
Laborprotokoll

DEZSYS09

**Systemtechnik Labor
5BHITT 2015/16, Gruppe Z**

Jakub Kopec

Version 0.1

Note:

Betreuer: Michael Borko

Begonnen am 26. Februar 2016

Beendet am 4. März 2016

Inhaltsverzeichnis

1 Einführung.....	3
1.1 Ziele.....	3
1.2 Voraussetzungen.....	3
1.3 Aufgabenstellung.....	3
2 Ergebnisse.....	4
2.1 Umsetzung.....	4
2.2 Testung.....	5
Registrierung.....	5
Login.....	7
3 Quellen.....	9

1 Einführung

Diese Übung zeigt die Anwendung von mobilen Diensten in Java.

1.1 Ziele

Das Ziel dieser Übung ist eine Webanbindung zur Benutzeranmeldung in Java umzusetzen. Dabei soll sich ein Benutzer registrieren und am System anmelden können.

Die Kommunikation zwischen Client und Service soll mit Hilfe von JAX-RS (Gruppe1+2) umgesetzt werden.

1.2 Voraussetzungen

- ! Grundlagen Java und Java EE
- ! Verständnis über relationale Datenbanken und dessen Anbindung mittels JDBC oder ORM-Frameworks
- ! Verständnis von Restful Webservices

1.3 Aufgabenstellung

Es ist ein Webservice mit Java zu implementieren, welches eine einfache Benutzerverwaltung implementiert. Dabei soll die Webapplikation mit den Endpunkten /register und /login erreichbar sein.

Registrierung

Diese soll mit einem Namen, einer eMail-Adresse als BenutzerID und einem Passwort erfolgen. Dabei soll noch auf keine besonderen Sicherheitsmerkmale Wert gelegt werden. Bei einer erfolgreichen Registrierung (alle Elemente entsprechend eingegeben) wird der Benutzer in eine Datenbanktabelle abgelegt.

Login

Der Benutzer soll sich mit seiner ID und seinem Passwort entsprechend authentifizieren können. Bei einem erfolgreichen Login soll eine einfache Willkommensnachricht angezeigt werden.

Die erfolgreiche Implementierung soll mit entsprechenden Testfällen (Acceptance-Tests bez. aller funktionaler Anforderungen mittels JUnit) dokumentiert werden. Es muss noch keine grafische Oberfläche implementiert werden! Verwenden Sie auf jeden Fall ein gängiges Build-Management-Tool (z.B. Maven). Dabei ist zu beachten, dass ein einfaches Deployment möglich ist (auch Datenbank mit z.B. file-based DBMS).

2 Ergebnisse

2.1 Abstract

Für diese Labor-Übung wurde als Buildtool Maven verwendet. Das pom.xml – File wurde im Laufe der Implementierung erweitert. Als erstes wurde eine Entität des Benutzers erstellt, welcher sich bei diesem Webservice registrieren und anmelden können soll. Hierbei ist erwähnenswert, dass die Email eines Users als eindeutige id verwendet wurde. Für die weitere Umsetzung der REST-Schnittstelle wurde das Spring-Framework [1] benutzt. Dies bringt gleich mehrere Vorteile. Einerseits bietet Spring ein CRUD-Repository [2] an, was zur Folge hat, dass dieses in dieser Labor-Übung verwendet werden kann und somit die ganze CRUD-Funktionalität nicht selbst implementiert werden muss. Andererseits kann dank Spring Boot [3][5] die Applikation sehr einfach gestartet werden. Als Datenbank wurde H2 [4] gewählt, damit die Abhängigkeit zu externen Datenbanken vermieden wird. Gewählt wurde diese Datenbank aus dem Grund, dass sie im Maven-Repository enthalten ist und auch von Spring ein guter Support angeboten wird. Die Datenbank wird automatisch im Verzeichnis erstellt, in welchem das Programm ausgeführt wird. Sobald die Entität und die Datenbank feststanden mussten lediglich die zwei Endpoints /register und /login ausprogrammiert werden.

2.2 Definition einer User-Entity

Für den User muss eine Entity erstellt werden, welche in der Datenbank angelegt werden soll. Dazu wird die Klasse `User` mit der Annotation `@Entity` erstellt, wobei die Notwendigen Spalten als Attribute dargestellt werden.

```
@Entity
public class User {
```

Der User erhält die Email-Adresse als Primary-Key, was mit der Annotation `@Id` sichergestellt wird. Die Email-Adresse darf nie leer und sollte nicht länger als 50 Zeichen sein, weshalb die Annotationen `@NotEmpty` und `@Size` verwendet wurden.

```
    @Id
    @Size(max = 50)
    @NotEmpty
    private String email;
```

Ähnliche Anforderungen gelten auch für die Attribute „name“ und „password“, weshalb hier die selben Annotations verwendet wurden.

```
@Size(max = 50)
@NotEmpty
private String name;

@NotEmpty
@Size(min = 5)
private String password;
```

2.3 Persistierung der User Entity

Zur Persistierung wird H2 verwendet. Die Konfiguration erfolgt in der Datei „application.properties“ im Ressourcenordner. Diese schaut folgendermaßen aus:

```
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.url = jdbc:h2:file:./User;FILE_LOCK=FS
spring.datasource.username = dezs09
spring.datasource.password = dezs09
spring.jpa.hibernate.ddl-auto = update
```

Die Datenbank wird im Projektverzeichnis gespeichert. „ddl-auto=update“ stellt sicher, dass die Datenbank beim starten an die Entities angepasst wird.

2.4 Konfiguration von Jersey

Damit Jersey zusammen mit Spring verwendet werden kann, ist eine Konfigurationsklasse erforderlich, mit Hilfe welcher die RESTful Endpoints registriert werden.

```
@Named
public class JerseyConfig extends ResourceConfig {

    public JerseyConfig() {
        this.register(Register.class);
        this.register(Login.class);
        this.register(JacksonFeature.class);
    }
}
```

2.5 Definition des CRUD-Interfaces

Um Objekte persistieren zu können, oder abzurufen ist eine Schnittstelle erforderlich die das kann. Im Falle von Spring wird dafür ein `CrudRepository` zur Verfügung gestellt, welches alle CRUD-Funktionalitäten übernimmt. Das Objekt (in diesem Fall der User) wird zusammen mit der ID als generischer Parameter übergeben.

```
public interface CrudInterface extends CrudRepository<User,
String> {
}
```

2.6 Implementierung der Endpoints

Für die Endpoints `/register` und `/login` wurden jeweils Klassen erstellt. So schaut beispielsweise der Endpoint für die Registrierung aus. Mit der `@Path` Annotation kann der Pfad des Endpoints gesetzt werden. Weiters muss angegeben werden, dass JSON Objekte vom Endpoint produziert werden. Das wird durch die Annotation `@Produces` sichergestellt.

```
@Named
@Path("/register")
@Produces({MediaType.APPLICATION_JSON})
public class Register {
```

Damit das bereits erstellte CRUD-Interface verwendet werden kann, muss es vorher mit der Annotation `@Inject` versehen werden.

```
@Inject
private CrudInterface crudInterface;
```

Durch die `@POST` Annotation werden Nachrichten des Typ `Post` akzeptiert.

```
@POST
public Response register(User user) {

}
```

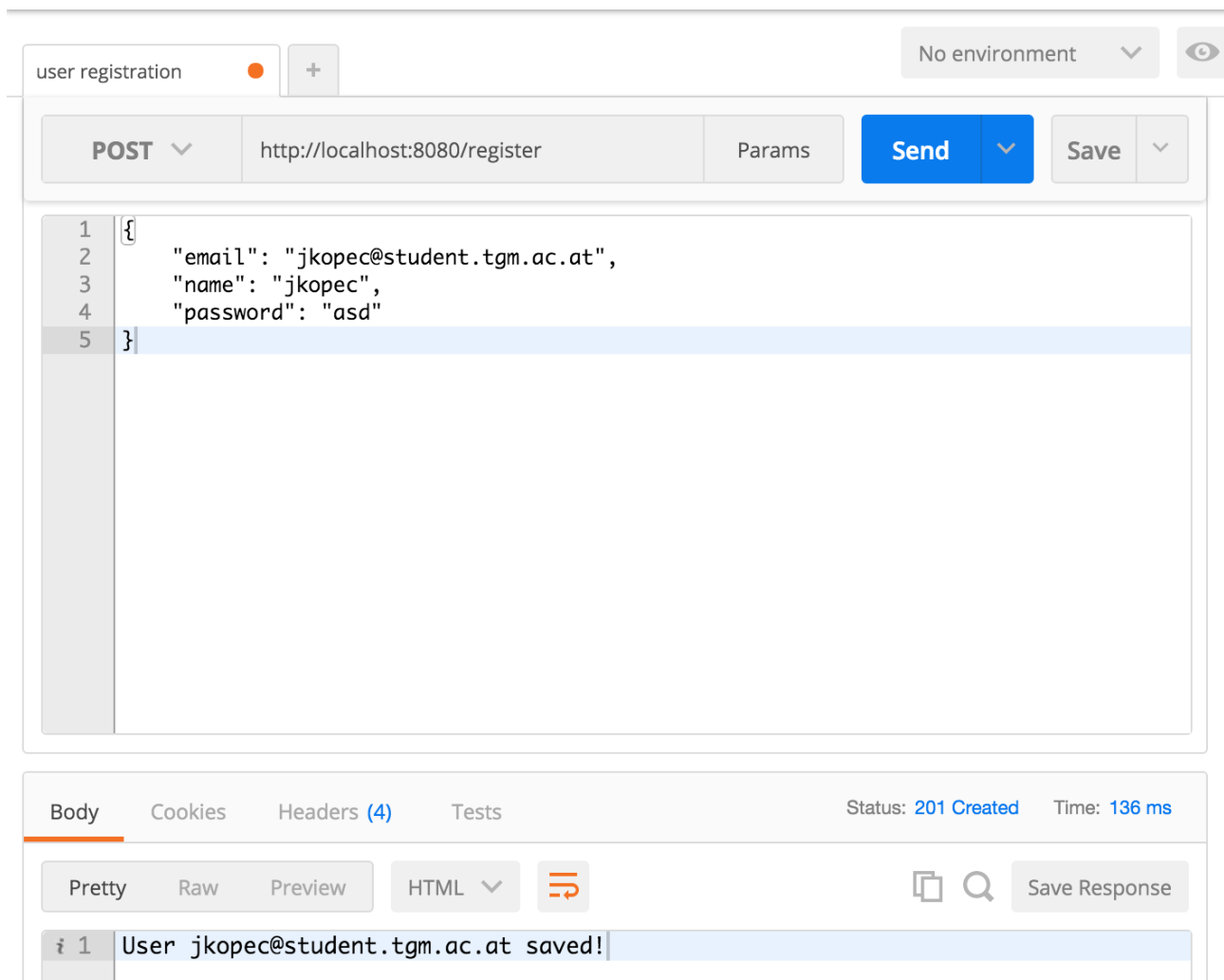
Der Endpoint zum Einloggen sieht von den Annotations her gleich aus, wobei sich die Annotation `@Path(„/login“)` von der Registrierung unterscheidet.

2.7 Testung

Bei der Implementierung wurde die Funktionalität mit dem Tool Postman, einer Google Chrome Erweiterung, getestet. In weiterer Folge wurden auch Junit-Tests erstellt.

Registrierung

Wurden alle nötigen Daten angegeben erhält der Benutzer eine positive Rückmeldung.



Will sich ein Benutzer mit einer bereits vergebenen E-Mail registrieren, erhält dieser eine Fehlermeldung.

The screenshot shows a REST client interface with a tab labeled "user registration". The request is a POST to `http://localhost:8080/register`. The request body is a JSON object:

```
1 {  
2   "email": "jkopec@student.tgm.ac.at",  
3   "name": "jkopec",  
4   "password": "asd"  
5 }
```

The response status is **400 Bad Request** with a time of **9 ms**. The response body, viewed in "Pretty" mode, contains the message:

```
1 User jkopec@student.tgm.ac.at already exists!
```


Login

Versucht sich ein Benutzer mit den richtigen Accountdaten anzumelden, wird er in weiterer Folge willkommen geheißen.

The screenshot displays a REST client interface with a tab labeled "user registration". The request is a POST to `http://localhost:8080/login`. The request body is a JSON object: `{ "email": "jkopec@student.tgm.ac.at", "name": "jkopec", "password": "asd" }`. The response status is `200 OK` with a time of `12 ms`. The response body is `Welcome jkopec!`.

user registration

No environment

POST `http://localhost:8080/login` Params Send Save

```
1 {  
2   "email": "jkopec@student.tgm.ac.at",  
3   "name": "jkopec",  
4   "password": "asd"  
5 }
```

Body Cookies Headers (4) Tests Status: 200 OK Time: 12 ms

Pretty Raw Preview HTML Save Response

1 Welcome jkopec!

Gibt der Benutzer jedoch falsche Daten ein, bekommt er eine Fehlermeldung zurück.

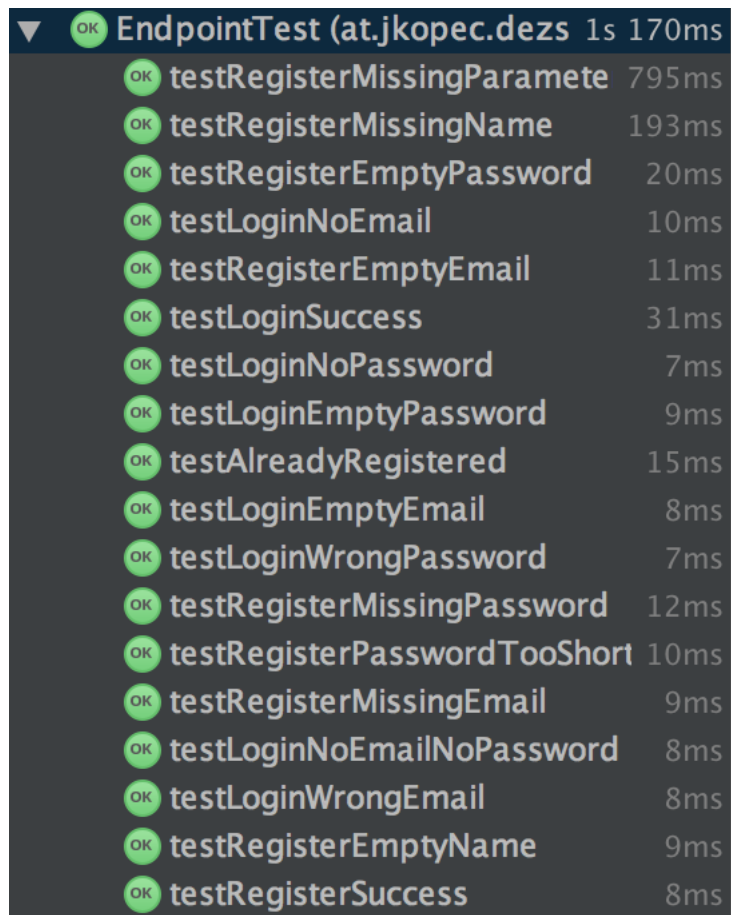
The screenshot shows a REST client interface with the following details:

- Environment:** user registration (orange dot), No environment (dropdown), and an eye icon.
- Method:** POST (dropdown).
- URL:** http://localhost:8080/login.
- Params:** Params (button).
- Buttons:** Send (blue), Save (dropdown).
- Request Body:**

```
1 {  
2   "email": "aernhofer@student.tgm.ac.at",  
3   "name": "aernhofer",  
4   "password": "asd"  
5 }
```
- Response Status:** Status: 403 Forbidden, Time: 10 ms.
- Response Body:** Invalid account data!

JUnit Tests

Folgende JUnit Tests wurden implementiert und erfolgreich durchgeführt. Die genaue Implementierung ist im SourceCode zu sehen.



The image shows a screenshot of a JUnit test runner interface. At the top, a summary bar indicates that the test suite 'EndpointTest (at.jkopec.dezs)' passed with a duration of 1s 170ms. Below this, a list of 19 individual test methods is shown, each preceded by a green 'OK' icon and followed by its execution time in milliseconds.

▼ OK	EndpointTest (at.jkopec.dezs)	1s 170ms
OK	testRegisterMissingParamete	795ms
OK	testRegisterMissingName	193ms
OK	testRegisterEmptyPassword	20ms
OK	testLoginNoEmail	10ms
OK	testRegisterEmptyEmail	11ms
OK	testLoginSuccess	31ms
OK	testLoginNoPassword	7ms
OK	testLoginEmptyPassword	9ms
OK	testAlreadyRegistered	15ms
OK	testLoginEmptyEmail	8ms
OK	testLoginWrongPassword	7ms
OK	testRegisterMissingPassword	12ms
OK	testRegisterPasswordTooShort	10ms
OK	testRegisterMissingEmail	9ms
OK	testLoginNoEmailNoPassword	8ms
OK	testLoginWrongEmail	8ms
OK	testRegisterEmptyName	9ms
OK	testRegisterSuccess	8ms

3 Quellen

- [1] Spring Framework:

<http://spring.io>

[zuletzt abgerufen am 14.04.2016]

- [2] Spring repository resources:

<http://docs.spring.io/spring-data/rest/docs/2.1.5.RELEASE/reference/html/repository-resources.html#repository-resources.collection-resource>

[zuletzt abgerufen am 14.04.2016]

- [3] Spring Boot:

<http://projects.spring.io/spring-boot/>

[zuletzt abgerufen am 14.04.2016]

- [4] H2:

<http://www.h2database.com/html/main.html>

[zuletzt abgerufen am 14.04.2016]

- [5] Spring Bootiful:

<http://spring.io/blog/2014/11/23/bootiful-java-ee-support-in-spring-boot-1-2>

[zuletzt abgerufen am 14.04.2016]