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 Bereistellung 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 12. 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 Überpüfung 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 strukturtierte 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 kompatibeles 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-Anwedungen, die es Dienstleitern, Betreibern und Aufsichtsbehördern ermöglichen, ein kosteneffizientes Management für Geodaten (Grundwassser,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 der Applikation enthalten eine übergreifende Suche von Geodaten und deren Umformatierung in ein passendes vorgegebenes Zielformat. Das weitere Zeil ist die Einbindung von mindenstens einer OGC-Geodatendiensten.

Durch die Umsetzungen dieser Ziele soll perspektivisch eine Software entstehen, die die offenen Problematik abdeckt.

2 Übersicht von Technologien

2.1 Vaadin

Vaadin ist ein serverseitiges Web-Framework, das dem Entwickler erlaubt, moderne Webanwendung in Java zu entwicklen, 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.

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.

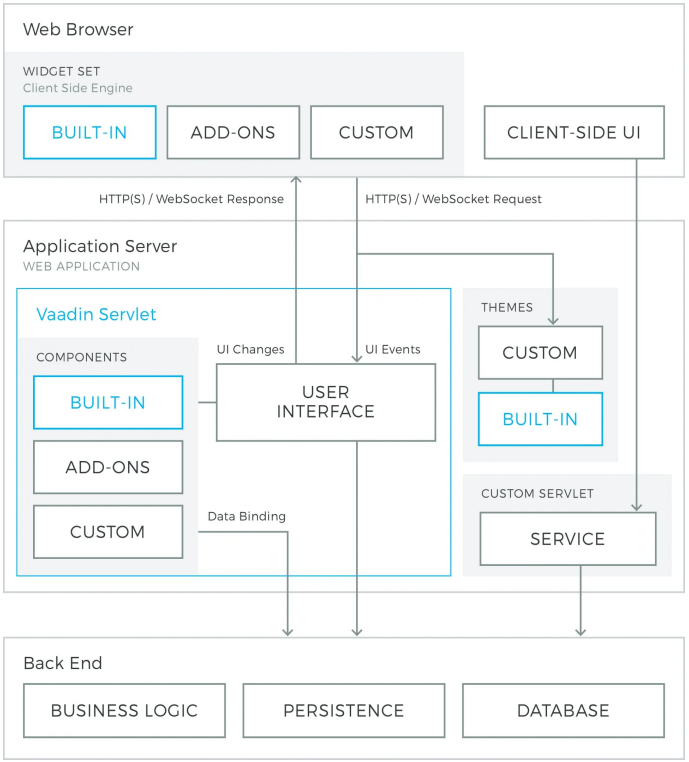

Abbildung 2.1: Übersicht Vaadin Architektur **architecture21**

2.2 Spring Framework

Spring Framework ist ein Java-Framework, das um 2003 als Reaktion auf die damals noch zu komplizierte J2EE-Platform 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 eigennen 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 konsutruieren 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 baisert auf einem deklaravitven Ansatz. Das bedeutet, dass der Entwickler alle relevanten Informationen über das Projekt in einer zentralen Konfigurartionsdatei, 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.
- Anhängigkeiten: 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 als Einstieg

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 anweden 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:

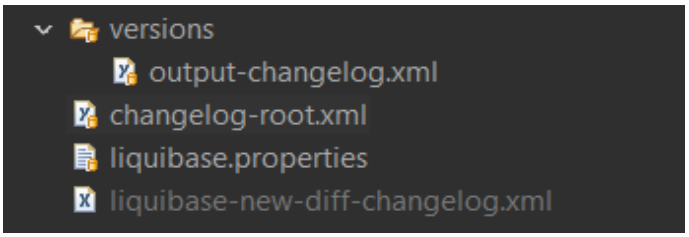

Abbildung 2.2: Liquibase-Konfiguration

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 Folgnden ist ein Abschniit 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. Änhlich wie in `output-changelog.xml` werden Änderungen auch in Form von ChangeSet beschrieben.

3 Anwendungsarchitektur