FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

FH Aachen

Fachbereich
Elektrotechnik und Informationstechnik
Studiengang Informatik

Bachelorarbeit

Konzeption und prototypische Entwicklung einer webbasierten Applikation
zur Wertschöpfung und Bereitstellung von Geodaten
(Data-Konnector für Geodaten)

Tuan Anh Cong Nguyen
Matr.-Nr.: 3517392

Referent: Prof. Dr. rer. nat. Heinrich Faßbender

In Zusammenarbeit mit Ausbildungsbetrieb:
ahu GmbH Wasser Boden Geomatik

Externer Betreuer: Dr. David Loibl

Selbstständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Aachen, den 14. Mai 2025 .....

Inhaltsverzeichnis

Abkürzungsverzeichnis

REST-API	Representational State Transfer Application Program Interface
Nu	Nußelt-Zahl
ν_{Luft}	Kinematische Viskosität von Luft
Pr	Prandtl-Zahl
$\dot{Q}$	Wärmestrom
Ra	Rayleigh-Zahl
ρ_{Luft}	Dichte von Luft
T	Temperatur
T_∞	Umgebungstemperatur

Abstract

In einer zunehmend digitalisierten und automatisierten Welt wächst täglich die Menge an neu generierten Geodaten – insbesondere Mess- und räumlichen Daten.

Die Suche nach und Einbindung von OGC-Geodatendiensten wie WMS (Web Map Service) und WFS (Web Feature Service) in Geoinformationssysteme wie QGIS kann tatsächlich schwierig sein. Die Suche erfordert den Anwender oft das Durchforsten verschiedener Quellen und die manuelle Überprüfung von Dienstbeschreibung. Eine automatisiert und effiziente Lösung kann teilweise helfen, diesen Prozess zu verbessern.
Die Suche nach Messdaten wie Grundwasserständen, Pegelständen und Klimadaten (Niederschlag,Temperatur) und deren Umformatierung in ein einheitliches Format gestalten sich als schwierig und kompliziert, da es eine Vielzahl von Datenanbietern mit unterschiedliche Datenstrukturen, Formen, etc. gibt.

Die Bereitstellung dieser Rohdaten erfolgt meinst im CSV-Format oder als individuell strukturierte ASCII-Dateien. Im Kontext von der Datenkonsum, gibt es bei der Ahu den sogenannten ahuManager-Client, der diese Daten konsumiert. Bevor diese konsumieren können, müssen die Rohdaten in ein kompatibles Format umformatiert werden, was bisher teilweise von Hand erledigt werden (in Excel o.ä.). Diese manuelle Vorgehen kann perspektivisch zum Teil oder ganz automatisiert werden, was die Produktivität steigert und die Fehleranfälligkeit vermeidet.

Daher befasst sich diese Arbeit mit dem Thema, wie eine webbasierte Applikation zur Wertschöpfung und Bereitstellung von rohen Geodaten im Kontext von der Abteilung Geomatik von Ahu GmbH entwickelt werden kann.

1 Einleitung

1.1 Motivation der Arbeit

Diese Bachelorarbeit wird in Zusammenarbeit mit dem Bereich Geomatik der Ahu GmbH verfasst.Der Bereich beschäftigt sich mit Konzeption und Softwareentwicklung der Monitoringsysteme und Web-Anwendungen, die es Dienstleitern, Betreibern und Aufsichtsbehörden ermöglichen, ein kosteneffizientes Management für Geodaten(Grundwasser, Oberflächenwasser, Boden,etc.) umzusetzen.

Momentan besteht es Bedarf für ein zentrales System zur automatisierten Suche und Verwaltung der Geodaten aus verschiedenen Datenanbietern und diese Daten in einer homogen Struktur zusammenzubringen und einen vorhandenen Client(z.B ahuManager) bereitzustellen. Unter anderem wird auch die Einbindung von OGC-Geodatendiensten für besseren und effizienten Prozess zur Durchsuche und Verwaltung gesorgt.

1.2 Ziel der Arbeit

Die Hauptaufgabe dieser Arbeit besteht darin, ein Konzeption für die prototypische Entwicklung einer webbasierten Applikation zu erstellen und umzusetzen. Die Ziele enthalten eine übergreifende Suche von Geodaten und deren Umformatierung in ein passendes vorgegebenes Zielformat. Das weitere Zeil ist die Einbindung von mindestens einer OGC-Geodatendiensten.

Durch die Umsetzungen dieser Ziele soll perspektivisch eine Software entstehen, die offene Problematik eine mögliche Lösung anbietet.

1.3 Aufbau der Arbeit

Kapitel 1: Einleitung

Im ersten Kapitel werden die Motivation und Zwecke für diese Arbeit erklärt. Es wird kurz auf die Problematik des gewählten Themas eingegangen und dieses perspektivisch für die

Firma auch Vorteile bringt.

Kapitel 2: Übersicht von Technologien

In diesem Kapitel werden wichtige Begriffe und in der Entwicklungsphase verwendete Technologien vorgestellt. Von der Bibliothek zur Verfolgung der Datenbankänderungen über das Backend-Framework bis hin zu in der Benutzeroberfläche verwendetem UI-Framework.

Kapitel 3: Anforderungsanalyse und Spezifikation

Diese Kapitel

Kapitel 4: Konzeption und Architektur

Dieses Kapitel erläutert die angewandte Architektur des Prototyps anhand von unterschiedlichen Sichten.

Kapitel 5: Implementierung

Anhand zahlreichen Code-Beispiele und technischen Erläuterungen wird die wirkliche Implementierung nochmal veranschaulicht.

Kapitel 6: Evaluierung

Das vorletzte Kapitel bewertet, inwiefern die vom Anfang festgelegte und erarbeitete Anforderungen erfüllt wurde.

Kapitel 7: Fazit und Ausblick für weitere Entwicklung

Abschließend wird noch einmal die gesamte Arbeit zusammenfassend bewertet und ein kurzer Ausblick für weiteren Entwicklungen oder Verbesserungen in der Zukunft gegeben.

2 Übersicht von Technologien

2.1 Vaadin

Vaadin ist ein serverseitiges Web-Framework, das dem Entwickler erlaubt, moderne Webanwendung in Java zu entwickeln, ohne dass explizit HTML,CSS oder JavaScript geschrieben werden muss. Vaddin verfügt über eine große Komponentenbibliothek, die eine Vielzahl an vorgefertigten UI-Elementen wie Buttons, Tabellen, Formulare, Dialoge und Layouts bietet.

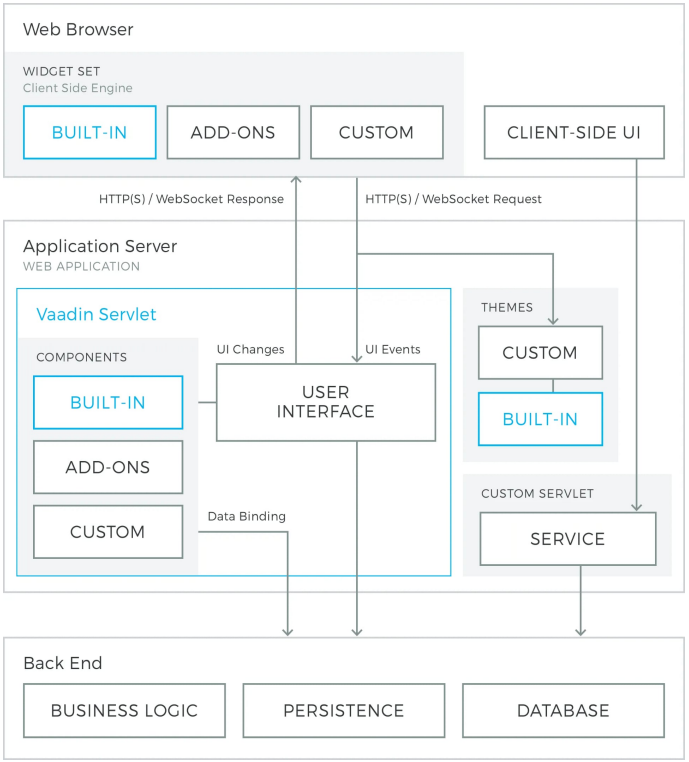

Abbildung 2.1: Übersicht Vaadin Architektur **architecture21**

Vaadin verfolgt einen serverseitigen Rendering-Ansatz. Während eine AJAX-basierte Vaadin Client-Side Engine dafür sorgt, dass die Benutzeroberfläche im Browser durch ein Widget-Set gerendert wird. Die Benutzeroberfläche kann aus eingebauten Komponenten, Add-Ons und benutzerdefinierten Komponenten bestehen. Diese Client-Side Engine kommuniziert über HTTP oder WebSockets mit dem Server. Die gesamte UI-Logik wird dann auf dem Server ausgeführt. Die Vaadin Servlet empfängt Client-Anfragen und aktualisiert die Benutzeroberfläche. Die UI-Komponenten werden serverseitig erstellt und verwaltet. Änderungen

in der UI werden durch die Vaddin Client-Side Engine an den Browser zurückgespielt.

2.2 Spring Framework

Spring Framework ist ein Java-Framework, das um 2003 als Reaktion auf die damals noch zu komplizierte J2EE-Plattform entwickelt wurde. Spring ermöglicht eine einfachere und unkompliziertere Entwicklung von Enterprise-Applikation in Java. Eines der wichtigsten Konzepte von Spring war die Inversion of Control (IoC). IoC ist ein Prinzip, bei dem der Kontrollfluss an eine externe Quelle(z.B. ein Framework) übergeben wird. Das Framework ist dann gemäß einer Spezifikation für die Erstellung und Löschung der Objekte und den Aufruf von Methoden verantwortlich.

Spring führt das Konzept des Bean-Containers ein. Beans sind Java-Objekte, die von Spring instanziiert und verwaltet sind. Wenn die Haupt-Bean von der Hilfs-Bean abhängig ist, stellt Spring sicher, dass die Hilfs-Bean vor der Haupt-Bean initialisiert wird. Spring injizierte außerdem die Instanz der Hilfs-Bean in die Haupt-Bean, sodass die Haupt-Bean nicht mehr nach ihren eigenen Abhängigkeiten suchen muss. Dieses Muster wird als Dependendy Injection bezeichnet.

Damals musste der Entwickler die Bean-Konfiguration in XML schreiben, die Spring anweist, wie die Beans zu konstruieren sind. Inzwischen wird diese mithilfe einer Kombination aus Java-Annotation, Java-Code und Konventionen erstellt.

2.3 Spring Boot

Mit dem Wachstum der Spring-Plattform nahm auch die Komplexität der Entwicklung von Spring-Anwendungen zu. Gleichzeitig verbreitet sich der Einsatz von Microservices und containerisierten Umgebungen in der Softwareentwicklung. Entwickler wünschten sich einfach Methode, um schlanke Webanwendungen zu erstellen, die unabhängig als eigenständige Dienst ausgeführt werden können, anstatt auf einem dedizierten Anwendungsserver zu laufen.

Spring Boot wurde als Reaktion auf diese Anforderungen entwickelt. Es erleichtert die Erstellung von Spring-Anwendungen durch sinnvolle Standard-Einstellungen, Starter-Abhängigkeiten und produktionsreife Funktionen für Konfiguration und Überwachung. Durch die Einführung eingebetteter Servlet-Container wurde es möglich, Anwendungen als eigenständige, ausführbare JAR-Dateien zu verpacken.

2.4 Maven

Maven ist ein Build-Management- und Projektverwaltungswerkzeug, das häufig in der Java-Entwicklung eingesetzt wird. Es automatisiert Aufgaben wie das Herunterladen von Abhängigkeiten, das Kompilieren von Quellcode und das Ausführen von Builds sowie Tests. Maven verwendet eine zentrale Konfigurationsdatei, die `pom.xml`, um alle Aspekte des Projekts wie Libraries, Plugins und Build-Prozesse zu steuern.

Die Funktionsweise von Maven basiert auf einem deklarativen Ansatz. Das bedeutet, dass der Entwickler alle relevanten Informationen über das Projekt in einer zentralen Konfigurationsdatei, der sogenannten `pom.xml`(Projekt Object Model) angibt, wie z.B. Abhängigkeiten, Build-Prozesse und Plugins : **deinhard24**.

1. Die pom.xml-Datei (Project Object Model)

Die `pom.xml` ist das Herzstück eines Maven-Projekts. Sie enthält Informationen wie:

- Projektinformationen: Name des Projekts, Version, Beschreibung.
- Abhängigkeiten: Welche Bibliotheken und Framworks das Projekt benötigt.
- Plugins: Zusätzliche Werkzeuge, die den Build-Prozess erweitern (z.B. Compiler-Plugins, Test-Frameworks).
- Repositories: Woher Maven externe Abhängigkeiten herunterladen soll, typischerweise das Maven Central Repository.
- Build-Spezifikationen: Kompilierungsanweisungen, Testkonfigurationen und Deployment-Optionen.

2. Build-Lifecycle

Maven besitzt einen vordefinierten Build-Lifecycle, der aus mehreren Phasen besteht. Zu den wichtigsten Phasen gehören:

- validate: Überprüft, ob alle erforderlichen Informationen im Projekt vorhanden sind.
- compile: Kompiliert den Quellcode.
- Führt automatisierte Tests aus.
- package: Verpackt den kompilierten Code in ein Distributionsformat, typischerweise eine JAR- oder WAR-Datei.

- install: Installiert das Paket in das lokale Maven-Repository, damit es in anderen Projekten verwendet werden kann.

2.5 Vaadin Initializer

Gegenüber dem üblichen Spring Initializer, der ein vorkonfiguriertes Spring-Projekt bereitstellt, bietet der Vaadin Initializer den Vorteil, dass er speziell auf Vaadin-Projekte zugeschnitten ist und eine vereinfachte und schnellere Projektgenerierung ermöglicht, insbesondere wenn man Spring Boot als Backend verwendet.

Mit dem Vaadin- Initializer kann man direkt ein Vaadin Flow-basiertes Projekt mit einem Spring Boot-Backend erstellen, während man mit dem Spring Initializer ein allgemeines Spring Boot-Projekt generiert und dann manuell die Vaadin-Abhängigkeiten hinzufügen muss.

2.6 Liquibase

Dank Versionsverwaltung wie Git kann man in den meisten Projekten den Weg einer Code-Änderung von der der Entwicklung über Test bis hin zum produktiven Deployment richtig verfolgen und nachvollziehen. Was für den Code gilt, sollte auch für die Datenbankanpassungen gelten. Liquibase ist eine Bibliothek um Änderungen an einem Datenbankschema verfolgen, verwalten und anwenden zu können. Mit folgendem Eintrag in `pom.xml` kann man in ein Spring-Projekt integrieren.

```
<dependency>
    <groupId>org.liquibase</groupId>
    <artifactId>liquibase-core</artifactId>
    <version>4.23.1</version>
</dependency>
```

Für die Speicherung der Daten wird die PostgreSQL-Datenbank entschieden. PostgreSQL selbst eine leistungsstarke, offene Datenbank, die kompatibel zu Liquibase passt. Die Aktivierung der Datenbank wird in der Datei `application.properties` innerhalb der Spring-Applikation wie folgt konfiguriert:

```
spring.datasource.url=jdbc:postgresql://192.168.252.140:5432/
   opendataconn_ng
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.username=opendataconn_ng_usr
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.liquibase.change-log=classpath:/db/changelog-root.xml
```

Für die Verwendung von Liquibase sind folgende Dateien notwendig:

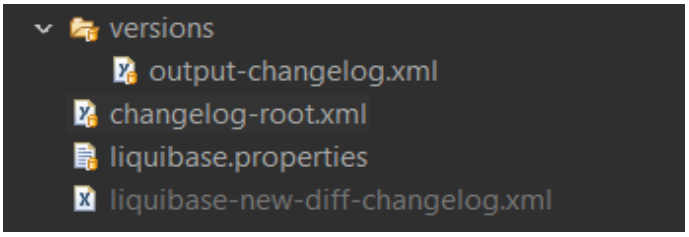

Abbildung 2.2: Liquibase-Konfiguration

1. Liquibase Konfiguration - liquibase.properties

In dieser Datei werden allgemeine Einstellungen wie der Datenbankzugang und die zu verwendenden Changelog-Konfiguration festgelegt.

```
# Database Configuration
url=jdbc:postgresql://192.168.252.140:5432/opendataconn_ng
username=opendataconn_ng_usr
password=
driver=org.postgresql.Driver
# Reference Configuration
referenceUrl=hibernate:spring:de.ahu.opendata?dialect=org.hibernate.
   dialect.PostgreSQLDialect
referenceDriver=liquibase.ext.hibernate.database.connection.
   HibernateDriver
# Output Files
changeLogFile=src/main/resources/db/changelog-root.xml
diffChangeLogFile=src/main/resources/db/liquibase-new-diff-changelog.xml
outputChangeLogFile=src/main/resources/db/versions/output-changelog.xml
```

2. Changelog Konfiguration - changelog-root.xml

Ein so genannter Root-ChangeLog wird erstellt, welcher in XML geschrieben ist und eine output-changelog.xml inkludiert, welche im Ordner „versions" zu finden ist. Diese Datei sorgt dafür, dass Liquibase das ChangeSet findet.

```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog
   xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:pro="http://www.liquibase.org/xml/ns/pro"
```

```
    xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
       http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-4.23.xsd
       http://www.liquibase.org/xml/ns/pro
       http://www.liquibase.org/xml/ns/pro/liquibase-pro-4.23.xsd">
     <includeAll path="db/versions"/>
</databaseChangeLog>
```

3. Changelog Konfiguration - output-changelog.xml

Diese Datei enthält sämtliche Datenbankänderungen und die Änderungen werden in Form von ChangeSet beschrieben und verwaltet.Im Folgenden ist ein Abschnitt im XML-Format für das Anlagen einer Tabelle Abonnement wiedergegeben.

```
<changeSet author="ng (generated)" id="1743435452264-1">
    <createTable tableName="abonnement">
        <column name="id" type="VARCHAR(32)">
            <constraints nullable="false" primaryKey="true"
               primaryKeyName="abonnementPK" />
        </column>
        <column name="description" type="TEXT" />
        <column name="label" type="VARCHAR(255)">
            <constraints nullable="false" />
        </column>
        <column name="base_url" type="VARCHAR(255)" />
        <column name="file_format" type="VARCHAR(255)" />
        <column name="end_datum" type="date" />
        <column name="location_id" type="VARCHAR(255)" />
        <column name="parameter" type="VARCHAR(255)" />
        <column name="start_datum" type="date" />
        <column name="sub_url" type="VARCHAR(255)" />
    </createTable>
</changeSet>
```

4. Changelog Konfiguration - liquibase-new-diff-changelog.xml

Diese Datei enthält nur bestimmte Änderungen von bestimmten Tabellen, die man in die Datei `output-changelog.xml` einfügt, wenn man das Schema der Datenbank modifiziert. Ähnlich wie in `output-changelog.xml` werden Änderungen auch in Form von ChangeSet beschrieben.

2.7 Datenbankmanagementsystem

Wie bereits in dem letzten Abschnitt offengelegt, wird ein PostgreSQL als das Datenbankmanagementsystem gewählt. Da alle Datenbanken von Ahu GmbH zentral im lokalen Netzwerk liegen und die meisten Projekte PostgreSQL-Datenbank verwenden, ist es deswegen ersichtlich, dass für dieses Projekt auch eine PostgreSQL-Datenbank verwendet wird.
Außerdem spricht der Einsatz einer PostgreSQL-Datenbank aufgrund ihrer zahlreichen Vorteile für sich.

1. Open-Source: geringe Kosten und hohe Flexibilität und Innovation, die bei anderen Datenbanklösung nicht immer möglich sind.

2. Leistung und Skalierbarkeit : PostgreSQL unterstützt eine Vielzahl von Leistungsoptimierung und verfügt über eine hohe Lese-/Schreibgeschwindigkeit, insbesondere bei der Unterstützung von Geodaten.

3. Fundierte Sprachunterstützung : Aufgrund seiner Kompatibilität und Unterstützung mehrerer Programmiersprachen ist PostgreSQL eine der flexibelsten Datenbanken für Entwickler. Python, JavaScript, C/C++, Java und weitere beliebte Programmiersprachen bieten ausgereifte Unterstützung für PostgreSQL.

3 Anforderungsanalyse und Spezifikation

3.1 Analyse

Während der Analysephase werden die ***Stakeholder*** identifiziert und deren Anforderungen mittels ***User Stories*** gesammelt. Die Anforderungen können in drei verschiedenen Untergruppen eingeteilt werden, um die systematisch und spezifisch behandeln zu können.

1. Funktionale Anforderungen beschreiben die Funktionen, die eine ganze Anwendung oder auch nur eine von ihren Komponenten erfüllen soll. Eine Funktion besteht aus drei Schritten: Eingabe der Daten-Systemverhalten-Ausgabe der Daten. Sie kann die Daten berechnen und manipulieren, Geschäftsprozesse ausführen, Benutzerinteraktionen herstellen oder andere Aufgabe ausführen.

2. Nicht-funktionale Anforderungen beschreiben, wie das System es tut, während die funktionalen Anforderungen bestimmen, was das System tut. Darunter zählen die Leistungsstandards und Qualitätsmerkmale von Software, z.B. die Benutzerfreundlichkeit, Effektivität, Sicherheit, Skalierbarkeit usw.

3. Randbedingungen sind Vorgaben, die den Lösungsraum einschränken und das Verhalten der Software beeinflussen.

3.1.1 Anwendungsszenario

Die zu entwickelnde Applikation soll als zentrale Stelle für die Durchsuche der Geodaten und Verwaltung der relevanten Geodaten dienen und ist auch in der Lage, diese Geodaten durch eine Schnittstelle einen anderen Client zur Verfügung stellen zu können.

3.1.2 Stakeholder

In der Softwareentwicklung sind Stakeholder Personen, Gruppen oder Organisation, die ein Interesse am Erfolg eines Softwareprojekts haben und von dessen Ergebnissen direkt oder indirekt beeinflusst werden.

3.1.3 User-Stories

Mittels User-Stories können funktionale Anforderungen der Stakeholder erfasst werden.

1. Als Benutzer möchte ich in der Web-Oberfläche eine Übersicht über populäre Datenquellen sehen, die relevante Geodaten bereitstellen.
2. Als Benutzer möchte ich mithilfe von Schlagwörtern nach OGC-Webservices durchsuchen und die entsprechenden Referenzen als URL-Links zu den Schlagwörtern erhalten.
3. Als Benutzer möchte ich gezielt nach Messdaten wie Grundwasserständen, Pegelständen, Niederschlag, etc. recherchieren und dabei Informationen über das Datenformat und wesentliche Inhalte der Datensätze erhalten.
4. Als Benutzer möchte ich Messdaten, die meist im CSV-Format oder individuell strukturierte ASCII-Dateien vorliegen, in ein passendes Zielformat konvertieren lassen.
5. Als Benutzer möchte ich die Daten in Form einer List von Datenstreams angezeigt bekommen.
6. Als Benutzer möchte ich in der Lage sein, bestimmte Datensätze abonnieren zu können.
7. Als Benutzer möchte ich die gefundenen Geo- und Messdaten visuell auf einer Karte darstellen lassen.

3.2 Spezifikation

Mit den im letzten Abschnitt erhobenen User-Stories können nun die funktionalen Anforderung genauer spezifiziert werden. Dort werden Use-Cases genauer definiert.

3.2.1 Use-Cases

Einfach ausgedrückt wird mit einem Use-Case die nach außen sichtbare Interaktion eines Nutzers mit einem System dokumentiert. Dieser Nutzer kann entweder eine Person, eine Organisation oder ein anderes System sein.
Für die prototypische Anwendung werden folgenden Use-Cases ausgearbeitet.

Use-Case 1: Historische Wetterdaten von Wetterdatendienst einsehen

Akteur : Benutzer
Ziel: Historische Wetterdaten einer bestimmten Messstation einsehen
Vorbedingung: keine

Nachbedingung: keine
Ablauf:

1. Benutzer navigiert zu der Historische Wetterdaten - Seite
2. Am Anfang zeigt die Seite einige Dropdowns zur Durchsuche der Daten und eine Karte, auf der noch keine Station zu sehen ist
3. Benutzer wählt die gewünschte Auflösung aus (stündlich, täglich, monatlich, jährlich)
4. Benutzer wählt dann die Hauptparameter aus und ein Dropdown für Unterparamter blendet sich ein.
5. Benutzer wählt anschließend die Unterparamter aus.
6. Das System reagiert auf den Wert von Unterparamter und zeigt sich eine deutschlandweite Karte, wo alle relevanten Messstationen zu dem ausgewählten Unterparamter zu sehen sind.
7. Weitere Dropdowns werden danach eingeblendet, entweder möchte Benutzer die Messstationen in einem bestimmten Bundesland sehen oder direkt eine gezielte Messstation auswählen
8. Nach Auswahl einer Station wird die Station auf der Karte fokussiert und die Daten zur gewählten Station als Diagramm dargestellt.

Use-Case 2: Wettervorhersagedaten von Wetterdatendienst einsehen

Akteur : Benutzer
Ziel: Vorhersagedaten einer bestimmten Messtation einsehen
Vorbedingung: keine
Nachbedingung: keine
Ablauf:

1. Benutzer navigiert zu der Wettervorhersage - Seite
2. Die Seite stellt bereits eine Tabelle mit verschiedenen Parameter zum Ansehen der Wetterdaten.
3. Benutzer wählt eine Station aus dem Dropdown aus.
4. Nach der Auswahl erscheint automatisch ein Button zur Überprüfung der Verfügbarkeit der Wetterdaten.
5. Benutzer klickt auf dieses Button und wartet auf die Meldung des Systems.

6. Das System überprüft all in der Grid befindliche Parameter,ob jeweiliger Parameter überhaupt Daten liefert.
7. Anhand des Status,ob es die Daten zu jeweiligen Parameter gibt, wird es in der Grid ein weitere Spalte zur Einsicht der Daten geben.
8. Benutzer klickt auf das Icon in der neu erscheinenden Spalte.
9. Ein Fenster öffnet sich und stellt die Daten in Form eines Diagramm bereits.

Use-Case 3: Pegelstanddaten von PegelOnline einsehen

1. Benutzer navigiert zu der Pegelständedaten - Seite
2. Benutzer sieht eine Karte mit Pegelstationen deutschlandweit, die je nach aktueller Lage farblich kategorisiert sind.
3. Inhalt...

Use-Case 4:

4 Konzeption und Architektur

5 Implementierung

6 Evaluierung

7 Fazit und Ausblick