Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Informatik

SWDE – Software-Entwicklung

Fallstudie

Applikationsdokumentation

Gruppe : 2

1. Kilchherr Raphael
2. Michel Philippe
3. Lötscher Till
4. Geissmann Robin

Gruppensprecher: Robin Geissmann

Version: 2.0

Datum: 14.12.2020

Inhaltsverzeichnis

1 Einführung

Dieses Dokument beschreibt eine Software, welche Wetterdaten über einen Web-Service zur Verfügung stellt. Die Daten werden von der Applikation abgerufen, bearbeitet und anschliessend in einer Datenbank abgelegt. Die Abfragen werden danach von der Datenbank abgerufen. Zu einem späteren Zeitpunkt wird die Applikation in ein Client-Server Modell umgewandelt.

Funktionen der Wetterdaten-App

Business:

Im Business werden die Anfragen verarbeitet. Anschliessend werden diese an den Persister weitergeleitet.

Hier befindet sich auch die Reader-Einheit, welche die Daten von der Rest Api an die Persister Unit übergibt

Domain:

In der Domain werden die Daten des Web-Services in Objekte (z.B. Wetterdatenobjekte) umgewandelt.

Persister:

Der Persister kommuniziert mit der Datenbank und speichert die Objekte darin ab.

RMI:

Mithilfe der RMI wird die Möglichkeit geschaffen, die Applikation zum Client - Server

UI:

Mit dem UI kann der Benutzer die gewünschten Wetterdaten (Ortschaft und Zeitperiode) abrufen und lokal speichern.

2 Anforderungen

Die Applikation soll folgende Abfragen durchführen:

A01	Für welche Ortschaften stehen die Wetterdaten zur Verfügung?
A02	Wie sehen die Temperatur, Luftdruck und Feuchtigkeit für eine angegebene Ortschaft während einer angegebenen Zeitperiode aus?
A03	Wie gross waren die Durchschnittswerte für Temperatur, Luftdruck und Feuchtigkeit für die angegebene Ortschaft in einer angegebenen Zeitperiode?
A04	Wie sahen die Maximal - und Minimalwerte für Temperatur, Luftdruck und Feuchtigkeit für eine angegebene Ortschaft während einer angegebenen Zeitperiode aus?
A05	Wie sahen die Maximal - und Minimalwerte für Temperatur, Luftdruck und Feuchtigkeit über alle Ortschaften zu einem bestimmten Zeitpunktaus3?
A06	Verwaltung von Daten der eigenen Benutzer: CRUD (Create, Read, Update, Delete).
A07	Export von bestellten Daten (gemäss den Abfragen in der obigen Tabelle) in einem passenden Format (z.B. xml, csv, json, pdf etc.)
A08	Der Zugriff auf die Applikation muss mit einem Login geschützt werden

.

3 Architektur

3.1 Module / Komponenten

Hier sollen die wesentlichen Module / Komponenten mit deren Zuständigkeiten und Beziehungen zu anderen Modulen / Komponenten mit Hilfe eines passenden Diagramms angegeben werden[1]. Daraus sollen sowohl die von einem Modul offerierten Schnittstellen (*provided interfaces*) als auch die von einem Modul benötigten Schnittstellen (*required interfaces*) ersichtlich sein.

Siehe Diagramm Folgeseite

[1] Hier könnte beispielsweise ein Komponentendiagramm sinnvoll verwendet werden. Eine einfache Erläuterung kann an der folgenden Adresse gefunden werden: https://online.visual-paradigm.com/diagrams/tutorials/component-diagram-tutorial/

UI

Stub

a1(): List<Ortschaft>
a2(): List<Wetterdatensatz>
a3():Map<String,Float>
a4():Map<String,Float>
a5():Map<String,Float>
a8():BenutzerVerwaltungImpl

Main(UI)

-Stub
main()
eingabeOrtschaft(): String
startDatumIsValid(): Date
endDatumIsValid(Date): Date
wetterdatenAusgabe()
durchschnittswerte()

Export

toFile(): File
mapWetterdatensatzToJSON():String
mapFloatToJSON(): String
listToJSON(): String

Domain

Benutzer

-id: int
-userName: String
-email: String
-password: String

get()
set()
(für alle param)

Wetterdatensatz

-id: int
-temperatur: float
-feuchtigkeit: float
-luftdruck: float
-datum: Date
-ortschaft: Ortschaft

get()
set()
(für alle param)

Ortschaft

-id: int
-name: String
- plz : int

get()
set()
(für alle param)

Business

Server

-BusinessLogikImpl

main()

<<Schnittstelle>>
Businesslogik

a1(): List<Ortschaft>
a2(): List<Wetterdatensatz>
a3():Map<String,Float>
a4():Map<String,Float>
a5():Map<String,Float>
a8():BenutzerVerwaltungImpl

BusinesslogikImpl

-JPAPersister

a1(): List<Ortschaft>
a2(): List<Wetterdatensatz>
a3():Map<String,Float>
a4():Map<String,Float>
a5():Map<String,Float>
a8():BenutzerVerwaltungImpl

AdminInterface

- Scanner
-OrtschaftsReader
WetterdatensatzReader

main()
fetchOrtschaften()
fetchWetterDaten()
deleteWetterDaten()
createUser()
changePassword()
deleteUser()

OrtschaftsReader

-JPAPersister

main()
parseOrtschaften(String): List<Ortschaften>

WetterdatenReader

-JPAPersister

main()
parseWetterdaten(): List<Wetterdatensatz>
wetterdatensatzRESTUrl(String, String)
createWetterdatensatz(String)

Persister

JPAPersister

-mitgliedsName

speichern():Wetterdatensatz
speichernListe():List<Wetterdatensatz>
speichernListeOrtschaft():List<Ortschaft>
aktualisieren(): Wetterdatensatz
loeschen(Wetterdatensatz): boolean
loeschen(ID): boolean
findeWetterdatensatzByOrt(String):Wetterdatensatz
findeWetterdatensatzById(**int**):Wetterdatensatz
findeZwischenDatum(String , Date , Date) List<Wetterdatensatz>
alle():List<Wetterdatensatz>
alleByDatum(Date):List<Wetterdatensatz>
alleOrtschaften():List<Ortschaft>
findeOrtschaftByName(String):Ortschaft

<<Schnittstelle>>
Persister

speichernWetterdatensatz(Wetterdatensatz): Wetterdatensatz
speichernListeWetterdatensatz(List<Wetterdatensatz>): boolean
speichernListeOrtschaft(List<Ortschaft>): boolean
aktualisierenWetterdatensatz(Wetterdatensatz): Wetterdatensatz
loeschenWetterdatensatz(Wetterdatensatz): boolean
loeschenAlleWetterdatensaetze(): boolean
loeschenWetterdatensatzById(ID): boolean
findeWetterdatensatzByOrt(String, Date):List <Wetterdatensatz>
findeWetterdatensatzById(int):Wetterdatensatz
findeWetterdatensatzZwischenDatum(String , Date , Date): List<Wetterdatensatz>
alleWetterdatensatz():List<Wetterdatensatz>
alleWetterdatensatzByDatum(Date):List<Wetterdatensatz>
alleOrtschaften():List<Ortschaft>
findeOrtschaftByName(String):Ortschaft
speichernBenutzer(Benutzer): boolean
loeschenBenutzer(Benutzer): boolean
aktualisierenBenutzer(Benutzer): boolean
getBenutzerbyName(String): Benutzer

3.2 Schichten und Module

Hier sollen die Zusammenhänge zwischen Schichten und Modulen / Komponenten passend dargestellt werden. Daraus soll die Zugehörigkeit von Modulen / Komponenten zu den entsprechenden Schichten der Applikation ersichtlich sein.

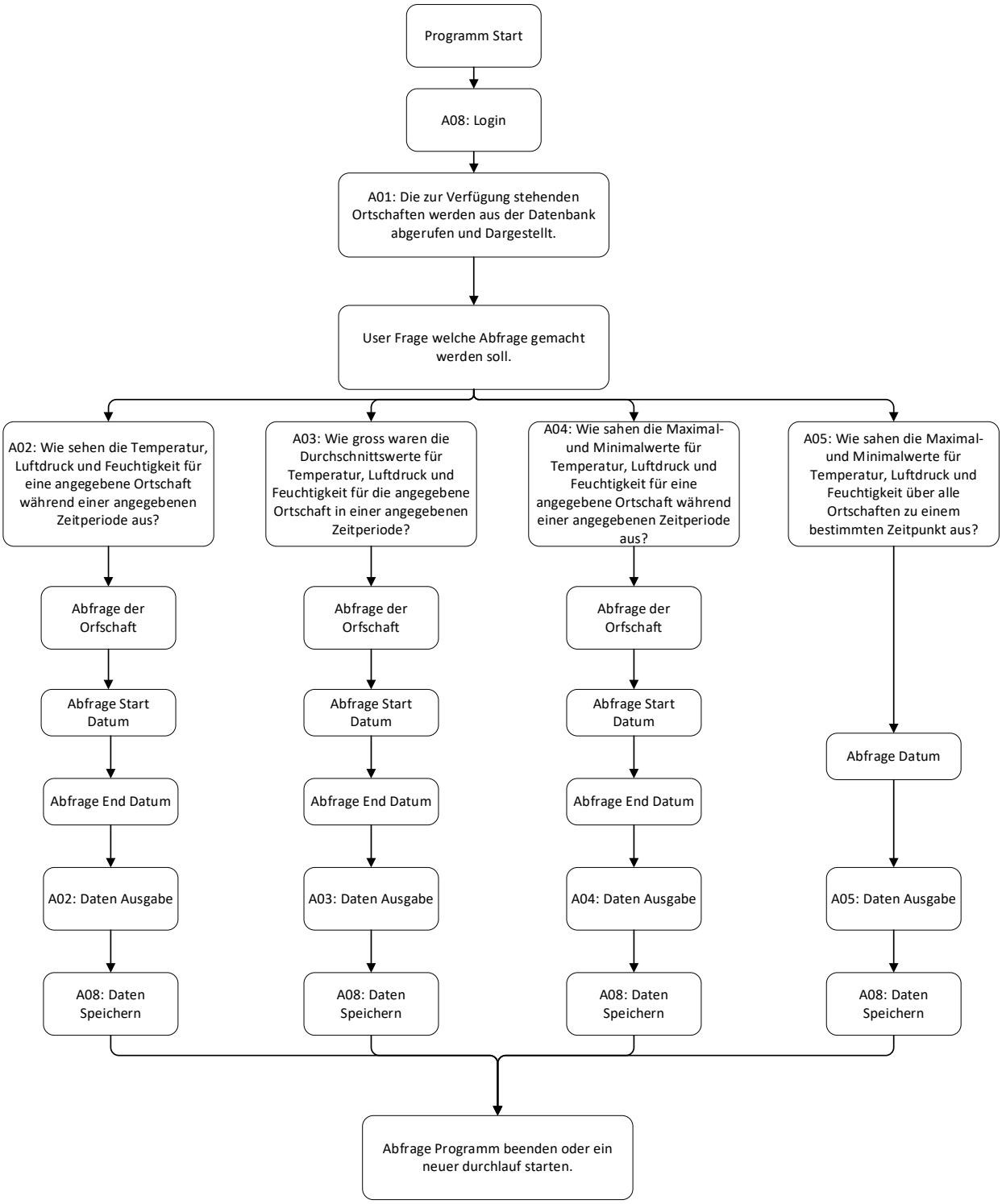
Siehe Diagramm Vorderseite

4 Datenstrukturen

Hier sollen die Klassen (als Klassendiagramm) aufgeführt werden, die als Entities[2] verwendet werden (und nur diese Klassen). Aus dem Diagramm sollen die Beziehungen unter den einzelnen Entity-Klassen ersichtlich werden. Um das Diagramm übersichtlich zu halten kann auf die Abbildung von Konstruktoren und set- und get-Methoden verzichtet werden: Die Angabe von Attributen (Instanzvariablen) reicht aus.

[2] Klassen, deren Instanzen in der Datenbank verwaltet werden.

5 User Interface



6 Interfaces

In diesem Kapitel werden die Business-Interfaces aufgeführt und kurz beschreiben. Daraus soll ersichtlich sein, wozu das Interface benötigt wird und welche Funktionalitäten das Interface vorgibt. Die Abbildung von einzelnen Interfaces soll in UML Form erfolgen (kein Java-Code verlangt).

6.1 Businesslogik

6.1.1 a1(): List<Ortschaft>

Die Methode a1() ruft mit dem jpaPersister alle Ortschaften auf, die in der Datenbank gespeichert sind. Die Ortschaften werden in die `List<Ortschaft> listReturn` gespeichert. Das UI verwendet den `return listReturn` um die Ortschaften, die in der Datenbank gespeichert sind auszugeben.

6.1.2 a2(): List<Wetterdatensatz>

Die Methode a2() ruft mit dem jpaPersister alle Wertterdaten für eine bestimmte Ortschaft für eine bestimmte Zeitperiode auf. Der Methode muss die Ortschaft als String, das Start Datum als Date und das Schluss Datum als Date übergeben werden. Diese Parameterliste wird dem `jpaPersister.findeWetterdatensatzZwischenDatum` übergeben, welcher alle Wetterdaten für die übergebene Ortschaft in der angegebenen Zeit auflistet. Diese werden dann in die `List<Wetterdatensatz> listReturn` gespeichert. Das UI verwendet den `return listReturn` um für die Ortschaft in der gewünschten Zeitangabe die Wetterdaten auszugeben.

6.1.3 a3(): Map<String,Float>

Die Methode a4() gibt die durchschnittlichen Wertterdaten (Temperatur, Luftdruck und Luftfeuchtigkeit) für eine bestimmte Ortschaft und eine bestimmte Zeitperiode aus. Der Methode muss die Ortschaft als String, das Start Datum als Date und das Schluss Datum als Date übergeben werden. Diese Parameterliste wird dem `jpaPersister.findeWetterdatensatzZwischenDatum` übergeben, welcher alle Wetterdaten für die übergebene Ortschaft in der angegebenen Zeit in die `List<Wetterdatensatz> inputList` speichert. Mit einer For Schleife werden die Wetterdaten addiert. Die durchschnitt Wetterdaten werden in die `Map<String, Float> returnValues` gespeichert, in dem die addierten Wetterdaten durch die Grösse der `List<Wetterdatensatz> inputList` geteilt werden. Das UI verwendet den `return returnValues` um für die Ortschaft in der gewünschten Zeitangabe die durchschnittlichen Wetterdaten auszugeben.

6.1.4 a4(): Map<String,Float>

Die Methode a4() gibt die minimale und maximale Wertterdaten (Temperatur, Luftdruck und Luftfeuchtigkeit) für eine bestimmte Ortschaft und eine bestimmte Zeitperiode aus. Der Methode muss die Ortschaft als String, das Start Datum als Date und das Schluss Datum als Date übergeben werden. Diese Parameterliste wird dem `jpaPersister.findeWetterdatensatzZwischenDatum` übergeben, welcher alle Wetterdaten für die übergebene Ortschaft in der angegebenen Zeit in die `List<Wetterdatensatz> inputList` speichert. Aus der `List<Wetterdatensatz> inputList` werden die maximal und minimal Werte ausgelesen und in die `Map<String, Wetterdatensatz> returnValues` gespeichert. Das UI verwendet den `return returnValues` um für die Ortschaft in der gewünschten Zeitangabe die maximal und minimal Wetterdaten auszugeben.

6.1.5 a5(): Map<String,Float>

Die Methode a5() gibt die minimale und maximale Wertterdaten (Temperatur, Luftdruck und Luftfeuchtigkeit) für alle Ortschaften in der Datenbank und einem bestimmten Zeitpunkt (Datum) aus. Der Methode muss die Ortschaft als String und das Datum als Date übergeben werden. Der Parameter Zeitpunkt wird dem `jpaPersister.alleWetterdatensatzByDatum` übergeben, welcher alle Wetterdaten für alle Ortschaften für den bestimmten Zeitpunkt in die `List<Wetterdatensatz> inputList` speichert. Aus der `List<Wetterdatensatz> inputList` werden die maximal und minimal Werte ausgelesen und in die `Map<String, Wetterdatensatz> returnValues` gespeichert. Das UI verwendet den `return returnValues` um für alle Ortschaften für den bestimmten Zeitpunkt die maximal und minimal Wetterdaten auszugeben.

6.1.6 A8(): BenutzerVerwaltungImpl

Die Methode a8() prüft, ob der Benutzer und das Passwort korrekt sind. Der Methode muss der Benutzername und das Passwort übergeben werden.

6.2 Persister

<<Schnittstelle>>
Persister

speichernWetterdatensatz(Wetterdatensatz): Wetterdatensatz
speichernListeWetterdatensatz(List<Wetterdatensatz>): boolean
speichernListeOrtschaft(List<Ortschaft>): boolean
aktualisierenWetterdatensatz(Wetterdatensatz): Wetterdatensatz
loeschenWetterdatensatz(Wetterdatensatz): boolean
loeschenAlleWetterdatensaetze(): boolean
loeschenWetterdatensatzById(ID): boolean
findeWetterdatensatzByOrt(String, Date):List <Wetterdatensatz>
findeWetterdatensatzById(int):Wetterdatensatz
findeWetterdatensatzZwischenDatum(String , Date , Date): List<Wetterdatensatz>
alleWetterdatensatz():List<Wetterdatensatz>
alleWetterdatensatzByDatum(Date):List<Wetterdatensatz>
alleOrtschaften():List<Ortschaft>
findeOrtschaftByName(String):Ortschaft
speichernBenutzer(Benutzer): boolean
loeschenBenutzer(Benutzer): boolean
aktualisierenBenutzer(Benutzer): boolean
getBenutzerbyName(String): Benutzer

6.2.1 speichernWetterdatensatz(Wetterdatensatz): Wetterdatensatz

Liest die übergebenen Wetterdaten ein und speichert diese.

6.2.2 speichernListeWetterdatensatz(List<Wetterdatensatz>): boolean

Liest die übergebenen Wetterdaten ein und speichert diese.

6.2.3 speichernListeOrtschaft(List<Ortschaft>): boolean

Liest die übergebenen Ortschaften ein und speichert diese.

6.2.4 aktualisierenWetterdatensatz(Wetterdatensatz): Wetterdatensatz

Diese Methode aktualisiert ein Wetterdatenobjekt.

6.2.5 loeschenWetterdatensatz(Wetterdatensatz): boolean

Diese Methode löscht ein Wetterdatenobjekt.

6.2.6 loeschenAlleWetterdatensaetze(): boolean

Diese Methode löscht alle Wetterdatensätze.

6.2.7 loeschenWetterdatensatzById(ID): boolean

Diese Methode löscht ein Wetterdatenobjekt anhand seiner Id.

6.2.8 findeWetterdatensatzByOrt(String, Date):List <Wetterdatensatz>

Diese Methode gibt eine Liste aller zutreffenden Ergebnisse der Abfrage zurück.

6.2.9 findeWetterdatensatzById(int):Wetterdatensatz

Diese Methode gibt das gesuchte Wetterdatenobjekt anhand der Id zurück.

6.2.10 findeWetterdatensatzZwischenDatum(String , Date , Date): List<Wetterdatensatz>

Diese Methode gibt die gesuchte Wetterdatenobjekte anhand des Datums zurück.

6.2.11 alleWetterdatensatz():List<Wetterdatensatz>

Diese Methode gibt eine Liste aller Wetterdaten zurück.

6.2.12 alleWetterdatensatzByDatum(Date):List<Wetterdatensatz>

Diese Methode gibt eine Liste aller Wetterdaten an einem bestimmten Zeitpunkt zurück.

6.2.13 alleOrtschaften():List<Ortschaft>

Diese Methode gibt eine Liste aller Ortschaften zurück.

6.2.14 findeOrtschaftByName(String):Ortschaft

Diese Methode überprüft die übergebene Ortschaft mit den Ortschaften in der Datenbank und gibt die Ortschaft aus der Datenbank zurück.

6.2.15 speichernBenutzer(Benutzer): boolean

Mit dieser Methode kann ein Benutzer in der Datenbank gespeichert werden und gibt bei erfolgreichem Speichern ein boolean true zurück.

6.2.16 loeschenBenutzer(Benutzer): boolean

Mit dieser Methode kann ein Benutzer in der Datenbank gelöscht werden und gibt bei erfolgreichem Löschen ein boolean true zurück.

6.2.17 aktualisierenBenutzer(Benutzer): boolean

Mit dieser Methode kann ein Benutzer in der Datenbank aktualisiert werden und gibt bei erfolgreichem Aktualisieren ein boolean true zurück.

6.2.18 getBenutzerbyName(String): Benutzer

Mit dieser Methode kann ein Benutzer aus der Datenbank abgerufen werden und gibt den Benutzer zurück.

7 Testen

In diesem Kapitel soll das Testkonzept beschrieben und auf die im Code erstellten Tests hingewiesen werden. Daraus soll ersichtlich werden, wie und was getestet wurde (Unit Tests, Integrationstests).

RMI:

Da RMI fortlaufend mit der effektiven Umgebung verknüpft ist, haben wir die Entwicklung und Test direkt im Code durchgeführt.

UI:
Für das Ui wurden diverse Eingaben von Ortschaften und Datumsabfragen getestet. Dies garantiert die das beim Eingeben nichts schief läuft.

BusinessLogik:

Das Implementieren der Businesslogik wurde in der Entwicklungsumgebung direkt an der TestPu umgesetzt. Allfällige Fehler wurden direkt im Code umgesetzt

Persistence Test:

Der Test für den Persister wurden für den anfänglichen Verbindungsaufbau mittels Unit und IT-Test umgesetzt.
Anschliessend wurden “Test” in der Testumgebung/Testdatenbank/TestPU ohne Testdateien durchgeführt. Die verschiedenen Fortschritte sind daher nur in der History (Gitlab) der Dateien des Persistence Moduls zu sehen.

8 Verteilung (Deployment)

Server:

G02-wda-business

Business api

Domain

Persister

Rmi

Launchpoint is ch.hslu.swde.wsa.rmi.server

Client:

Business-api

domain

Ui

Launchpoint ch.hslu.swde.ui.UI

ServerAdmin:

Business

Business api

Domain

Persister

Launchpoint ch.hslu.swde.wda.business.adminInterface.AdminInterface

Im RMI-Modul befindet sich der Server. Das Interface befindet sich im Business-Modul.

9 Konfigurationsangaben

In diesem Kapitel müssen alle nötigen Angaben enthalten sein, die ein Administrator für die Installation und Inbetriebnahme der Applikation benötigt. Insbesondere ist es wichtig, dass die Initialdaten nicht fehlen: Benutzername und Kennwörter, die am Anfang nötig sind, Angaben zu DBMS und Datenbank, Angaben von IP-Adressen und Ports, die in den einzelnen **config**-Dateien evtl. angepasst werden müssen usw. Gehen Sie davon aus, dass der Administrator die Applikation nicht kennt und bei der Installation und Inbetriebnahme der Applikation auf die Angaben von Ihnen angewiesen ist[3].

Zugang zur Datenbank:

Zugriff über das Netzwerk der HSLU (VPN-Verbindung) Diese werden in der persistence.xml Datei hinterlegt, damit die Verbindung zur Datenbank erfolgen kann.

Host: stud.el.eee.intern

Port 5432

Name der Datenbank: swde_user_hs20

Passwort: swde_user_hs20_user

Policy.policy File richtig ablegen

Falls beim erstmaligen Ausführen des Programmes das policy.policy nicht am Richtigen Ort geschrieben wird und eine Access denied Meldung ausgeworfen wird, legen Sie das policy.policy file unter C://Users/%Benutzername%/policy.policy

[3] Oder noch besser: Versuchen Sie die generierten Artefakte selbst auf einem neuen System zu installieren und dann werden Sie schnell sehen, welche Informationen / Angaben noch fehlen und was noch nicht klar angegeben ist.

10 Gemachte Erfahrungen (lessons learned)

In diesem Kapitel können die gemachten Erfahrungen, Höhen und Tiefen adressiert werden. Des Weiteren kann man angeben, was gut und was weniger gut gelaufen ist und was man im Nachhinein anders machen würde.

1. Abgabe

Die Zusammenarbeit ist auf Grund der verteilten Arbeitsplätze äusserst erschwert. Neuerungen und Anpassungen sind schwierig zu kommunizieren. Deshalb haben wir in der Zwischenabgabe zwar die Funktionalität erreicht, jedoch noch nicht die von uns angestrebte Sauberkeit des Codes.

2. Abgabe

Es ist schwierig bei der Migration von RMI von einer Testumgebung in die Aktivumgebung alles richtig zu verknüpfen. Daher hat es bei uns nicht im ersten Anlauf geklappt und es musste genaustens nachgeprüft werden.

3. API-Datenqualität

Man kann nicht immer davon ausgehen, dass die Datenqualität von APIs den benötigten Spezifikationen entspricht. Wir haben das am konkreten Beispiel vom Ortsnamen Biel/Bienne festgestellt.

Dies konnten wir durch Anpassungen an unserer Parserkomponente zwar beheben, hat uns aber doch erst etwas Kopfzerbrechen bereitet.

4. UI

Die Kommunikation zwischen dem UI und dem Persister mittels RMI hat gewisse Schwierigkeiten geboten, diese konnten aber mit aussenstehender Unterstützung beseitigt werden.

11 Tutorials

- Component Diagram Tutorial[4]
- Deployment Diagram Tutorial[5]

[4] https://online.visual-paradigm.com/diagrams/tutorials/component-diagram-tutorial/

[5] https://online.visual-paradigm.com/diagrams/tutorials/deployment-diagram-tutorial/