JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

DezSys09 – Web Services in Java

Systemtechnik Labor
5BHITT 2015/16, Gruppe Y

Alexander Kölbl

Version 1.0

Note:

Betreuer: Prof. Borko

Begonnen am 11. März 2016

Beendet am 14. April 2016

Inhaltsverzeichnis

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 Datebanktabelle 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

Java API for RESTful Web Services (JAX-RS) wird für die Kommunikation zwischen Client und Service verwendet. Als Ansatz habe ich mich am folgenden Repository orientiert: [5]

2.1 Buildtool

Das Projekt wurde mit Maven als Buildtool aufgesetzt. Das POM File wurde dem File aus dem Tutorial Bootiful Java EE in Spring [6] nachempfunden.

2.2 Jersey Konfiguration

Jersey wird verwendet um die REST APIs zur Verfügung zu stellen. Dazu ist folgende Konfiguration notwendig:

```
@Configuration
public class JerseyConfiguration extends ResourceConfig {
    public JerseyConfiguration() {

        this.property(ServerProperties.BV_SEND_ERROR_IN_RESPONSE, true);
        this.register(JsonMappingExceptionMapper.class);
        this.register(ConstraintViolationExceptionMapper.class);
        this.register(BasicExceptionMapper.class);
        this.register(UserEndpoint.class);
    }
}
```

„In order to simplify development of RESTful Web services and their clients in Java, a standard and portable JAX-RS API has been designed. Jersey RESTful Web Services framework is open source, production quality, framework for developing RESTful Web Services in Java that provides support for JAX-RS APIs and serves as a JAX-RS (JSR 311 & JSR 339) Reference Implementation. Jersey framework is more than the JAX-RS Reference Implementation. Jersey provides it's own API that extend the JAX-RS toolkit with additional features and utilities to further simplify RESTful service and client development. Jersey also exposes numerous extension SPIs so that developers may extend Jersey to best suit their needs.“ [7]

2.3 User Entity

Ein User besteht aus einer Email-Adresse (Primary Key) und einem Passwort. Die Email-Adresse darf nur für einen User verwendet werden. Weiters muss das Password eine Mindestlänge von fünf Zeichen haben. Mittels der

Annotation „@Entity“ kann eine Klasse erzeugt werden, die Entities definieren kann. Mittels der Annotation „@NotNull“ wird festgelegt, dass das Passwort nicht null sein darf (wird benötigt). Mittels der Annotation „@Size“ wird die Länge des Passworts festgelegt (mind. Fünf Zeichen in diesem Fall). Das Passwort wird als Hash in die Datenbank gespeichert, die Übertragung findet jedoch als Klartext statt. Deshalb wir zusätzlich auch die Annotation „@Transient“ verwendet, um eine Persistierung des Klartextpasswortes zu verhindern.

```
@Entity
public class User {

    @Id
    @NotBlank(message="{NotBlank.user.email}")
    @Email(message="{Email.user.email}")
    private String email;

    @NotNull(message="{NotNull.user.passwordHash}")
    private byte[] passwordHash;

    @Transient
    @Size(min=5, message="{Size.user.passwordRaw}")
    @JsonProperty("password")
    private String passwordRaw;

    @JsonCreator
    public User(@JsonProperty("email") String email, @JsonProperty("password")
String password) {
        this.email = email;
        this.passwordRaw = password;
        this.passwordHash = this.createHash();
    }

    private User() {
    }

    private byte[] createHash() {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            md.update(this.passwordRaw.getBytes("UTF-8"));
            return md.digest();
        } catch (Exception e) {
            return null;
        }
    }

}
```

2.4 User Persistierung

Damit die User Objekte persistiert werden können, wird eine Schnittstelle benötigt. Spring bietet hierfür Repositories an. In diesem Fall wird das

CrudRepository verwendet.

```
@Repository
public interface UserRepository extends CrudRepository<User, String> {
}
```

2.5 RESTful Endpoint

Endpoint zum Login und zur Registrierung erstellt. Mittels der Annotation „@Produces“ wird eingestellt, welcher MediaTyp zurückgegeben wird (JSON). Der Endpoint reagiert auf sämtliche Anfragen mit einer ResponseMessage. Diese besteht aus den Attributen „message“ und „status“. In „message“ befindet sich eine Nachricht (Erfolgsmeldung, Fehlermeldung) und in „status“ befindet sich der HTTP-Statuscode. Die URI http://IP/register wird für die Registrierung und http://IP/login für das Login verwendet. Für die Registrierung muss zur Registrierungs-URI eine POST-Anfrage mit folgendem Aufbau gesendet werden:

```
{
"email": "<Email-Adresse>",
"password": "<Passwort>"
}
```

Die Methode register verarbeitet diese Anfragen. Innerhalb dieser Methode wird kontrolliert, ob die Eingaben in der POST-Anfrage korrekt waren. Wenn die Anfrage erfolgreich verarbeitet wurde erhält der Benutzer eine Erfolgsmeldung (Benutzer erfolgreich erstellt), ansonsten eine Fehlermeldung. Für das Login muss zur Login-URI eine POST-Anfrage mit folgendem Aufbau gesendet werden:

```
{
"email": "<Email-Adresse>",
"password": "<Passwort>"
}
```

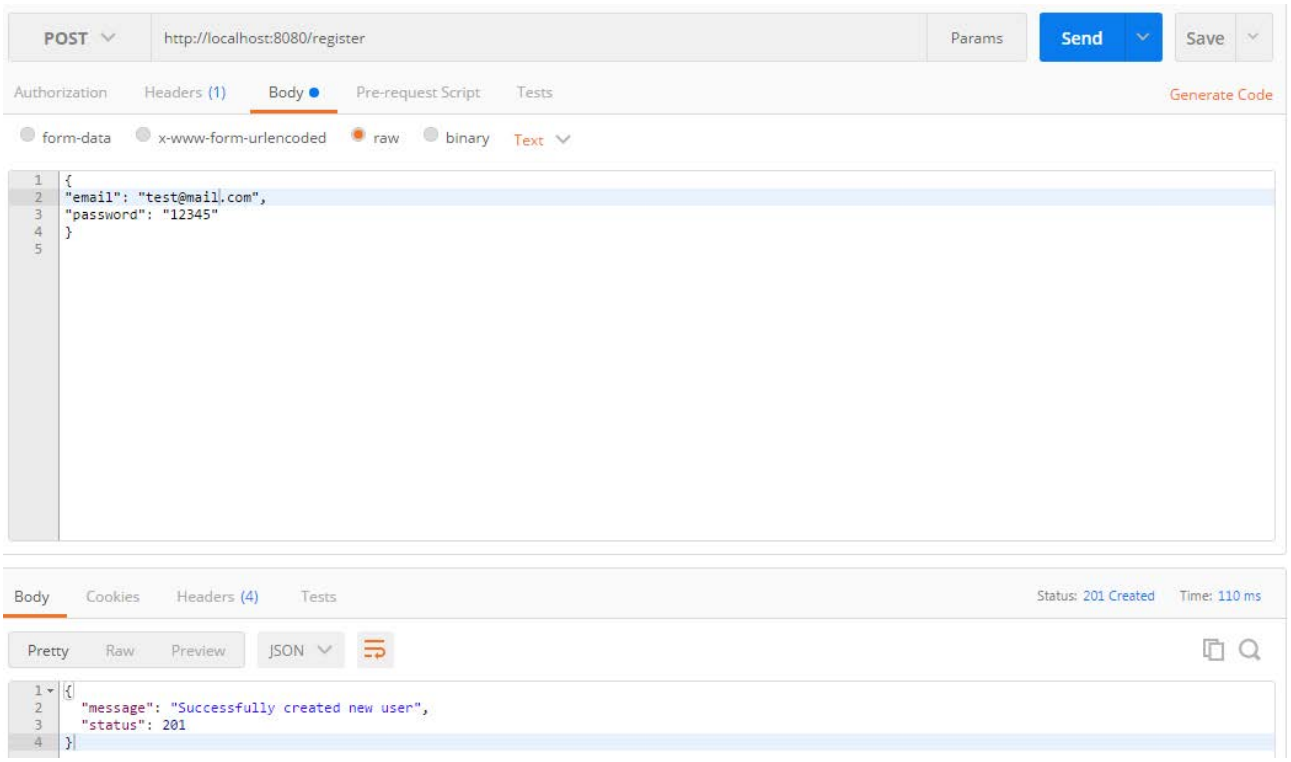
Die Methode login verarbeitet und validiert die POST-Anfragen. Wenn die Anfrage erfolgreich verarbeitet wurde erhält der Benutzer eine Erfolgsmeldung (Login erfolgreich), ansonsten eine Fehlermeldung

```
@Component
@Produces(MediaType.APPLICATION_JSON)
@Path("/")
public class UserEndpoint {

    @Autowired
    private UserRepository repo;

    @POST
    @Path("/register")
    public Response register(@Valid User user) {
        if (user == null)
            return ResponseUtil.badRequest("Payload is empty");
        if (repo.exists(user.getEmail()))
            return ResponseUtil.badRequest("User with given email address
already exists");
        repo.save(user);
        return ResponseUtil.created("Successfully created new user");
    }

    @POST
    @Path("/login")
    public Response login(@Valid User user) {
        if (user == null)
            return ResponseUtil.badRequest("Payload is empty");
        User requestUser = repo.findOne(user.getEmail());
        if (requestUser != null && Arrays.equals(requestUser.getPasswordHash(),
user.getPasswordHash()))
            return ResponseUtil.ok("Successfully logged in");
        return ResponseUtil.forbidden("Wrong email or password");
    }

}
```

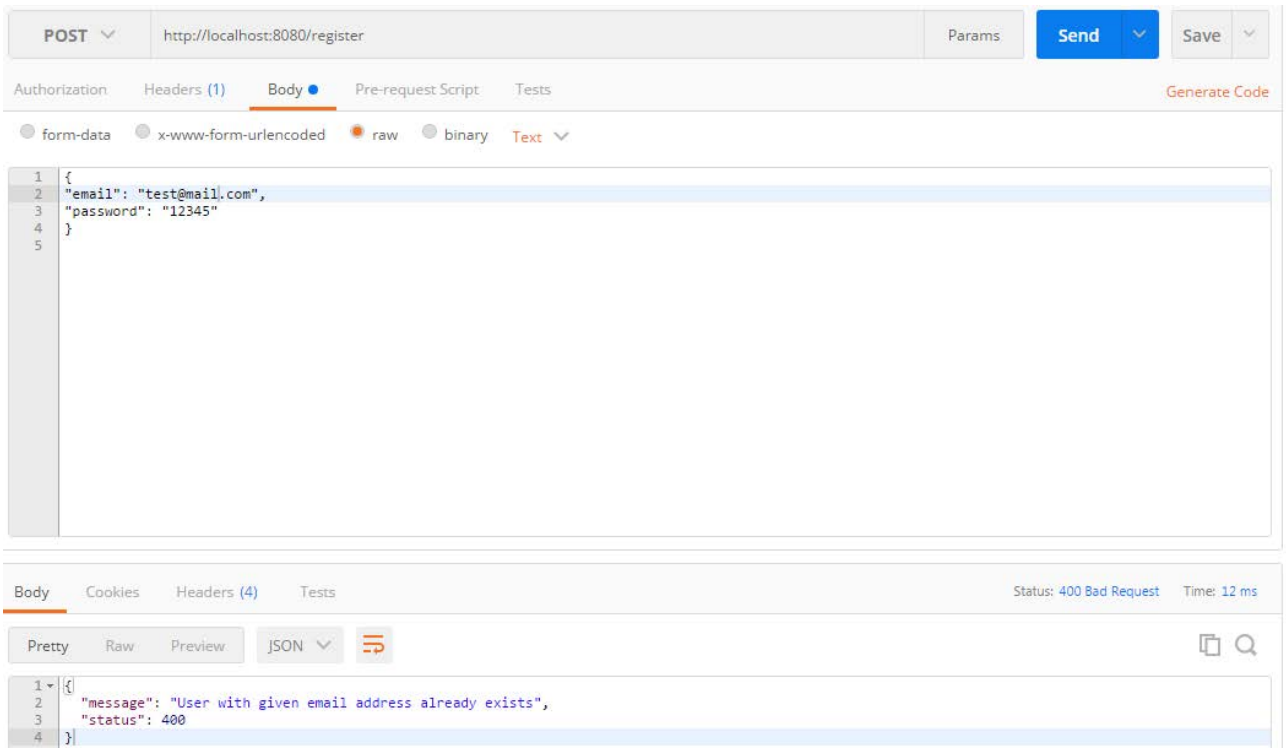
2.6 Funktionalitätstest

Mittels des Google Chrome Plugins Postman [8] wurden die Funktionalitäten überprüft

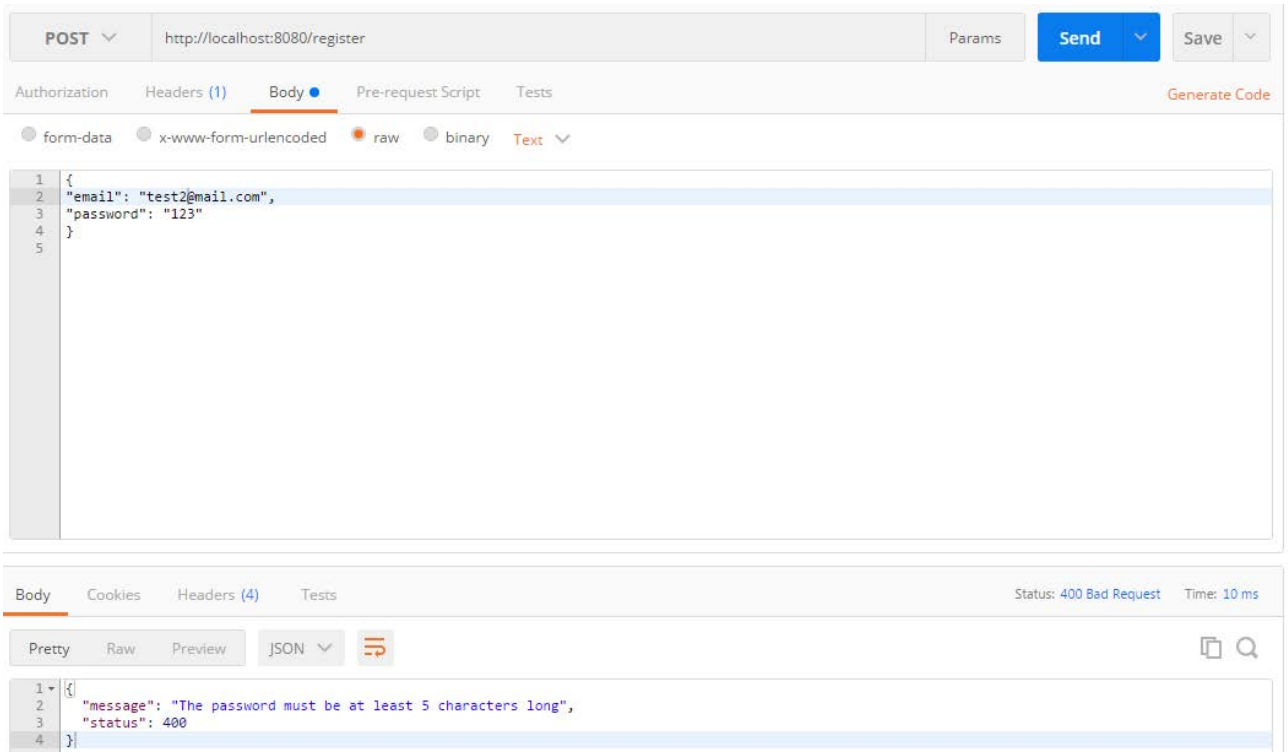
User erstellen



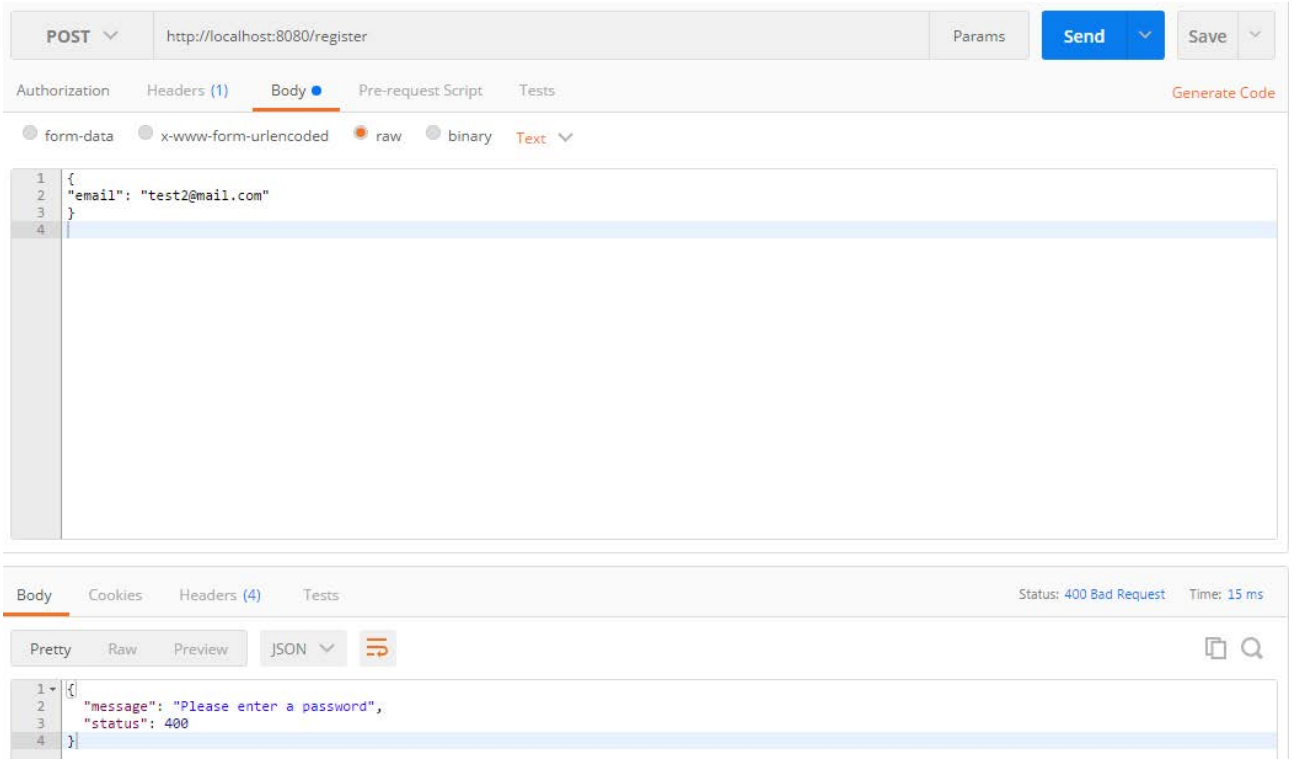
Gleiche Email Adresse verwendet



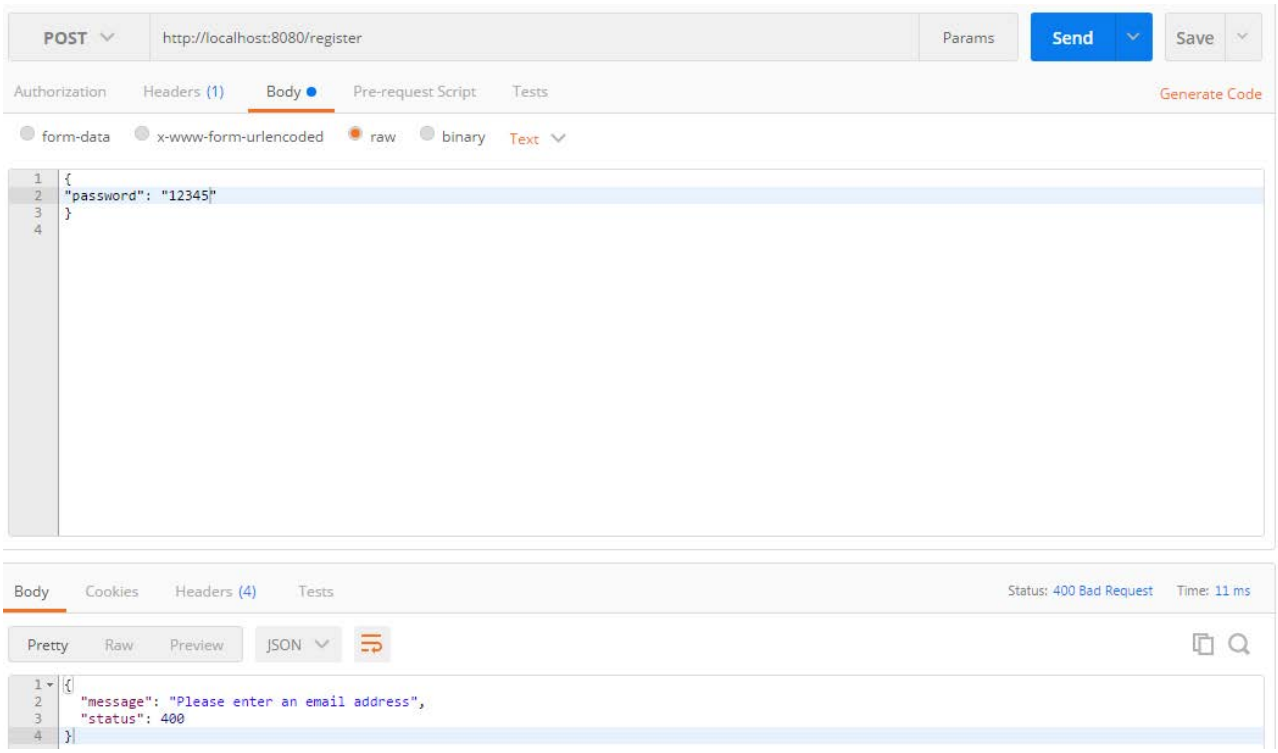
Zu kurzes Passwort verwendet



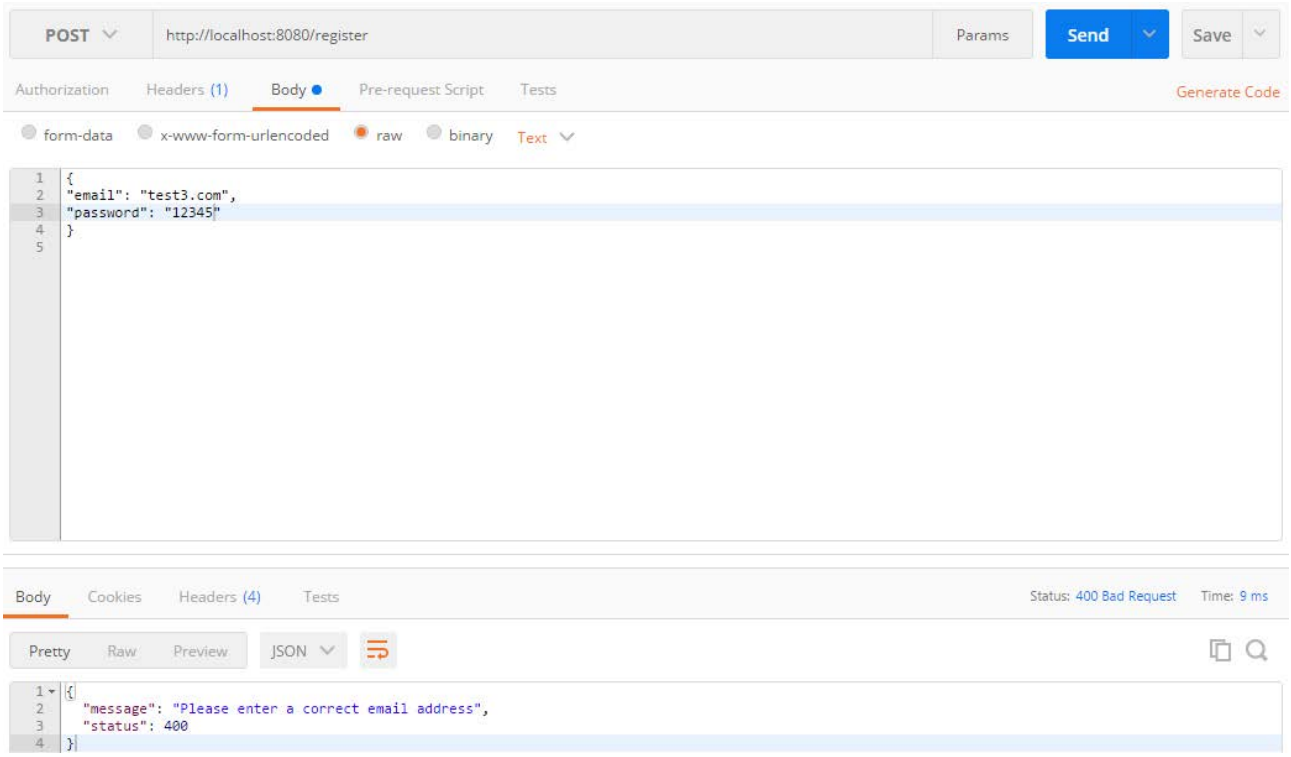
Kein Passwort angegeben



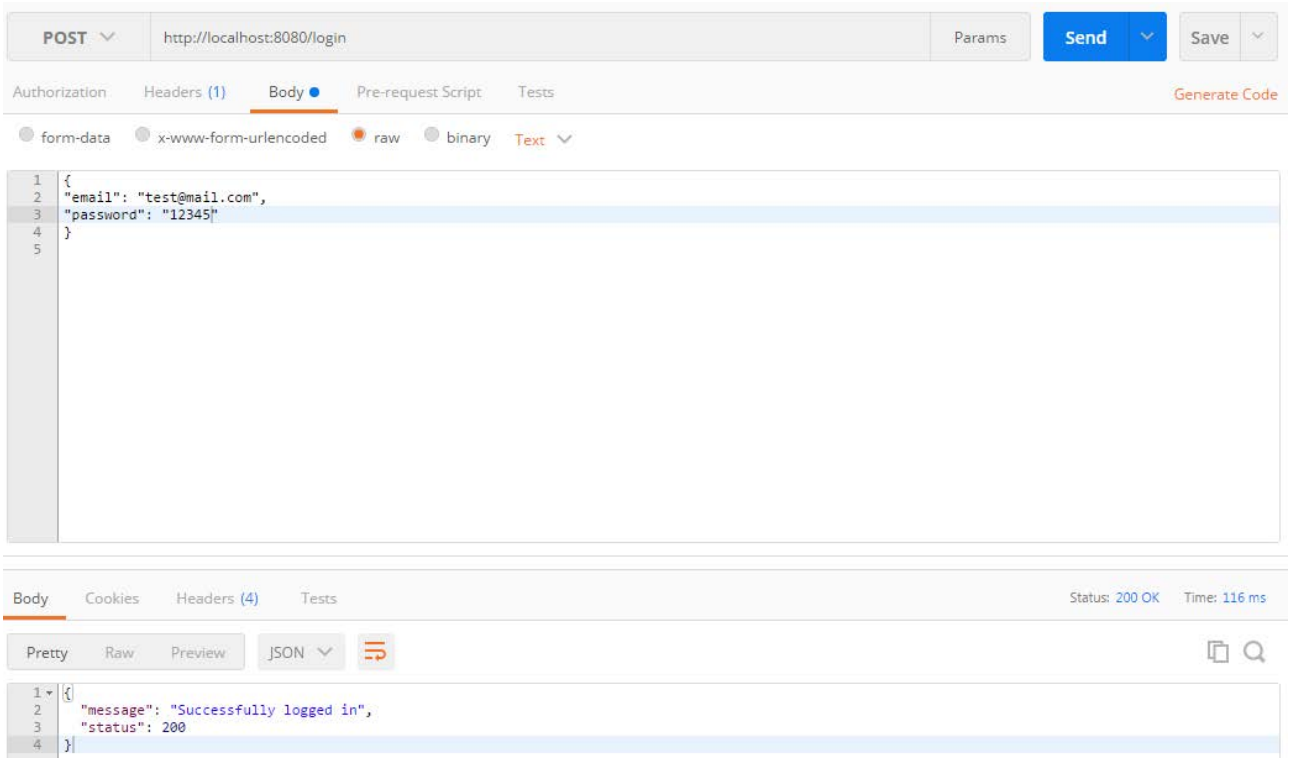
Keine Email Adresse angegeben



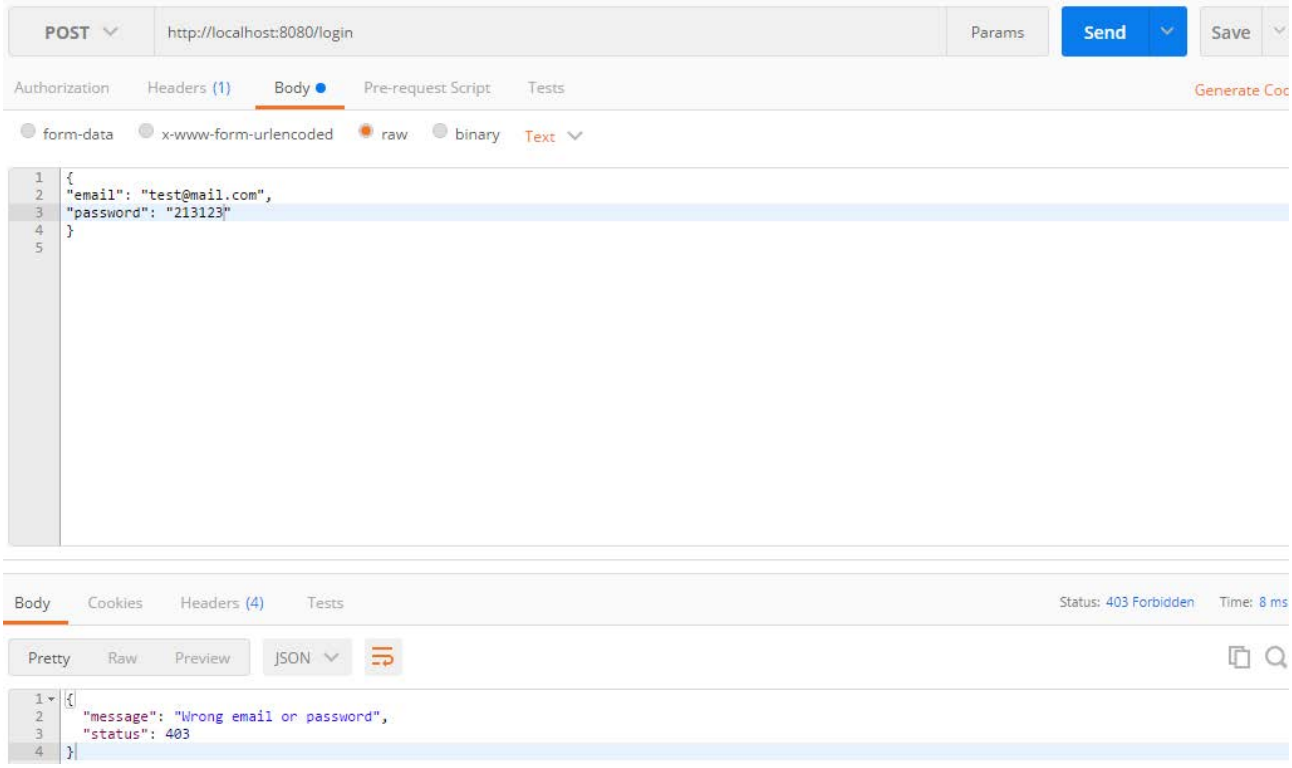
Keine gültige Email Adresse angegeben



Erfolgreicher Login



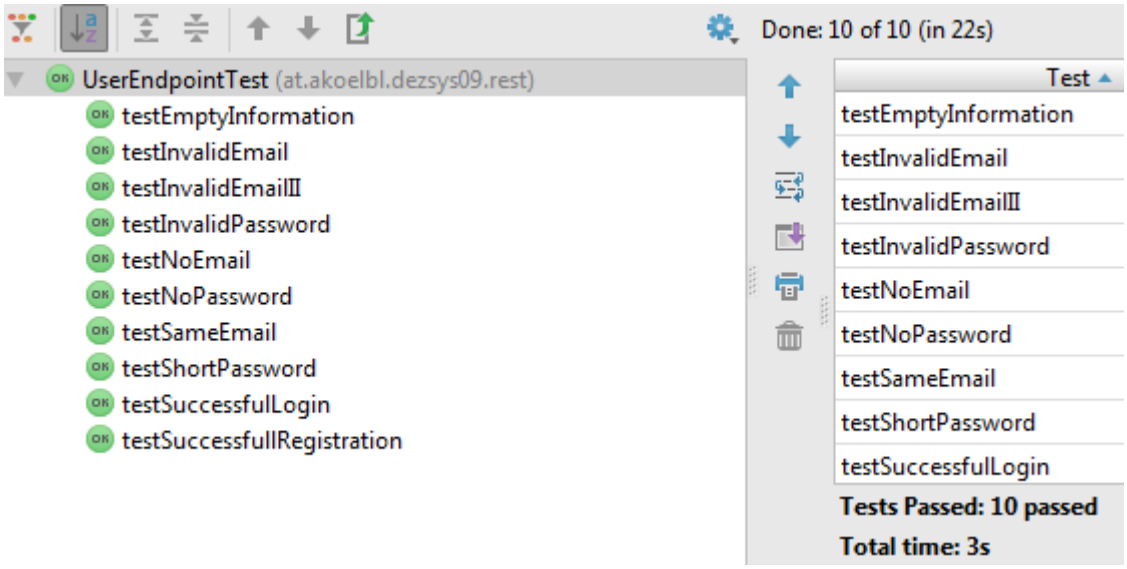
Flasche Login-Daten



2.7 Testfälle

Acceptance Tests (10 Stück) für folgende Funktionalitäten wurden durchgeführt (Testbericht wird bei Maven install generiert):

- Erfolgreiche Registrierung
- Zu kurzes Passwort bei Registrierung
- gleiche Email Adresse bei Registrierung verwenden
- keine Email Adresse bei Registrierung angeben
- kein Passwort bei Registrierung angeben
- keine Informationen bei Registrierung angeben
- ungültige Email Adresse bei Registrierung angeben
- erfolgreiches Login



2.8 Ausführen

Mittels des Befehls „maven install“ wird ein jar-File generiert, das durch Spring einen Applicationserver integriert hat. Durch den Befehel „java –jar <Jar-File>“ kann das Programm gestartet werden (standardmäßig auf Port 8080). Mit dem Parameter --server.port=<Portnummer> kann der Port geändert werden.

Registrierung-URI: http://127.0.0.1:8080/register

Login-URI: „http://127.0.0.1:8080/login

Sowohl bei der Registrierung als auch beim Login muss eine POST-Anfrage mit

folgender Struktur versendet werden:

```
{
"email": "<Email-Adresse>",
"password": "<Passwort>"
}
```

3 Zeitaufwand

Der Zeitaufwand für diese Aufgabe (Programmieren & Dokumentation) betrug 7 Stunden.

4 Repository

Das Git-Repository zu dieser Aufgabe ist unter folgender URL verfügbar:

https://github.com/akoelbl-tgm/DezSys09-WebServices

5 Quellen

[1] "Android Restful Webservice Tutorial – Introduction to RESTful webservice – Part 1"; Posted By Android Guru on May 1, 2014; online: http://programmerguru.com/android-tutorial/android-restful-webservicetutorial-part-1/

[2] REST with Java (JAX-RS) using Jersey - Tutorial"; Lars Vogel; Version 2.5; 15.12.2015; online: http://www.vogella.com/tutorials/REST/article.html

[3] "O Java EE 7 Application Servers, Where Art Thou? Learn all about the state of Java EE app servers, a rundown of various Java EE servers, and benchmarking."; by Antonio Goncalves; Java Zone; Feb. 10, 2016; online: https://dzone.com/articles/o-java-ee-7-application-serverswhere-art-thou

[4] "Heroku makes it easy to deploy and scale Java apps in the cloud"; online: https://www.heroku.com/

[5] Web Services in Java, Paul Kalauner, online: https://github.com/pkalauner-tgm/dezsys09-java-webservices

[6] Bootiful Java EE Support in Spring Boot 1.2, online: http://spring.io/blog/2014/11/23/bootiful-java-ee-supportin-spring-boot-1-2

[7] Jersey RESTful Web Services in Java, online: https://jersey.java.net/

[8] Postman, online: http://www.getpostman.com/