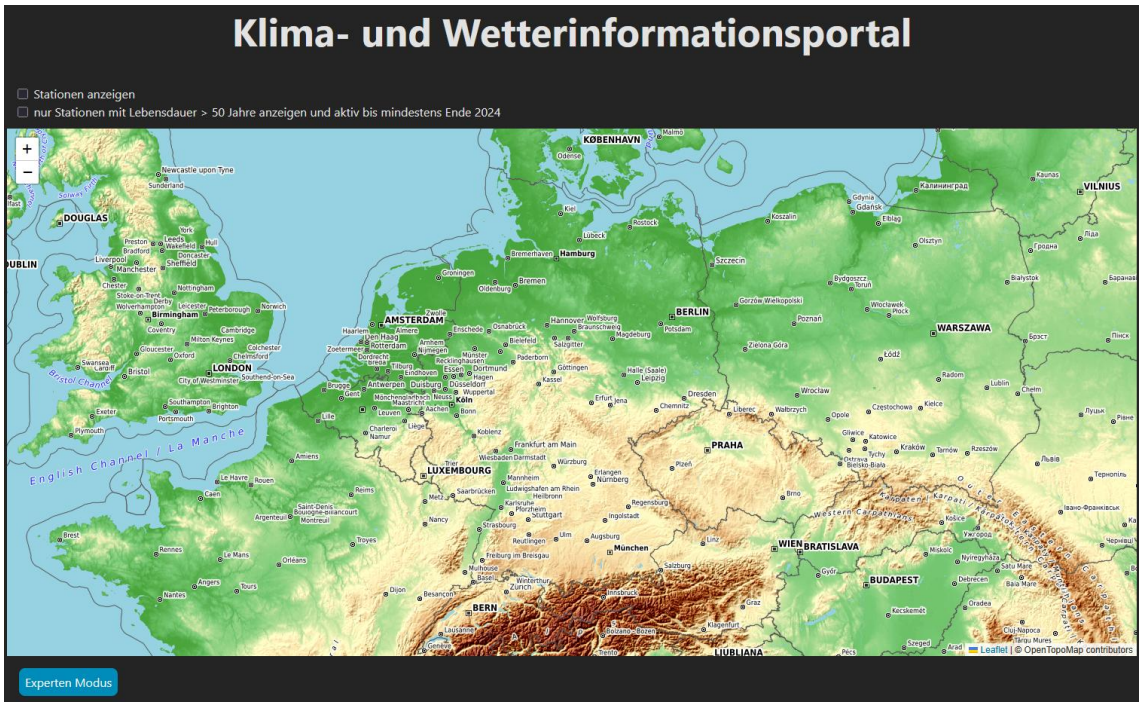



Abbildung 18: Startseite des Portals (Aslan, 2025)

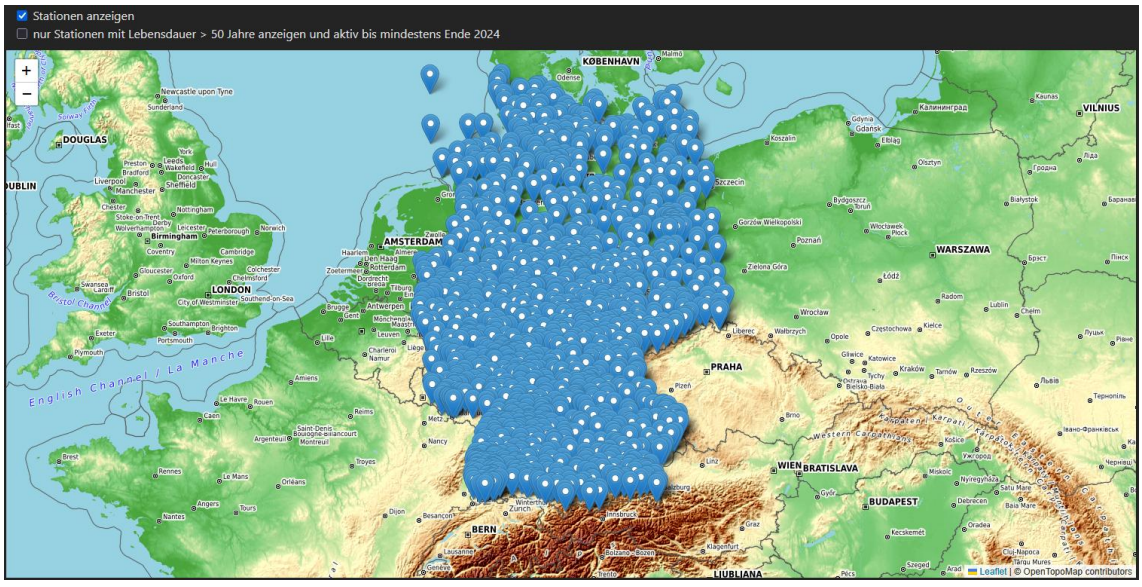



Abbildung 19: Stationen und Umkreis Anzeige angeschaltet (Aslan, 2025)

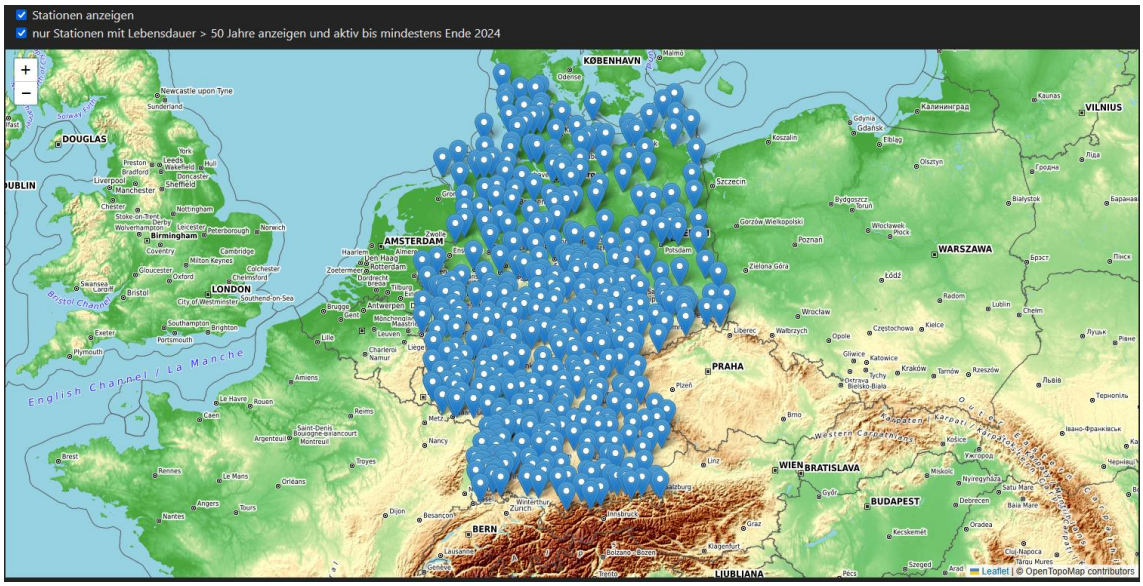



Abbildung 20: Stationen mit Lebensdauer > 50 Jahre und aktiv bis Ende 2024 (Aslan, 2025)

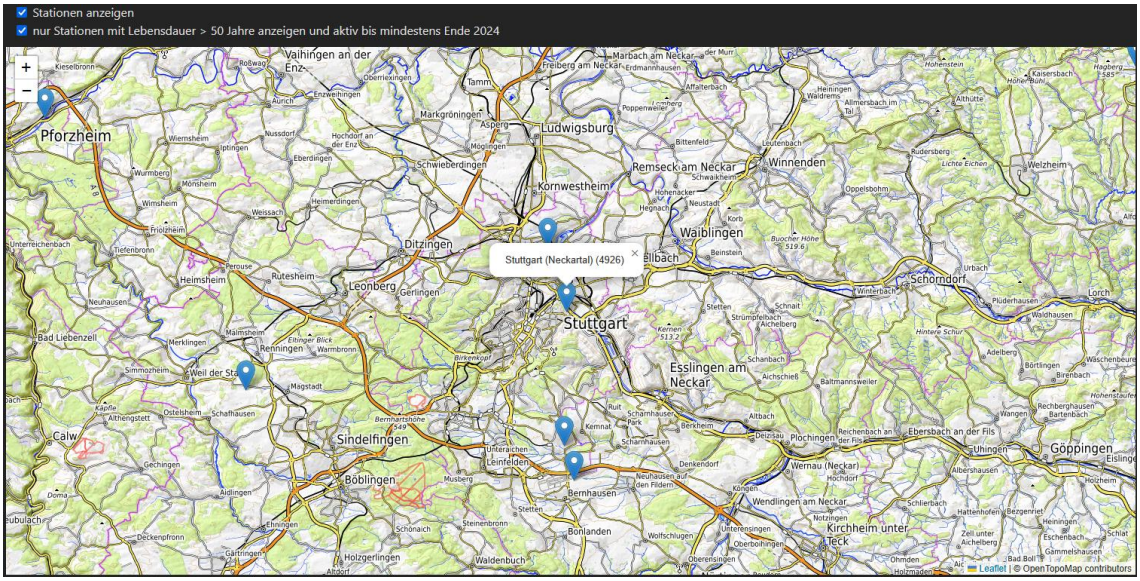



Abbildung 21: Die „Zoom-In“-Funktion in der Umgebung Stuttgart (Aslan, 2025)

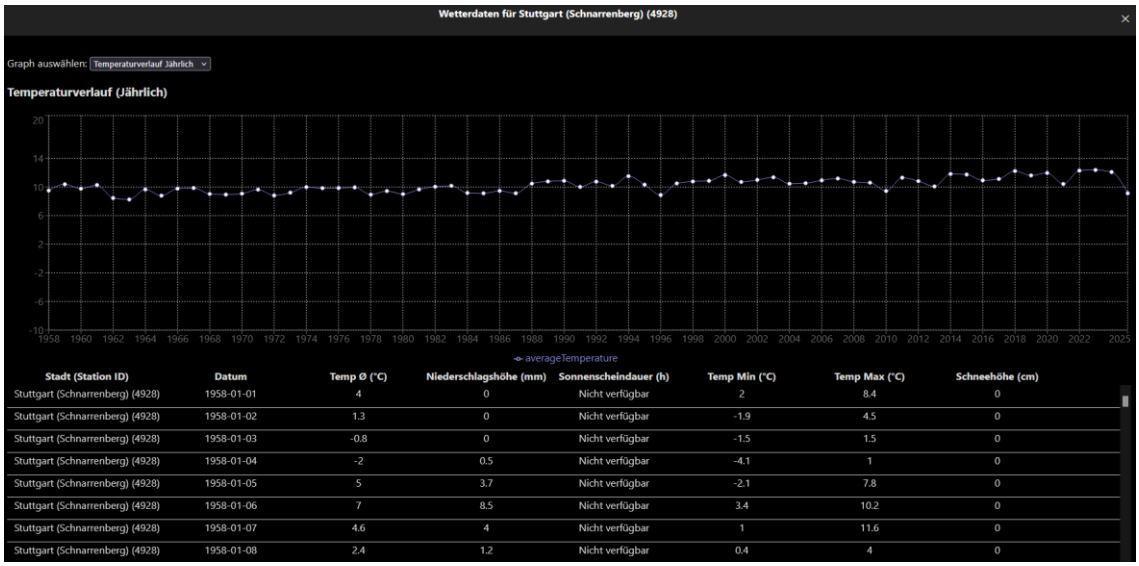



Abbildung 22: Klimadaten von Stuttgart (Schnarrenberg) (Aslan, 2025)

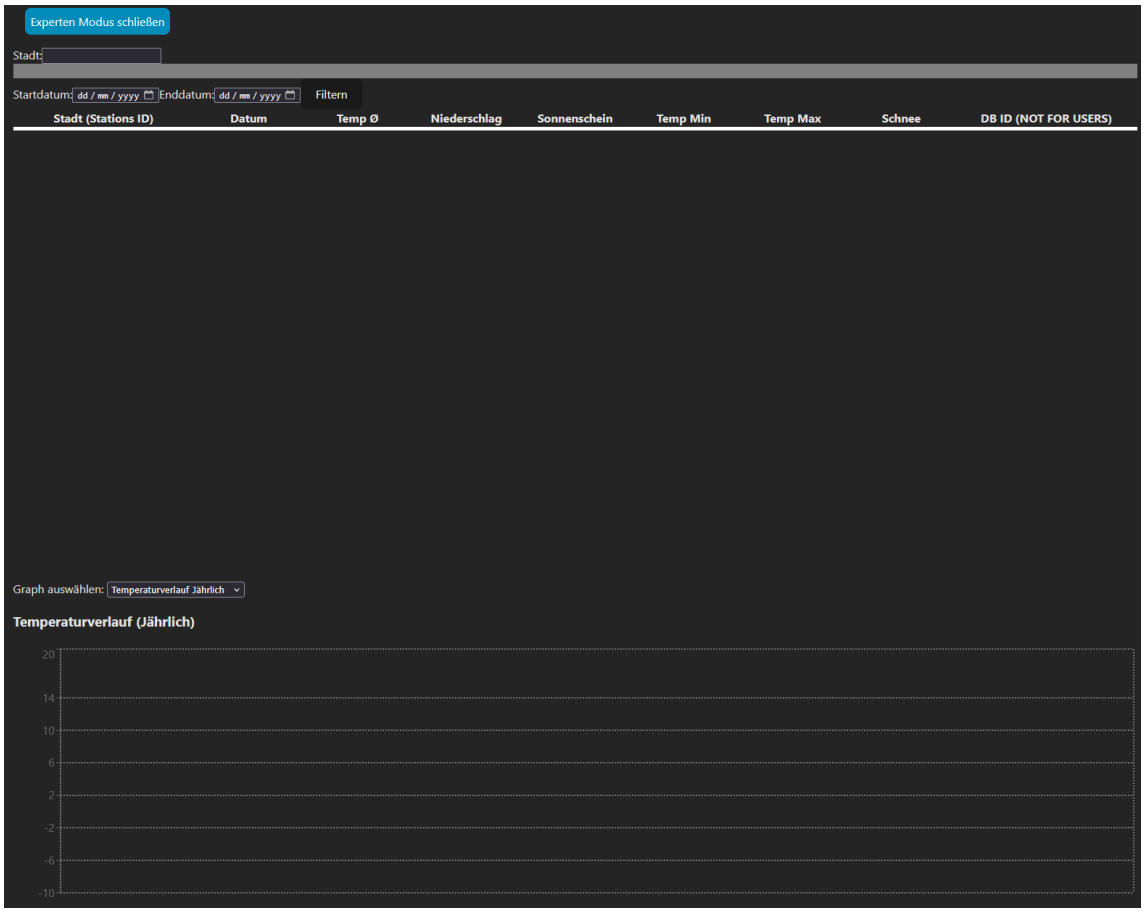
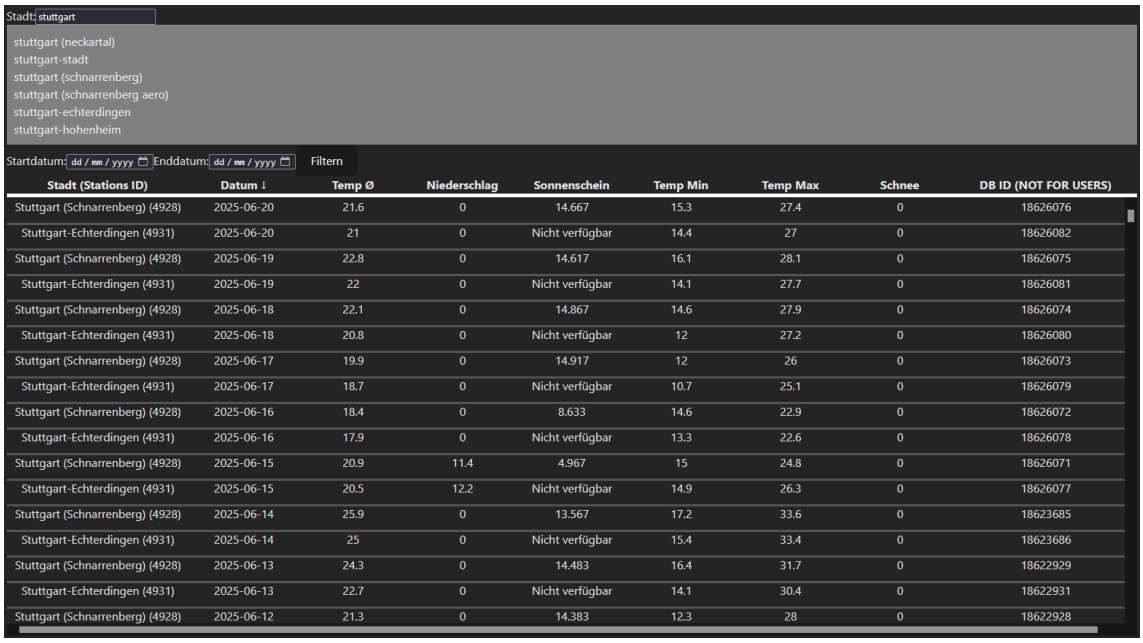



Abbildung 23: Experten Modus unter der Karte (Aslan, 2025)




| Stadt (Stations ID) | Datum ↓ | Temp Ø | Niederschlag | Sonnenschein | Temp Min | Temp Max | Schnee | DB ID (NOT FOR USERS) |
|---|---|---|---|---|---|---|---|---|
| Stuttgart (Schnarrenberg) (4928) | 2025-06-20 | 21.6 | 0 | 14.667 | 15.3 | 27.4 | 0 | 18626076 |
| Stuttgart-Echterdingen (4931) | 2025-06-20 | 21 | 0 | Nicht verfügbar | 14.4 | 27 | 0 | 18626082 |
| Stuttgart (Schnarrenberg) (4928) | 2025-06-19 | 22.8 | 0 | 14.617 | 16.1 | 28.1 | 0 | 18626075 |
| Stuttgart-Echterdingen (4931) | 2025-06-19 | 22 | 0 | Nicht verfügbar | 14.1 | 27.7 | 0 | 18626081 |
| Stuttgart (Schnarrenberg) (4928) | 2025-06-18 | 22.1 | 0 | 14.867 | 14.6 | 27.9 | 0 | 18626074 |
| Stuttgart-Echterdingen (4931) | 2025-06-18 | 20.8 | 0 | Nicht verfügbar | 12 | 27.2 | 0 | 18626080 |
| Stuttgart (Schnarrenberg) (4928) | 2025-06-17 | 19.9 | 0 | 14.917 | 12 | 26 | 0 | 18626073 |
| Stuttgart-Echterdingen (4931) | 2025-06-17 | 18.7 | 0 | Nicht verfügbar | 10.7 | 25.1 | 0 | 18626079 |
| Stuttgart (Schnarrenberg) (4928) | 2025-06-16 | 18.4 | 0 | 8.633 | 14.6 | 22.9 | 0 | 18626072 |
| Stuttgart-Echterdingen (4931) | 2025-06-16 | 17.9 | 0 | Nicht verfügbar | 13.3 | 22.6 | 0 | 18626078 |
| Stuttgart (Schnarrenberg) (4928) | 2025-06-15 | 20.9 | 11.4 | 4.967 | 15 | 24.8 | 0 | 18626071 |
| Stuttgart-Echterdingen (4931) | 2025-06-15 | 20.5 | 12.2 | Nicht verfügbar | 14.9 | 26.3 | 0 | 18626077 |
| Stuttgart (Schnarrenberg) (4928) | 2025-06-14 | 25.9 | 0 | 13.567 | 17.2 | 33.6 | 0 | 18623685 |
| Stuttgart-Echterdingen (4931) | 2025-06-14 | 25 | 0 | Nicht verfügbar | 15.4 | 33.4 | 0 | 18623686 |
| Stuttgart (Schnarrenberg) (4928) | 2025-06-13 | 24.3 | 0 | 14.483 | 16.4 | 31.7 | 0 | 18622929 |
| Stuttgart-Echterdingen (4931) | 2025-06-13 | 22.7 | 0 | Nicht verfügbar | 14.1 | 30.4 | 0 | 18622931 |
| Stuttgart (Schnarrenberg) (4928) | 2025-06-12 | 21.3 | 0 | 14.383 | 12.3 | 28 | 0 | 18622928 |

Abbildung 24: Tabellarische Anzeige der Daten von Stuttgart (Aslan, 2025)

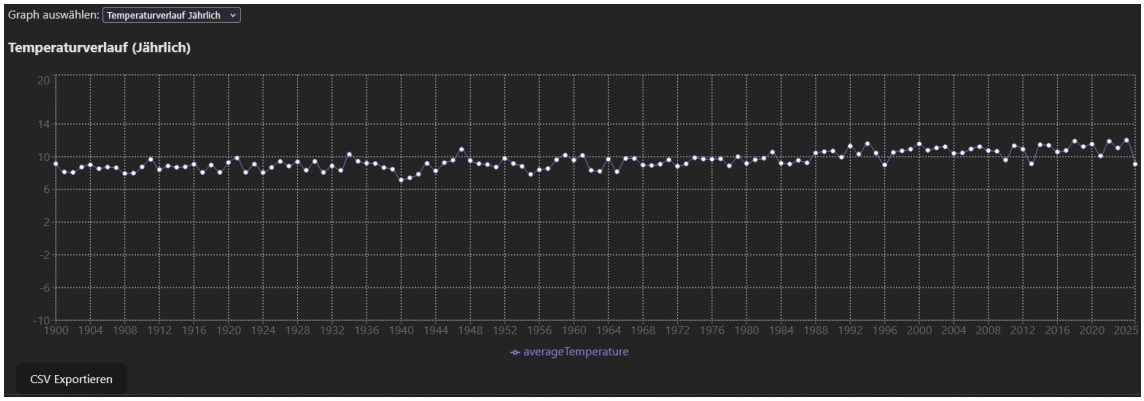



Abbildung 25: Grafische Anzeige der Daten von Stuttgart (Aslan, 2025)

### 6.1.4 Ergebnisbewertung im Abgleich mit den funktionalen Anforderungen

In der nachfolgenden Tabelle 3 sind die in Kapitel 4.2 und 4.3 definierten Anforderungen bewertet. Wie zu sehen ist, sind insbesondere die backendspezifischen Anforderungen alle umgesetzt. Die einzige Anforderung, welche nicht umgesetzt werden konnte, ist die frontendspezifische Anforderung „Benutzerfreundliche und ansprechende UI“. Dies ist auf die fehlende Erfahrung des Verfassers dieser Arbeit sowie der begrenzten Bearbeitungszeit zurückzuführen.

| Anforderung | Umgesetzt | Bemerkung |
|---|---|---|
| Automatischer Import von DWD-Daten | Ja | Erfolgt bei Start |
| Speicherung und Abfrage von Wetterdaten | Ja | |
| Dynamische Filterung über API | Ja | Vollständig über Parameter |
| CSV-Export | Ja | Formatwahl über URL-Query und per Button |
| Darstellung in Diagrammen und Tabellen | Ja | Leaflet + Recharts |
| Interaktive Stationsauswahl über Karte | Ja | Nächstgelegene Station wird berechnet |
| Benutzerfreundliche und ansprechende UI | Nein | Begrenzte Zeit dieser Arbeit und Erfahrung |

Tabelle 3: Ergebnisauswertung der Anforderungen (Aslan, 2025)

Zusammenfassend ist festzustellen, dass die wesentlichen Funktionen des Portals erfolgreich umgesetzt worden sind. Dabei arbeiten alle Komponenten wie geplant und gewünscht zusammen. Die Schnittstellen fungieren zuverlässig. In den weiteren Unterkapiteln der Evaluation werden über die Funktionalität hinaus weitere Aspekte des Portals durchleuchtet.

## 6.2 Usability und Zugänglichkeit

Ein zentrales Ziel dieser Arbeit ist, meteorologische Daten nicht nur bereitzustellen, sondern sie auch leicht zugänglich und benutzerfreundlich aufzubereiten. Dieser Aspekt ist insbesondere für Nutzer ohne Fachwissen essenziell. In diesem Abschnitt wird bewertet, wie gut dieses Ziel durch das aktuelle Design und die technische Umsetzung erreicht wird.

### 6.2.1 Zielgruppengerechte Darstellung

Da das Thema Klimawandel allgegenwärtig präsent ist, richtet sich das Portal an eine schwer abzugrenzende große Gruppe der Bevölkerung. Es wird antizipiert, dass sich diese Gruppe verstärkt für das Klima interessiert. Demnach wird dieser Gruppe das Portal zur Verfügung gestellt, um historische und gegenwärtige Klimadaten zu untersuchen. Insbesondere liegt der Fokus darauf, die Klimadaten des gewünschten Standorts (z.B. eigener Wohnort) einsehen zu können.

Dafür müssen folgende Elemente integriert sein:

- Allgemeinverständliche Sprache im Frontend (z. B. „Temperaturverlauf jährlich“ statt „Zeitreihenanalyse),
- Intuitive Filteroberfläche zur Einschränkung nach Ort, Zeitraum und Wetterparametern,
- Standardisiertes Design, das responsiv auf verschiedene Bildschirmgrößen reagiert.

### 6.2.2 Visuelle Aufbereitung

Die Nutzung von Diagrammbibliotheken (Recharts) ermöglicht eine grafische Darstellung meteorologischer Daten, z. B. Temperaturverläufe über die letzten Jahrzehnte, wie bereits in Abbildung 25 gezeigt wird. Dadurch wird eine schnellere Erfassung von Trends und Abweichungen unterstützt.

Ergänzt wird dies durch die interaktive Karte mit Leaflet, welche die geografische Einordnung der Wetterstationen vereinfacht. Die Auswahl der jeweils nächstgelegenen Station per Mausklick erhöht zusätzlich die Nutzerfreundlichkeit, da keine spezifischen Stationsnummern bekannt sein müssen.

### 6.2.3 Bedienbarkeit und Feedback

Die Anwendung bietet eine einfache Navigation:

- Wählbare Station auf einer interaktiven Leaflet-Karte,
- Schnelle Ladezeiten auch bei größeren Datenmengen.

Besonders hervorzuheben ist die Implementierung von react-window, die es ermöglicht, große Datenmengen performant zu rendern, ohne dass die Benutzeroberfläche blockiert wird.

## 6.3 Systemverhalten und Leistungsfähigkeit

Neben der korrekten funktionalen Umsetzung ist auch die Leistungsfähigkeit des Systems ein zentraler Aspekt bei der Evaluation eines Webportals. In diesem Abschnitt wird analysiert, wie sich das System bei der Datenverarbeitung, beim Datenabruf und unter Last verhält.

### 6.3.1 Datenverarbeitung und Importverhalten

Die Importlogik des Backends ist darauf ausgelegt, sowohl historische als auch täglich aktualisierte Wetterdaten des DWD automatisiert zu verarbeiten. Dabei kommen gezielte Optimierungen zum Einsatz:

- Duplikatvermeidung: Daten werden anhand der Kombination aus Station ID und Datum eindeutig identifiziert. Existiert bereits ein Datensatz für denselben Tag und dieselbe Station, wird dieser nur überschrieben, wenn sich Werte geändert haben.
- Vermeidung von Masseneinträgen: Durch gezielte Importkontrolle wird verhindert, dass identische Daten mehrfach geschrieben werden. Dies führt zu Ressourceneffizienz.
- Zeitgesteuerte Importe: Für aktuelle Daten sorgt die Einrichtung eines zeitstempelorientierten Tagesimports. Dadurch wird die Aktualität der Daten gewährleistet.

### 6.3.2 Datenbankperformance

Die PostgreSQL-Datenbank speichert Wetterdaten über mehrere Jahrzehnte und um die 18 Millionen Einträge. Die gewählte Normalisierung (Trennung zwischen Stationen und Wetterdaten) führt zu idealen Antwortzeiten selbst bei größeren Anfragen.

Die Messungen der Importdauer und API-Antwortzeiten wird mit dem Programm Postman[68] durchgeführt. Dabei werden gezielte GET-Anfragen an die definierte API-Endpunkte gestellt und die Antwortzeiten im integrierten Zeitfenster von Postman protokolliert. Alle Messungen werden im lokalen Netzwerk durchgeführt.

Die Tests erfolgten auf folgendem System:

- Betriebssystem: Windows 11 (24H2)
- Prozessor: AMD Ryzen 9 9800X3D
- Grafikkarte: AMD RX 9070 XT
- Arbeitsspeicher: 32 GB DDR5 @ 6400 MHz (2 x 16 GB)
- Speichermedium: Samsung SSD 990 Pro 2 TB

Diese Angaben dienen der Nachvollziehbarkeit der Systemleistung, da Ladezeiten stets auch durch die Hardware und die Umgebung beeinflusst werden.

Während der Messung gibt Postman detailliert an, welche Messpunkte wie viel Zeit in Anspruch nehmen. Die Erläuterung der Messpunkte ist wie folgt:

- Socket Initialization: Aufbau der Verbindung zum Server auf Betriebssystemebene.
- DNS Lookup: Auflösung des Domainnamens in eine IP-Adresse.
- TCP Handshake: Verbindungsaufbau zwischen Client und Server über das TCP-Protokoll.

[68] Postman ist ein Tool zur Entwicklung und zum Testen von APIs, mit dem HTTP-Anfragen gesendet und deren Antworten analysiert werden können (https://www.postman.com/)

- Transfer Start: Zeit, bis der Server die Anfrage verarbeitet und mit dem Senden der Antwort beginnt.
- Download: Dauer des Herunterladens der Antwortdaten.
- Process: Dauer der Verarbeitung der Antwortdaten von Postman (z. B. Formatierung). Diese wird nicht zur Serverantwortzeit hinzuaddiert.

Ladezeiten für 1 Millionen Datensätze: < 11,3 Sekunden (Abbildung 26)

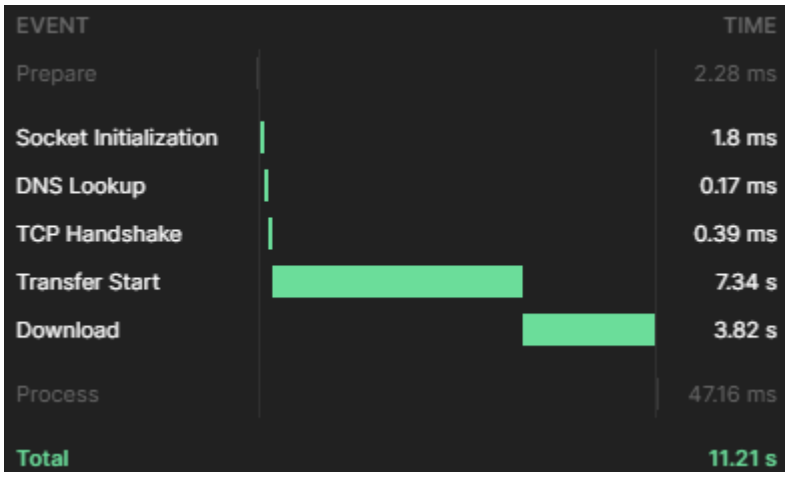



Abbildung 26: Ladezeit für 1 Millionen Datensätze (Aslan, 2025)

Ladezeit aller Datensätze von Stuttgart (> 100.000 Datensätze): <2,2 Sekunden (Abbildung 27)

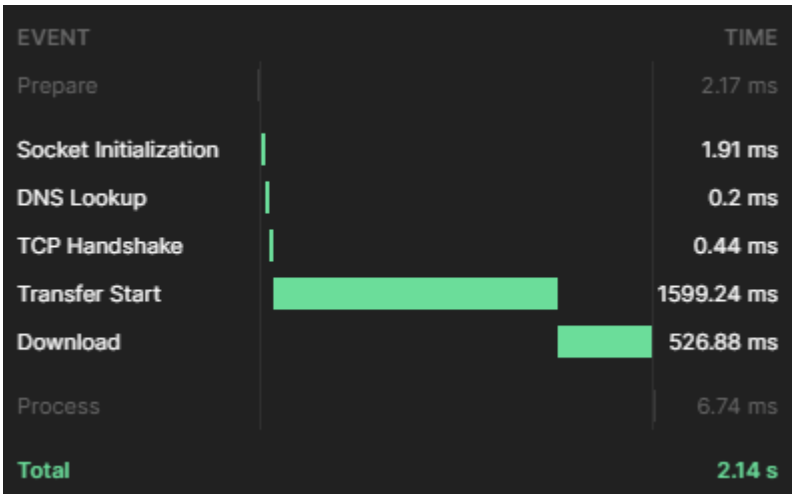



Abbildung 27: Ladezeit aller Datensätze Stuttgart (Aslan, 2025)

Ladezeit aller Datensätze von Berlin (> 300.000 Datensätze): <3,8 Sekunden (Abbildung 28)

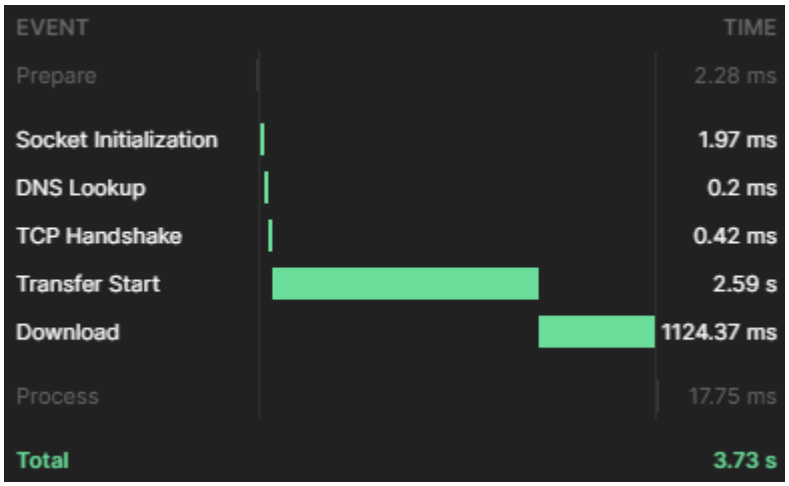



Abbildung 28: Ladezeit aller Datensätze Berlin (Aslan, 2025)

### 6.3.3 API-Zuverlässigkeit

Die REST-API wird getestet auf:

- Stabilität bei parallelen Abfragen (Szenario: Vier parallele Abfragen mit 1 Millionen zufälligen Datensätzen erfolgreich),
- Antwortzeit bei gefilterten Anfragen,
- Richtigkeit der gelieferten Daten (insbesondere beim Filtern nach Bereichen).

Das Verhalten ist durchgehend stabil. In lokalen Tests werden mehrere gleichzeitige Abfragen verarbeitet. Es werden keine Timeouts[69] oder Speicherüberläufe[70] festgestellt.

### 6.3.4 Frontend-Performance

Das React-Frontend verarbeitet die Daten clientseitig. Clientseitig bedeutet, dass etwas direkt im Browser des Nutzers passiert, z. B. das Anzeigen von Inhalten oder das Reagieren auf Klicks, und nicht auf dem Server verarbeitet wird. Die Daten werden in folgenden Varianten dargestellt bzw. visualisiert:

- Interaktive Diagramme (Recharts),

[69] Timeouts bezeichnen eine festgelegte Zeitspanne, nach der ein Vorgang automatisch abgebrochen wird, wenn er nicht rechtzeitig abgeschlossen ist.

[70] Ein Speicherüberlauf passiert, wenn ein Programm mehr Arbeitsspeicher verbraucht, als ihm zur Verfügung steht. Das kann zu Abstürzen, Fehlern oder langsamer Leistung führen.

- Tabellen (virtuell gerendert via react-window).

Durch Virtualisierung wird auch bei größeren Ergebnismengen (z. B. mehrere hunderttausend Datensätze) eine flüssige Benutzererfahrung ermöglicht. Die Ladezeit bleibt gering und das Interface reagiert schnell auf Nutzerinteraktion.

## 6.4 Grenzen und Limitationen

Trotz der erfolgreichen Umsetzung des Klima- und Wetterinformationsportals bestehen verschiedene Einschränkungen in funktionaler, technischer und gestalterischer Hinsicht. Diese sind in erster Linie auf die zeitliche Beschränkung der Bearbeitungsdauer sowie dem eingeschränkten Umfang der vorliegenden Bachelorarbeit zurückzuführen. Weiterhin ist die individuelle Erfahrung des Verfassers in bestimmten Technologiegebieten nicht ausgeprägt.

### 6.4.1 Gestalterische Limitierungen des Frontends

Die grafische Benutzeroberfläche erfüllt die definierten funktionalen Anforderungen, weist jedoch in Bezug auf Design, Nutzerführung und Barrierefreiheit Verbesserungspotenzial auf. Das Frontend ist derzeit minimalistisch gehalten und besteht im Wesentlichen aus:

- Einer Leaflet-Karte zur Auswahl geografischer Positionen,
- Einer automatischen Ermittlung der nächstgelegenen Wetterstation gemäß dem Mausklick,
- Einer Datenanzeige in Form von vereinfachten Diagrammen und Tabellen,
- Einer Filtermaske zur erweiterten Suche.

Komponenten wie Animationen, visuelle Rückmeldungen bei Benutzeraktionen, Hilfetexte oder ein responsives Layout für Mobilgeräte sind nur teilweise oder nicht vorhanden. Das schlichte Design ist auf den begrenzten zeitlichen Rahmen der vorliegenden Arbeit sowie der geringen Erfahrung des Verfassers im Bereich Frontend-Gestaltung mit React.js zurückzuführen. Eine Weiterentwicklung durch nachfolgende Projektbeteiligte könnte hier gezielt ansetzen, etwa durch:

- die Integration von Designsystemen wie Material UI[71],
- barrierefreie Gestaltung gemäß WCAG[72],
- optimierte Usability-Tests und Nutzerfeedbackrunden.

[71] Material UI ist eine React Bibliothek, die Googles Material-Design-Richtlinien umsetzt und vorgefertigte UI-Elemente für moderne Webanwendungen bereitstellt.

[72] vgl. Bundesregierung (2025, online)

### 6.4.2 Abdeckung meteorologischer Parameter

Das Portal greift derzeit auf die DWD-Daten der Kategorie „daily“ zurück. Diese beinhalten tägliche Werte wie Temperatur, Niederschlag, Sonnenscheindauer, Windgeschwindigkeit, etc. Andere relevante meteorologische Phänomene wie Luftschadstoffe sind nicht Teil der täglichen Datenbasis. Zudem werden keine Daten im 1/5/10 Minuten oder im Stunden Intervall verarbeitet. In Kapitel 7.1 wird auf diesen Sachverhalt näher eingegangen.

### 6.4.3 Systemtests unter realen Lastbedingungen

Die Anwendung wird lokal und im Entwicklerumfeld getestet. Umfangreiche Last- und Performancetests unter produktionsnahen Bedingungen (z. B. mit 100+ parallelen Nutzern) wird aus Ressourcengründen nicht durchgeführt. Die vorhandenen Leistungsdaten basieren auf Einzelmessungen und Entwicklungsbedingungen. Zur Gewährleistung der Stabilität und Perfomance des Systems für den späteren öffentlich-produktiven Einsatz sind weiterführende Tests empfehlenswert.

### 6.4.4 Zusammenfassung

Die genannten Limitierungen schränken den aktuellen Funktionsumfang des Portals zwar ein, stellen aber keine grundsätzlichen Mängel dar. Vielmehr sind sie als Folge der klaren Priorisierung innerhalb der vorgegebenen Bearbeitungsdauer zu verstehen. Die aktuelle Umsetzung bildet ein solides Fundament, auf das künftige Erweiterungen gezielt aufbauen können.

## 6.5 Entwicklungsverlauf

Der Entwicklungsprozess des Wetterinformationsportals verlief iterativ und ist durch zahlreiche technische Entscheidungen, Problemstellungen und kontinuierliche Verbesserungen geprägt. Die nachfolgenden Unterkapitel zeigen exemplarisch zentrale Phasen und Herausforderungen der Entwicklung sowie deren Lösungen auf. Der Entwicklungsverlauf basiert auf regelmäßig geführten Entwicklungsnotizen.

### 6.5.1 Herausforderungen beim Datenimport

Ein zentraler Aspekt war die Verarbeitung der DWD-Daten. Bereits zu Beginn der Konzeption zeigt sich, dass die Qualität und Konsistenz der Daten nicht durchgängig gegeben sind:

- Einzelne Werte sind als -999 markiert und werden als NULL interpretiert, damit Durchschnittsberechnungen nicht negativ beeinflusst werden,

- Die Importlogik ist so anzupassen, dass Duplikate verhindert und rückwirkende Korrekturen erkannt werden.

Zur Lösung wird eine HashMap[73] verwendet, um bereits importierte Datensätze zu vergleichen und nur geänderte Werte zu aktualisieren.

Ein zusätzliches Hilfsmechanismus ist die Datei *recent_time.txt*, um den letzten erfolgreichen Importzeitpunkt zu dokumentieren. Bei einer produktiven Umgebung, die rund um die Uhr läuft, ist bevorzugt eine Scheduled-Funktion zu benutzen. Scheduled ist eine Java-Funktion, mit der festgelegt wird, dass eine bestimmte Methode automatisch in regelmäßigen Abständen ausgeführt wird. Beispielsweise wird der Datenimport täglich einmal um 10 Uhr durchgeführt. Aufgrund der zeitlichen Begrenzung dieser Bachelorarbeit wird auf die Implementierung der Scheduled-Funktion sowie deren Testläufe verzichtet.

Nicht zu vernachlässigen ist die Importzeit der Klimadaten bei Erstbeladung der Datenbank. Messungen haben ergeben, dass diese etwa 90 Minuten benötigen. Nach der Erstbeladung der Datenbank beansprucht der tägliche Import der aktuellen Daten in der Regel ungefähr sechs Minuten.

### 6.5.2 Optimierung der Performance

Im Verlauf der Entwicklung zeigen sich Performanceengpässe aufgrund der enormen Menge an Klimadaten. Im Kalenderjahr 2025 beträgt die Anzahl der Datensätze seit Anbeginn der Aufzeichnung über 18 Millionen. Beispielsweise sind für die Großstadt Stuttgart in der Datenbank circa 100.000 Datensätze abgelegt. Die Performanceengpässe sind insbesondere in folgenden Fällen des Öfteren zu beobachten:

- Große Datenmengen beim Filterabruf im Backend,
- Rendering großer Tabellen im Frontend.

Um sowohl das Nutzererlebnis responsiver als auch die Last des Servers (Backend) zu verbessern, sind folgende Maßnahmen durchzuführen:

- Backend-seitig wird Paging über Pageable[74] aktiviert,
- Frontend-seitig kommt react-window zum Einsatz, um nur sichtbare Zeilen zu rendern.

[73] Eine HashMap ist ein digitales Nachschlagewerk im Arbeitsspeicher, indem man Werte unter einem eindeutigen Schlüssel speichert, um sie später schneller wiederzufinden.

[74] Pageable ist eine Schnittstelle in Spring Data zur effizienten Verarbeitung großer Datenmengen, mit der sich Datenbankabfragen seitenweise durchführen lassen.

### 6.5.3 Probleme mit CORS und Netzwerkzugriff

Während der Entwicklung treten beim API-Zugriff von der Frontend (localhost:5173) auf die Backend-API (localhost:8080) häufig Probleme auf. Im Problemfall erhält die Frontend über die vordefinierten API-Endpunkte keine Antwort vom Backend. In diesem Fall ist dies durch die CORS-Konfiguration der Klasse *CorsConfig* begründet. Wie in Kapitel 5.1.7 erläutert, erlaubt oder blockiert diese Klasse Zugriffe an die API von anderen Domains. Die Lösung erfolgt durch die Konfiguration der *CorsConfig* Klasse, welche beim derzeitigen Stand alle Ursprünge erlaubt. Für den produktiven Betrieb ist jedoch eine Einschränkung auf definierte Domains erforderlich.

### 6.5.4 Technische Modularisierung

Ein zentrales Ziel bei der Umsetzung des Portals ist, die Architektur so zu gestalten, dass neue Funktionen, Datenquellen oder Visualisierungselemente mit möglichst geringem Aufwand ergänzt werden können. Dies ist insbesondere für die Weiterentwicklung durch zukünftige Projektteams oder Forschungsgruppen von großer Bedeutung. Die technische Modularisierung stellt dabei sicher, dass Komponenten klar getrennt, wiederverwendbar und erweiterbar bleiben.

Die Backend-Logik ist nach dem Prinzip der Schichtenarchitektur aufgebaut. Die Importlogik ist in der dedizierten Klasse *WetterDWDImporter* gekapselt, was folgende Vorteile bietet:

- Klare Trennung von Datenbeschaffung und Datenverarbeitung,
- Möglichkeit neue Importquellen als zusätzliche Importservices zu implementieren,
- Wiederverwendbarkeit der Verarbeitungslogik.

### 6.5.5 Anpassungen der Datenbankstruktur

Die ursprüngliche Datenhaltung sah nur eine Tabelle für Wetterdaten vor. Im späteren Verlauf wird eine Normalisierung durch Einführung der Tabelle weather_station vorgenommen. Dies ist auf die Erweiterung des Dateninhalts der einzelnen Stationen zurückzuführen. Die Stationen werden zusätzlich um die Daten Stationsname sowie Stationskoordinate ergänzt. Dadurch wird die Wiederverwendbarkeit der Daten sowie die Datenintegrität gefördert. Damit geht ein Foreign-Key-Constraint einher, der bei neuen Station IDs zu Fehlern führt. Ein Foreign-Key-Constraint ist eine Datenbankregel, die sicherstellt, dass ein Wert in einer Spalte nur dann eingetragen werden darf, wenn er in der referenzierten Tabelle bereits als Primärschlüssel existiert. In diesem Fall bedeutet dies, dass der Import der Klimadaten von neuen Stationen nicht erfolgen wird, wenn der DWD neue Stationen in sein Open-Data Angebot einbringt. Diese potenzielle Fehlerquelle wird in Kapitel 7.6 genauer beschrieben.

# 7 Erweiterungspotenziale und Weiterentwicklung

In diesem Kapitel werden für die zukünftige Weiterentwicklung des Portals Erweiterungspotenziale analysiert und vorgestellt. Zusätzlich wird der Einsatz des Portals auf anderen Systemen untersucht. Abschließend wird das System auf potenzielle Fehlerquellen durchsucht.

## 7.1 Erweiterte Datenquellen

Der derzeitige Stand des Portals basiert ausschließlich auf den täglichen Klimadaten des DWD, welche eine solide Grundlage für die Analyse langfristiger Entwicklungen wie Temperaturverläufe oder Niederschlagsmuster bieten. Für eine umfassendere Analyse und zur Abdeckung weiterer meteorologischer Fragestellungen können jedoch grundsätzlich weitere Datenquellen und -typen integriert werden.

Der DWD bietet über sein Open-Data-Portal zahlreiche weitere Datensätze, die über die bereits eingebundenen Tageswerte hinausgehen. Insbesondere könnten folgende Datenquellen eine sinnvolle Erweiterung darstellen.

- Messdaten in verschiedenen Zeitintervallen:
    - Zusätzlich zu den täglichen Klimadaten, gibt es Klimadaten im 1-, 5-, 10- und 60-Minuten-Intervall (Abbildung 29).
        - Die Daten im 1- und 5-Minuten-Intervall enthalten nur Niederschlagsdaten,
        - Die Daten im 10- und 60-Minuten-Intervall enthalten zusätzlich zu den Niederschlagsdaten unter anderem auch Lufttemperatur und Windgeschwindigkeit.


Abbildung 29: Verschiedene Messdaten des DWDs
(https://opendata.dwd.de/climate_environment/CDC/observations_germany/climate/)

Die CSV-Struktur der Messdaten ist auch bei verschiedenen Kategorien gleich. Eine Erweiterung zum Import der verschiedenen Kategorien ist mit vergleichsweise geringem Aufwand möglich. Hierzu müsste jedoch die Datenbank um die verschiedenen

Datenkategorien erweitert werden (z.B. Tabelle „weather_data_hourly“ für stündliche Klimadaten). Während dem Import muss beachtet werden, dass alle Datenkategorien die einen Zeitraum abbilden, welcher kleiner als „daily“ (täglich) ist, anders strukturiert sind. Die Klimaparameter (wie z.B. Temperatur, Niederschlag, etc.) werden nicht wie bei den täglichen Daten gebündelt in einer CSV-Datei ausgegeben, sondern je nach Klimaparameter in mehreren separierten CSV-Dateien. Wie in Abbildung 30 zu sehen ist, sind die verschiedenen Klimaparameter in verschiedenen Unterordnern unterteilt.

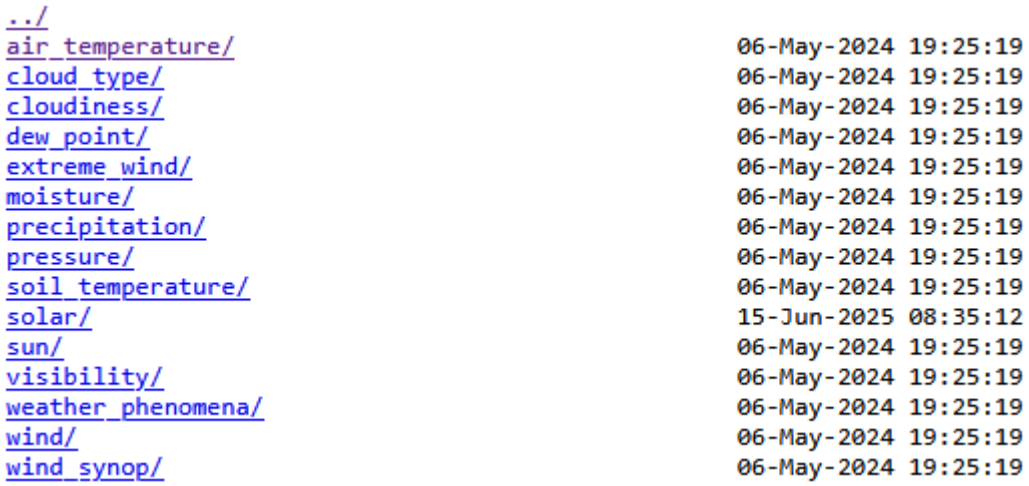



Abbildung 30: Klimaparameter Unterordner Stündlich
(https://opendata.dwd.de/climate_environment/CDC/observations_germany/climate/hourly/)

## 7.2 Verbesserte Visualisierung

Die derzeitige Visualisierung im Frontend ist funktional, aber gestalterisch und technisch bewusst dezent gehalten. Für eine breitere Zielgruppe sowie eine tiefere Analyse der Daten bestehen jedoch vielfältige Potenziale zur Erweiterung der grafischen Darstellung. Durch den gezielten Einsatz interaktiver, performanter und benutzerfreundlicher Visualisierungsformen lassen sich wissenschaftliche Erkenntnisse leichter erkennen. Beispielsweise kann, wie in Abbildung 25 zu sehen ist, durch einen optisch ansprechendes Temperaturverlaufsdiagramm der Trend der letzten Jahrzehnte besser und verständlicher erkannt werden.

### 7.2.1 Interaktive Zeitachsen und Diagrammtypen

Aktuell kommen einfache Liniendiagramme zum Einsatz, um etwa Temperaturverläufe über Jahre hinweg darzustellen. Folgende Visualisierungsfunktionen sind für zukünftige Erweiterungen und Verbesserungen möglich:

- Zoom- und Auswahlfunktionen: Nutzer könnten gezielt bestimmte Zeiträume auswählen (ohne neue Abfragen senden zu müssen).
- Heatmaps: Darstellung der Temperaturintensität oder Niederschlagsmengen über das Jahr verteilt.

- Boxplots: Zur Veranschaulichung von Extremwerten und Schwankungen über definierte Zeiträume hinweg (z. B. pro Monat oder Jahreszeit).
- Vergleichsdiagramme: Gegenüberstellung mehrerer Jahre in einem kombinierten Diagramm.

### 7.2.2 Kartenbasierte Visualisierung

Die bisherige Kartenansicht (Leaflet) zeigt lediglich Stationen als Marker und ermöglicht deren Auswahl. Mögliche Erweiterungen sind folgende:

- Kartenlayer mit Farbskalen: Darstellung von Temperaturabweichung, Niederschlagsintensität oder Windgeschwindigkeit als farbige Flächen in Anlehnung an das Beispiel von LoKlim in Abbildung 9 (Kapitel 3.1.3).
- Animation von Wetterverläufen über eine definierte Zeit: Darstellung von z. B. Niederschlagshöhe im Tages-, Monats- oder Jahresverlauf auf der Karte.

### 7.2.3 Erweiterte Benutzerinteraktion

Auch die Interaktivität kann gezielt erweitert werden:

- Tooltip-Erweiterungen: Zusatzinformationen bei Mouseover über die Diagramme zur erweiterten Dateneinsicht (z. B. Extremwerte, Quellenangaben, Monatsmittelwerte).
- Dynamische Einheitenumschaltung: In Hinsicht auf internationale Einheitensysteme sind für einige Klimadaten mehr als nur eine Einheit anzuzeigen. Beispielsweise ist zu beachten, dass Temperaturangaben in Grad Celsius oder Grad Fahrenheit oder Niederschlagshöhen im metrischen oder imperialen System erfolgen können. In diesem Fall sind die Ausgangswerte in die jeweils erwünschte Einheit zu konvertieren.

## 7.3 Benutzerkonten und API-Zugriffskontrolle

In der aktuellen Auslegung ist das Backend-API des Wetterportals ohne Einschränkung zugänglich. Dies würde jedoch bei öffentlichem und somit stetig wachsendem Zugriff potenziell zu Skalierungsproblemen, Missbrauch (z. B. durch Bots) und unnötiger Serverbelastung führen. Zur Vermeidung dieser unerwünschten Störungen, eignet sich die Einführung eines benutzerbasierten Zugriffssystems. In den nachfolgenden Unterkapiteln wird dieses Zugriffsystem, das Zugriffe strukturiert und steuert, durchleuchtet. Im ersten Schritt wird die Motivation dieses Systems angeführt.

### 7.3.1 Motivation für Zugriffsbeschränkung

Offene und nicht ausreichend abgesicherte Schnittstellen können, wenn sie keine Zugangskontrolle besitzen, Ziele von beispielsweise übermäßigen Anfragen (Flooding) durch Skripte oder automatisierte Tools werden. Weiterhin sind diese Schnittstellen der Gefahr ausgesetzt, dass unkontrollierte und missbräuchliche Downloadnutzung durch Dritte (z. B. für die eigene Plattformen) entstehen. Derartige missbräuchliche Nutzungen führen im Extremfall zu Performanceeinbußen bei regulären Nutzern, insbesondere bei großen Datenabfragen. Zur Sicherung der Systemintegrität empfiehlt sich daher eine tokenbasierte Zugriffskontrolle.

### 7.3.2 Limitierung und Monitoring von API-Calls

Zusätzlich zur Authentifizierung kann der API-Zugriff durch das sogenannte Rate Limiting eingeschränkt werden. Folgende Einschränkungen wären möglich:

- X Anfragen pro Minute/Stunde/Tag pro Benutzer-Token,
- Reaktion bei Überschreitung: HTTP 429 Too Many Requests,
- Technisch möglich mit Libraries wie Spring Buchet4j[75].

Zur Festlegung maximaler Anfragen innerhalb einer Zeitperiode (z.B. täglich) bietet sich die Einführung unterschiedlicher Nutzerrollen an. Tabelle 4 zeigt eine beispielhafte Einordnung in Nutzerrollen. In diesem Beispiel ist ein gebührenpflichtiger Premiumzugang möglich, durch den die Anzahl maximaler Anfragen seitens des Nutzers deutlich ansteigen darf.

| Nutzerrolle | Maximale Anfragen pro Tag |
|---|---|
| Gast (ohne Login) | 100 |
| Registriert | 1.000 |
| Premium (mit Gebühren) | 10.000 - 100.000 |
| Admin | Unbegrenzt |

Tabelle 4: Beispielhafte Nutzerrollen für API-Nutzung (Aslan, 2025)

### 7.3.3 Vor- und Nachteile

Ein Systementwurf, wie in Kapitel 7.3 vorgestellt, welches die Zugriffssteuerung auf die API definiert und je nach Benutzerrolle die Zahl der Anfragen an den Server limitiert, bietet sowohl für die Entwickler als auch für die Nutzer Vor- und Nachteile an.

[75] Spring Bucket4j ist eine Erweiterung für Java-/Spring-Anwendungen, mit der man Rate Limiting steuern kann, wie viele Anfragen ein Nutzer in einem bestimmten Zeitraum schicken darf.

Als ein Vorteil ist die Lastverteilung und Performancesicherung bei hoher Nutzung zu nennen. Weiterhin wird der Missbrauch von Dritten erschwert bzw. vermieden. Zusätzlich können für registrierte Nutzer erweiterte Funktionalitäten bereitgestellt werden, wie z.B. vordefinierte Suchanfragen. Nicht zuletzt generiert ein potenzieller kommerzieller Vertrieb Einnahmequellen, welche zur Weiterentwicklung und Wartung des Portals reinvestiert werden können.

An dieser Stelle sind ebenfalls Nachteile dieses Systementwurfs zu nennen. Die Einführung von Benutzerrollen birgt das Risiko, potenziell interessierte Nutzer abzuschrecken. Des Weiteren müssen durch Benutzerrollen persönliche Daten sowie Zahlungsdaten der Nutzer verarbeitet werden. Dies wiederum stellt im Sinne des Datenschutzes und der Datensicherheit eine Herausforderung dar. Hierbei sind bezüglich Daten rechtliche, technische sowie finanzielle Aspekte zu beachten. Ferner gehen die Implementierung und Wartung unterschiedlicher Benutzerrollen mit zusätzlichem Aufwand für die Entwickler einher.

## 7.4 Mobile und barrierefreie Nutzung

Die bisherige Umsetzung des Portals ist primär für die Nutzung auf stationären Rechnern (z.B. Heimcomputer) oder Notebooks ausgelegt, da diese in der Regel über eine Maussteuerung verfügen, die für die Nutzung des Portals unabdingbar ist. Für eine langfristige und leicht zugängliche Nutzung durch eine möglichst breite Zielgruppe, insbesondere außerhalb akademischer bzw. wissenschaftlicher Kreise, ist das Portal zusätzlich mobilfreundlich, responsiv und barrierefrei zu gestalten. Dies ist nicht zuletzt eine Frage der Usability, sondern sowie gesellschaftlicher Inklusion.

### 7.4.1 Mobile Optimierung und Responsive Design

Die aktuelle Oberfläche ist zwar technisch auf modernen Frameworks aufgebaut (React), allerdings nicht konsequent auf mobile Endgeräte ausgelegt. Eine Möglichkeit stellt hierzu das sogenannte Responsives Grid-System dar, welches eine an die Bildschirmgröße anpassbare Darstellung für Smartphones und Tablets automatisch bewerkstelligt. Zusätzlich sind bei diesem Vorhaben die Steuerung der Mobilgeräte über Touch zu beachten. Beispielsweise müssen größere Buttons implementiert werden, da sich die Steuerung von Mobilgeräten grundlegend zu der Steuerung von PCs unterscheidet.

### 7.4.2 Barrierefreiheit nach WCAG

Um die Barrierefreiheit des Portals zu gewährleisten, erfordert es Hilfsmittel. Ein barrierefreies Webportal muss die Anforderungen der Web Content Accessibility Guidelines (WCAG) erfüllen, insbesondere im Hinblick auf:[76]

- Tastatursteuerung: Alle Funktionen müssen mittels Tastatur bedienbar sein.
- Kontraste: Textfarben müssen sich deutlich vom Hintergrund abheben.
- Beschreibungen: Grafiken und Diagramme müssen mit Alternativtexte versehen werden.
- Screenreader-Kompatibilität: Bildschirminhalte müssen durch ein Screenreader-Programme lesbar sein oder in Brailleschrift ausgeben.
- Keine rein visuelle Kodierung: Farbunterschiede müssen durch Symbole oder Beschriftungen ergänzt werden.

### 7.4.3 Zielgruppen und Relevanz

Die Inhalte des Unterkapitels 7.4.1 und 7.4.2 sind von hoher Bedeutung, da sowohl barrierefreie als auch die mobile Nutzung nicht nur technisch notwendig ist, sondern auch gesellschaftlich relevant. Denn durch diese zwei Eigenschaften wird eine viel breitere Masse in der Gesellschaft angesprochen, die sich über den Klimawandel in Ihrer Lokalität informieren möchten. Diese sind zum Beispiel:

- Ältere Personen, die Informationen über ihre lokale Wetterentwicklung abrufen,
- Schüler und sonstige am Thema interessierte Nutzer, die mit Tablets arbeiten,
- Menschen mit Einschränkungen, die auf assistive Technologien angewiesen sind.

## 7.5 Einsatz auf anderen Systemen

Ursprünglich sind zu Beginn der Konzeption und Entwicklung des Portals der Einsatz über Docker[77] geplant, um die Anwendung auf verschiedenen Systemen problemlos zu ermöglichen. Erste Schritte zum Einsatz mit Docker werden durchgeführt, führen jedoch nicht zur erfolgreichen Umsetzung. Eine Fehlerursache ist das Importieren der Datenbanktabellen, da diese nicht vollständig importiert werden. Fehlermeldungen

[76] vgl. Bundesregierung (2025, online)

[77] Docker ist eine Plattform zur Containerisierung, mit der Anwendungen mitsamt ihren Abhängigkeiten in isolierten, leichtgewichtigen Containern ausgeführt werden können, unabhängig von der zugrunde liegenden Systemumgebung (https://www.docker.com/).

oder Logeinträge geben keinen Hinweis darauf, aus welchem Grund die Tabellen nicht vollständig importiert werden. Auf Grund des Foreign-Key-Constraints zwischen den Tabellen weather_data und weather_station führt ein unvollständiger Datenimport dazu, dass die Datenbank und die zusammenhängende Backend nicht funktionieren. Es wird angenommen, dass eine weitere Fehlerquelle in der Befüllung der weather_station Tabelle vorliegt, da keine automatisierte Funktion für die Befüllung existiert. Diese potenzielle Fehlerquelle wird im nachfolgendem Unterkapitel detaillierter behandelt.

## 7.6 Anfälligkeit des Foreign-Key-Constraints

Eine Schwachstelle in der aktuellen Systemarchitektur ist die Verarbeitung und Pflege der Tabelle weather_station, welche die Metadaten der Wetterstationen beinhaltet. Diese Tabelle wird initial über ein einfaches Skript generiert, das eine lokal gespeicherte JSON-Datei einliest und die enthaltenen Stationseinträge in die Datenbank überträgt.

Da in der Datenbank ein Foreign-Key-Constraint zwischen weather_station und der Haupttabelle weather_data besteht, dürfen in weather_data nur solche station_id-Werte vorkommen, die ebenfalls identisch in weather_station existieren. Wird jedoch im Zuge des Datenimports eine neue station_id erkannt, die noch nicht in weather_station hinterlegt ist, führt dies zu einem Integritätsfehler und der Importvorgang schlägt fehl.

Um diese potenzielle Schwachstelle zu beheben, ist die Implementierung einer automatisierten Prüf- und Aktualisierungslogik von Nöten, die regelmäßig die Stations-Beschreibungsquelle des DWD[78] auf neue Einträge untersucht. Demnach werden neue station_ids bei Bedarf automatisch in die Tabelle weather_station übernommen.

Die Beschreibung der Wetterstationen wird vom DWD in einem veralteten „Fixed-Width“[79]-Format festgehalten (Abbildung 31). Bei einer potenziellen Aktualisierungslogik muss dies beachtet werden, da es sich nicht um ein klassisches CSV-Format handelt.

[78] vgl. Deutscher Wetterdienst (Jahr unbekannt, online) (4)

[79] Ein Fixed Width Format ist ein Textformat, bei dem jede Spalte eine feste Zeichenlänge hat, statt Trennzeichen wie Kommas oder Semikolon, was vor allem bei alten Datenquellen oder Messdaten häufig verwendet wird.

| Stations_id | von_datum | bis_datum | Stationshoehe | geoBreite | geoLaenge | Stationsname | Bundesland | Abgabe |
|---|---|---|---|---|---|---|---|---|
| 00001 | 19370101 | 19860630 | 478 | 47.8413 | 8.8493 | Aach | Baden-Württemberg | Frei |
| 00003 | 18910101 | 20110331 | 202 | 50.7827 | 6.0941 | Aachen | Nordrhein-Westfalen | Frei |
| 00011 | 19800901 | 20250622 | 680 | 47.9736 | 8.5205 | Donaueschingen (Landeplatz) | Baden-Württemberg | Frei |
| 00044 | 19690101 | 20250622 | 44 | 52.9336 | 8.2370 | Großenkneten | Niedersachsen | Frei |
| 00052 | 19690101 | 20011231 | 46 | 53.6623 | 10.1990 | Ahrensburg-Wulfsdorf | Schleswig-Holstein | Frei |
| 00061 | 19750701 | 19780831 | 339 | 48.8443 | 12.6171 | Aiterhofen | Bayern | Frei |
| 00070 | 19730601 | 19860930 | 712 | 48.2052 | 9.0371 | Albstadt-Ebingen | Baden-Württemberg | Frei |
| 00071 | 19861101 | 20191231 | 759 | 48.2156 | 8.9784 | Albstadt-Badkap | Baden-Württemberg | Frei |
| 00072 | 19780901 | 19950531 | 794 | 48.2766 | 9.0001 | Albstadt-Onstmettingen | Baden-Württemberg | Frei |
| 00073 | 19590301 | 20250622 | 374 | 48.6183 | 13.0620 | Aldersbach-Kramersepp | Bayern | Frei |
| 00078 | 19610101 | 20250622 | 64 | 52.4853 | 7.9125 | Alfhausen | Niedersachsen | Frei |
| 00090 | 19880219 | 20250622 | 305 | 50.7557 | 9.2583 | Alsfeld | Hessen | Frei |
| 00091 | 19781101 | 20250622 | 304 | 50.7446 | 9.3450 | Alsfeld-Eifa | Hessen | Frei |
| 00093 | 19771201 | 19781031 | 269 | 50.7691 | 9.2542 | Alsfeld-Reibertenrod | Hessen | Frei |
| 00096 | 20190409 | 20250622 | 50 | 52.9437 | 12.8518 | Neuruppin-Alt Ruppin | Brandenburg | Frei |
| 00098 | 18870215 | 19541231 | 780 | 51.1890 | 8.4671 | Altastenberg | Nordrhein-Westfalen | Frei |
| 00102 | 19980101 | 20250622 | 0 | 53.8633 | 8.1275 | Leuchtturm Alte Weser | Niedersachsen | Frei |
| 00106 | 19710101 | 20061201 | 500 | 51.7976 | 10.4429 | Altenau | Niedersachsen | |
| 00116 | 18990601 | 19730531 | 213 | 50.9833 | 12.4333 | Altenburg | Thüringen | Frei |
| 00125 | 19740301 | 20250622 | 739 | 47.8342 | 10.8667 | Altenstadt | Bayern | Frei |
| 00126 | 19791101 | 20101130 | 330 | 49.5447 | 10.2213 | Uffenheim (Schulstr.) | Bayern | Frei |
| 00129 | 19910101 | 20061231 | 30 | 53.6893 | 13.2420 | Altentreptow | Mecklenburg-Vorpommern | Frei |
| 00131 | 20041101 | 20250622 | 296 | 51.0881 | 12.9326 | Geringswalde-Altgeringswalde | Sachsen | Frei |
| 00132 | 19531211 | 19870930 | 750 | 49.7725 | 12.3891 | Altglashütte | Bayern | Frei |
| 00142 | 19550101 | 20250622 | 511 | 48.4060 | 11.3117 | Altomünster-Maisbrunn | Bayern | Frei |
| 00150 | 19510101 | 20250622 | 215 | 49.7273 | 8.1164 | Alzey | Rheinland-Pfalz | Frei |
| 00151 | 19470101 | 20250622 | 382 | 49.4691 | 11.8546 | Amberg-Unterammersricht | Bayern | Frei |
| 00154 | 19940101 | 20250622 | 516 | 48.0197 | 12.2925 | Amerang-Pfaffing | Bayern | Frei |
| 00161 | 19801201 | 20250622 | 75 | 50.4237 | 7.4202 | Andernach | Rheinland-Pfalz | Frei |
| 00164 | 19080517 | 20250622 | 54 | 53.0316 | 13.9908 | Angermünde | Brandenburg | Frei |
| 00167 | 20040901 | 20250622 | 11 | 53.8409 | 13.6854 | Anklam | Mecklenburg-Vorpommern | Frei |
| 00169 | 19470101 | 19661231 | 630 | 50.5730 | 13.0053 | Annaberg-Buchholz | Sachsen | Frei |
| 00172 | 19690501 | 19911130 | 395 | 52.2486 | 9.5056 | Annaturm | Niedersachsen | Frei |
| 00174 | 19840301 | 19871231 | 250 | 49.1886 | 7.9766 | Annweiler-Bindersbach | Rheinland-Pfalz | Frei |
| 00175 | 19520101 | 19760430 | 413 | 49.2964 | 10.5751 | Ansbach | Bayern | Frei |

Abbildung 31: Stationsbeschreibungen vom DWD (https://opendata.dwd.de/climate_environment/CDC/observations_germany/climate/daily/kl/recent/KL_Tageswerte_Beschreibung_Stationen.txt)

# 8 Fazit und Ausblick

Das letzte Kapitel der vorliegenden Bachelorarbeit dient zur Wiedergabe des aus der Forschungsfragen resultierenden Fazits. Weiterhin wird ein Ausblick.

Inhaltlich und technisch wird im Rahmen dieser Bachelorarbeit ein funktionsfähiges Klima- und Wetterinformationsportal konzipiert und implementiert. Die zentrale Zielsetzung, meteorologische Daten für eine breite Öffentlichkeit verständlich aufzubereiten, wird erreicht. Das Portal ermöglicht den Nutzern, historische Klimadaten ihrer ausgewählten Region einzusehen und die Klimatrends der letzten Jahre bzw. Jahrzehnte aufzuzeigen.

Bei der Umsetzung kommen moderne Webtechnologien und offene Datenquellen zum Einsatz. Im Backend verarbeitet ein automatisierter Importservice täglich die Klimadaten des Deutschen Wetterdienstes und speichert sie in einer PostgreSQL-Datenbank mit über 18 Millionen Einträgen ab. Dank einer normalisierten Datenbankstruktur und Optimierungen, wie Duplikatvermeidung, werden auch umfangreiche Abfragen performant beantwortet. So liegen etwa die Antwortzeiten für Abfragen von hunderttausenden Datensätzen im Bereich weniger Sekunden.

Auch die Frontend-Komponente erfüllt die Anforderungen an Interaktivität und Usability. Die interaktive Kartenkomponente Leaflet erleichtert die Auswahl der nächstgelegenen Wetterstation per Mausklick auf der Landkarte. Dabei ist es nicht von Bedeutung, ob dem Nutzer die spezifische Station ID bekannt ist. Klimazeitreihen werden mittels interaktiver Grafiken (Recharts) visualisiert und durch tabellarische Daten ergänzt. Die Filterfunktion erlaubt zudem gezielte Abfragen nach Zeitraum, Ort und Wetterparameter. Durch die Verwendung von Virtualisierungstechniken (z. B. react-window) ist die durchgehend responsive Darstellung von selbst großen Datenmengen im Browser gewährleistet. Insgesamt stellt das Portal eine robuste sowie ganzheitliche Lösung aus Backend, Frontend und Datenbank dar. Das Portal bewerkstelligt erfolgreich die Verbindung von komplexen Klimadaten mit einer benutzerfreundlichen Aufbereitung.

Trotz dieser Erfolge weist die Lösung einige Einschränkungen in methodischer, technischer und gestalterischer Hinsicht auf. Aufgrund der zeitlichen Beschränkung der Bearbeitungsdauer und der Fokussierung auf Kernfunktionen des Portals bleiben bestimmte Aspekte unausgereift. Daher ist die Benutzeroberfläche des Portals dezent gehalten. Sie erfüllt die Grundanforderungen (Kartenansicht, Diagramme und Filter), bietet aber keine Nutzerführung oder interaktive Hilfestellungen an. Fortgeschrittene UI-Elemente wie Animationen oder visuelle Rückmeldungen bei Nutzeraktionen sind nicht Bestandteil dieser Arbeit und werden daher nicht programmiert. Dies gilt ebenso für ein konsequent responsives Design auf Mobilgeräten. Auch Aspekte der Barrierefreiheit (z. B. Tastaturbedienung, Screenreader-Unterstützung, etc.) werden nicht berücksichtigt.

Hinsichtlich der Methodik ist zudem anzumerken, dass keine systematische Evaluation der Anwendung mit Endnutzern stattfindet. Unter der Voraussetzung, dass dem Verfasser deutlich mehr Zeit für die Evaluation gegeben wird, sind Nutzertests und Feedbackschleifen theoretisch möglich, um die Verständlichkeit und Wirksamkeit des Portals zu validieren. Dieser Punkt eignet sich für zukünftige studentische Arbeiten als eine potenzielle Forschungsfrage. Sowohl inhaltlich als auch technisch beschränkt sich der Prototyp auf die Verarbeitung historischer Klimadaten (Tageswerte) des DWD. Echtzeitdaten, höhere zeitliche Auflösungen oder weitere Klimavariablen (wie z.B. Luftschadstoffe) werden nicht integriert. Eine Prognose zukünftiger Klimaentwicklungen, etwa basierend auf Klimamodellen, ist nicht Gegenstand dieser Arbeit.

Darüber hinaus erfolgt die Nutzung des Portals derweilen ausschließlich in einer lokalen Umgebung. Ein skalierbarer Produktivbetrieb (z. B. in der Cloud oder via Containerisierung) wird durch den begrenzten Rahmen der vorliegenden Arbeit nicht realisiert. Der geplante Einsatz von Docker bleibt unvollständig, was beim Import großer Datentabellen zu technischen Problemen führt. Es ist nicht möglich, Stresstests, die unter realistischen Lastszenarien (mit vielen gleichzeitigen Nutzern), durchzuführen. Daher ist die Performance unter Produktionsbedingungen empirisch unbestätigt. Diese Limitierungen stellen jedoch keine grundlegenden Mängel des Konzepts dar, sondern spiegeln in erster Linie die notwendige Priorisierung in der vorliegenden Bachelorarbeit wider. Die aktuelle Lösung bildet ein solides Fundament, auf dem zukünftige Erweiterungen gezielt aufbauen können.

Eine weitere Forschungsfrage künftiger studentischer Arbeiten stellt die Verbesserung der Visualisierung und der Nutzerinteraktion dar. Die aktuelle Darstellung ist zwar funktional, bietet aber Raum für attraktivere und aussagekräftigere Visualisierungsmethoden. Interaktive Zeitreihendiagramme mit Zoom- und Filtermöglichkeit, Heatmaps zur Veranschaulichung jahreszeitlicher Muster oder Boxplots zur Darstellung von Schwankungsbreiten und Extremwerten bereichern die Datenanalyse deutlich. Auch die Kartenansicht lässt sich gegebenenfalls ausbauen, indem Wetterinformationen als Overlay direkt auf der Karte visualisiert werden. Beispiele hierzu sind farbcodierte Temperaturverteilungen oder Niederschlagsverläufe.

Auch auf technischer Ebene bieten sich Potenziale für zukünftige Forschungsfragen an, um Verbesserungen am Portal durchzuführen. Insbesondere ist die Einführung einer Benutzerverwaltung mit Zugriffskontrolle unabdingbar, um bei steigendem Nutzungsaufkommen Missbrauch vorzubeugen und eine geregelte API-Nutzung zu gewährleisten. Durch ein Authentifizierungskonzept lassen sich API-Aufrufe begrenzen sowie unterschiedliche Nutzerrollen mit abgestuften Rechten etablieren.

Ein vollständig responsives Design, touch-optimierte Bedienelemente und die Einhaltung der WCAG-Richtlinien würden die Reichweite und Benutzerfreundlichkeit des Portals weiter erhöhen. Darüber hinaus sollte der Bereitstellungsprozess professionalisiert werden. Insbesondere ist der konsequente Einsatz von Container-Technologien (z. B. Docker) umzusetzen, um einen reibungslosen Betrieb auf verschiedenen Systemumgebungen sicherzustellen und das Aufsetzen des Portals in ein neues System zu vereinfachen.

Nicht zuletzt wird, wie bereits erwähnt, empfohlen, die Stabilität und Effektivität der Lösung im Rahmen eines Pilotbetriebs oder durch umfangreiche Tests unter realen

Bedingungen zu validieren. Langfristig könnte das Portal in Kooperation mit Forschungseinrichtungen oder Behörden weiterentwickelt werden, um es als offizielles Informationswerkzeug im Bereich Klima und Wetter zu etablieren.

Das Portal ist für den Standort Deutschland konzipiert. Die Standardisierung der Klimarohdaten durch den DWD haben geholfen, eine konsistente Datenstruktur und -grundlage aufzubauen. Ein mögliches Vorhaben ist die Erweiterung des Portals auf Europa oder auf die ganze Welt. Dies ist jedoch mit großen Hürden verbunden, da sogar innerhalb Europas die Rohdaten nicht einheitlich strukturiert sind. Daher ist auf internationaler Ebene eine Standardisierung der Klimarohdaten von Nöten. Antreiber dieses Vorhabens könnten Bildungseinrichtungen oder Behörden sein.

Insgesamt bietet die vorliegende Arbeit eine tragfähige Grundlage, auf der zukünftige Projekte aufbauen können, um die Lücke zwischen komplexen Klimadaten und der breiten Öffentlichkeit weiter zu schließen. Es ist nicht zu vergessen, dass der Klimawandel immer präsenter, spürbarer und auswirkungsvoller in allen möglichen Regionen Deutschlands ist. Der Menschheit steht nur diese eine Erde zur Verfügung. Umso wichtiger ist es, dass sich jeder mit dem Klimawandel befasst und im Rahmen seiner Möglichkeiten etwas dagegen unternimmt.

## Anhang A: Code-Dokumentation Klima-/Wetterinformations-Portal

# Code-Dokumentation

# Klima-/Wetterinformations-Portal

Von: Bora Aslan
Matrikelnummer: 1002700
E-Mail: 02asbo1bil@hft-stuttgart.de

# WetterDbEntity.java

```java
@Entity
@Table(name = "weather_data")
public class WetterDbEntity {



    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "station_id")  2 usages
    private Integer stationId;

   // @JsonIgnoreProperties({"weatherData"})
    @ManyToOne(fetch = FetchType.LAZY)  8 usages
    @JoinColumn(name = "station_id", referencedColumnName = "station_id", insertable = false, updatable = false)
    @JsonIgnore
    private WeatherStation station;

    @Column(name = "date")  2 usages
    private LocalDate date;

    @Column(name = "temperature")  2 usages
    private Double temperature;

    @Column(name = "precipitation")  2 usages
    private Double precipitation;

    @Column(name = "sunshine")  2 usages
    private Double sunshine;

    @Column(name = "wind_speed")  2 usages
    private Double windSpeed;

    @Column(name = "qualityi")  2 usages
    private Double qualityI;

    @Column(name = "wind_max")  2 usages
    private Double windMax;

    @Column(name = "qualityii")  2 usages
    private Double qualityII;

    @Column(name = "prec_form")  2 usages
    private Double precForm;

    @Column(name = "snow")  2 usages
    private Double snow;
```

Abbildung A-1: WetterDbEntity Variablen

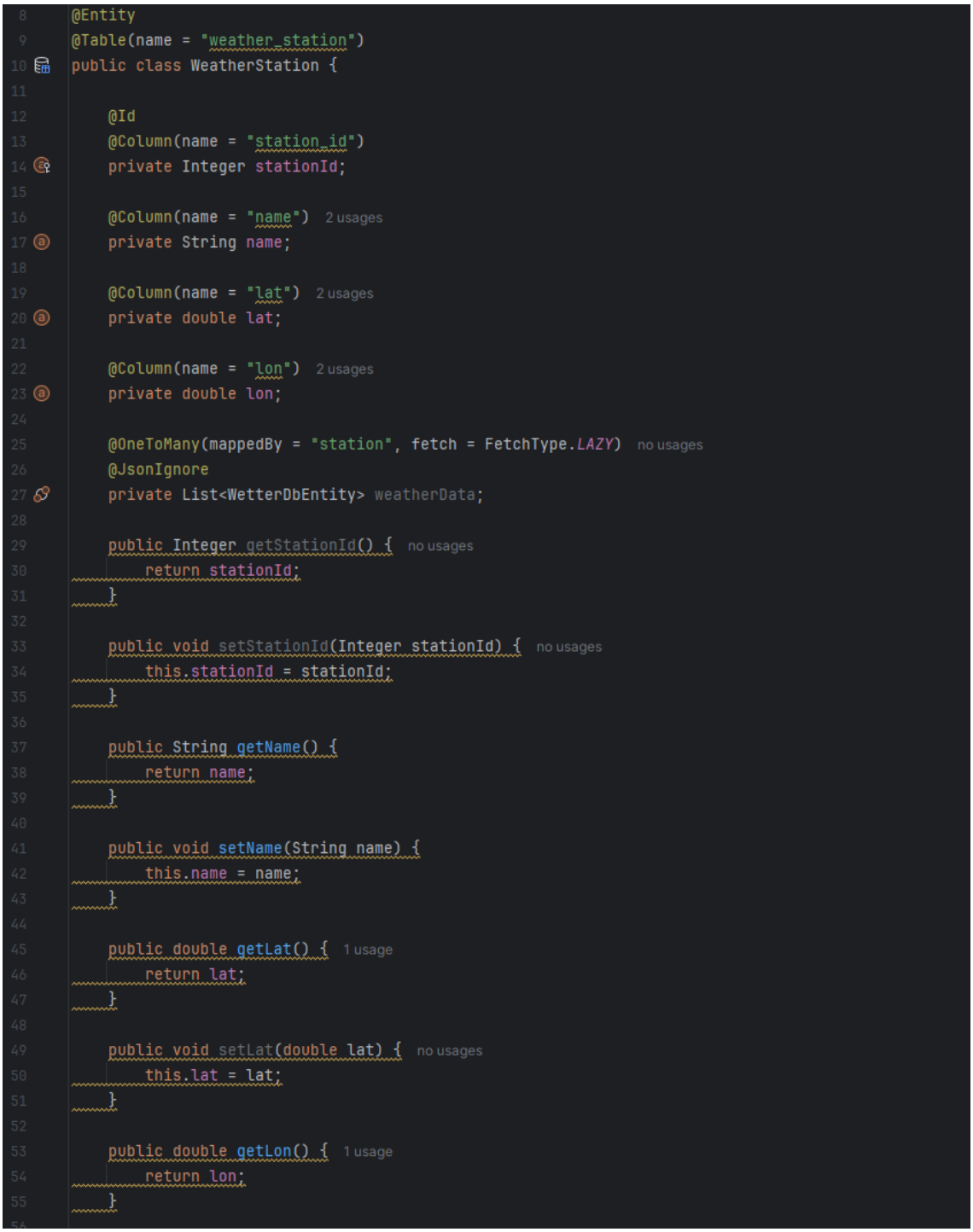
```
78
79      // Getters and setters
80
81      public void setId(Long id) {this.id = id;}  1 usage
82
83      public Long getId() { return id; }
86
87      public Integer getStationId() { return stationId; }
90
91      public void setStationId(Integer stationId) { this.stationId = stationId; }
94
95      public LocalDate getDate() { return date; }
98
99      public void setDate(LocalDate date) { this.date = date; }
102
103     public Double getTemperature() { return temperature; }
106
107     public void setTemperature(Double temperature) { this.temperature = temperature; }
110
111     public Double getPrecipitation() { return precipitation; }
114
115     public void setPrecipitation(Double precipitation) { this.precipitation = precipitation; }
118
119     public Double getSunshine() { return sunshine; }
122
123     public void setSunshine(Double sunshine) { this.sunshine = sunshine; }
126
127     public Double getWindSpeed() { return windSpeed; }
130
131     public void setWindSpeed(Double windSpeed) { this.windSpeed = windSpeed; }
134
135     public Double getQualityI() { return qualityI; }
138
139     public void setQualityI(Double qualityI) { this.qualityI = qualityI; }
142
143     public Double getWindMax() { return windMax; }
146
147     public void setWindMax(Double windMax) { this.windMax = windMax; }
150
151     public Double getQualityII() { return qualityII; }
154
155     public void setQualityII(Double qualityII) { this.qualityII = qualityII; }
158
159     public Double getPrecForm() { return precForm; }
```

Abbildung A-2: WetterDbEntity Getter/Setter

- WetterDbEntity bildet die „weather_data“-Tabelle in der Datenbank ab.
- Alle Variablen werden zusätzlich mit der Annotation „@Column(name = X)“ versehen, um die einzelnen Variablen den zugehörigen Spalten in der DB-Tabelle zuzuordnen.
    - Es wird Double anstatt double verwendet, da im DWD-Datensatz NULL-Werte vorkommen.
- Station ID wird als Fremdschlüssel gekennzeichnet.

- Die Fremdschlüsselkennzeichnung wird mit @JsonIgnore versehen, damit sie nicht in JSON-Antworten zurückgegeben wird.

# WeatherStation.java

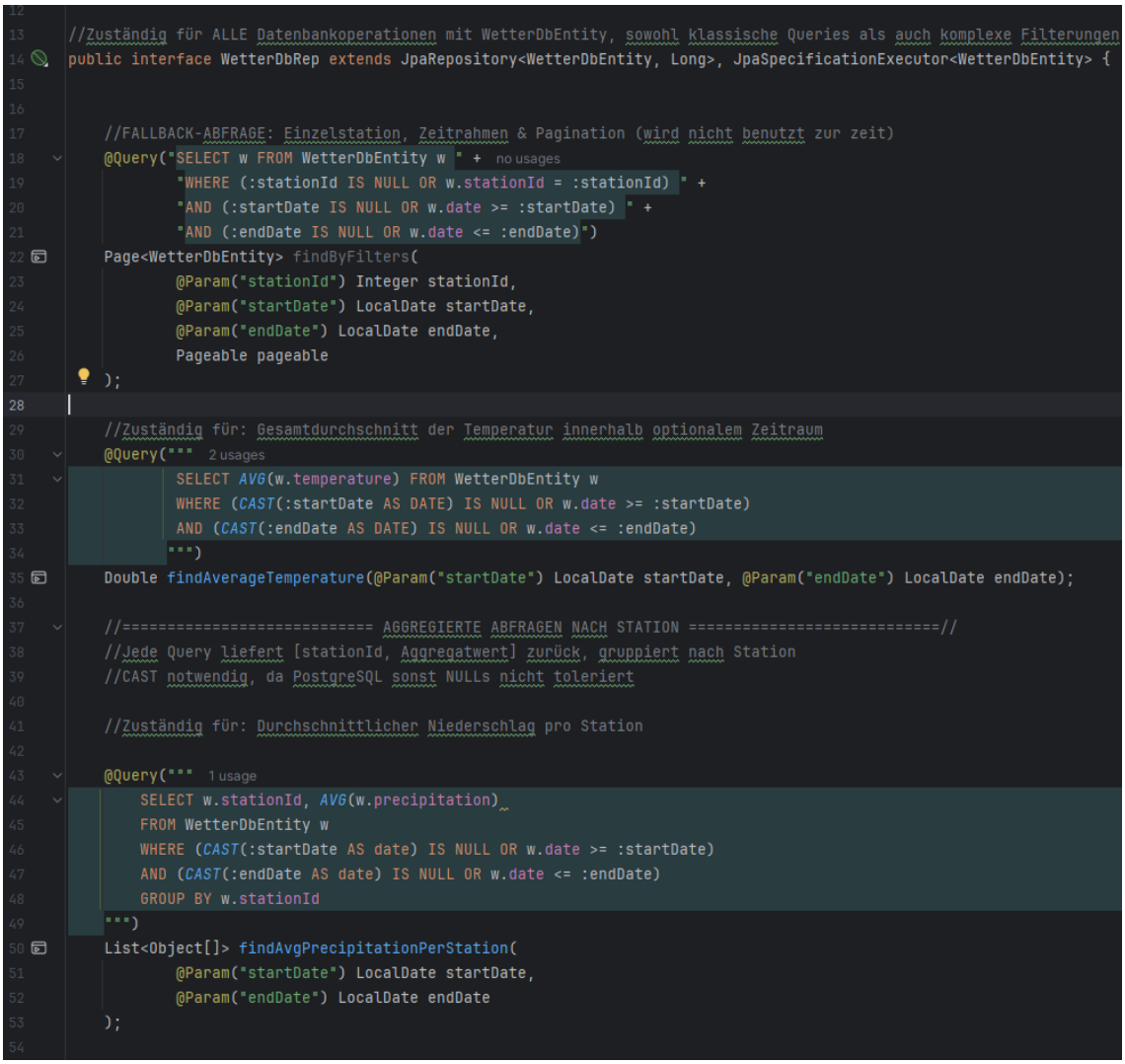


```java
@Entity
@Table(name = "weather_station")
public class WeatherStation {

    @Id
    @Column(name = "station_id")
    private Integer stationId;

    @Column(name = "name")  2 usages
    private String name;

    @Column(name = "lat")  2 usages
    private double lat;

    @Column(name = "lon")  2 usages
    private double lon;

    @OneToMany(mappedBy = "station", fetch = FetchType.LAZY)  no usages
    @JsonIgnore
    private List<WetterDbEntity> weatherData;

    public Integer getStationId() {  no usages
        return stationId;
    }

    public void setStationId(Integer stationId) {  no usages
        this.stationId = stationId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getLat() {  1 usage
        return lat;
    }

    public void setLat(double lat) {  no usages
        this.lat = lat;
    }

    public double getLon() {  1 usage
        return lon;
    }
```

Abbildung A-3: WeatherStation Variablen und Getter/Setter

- Abbildung der weather_station DB-Tabelle.
- Sowohl bei WetterDbEntity als auch hier wird die Liste mit LAZY Fetching verarbeitet, aufgrund von Performanceproblemen bei EAGER.

# WetterDBRep.java



```java
//Zuständig für ALLE Datenbankoperationen mit WetterDbEntity, sowohl klassische Queries als auch komplexe Filterungen
public interface WetterDbRep extends JpaRepository<WetterDbEntity, Long>, JpaSpecificationExecutor<WetterDbEntity> {



    //FALLBACK-ABFRAGE: Einzelstation, Zeitrahmen & Pagination (wird nicht benutzt zur zeit)
    @Query("SELECT w FROM WetterDbEntity w " +  no usages
            "WHERE (:stationId IS NULL OR w.stationId = :stationId) " +
            "AND (:startDate IS NULL OR w.date >= :startDate) " +
            "AND (:endDate IS NULL OR w.date <= :endDate)")
    Page<WetterDbEntity> findByFilters(
            @Param("stationId") Integer stationId,
            @Param("startDate") LocalDate startDate,
            @Param("endDate") LocalDate endDate,
            Pageable pageable
    );

    //Zuständig für: Gesamtdurchschnitt der Temperatur innerhalb optionalem Zeitraum
    @Query("""  2 usages
            SELECT AVG(w.temperature) FROM WetterDbEntity w
            WHERE (CAST(:startDate AS DATE) IS NULL OR w.date >= :startDate)
            AND (CAST(:endDate AS DATE) IS NULL OR w.date <= :endDate)
           """)
    Double findAverageTemperature(@Param("startDate") LocalDate startDate, @Param("endDate") LocalDate endDate);

    //========================== AGGREGIERTE ABFRAGEN NACH STATION ==========================//
    //Jede Query liefert [stationId, Aggregatwert] zurück, gruppiert nach Station
    //CAST notwendig, da PostgreSQL sonst NULLs nicht toleriert

    //Zuständig für: Durchschnittlicher Niederschlag pro Station

    @Query("""  1 usage
        SELECT w.stationId, AVG(w.precipitation)
        FROM WetterDbEntity w
        WHERE (CAST(:startDate AS date) IS NULL OR w.date >= :startDate)
        AND (CAST(:endDate AS date) IS NULL OR w.date <= :endDate)
        GROUP BY w.stationId
    """)
    List<Object[]> findAvgPrecipitationPerStation(
            @Param("startDate") LocalDate startDate,
            @Param("endDate") LocalDate endDate
    );

```

Abbildung A-4: WetterDBRep Einblick

- Besteht aus manuellen SQL-Queries, die abgerufen werden können.
- Standardmethoden wie findAll(), save() etc. sind automatisch über JpaRepository<WetterDbEntity, Long> verfügbar.
- Beachte die CAST-Funktion beim Datum, da es sonst zu Fehlern kommt, weil PostgreSQL keine NULL-Werte bei Datum akzeptiert (z. B. wenn der Nutzer kein Datum bei findAvgPrecipitationPerStation eingibt.
- Anfragen, die nach Station sortiert werden, wie findAvgPrecipitationPerStation, benötigen keine Pageable-Funktion, da die Anzahl an Stationen mit ca. 1300 relativ gering ist.

# WetterDbSpecs.java

```java
//Zuständig für Filterkriterien (also Specifications), welche dynamisch auf die JPA-Queries angewendet werden
//Ermöglicht flexible WHERE-Klauseln auf der Datenbankseite, ohne dass manuell SQL geschrieben werden muss

public class WetterDbSpecs {  16 usages

    //Zuständig für Filterung nach EINER Station ID (wird aktuell nicht mehr verwendet)
    //DEPRACTED? später wird nur noch mit Listen gearbeitet (mehrere Stationen auf einmal)

    public static Specification<WetterDbEntity> hasStationId(Integer stationId) {  no usages
    return ( Root<WetterDbEntity> root,  CriteriaQuery<?> query,  CriteriaBuilder cb) -> stationId == null ? null : cb.equal(root
}



    //Zuständig für Filterung nach MEHREREN Station IDs (z.B. alle Stationen einer Stadt)

    public static Specification<WetterDbEntity> hasStationIds(List<Integer> stationIds) {  1 usage
        return ( Root<WetterDbEntity> root,  CriteriaQuery<?> query,  CriteriaBuilder cb) -> {
            if (stationIds == null || stationIds.isEmpty()) return null;
            return root.get("stationId").in(stationIds);
        };
    }

    //Zuständig für Datumsfilter --> nur Einträge ab einem bestimmten Datum

    public static Specification<WetterDbEntity> hasDateAfter(LocalDate startDate) {  1 usage
        return ( Root<WetterDbEntity> root,  CriteriaQuery<?> query,  CriteriaBuilder cb) -> startDate == null ? null : cb.greate
    }

    //Zuständig für Datumsfilter --< nur Einträge bis zu einem bestimmten Datum

    public static Specification<WetterDbEntity> hasDateBefore(LocalDate endDate) {  1 usage
        return ( Root<WetterDbEntity> root,  CriteriaQuery<?> query,  CriteriaBuilder cb) -> endDate == null ? null : cb.lessThan
    }

    //=====================================================================
    // AB HIER GENERISCHE METHODE FÜR ZAHLENBEREICHE (z.B. Temperatur, Niederschlag usw.)
    //=====================================================================

    //Generische Methode → kann für jedes numerische Feld verwendet werden
    //Wird intern von allen `XYZBetween(...)` Methoden verwendet
```

Abbildung A-5: WetterDbSpecs Einblick

```java
    //Wird von jedem Zahlenbasiertem Parameter aufgerufen um minimal und oder maximale werte aufzurufen
    private static <T extends Number & Comparable<T>> Specification<WetterDbEntity> isInRange(  12 usages
            String field, T min, T max
    ) {
        return ( Root<WetterDbEntity> root,  CriteriaQuery<?> query,  CriteriaBuilder cb) -> {
            if (min != null && max != null) {
                //Beispiel: temperature BETWEEN 10 AND 20
                return cb.between(root.get(field), min, max);
            } else if (min != null) {
                //Beispiel: temperature >= 10
                return cb.greaterThanOrEqualTo(root.get(field), min);
            } else if (max != null) {
                //Beispiel: temperature <= 20
                return cb.lessThanOrEqualTo(root.get(field), max);
            } else {
                //Kein Filter => return null
                return null;
            }
        };
    }
    //Zuständig für Temperaturbereich-Filter (Durchschnittstemperatur)
    public static Specification<WetterDbEntity> temperatureBetween(Double min, Double max) {  1 usage
        return isInRange( field: "temperature", min, max);
    }

    //Zuständig für Niederschlagsbereich-Filter
    public static Specification<WetterDbEntity> precipitationBetween(Double min, Double max) {  1 usage
        return isInRange( field: "precipitation", min, max);
    }
    //Zuständig für Sonnenstundenbereich-Filter
    public static Specification<WetterDbEntity> sunshineBetween(Double min, Double max) {  1 usage
        return isInRange( field: "sunshine", min, max);
    }
    //Zuständig für Windgeschwindigkeit-Filter
    public static Specification<WetterDbEntity> windSpeedBetween(Double min, Double max) {  1 usage
        return isInRange( field: "windSpeed", min, max);
    }
    //Zuständig für Schneehöhenbereich-Filter
    public static Specification<WetterDbEntity> snowBetween(Double min, Double max) {  1 usage
        return isInRange( field: "snow", min, max);
    }
    //Zuständig für Schneebedeckungsgrad (z.B. 0-100%)
    public static Specification<WetterDbEntity> coveringBetween(Double min, Double max) {  1 usage
        return isInRange( field: "covering", min, max);
    }
```

Abbildung A-6: WetterDbSpecs Bereichsfunktionen

- Wird nur für den Standardfilter verwendet (also /api/weather/filter).
- Statt fixer Abfragen werden nur Bedingungen implementiert (z. B. wird hasStationIds aufgerufen, falls eine Station-ID angegeben wurde) und nur dann angewendet, wenn der entsprechende Wert übergeben wird.
- Ermöglicht die flexible Nutzung des Standardfilters, ohne alle Parameter angeben zu müssen, und bietet zusätzlich die „inRange()“-Funktion, welche es erlaubt, Zahlenwerte (z. B. bei Temperatur) als Minimal- und/oder Maximalwert für die Datensätze festzulegen.

# ImportedFile.java und ImportedFileRepository.java

```java
@Entity
public class ImportedFile {

    @Id
    private String filename;

    public ImportedFile() {}

    public ImportedFile(String filename) {  2 usages
        this.filename = filename;
    }

    public String getFilename() {  no usages
        return filename;
    }

    public void setFilename(String filename) {  no usages
        this.filename = filename;
    }
}
```

Abbildung A-7: ImportedFile.java Einblick

```java
public interface ImportedFileRepository extends JpaRepository<ImportedFile, String> {  1 usage
    boolean existsByFilename(String filename);  1 usage
}
```

Abbildung A-8: ImportedFileRepository.java Einblick

- ImportedFile.java bildet die imported_file DB-Tabelle ab.
- Speichert die Dateinamen der historischen Daten in der Tabelle ab.
- existsBy ist eine Methode des JpaRepository (wird von Spring generiert), weshalb keine explizite Definition erforderlich ist, auch wenn sie nicht direkt deklariert wurde.

# WetterDWDImporter.java

```java
@Component  1 usage
public class WetterDWDImporter {

    @Autowired
    private WetterDbRep wetterDbRep;

    @Autowired
    private ImportedFileRepository importedFileRepository;


    private static final String DWD_BASE_URL = "https://opendata.dwd.de/climate_environment/CDC/observations_germany/cli
    private static final String DWD_RECENT_URL = "https://opendata.dwd.de/climate_environment/CDC/observations_germany/c
    private static final DateTimeFormatter DWD_DATE_FORMAT = DateTimeFormatter.ofPattern("yyyyMMdd");  2 usages

    //Zuständig für das Importieren ALLER historischen DWD ZIP-Dateien
    public void importAllHistoricalData() {  1 usage
        try {
            Document doc = Jsoup.connect(DWD_BASE_URL).get();
            Elements links = doc.select( cssQuery: "a[href$=.zip]");

            for (Element link : links) {
                String zipFilename = link.attr( attributeKey: "href");

                if (importedFileRepository.existsByFilename(zipFilename)) {
                    System.out.println("FILE HAS ALREADY BEEN IMPORTED: " + zipFilename);
                    continue;
                }

                System.out.println("IMPORTING : " + zipFilename);
                URL fileUrl = new URL( spec: DWD_BASE_URL + zipFilename);

                try (InputStream in = fileUrl.openStream()) {
                    importZipFile(in, zipFilename);
                    importedFileRepository.save(new ImportedFile(zipFilename));
                } catch (Exception e) {
                    System.err.println("ERROR AT FILE  ==> " + zipFilename + " REASON ==> " + e.getMessage());
                }
            }

            System.out.println("<====================>IMPORT FINISHED<====================> ");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

```

Abbildung A-9: WetterDWDImporter Einblick und Historical Import

- Der Ablauf der importAllHistoricalData läuft folgendermaßen:

1. Die Methode importAllHistoricalData scannt zunächst den Inhalt der URL (DWD_BASE_URL) und wählt nur die Elemente aus, die mit .zip enden
2. Anschließend wird überprüft, ob der Name der ZIP-Datei bereits in der imported_file-Tabelle vorhanden ist; falls ja, wird diese ZIP-Datei übersprungen.
3. Falls der ZIP-Name nicht in der Datenbank vorhanden ist, wird für jedes ausgewählte Element (Link) eine for-Schleife durchlaufen, welche die benötigten ZIP-Dateien herunterlädt, sie mit der Methode importZipFile verarbeitet und anschließend den Namen der ZIP-Datei in die imported_file-Tabelle einträgt, schreibt.

```java
//Zuständig für das Importieren der aktuellen RECENT-Daten, aber nur einmal pro Tag, ableich mit recent_time.txt
public void importRecentDataOncePerDay() {  1 usage
    File markerFile = new File( pathname: "recent_time.txt");
    LocalDate today = LocalDate.now();

    //Wenn Datei existiert, prüfen ob schon heute importiert wurde
    if (markerFile.exists()) {
        try (BufferedReader reader = new BufferedReader(new FileReader(markerFile))) {
            String line = reader.readLine();
            if (line != null) {
                LocalDate lastImportDate = LocalDate.parse(line);
                if (!lastImportDate.isBefore(today)) {
                    System.out.println("RECENT-Daten wurden heute bereits importiert.");
                    return;
                }
            }
        } catch (Exception e) {
            System.err.println("Fehler beim Lesen von recent_time.txt: " + e.getMessage());
        }
    }

    //Import durchführen
    try {
        System.out.println("IMPORTING RECENT DATA...");
        Document doc = Jsoup.connect(DWD_RECENT_URL).get();
        Elements links = doc.select( cssQuery: "a[href$=.zip]");

        for (Element link : links) {
            String zipFilename = link.attr( attributeKey: "href");
            System.out.println("IMPORTING : " + zipFilename);
            URL fileUrl = new URL( spec: DWD_RECENT_URL + zipFilename);

            try (InputStream in = fileUrl.openStream()) {
                importZipFile(in, zipFilename); //Benutzt dieselbe Logik wie historische Daten
            } catch (Exception e) {
                System.err.println("FEHLER BEIM RECENT-DATEI-IMPORT ==> " + zipFilename + " REASON ==> " + e.getM
            }
        }

        //Datum in recent_time.txt schreiben
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(markerFile))) {
            writer.write(today.toString());
        }

        System.out.println("<====================>RECENT IMPORT ABGESCHLOSSEN<====================> ");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Abbildung A-10: WetterDWDImported Recent-Import

- Die importRecentDataOncePerDay() Methode läuft folgendermaßen:

1. Zunächst wird das aktuelle Datum gespeichert und anschließend eine recent_time.txt erstellt.
2. Falls bereits eine recent_time.txt existiert, wird diese stattdessen gelesen. Wenn die erste Zeile nicht leer ist, wird per if-Abfrage geprüft, ob das darin enthaltene Datum mit dem aktuellen Datum übereinstimmt. Ist das der Fall, wird der Recent-Import nicht weiter ausgeführt.
3. Falls keiner der zuvor genannten Fälle zutrifft, wird der Recent-Import gestartet.
4. Die URL (DWD_RECENT_URL) wird gescannt, und es werden nur die Elemente ausgewählt, die mit .zip enden.

5. Für jedes ausgewählte Element (Link) wird eine for-Schleife durchlaufen, welche die ZIP-Dateien mithilfe der Methode importZipFile() verarbeitet.
6. Sobald die for-Schleife erfolgreich beendet wurde, wird das heutige Datum in die recent_time.txt eingetragen und die Methode abgeschlossen.

```
114
115     //Zuständig für das Verarbeiten einer einzelnen ZIP-Datei
116     public void importZipFile(InputStream zipStream, String zipFilename) throws IOException {  2 usages
117         try (ZipInputStream zis = new ZipInputStream(zipStream)) {
118             ZipEntry entry;
119             while ((entry = zis.getNextEntry()) != null) {
120                 if (entry.getName().startsWith("produkt") && entry.getName().endsWith(".txt")) {
121                     BufferedReader reader = new BufferedReader(new InputStreamReader(zis));
122                     reader.readLine(); //Header überspringen
123
124                     //Map<stationId, Map<LocalDate, WetterDbEntity>> => Cache nur für relevante Stationen
125                     Map<Integer, Map<LocalDate, WetterDbEntity>> stationCache = new HashMap<>();
126                     Set<Integer> seenStationIds = new HashSet<>();
127                     Map<Integer, LocalDate[]> stationDateRanges = new HashMap<>();
128
129                     List<String> lines = new ArrayList<>();
130                     while (reader.ready()) {
131                         lines.add(reader.readLine());
132                     }
133
134                     //Sammle pro stationId den min/max Zeitraum
135                     for (String line : lines) {
136                         try {
137                             String[] parts = line.split( regex: ";");
138                             Integer stationId = Integer.parseInt(parts[0].trim());
139                             LocalDate date = LocalDate.parse(parts[1], DWD_DATE_FORMAT);
140                             seenStationIds.add(stationId);
141
142                             stationDateRanges.putIfAbsent(stationId, new LocalDate[]{date, date});
143                             LocalDate[] range = stationDateRanges.get(stationId);
144                             range[0] = date.isBefore(range[0]) ? date : range[0];
145                             range[1] = date.isAfter(range[1]) ? date : range[1];
146                         } catch (Exception e) {
147                             System.err.println("IGNORING LINE DURING RANGE PARSE");
148                         }
149                     }
150
151                     //Hole alle relevanten Daten pro Station aus DB (zeitlich begrenzt)
152                     for (Integer stationId : seenStationIds) {
153                         LocalDate[] range = stationDateRanges.get(stationId);
154                         List<WetterDbEntity> dbEntries = wetterDbRep.findByStationIdAndDateRange(stationId, range[0], ra
155                         Map<LocalDate, WetterDbEntity> dateMap = new HashMap<>();
156                         for (WetterDbEntity e : dbEntries) {
157                             dateMap.put(e.getDate(), e);
158                         }
159                         stationCache.put(stationId, dateMap);
160                     }
161
162                     int savedCount = 0;
163                     int skippedCount = 0;
164                     int updatedCount = 0;
165
```

Abbildung A-11: Import Logik Teil 1/2

```
                //Importieren
                for (String line : lines) {
                    String[] parts = line.split( regex: ";");
                    try {
                        Integer stationId = Integer.parseInt(parts[0].trim());
                        LocalDate date = LocalDate.parse(parts[1], DWD_DATE_FORMAT);

                        WetterDbEntity entity = new WetterDbEntity();
                        entity.setStationId(stationId);
                        entity.setDate(date);
                        entity.setQualityI(parseDouble(parts[2]));
                        entity.setWindMax(parseDouble(parts[3]));
                        entity.setWindSpeed(parseDouble(parts[4]));
                        entity.setQualityII(parseDouble(parts[5]));
                        entity.setPrecipitation(parseDouble(parts[6]));
                        entity.setPrecForm(parseDouble(parts[7]));
                        entity.setSunshine(parseDouble(parts[8]));
                        entity.setSnow(parseDouble(parts[9]));
                        entity.setCovering(parseDouble(parts[10]));
                        entity.setSteamPressure(parseDouble(parts[11]));
                        entity.setAirPressure(parseDouble(parts[12]));
                        entity.setTemperature(parseDouble(parts[13]));
                        entity.setHumidity(parseDouble(parts[14]));
                        entity.setTempMax(parseDouble(parts[15]));
                        entity.setTempMin(parseDouble(parts[16]));
                        entity.setTempMin5cm(parseDouble(parts[17]));

                        Map<LocalDate, WetterDbEntity> stationMap = stationCache.getOrDefault(stationId, new HashMap
                        WetterDbEntity existing = stationMap.get(date);

                        if (existing != null) {
                            if (entitiesEqual(existing, entity)) {
                                skippedCount++;
                                continue;
                            }
                            entity.setId(existing.getId());
                            updatedCount++;
                        } else {
                            savedCount++;
                        }

                        wetterDbRep.save(entity);

                    } catch (Exception e) {
                        System.err.println("SKIPPING INVALID ROW: " + line);
                    }
                }

                System.out.println("Saved: " + savedCount + ", Updated: " + updatedCount + ", Skipped: " + skippedCo
                break;
            }
        }
    }
```

Abbildung A-12: Import Logik Teil 2/2

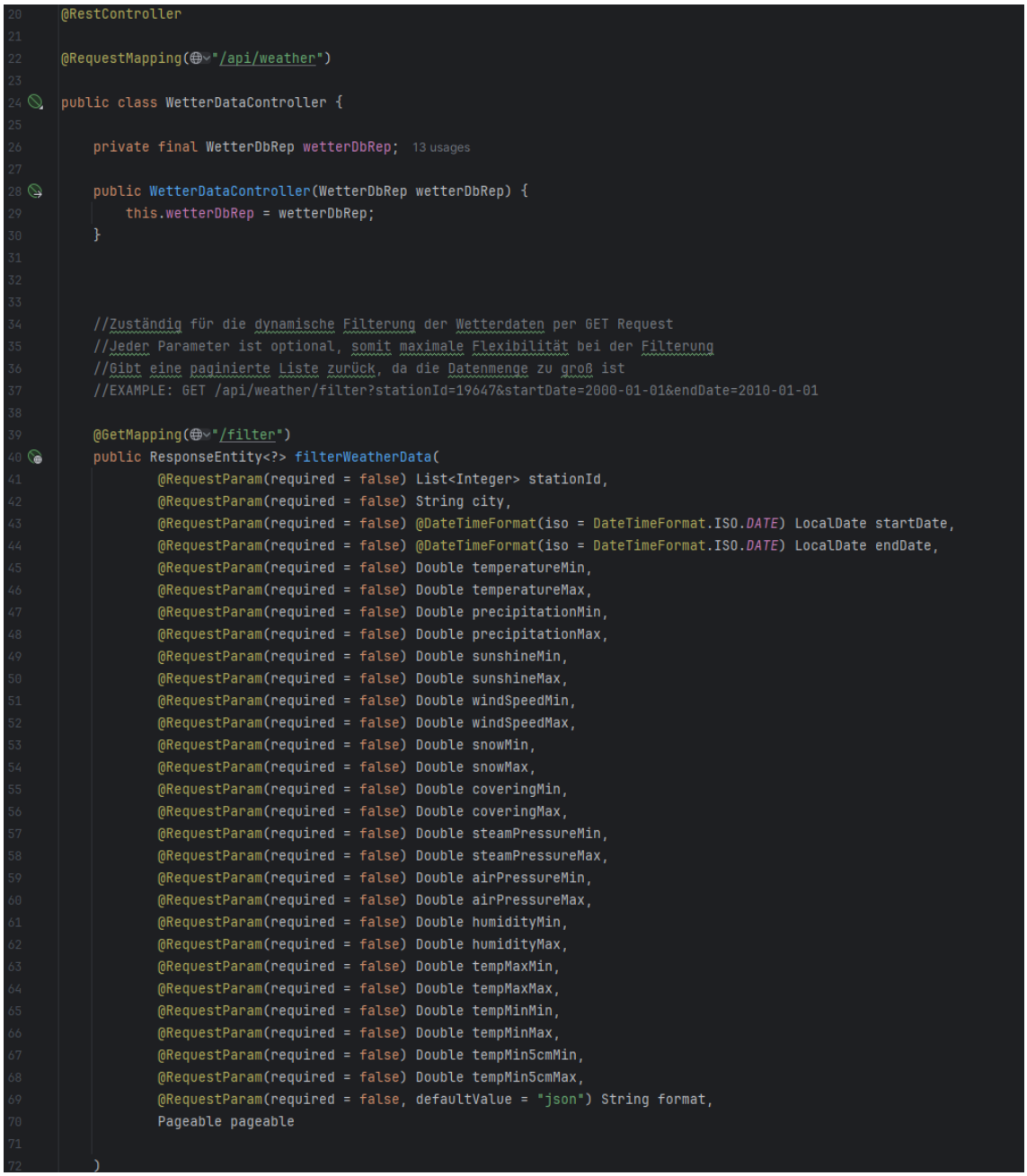
- Die importZipFile() Methode läuft folgendermaßen:

1. Die ZIP-Dateien werden einzeln auf eine „produkt“-Datei geprüft, die mit .txt endet. Dateien mit diesem Format enthalten die Klimadaten, die verarbeitet werden.
2. Daraufhin werden die Textdaten gelesen und dabei drei Datenstrukturen erstellt:
    - stationCache dient zur Zwischenspeicherung bereits bekannter Stationen in der Datenbank.
    - seenStationIds enthält alle vorkommenden Stations-IDs in der TXT-Datei.

- stationDateRange enthält das Start- und Enddatum der jeweiligen Stationen, die in der TXT-Datei vorkommen.

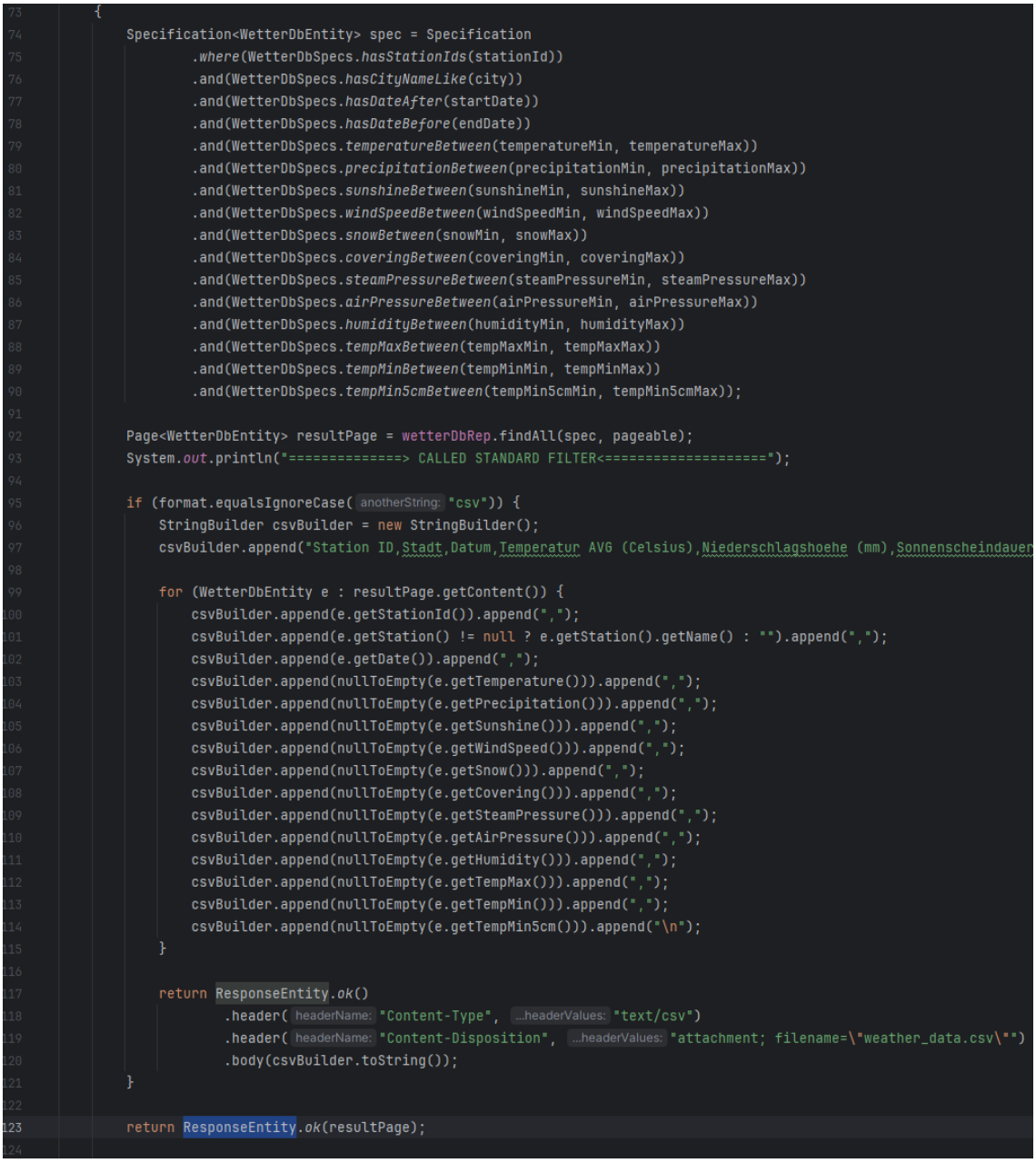
3. In der ersten for-Schleife wird ermittelt, von wann bis wann die Messdaten pro Station vorliegen, und diese Zeiträume werden in stationDateRanges gespeichert.
4. In der zweiten for-Schleife wird für jede Station und ihren Zeitraum eine Datenbankanfrage durchgeführt. Falls Ergebnisse gefunden werden, werden die Datensätze in stationCache gespeichert, für einen späteren Vergleich.
5. In der dritten for-Schleife findet der eigentliche Importprozess statt. Alle Zeilen werden anhand von Semikolon getrennt und in die jeweiligen Entity-Variablen gespeichert.
    - Nach der Speicherung wird geprüft, ob bereits ein Eintrag mit gleichem Datum und gleicher Station vorhanden ist. Der Eintrag wird je nach Ergebnis entweder übersprungen, überschrieben oder neu gespeichert.
6. Abschließend wird für jede ZIP-Datei ein kurzer Report ausgegeben, der angibt, wie viele Datensätze gespeichert, aktualisiert oder übersprungen wurden.

# WetterDataController.java



```java
@RestController

@RequestMapping("/api/weather")

public class WetterDataController {

    private final WetterDbRep wetterDbRep;  13 usages

    public WetterDataController(WetterDbRep wetterDbRep) {
        this.wetterDbRep = wetterDbRep;
    }




    //Zuständig für die dynamische Filterung der Wetterdaten per GET Request
    //Jeder Parameter ist optional, somit maximale Flexibilität bei der Filterung
    //Gibt eine paginierte Liste zurück, da die Datenmenge zu groß ist
    //EXAMPLE: GET /api/weather/filter?stationId=19647&startDate=2000-01-01&endDate=2010-01-01

    @GetMapping("/filter")
    public ResponseEntity<?> filterWeatherData(
            @RequestParam(required = false) List<Integer> stationId,
            @RequestParam(required = false) String city,
            @RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate startDate,
            @RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate endDate,
            @RequestParam(required = false) Double temperatureMin,
            @RequestParam(required = false) Double temperatureMax,
            @RequestParam(required = false) Double precipitationMin,
            @RequestParam(required = false) Double precipitationMax,
            @RequestParam(required = false) Double sunshineMin,
            @RequestParam(required = false) Double sunshineMax,
            @RequestParam(required = false) Double windSpeedMin,
            @RequestParam(required = false) Double windSpeedMax,
            @RequestParam(required = false) Double snowMin,
            @RequestParam(required = false) Double snowMax,
            @RequestParam(required = false) Double coveringMin,
            @RequestParam(required = false) Double coveringMax,
            @RequestParam(required = false) Double steamPressureMin,
            @RequestParam(required = false) Double steamPressureMax,
            @RequestParam(required = false) Double airPressureMin,
            @RequestParam(required = false) Double airPressureMax,
            @RequestParam(required = false) Double humidityMin,
            @RequestParam(required = false) Double humidityMax,
            @RequestParam(required = false) Double tempMaxMin,
            @RequestParam(required = false) Double tempMaxMax,
            @RequestParam(required = false) Double tempMinMin,
            @RequestParam(required = false) Double tempMinMax,
            @RequestParam(required = false) Double tempMin5cmMin,
            @RequestParam(required = false) Double tempMin5cmMax,
            @RequestParam(required = false, defaultValue = "json") String format,
            Pageable pageable

    )
```

Abbildung A-13: WetterDataController filterWeatherData() Kopf



```
    {
        Specification<WetterDbEntity> spec = Specification
                .where(WetterDbSpecs.hasStationIds(stationId))
                .and(WetterDbSpecs.hasCityNameLike(city))
                .and(WetterDbSpecs.hasDateAfter(startDate))
                .and(WetterDbSpecs.hasDateBefore(endDate))
                .and(WetterDbSpecs.temperatureBetween(temperatureMin, temperatureMax))
                .and(WetterDbSpecs.precipitationBetween(precipitationMin, precipitationMax))
                .and(WetterDbSpecs.sunshineBetween(sunshineMin, sunshineMax))
                .and(WetterDbSpecs.windSpeedBetween(windSpeedMin, windSpeedMax))
                .and(WetterDbSpecs.snowBetween(snowMin, snowMax))
                .and(WetterDbSpecs.coveringBetween(coveringMin, coveringMax))
                .and(WetterDbSpecs.steamPressureBetween(steamPressureMin, steamPressureMax))
                .and(WetterDbSpecs.airPressureBetween(airPressureMin, airPressureMax))
                .and(WetterDbSpecs.humidityBetween(humidityMin, humidityMax))
                .and(WetterDbSpecs.tempMaxBetween(tempMaxMin, tempMaxMax))
                .and(WetterDbSpecs.tempMinBetween(tempMinMin, tempMinMax))
                .and(WetterDbSpecs.tempMin5cmBetween(tempMin5cmMin, tempMin5cmMax));

        Page<WetterDbEntity> resultPage = wetterDbRep.findAll(spec, pageable);
        System.out.println("==============> CALLED STANDARD FILTER<====================");

        if (format.equalsIgnoreCase( anotherString: "csv")) {
            StringBuilder csvBuilder = new StringBuilder();
            csvBuilder.append("Station ID,Stadt,Datum,Temperatur AVG (Celsius),Niederschlagshoehe (mm),Sonnenscheindauer

            for (WetterDbEntity e : resultPage.getContent()) {
                csvBuilder.append(e.getStationId()).append(",");
                csvBuilder.append(e.getStation() != null ? e.getStation().getName() : "").append(",");
                csvBuilder.append(e.getDate()).append(",");
                csvBuilder.append(nullToEmpty(e.getTemperature())).append(",");
                csvBuilder.append(nullToEmpty(e.getPrecipitation())).append(",");
                csvBuilder.append(nullToEmpty(e.getSunshine())).append(",");
                csvBuilder.append(nullToEmpty(e.getWindSpeed())).append(",");
                csvBuilder.append(nullToEmpty(e.getSnow())).append(",");
                csvBuilder.append(nullToEmpty(e.getCovering())).append(",");
                csvBuilder.append(nullToEmpty(e.getSteamPressure())).append(",");
                csvBuilder.append(nullToEmpty(e.getAirPressure())).append(",");
                csvBuilder.append(nullToEmpty(e.getHumidity())).append(",");
                csvBuilder.append(nullToEmpty(e.getTempMax())).append(",");
                csvBuilder.append(nullToEmpty(e.getTempMin())).append(",");
                csvBuilder.append(nullToEmpty(e.getTempMin5cm())).append("\n");
            }

            return ResponseEntity.ok()
                    .header( headerName: "Content-Type",  ...headerValues: "text/csv")
                    .header( headerName: "Content-Disposition",  ...headerValues: "attachment; filename=\"weather_data.csv\"")
                    .body(csvBuilder.toString());
        }

        return ResponseEntity.ok(resultPage);
```

Abbildung A-14: WetterDataController filterWeatherData() Inhalt

- Die Klasse bietet die REST-API-Endpunkte an. Über @GetMapping() kann eine Rückgabemethode aufgerufen werden, um die gewünschte Antwort zu erhalten.
- Die Methode filterWeatherData() besitzt einen sehr langen Methodenkopf aufgrund der Aufzählung aller Klimaparameter, die vom DWD angegeben werden, zusätzlich erweitert durch die Bereichsfunktion mit Minimum/Maximum.
- Alle Parameter, die an filterWeatherData() übergeben werden, rufen die Specifications aus der Klasse WetterDbSpecs auf. Die Angabe aller Parameter ist optional.

- Zusätzlich zur JSON-Ausgabe kann die Antwort auch im CSV-Format erfolgen. Dabei ist zu beachten, dass die Daten kommagetrennt ausgegeben werden und nicht mit Semikolon, wie es beim DWD der Fall ist.

# CorsConfig.java

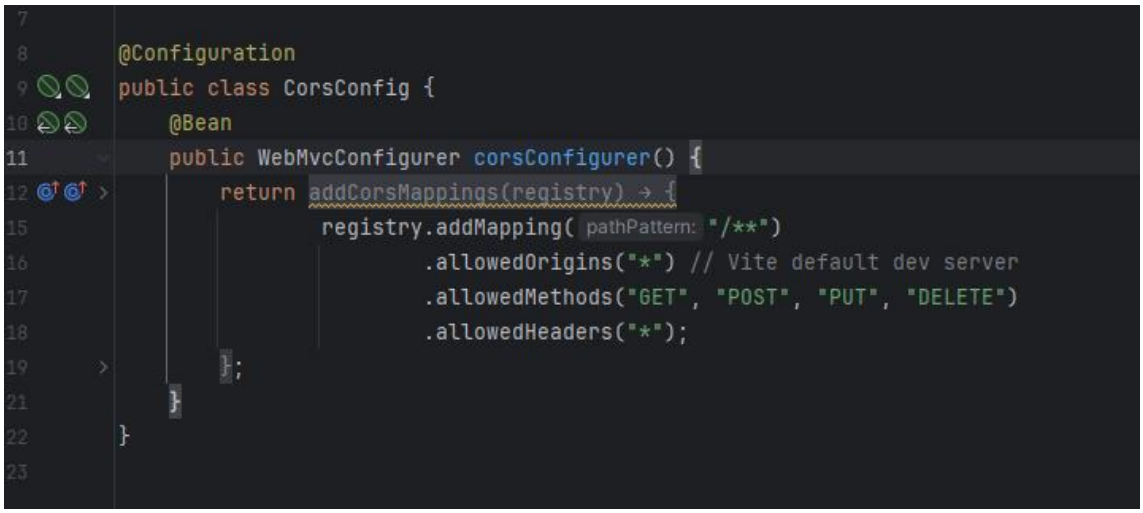



Abbildung A-15: CorsConfig.java Einblick

- Durch allowedOrigins() kann die Domain angegeben werden, die erlaubt werden soll; bei „*“ wird jede Domain zugelassen.
- allowedMethods() steuert den Zugriff auf verschiedene REST-Funktionen. Falls Nutzer ausschließlich lesend (GET) zugreifen sollen, können die Schreib-, Update- und Löschfunktionen entfernt werden.
- Falls eine Konfiguration für spezifische Header vorgenommen wird, sollte entsprechend allowedHeaders angepasst werden.

# WetterPortalApplication.java

```java

@SpringBootApplication
public class WetterPortalApplication {

    public static void main(String[] args) {

        SpringApplication.run(WetterPortalApplication.class, args);
    }

    @Bean
    CommandLineRunner runImport(WetterDWDImporter importer) {
        return String[] args -> {
            importer.importAllHistoricalData();
            System.out.println("HISTORICAL IMPORT FINISHED.");

            importer.importRecentDataOncePerDay();
            System.out.println("RECENT IMPORT FINISHED.");
        };
    }

}

```

Abbildung A-16: WetterPortalApplication Einblick

- Beinhaltet die Main-Methode, die das Programm startet.
- Durch den CommandLineRunner können spezifische Methoden beim Start ausgeführt werden, in diesem Fall werden importAllHistoricalData() und importRecentDataOncePerDay() aufgerufen.

# Literaturverzeichnis

**Bundesamt für Justiz** (Jahr unbekannt, online): *„Gesetze über den Deutschen Wetterdienst (DWD-Gesetz) § 4 Aufgaben“*
https://www.gesetze-im-internet.de/dwdg/__4.html
(Datum des Zugriffs: 19.06.2025)

**Bundesregierung** (2021, online): *„Zwischenbericht zur Flutkatastrophe 2021: Katastrophenhilfe, Soforthilfen und Wiederaufbau“*
https://www.bundesregierung.de/resource/blob/974430/1963706/613b934d3f359a5118df16755e9e527c/2021-09-27-zwischenbericht-hochwasser-data.pdf
(Datum des Zugriffs: 19.06.2025)

**Bundesregierung** (2021, online) (2): *„Open-Data-Strategie der Bundesregierung“*
https://www.bmi.bund.de/SharedDocs/downloads/DE/publikationen/themen/moderne-verwaltung/open-data-strategie-der-bundesregierung.pdf?__blob=publicationFile&v=6
(Datum des Zugriffs: 19.06.2025)

**Bundesregierung** (2025, online): *„Web Content Accessibility Guidelines 2.1 (WCAG 2.1)“*
https://www.barrierefreiheit-dienstekonsolidierung.bund.de/Webs/PB/DE/gesetze-und-richtlinien/wcag/wcag-node.html
(Datum des Zugriffs: 19.06.2025)

**Carreras-Coch, A.; Navarro, J.; Sans, C.; Zaballos, A.** (2022, online): *„Communication Technologies in Emergency Situations“*
https://www.mdpi.com/2079-9292/11/7/1155
(Datum des Zugriffs: 23.06.2025)

**Climate Outreach** (2020, online): *„Engaging the public on climate risks and adaptation“*
https://www.ukclimateresilience.org/wp-content/uploads/2020/03/resilrisk-briefing-ONLINE.pdf
(Datum des Zugriffs: 23.06.2025)

**Climate Outreach** (2022, online): *„Ten Key Principles: How to Communicate Climate Change for Effective Public Engagement“*
https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4151465
(Datum des Zugriffs: 23.06.2025)

**Climate Outreach** (Jahr unbekannt, online): *„ Seven principles for visual climate change communication“*
https://climateoutreach.org/reports/climate-visuals-seven-principles-for-visual-climate-change-communication/
(Datum des Zugriffs: 23.06.2025)

**Deutscher Wetterdienst** (2022, online): *„Faktenpapier 2022: Was wir 2022 über das Extremwetter in Deutschland wissen“*
https://www.dwd.de/DE/klimaumwelt/aktuelle_meldungen/220928/Faktenpapier-Extremwetterkongress_download.pdf?__blob=publicationFile&v=5
(Datum des Zugriffs: 19.06.2025)

**Deutscher Wetterdienst** (2023, online): *„Zahlen und Fakten zum Deutschen Wetterdienst 2023“*
https://www.dwd.de/DE/presse/pressekonferenzen/DE/2023/PK_2023_03_21/zundf_zum_dwd_2023.pdf?__blob=publicationFile
(Datum des Zugriffs: 19.06.2025)

**Deutscher Wetterdienst** (2025, online): *„Zahlen und Fakten zum Deutschen Wetterdienst 2025“*
https://www.dwd.de/DE/presse/pressekonferenzen/DE/2025/PK_2025_04_01/zahlen-und-fakten_zum_dwd_2024.pdf?__blob=publicationFile&v=2
(Datum des Zugriffs: 19.06.2025)

**Deutscher Wetterdienst** (Jahr unbekannt, online): *„Qualitätssicherung historischer Klimadaten“*
https://www.dwd.de/DE/klimaumwelt/klimaueberwachung/klimadatenverarbeitung/qualitaetssicherung/quali_hist.html
(Datum des Zugriffs: 19.06.2025)

**Deutscher Wetterdienst** (Jahr unbekannt, online) (2): *„Rechtliche Hinweise“*
https://www.dwd.de/DE/service/rechtliche_hinweise/rechtliche_hinweise_node.html
(Datum des Zugriffs: 19.06.2025)

**Deutscher Wetterdienst** (Jahr unbekannt, online) (3): *„Häufig gestellte Fragen - Open Data“*
https://www.dwd.de/DE/leistungen/opendata/faqs_opendata.html
(Datum des Zugriffs: 19.06.2025)

**Deutscher Wetterdienst** (Jahr unbekannt, online) (4):
https://opendata.dwd.de/climate_environment/CDC/observations_germany/climate/daily/kl/historical/KL_Tageswerte_Beschreibung_Stationen.txt
(Datum des Zugriffs: 19.06.2025)

**Europäische Kommission** (2021, online): *„Wie Städte dank neuartiger Plattform Klimaresilienz und -anpassung aufbauen“*
https://cordis.europa.eu/article/id/451845-novel-platform-empowers-cities-to-build-climate-resilience-and-adaptation/de
(Datum des Zugriffs: 19.06.2025)

**Europäische Kommission** (Jahr unbekannt, online): *„Open source software strategy“*
https://commission.europa.eu/about/departments-and-executive-agencies/digital-services/open-source-software-strategy_en?prefLang=de
(Datum des Zugriffs: 19.06.2025)

**GO FAIR International Support and Coordination Office** (Jahr unbekannt, online): *„FAIR Principles“*
https://www.go-fair.org/fair-principles/
(Datum des Zugriffs: 19.06.2025)

**Intergovernmental Panel on Climate Change (IPCC)** (2021, online): *„Summary for Policy-makers“*
https://www.ipcc.ch/report/ar6/wg1/downloads/report/IPCC_AR6_WGI_SPM.pdf
(Datum des Zugriffs: 19.06.2025)

**Ipsos GmbH** (2025, online): *„Earth Day 2025: Deutsche verlieren Interesse am Klimaschutz“*
https://www.ipsos.com/sites/default/files/ct/news/documents/2025-04/Ipsos-PM_Earth%20Day_2025-04-17.pdf
(Datum des Zugriffs: 19.06.2025)

**LoKlim** (Jahr unbekannt, online): *„Das Forschungsprojekt LoKlim“*
https://lokale-klimaanpassung.de/
(Datum des Zugriffs: 19.06.2025)

**Meteostat** (Jahr unbekannt, online): *„Wetterrückblick und Klimadaten“*
https://meteostat.net/de/
(Datum des Zugriffs: 19.06.2025)

**Met Office** (2019, online): *„UK Climate Resilience Programme“*
https://www.metoffice.gov.uk/binaries/content/assets/metofficegovuk/pdf/research/spf/ukcrp_joint_science_plan.pdf
(Datum des Zugriffs: 23.06.2025)

**Mozilla** (Jahr unbekannt, online): *„API“*
https://developer.mozilla.org/de/docs/Glossary/API
(Datum des Zugriffs: 19.06.2025)

**Open Geospatial Consortium** (2022, online): *„OGC API - Features - Part 1: Core corrigendum“*
https://docs.ogc.org/is/17-069r4/17-069r4.html
(Datum des Zugriffs: 19.06.2025)

**Open Knowledge Foundation** (Jahr unbekannt, online): *„Open Definition – Defining Open in Open Data, Open Content and Open Knowledge“*
https://opendefinition.org/od/2.1/de/
(Datum des Zugriffs: 19.06.2025)

**Open Source Initiative** (2024, online): *„The Open Source Definition“*
https://opensource.org/osd
(Datum des Zugriffs: 19.06.2025)

**Umweltbundesamt** (2022, online): *„Umweltbewusstsein in Deutschland 2020“*
https://www.umweltbundesamt.de/sites/default/files/medien/479/publikationen/ubs_2020_0.pdf
(Datum des Zugriffs: 19.06.2025)

**Umweltbundesamt** (2023, online): *„Umweltbewusstsein in Deutschland 2022“*
https://www.umweltbundesamt.de/sites/default/files/medien/3521/publikationen/umweltbewusstsein_2022_bf-2023_09_04.pdf
(Datum des Zugriffs: 19.06.2025)

**United Nations** (Jahr unbekannt, online): *„Communicating on Climate Change“*
https://www.un.org/en/climatechange/communicating-climate-change
(Datum des Zugriffs: 23.06.2025)

**WeatherSpark** (Jahr unbekannt, online): *„Das Wetter das ganze Jahr über an einem beliebigen Ort der Erde“*
https://de.weatherspark.com/
(Datum des Zugriffs: 19.06.2025)

**WeatherSpark** (Jahr unbekannt, online) (2): *„Klima und durchschnittliches Wetter das ganze Jahr über in Waiblingen“*
https://de.weatherspark.com/y/63769/Durchschnittswetter-in-Waiblingen-Baden-W%C3%BCrttemberg-Deutschland-das-ganze-Jahr-%C3%BCber
(Datum des Zugriffs: 19.06.2025)

# Erklärung zur Verwendung generativer KI-Systeme

Bei der Erstellung der Arbeit habe ich die folgenden auf künstlicher Intelligenz (KI) basierten Systeme benutzt:

1. OpenAI: ChatGPT 4o

Ich erkläre weiterhin, dass ich

- ✓ mich aktiv über die Leistungsfähigkeit und Beschränkungen der oben genannten KI-Systeme informiert habe,
- ✓ die aus den oben angegebenen KI-Systemen übernommenen Passagen gekennzeichnet habe,
- ✓ überprüft habe, dass die mithilfe der oben genannten KI-Systeme generierten und von mir übernommenen Inhalte faktisch richtig sind,
- ✓ mir bewusst bin, dass ich als Autor:in dieser Arbeit die Verantwortung für die in ihr gemachten Angaben und Aussagen trage.

Die oben genannten KI-Systeme habe ich wie im Folgenden dargestellt eingesetzt:

| **Arbeitsschritt** | **Eingesetzte(s) KI-System(e)** | **Beschreibung der Verwendungsweise** |
|---|---|---|
| Literatursuche | ChatGPT 4o | ChatGPT Deep Research Funktion zur Suche von verwandten Projekten und Untersuchungen |
| Auswahl von Technologien | ChatGPT 4o | Vorschläge untersucht welche Technologien benutzt werden können für große Datenmengen |
| Generierung von Dateien | ChatGPT 4o | Generierung einer „stationMap.json“ die eine .txt-Datei in eine JSON-Datei umwandelt |
| Interpretation und Validierung | ChatGPT 4o | Untersuchung von Fehlercodes während Programmierung der Backend und Frontend |
| Korrektur der Arbeit (Grammatik und Rechtschreibung) | ChatGPT 4o | Zur Feststellung und Korrektur von Texten in der Arbeit |

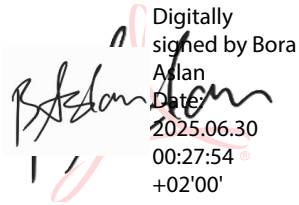
# Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig und ohne Verwendung anderer als der angegebenen Hilfsmittel verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäß den gängigen wissenschaftlichen Regeln zitiert. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Ort, Datum

Waiblingen, 30.06.2025

Unterschrift



# Stichwortverzeichnis