Final Report

Microservice 1 (MS1)

Cordula Eggerth
00750881

Team 301

BSc Business Informatics

VU Distributed Systems Engineering

Winter Semester 2017

Table of Contents

1. Discussion of deviation from the original plan (laid in SUPD)

The plan laid in the SUPD served as a viable guidance for the final implementation of MS1. The final implementation includes most aspects that were previously stated in the SUPD report, at some places with additions, notably as to new services offered and regarding the access control problem that occurred when the client (user interface) of MS3 (which is written in JavaScript (react.js/jsx)) started using my then available version of MS1. All services offered from my side, are now deployed on the Tomcat server of the university on port 31811. Therefore, I needed to coordinate with the team member who was implementing MS3 in order to tackle the cross-origin problem that had occurred, which resulted in a deviation from my original plan. To make her client code work, we decided to use a Cross-Origin Resource Sharing (CORS) filter to permit the client written in JavaScript to carry out requests (e.g. in Ajax and react.js) to the domain of my MS1 on the server-side, which is differs from the domain of the client that contains the user interface for the To-Do-List application. The problem, we experienced first, was that the web request made by the client was forbidden and caused the cross-origin resource sharing problem.[1] Therefore, a CORS filter was added to the dependencies in the pom.xml and in the web content in the web.xml, to permit that all requests (incl. cross-domain requests) are served accordingly by MS1.[2] The "access-control-allow-origin" details the domains, which are allowed to make cross-domain requests. In this project, it was set to "*", to allow all domains to make requests. So, the client application (MS3) can actually request and get the data and responses from my MS1 implementation deployed on the Tomcat server of the university at port 31811.

To test if the cross-origin access problem could be solved, I created the dse.html file, which includes JavaScript code and uses Ajax (dse.html) to make requests to my MS1 service. By clicking on the button "User List" or "TODO List", the functioning of the userlist and the toDoListPerUser service can be checked respectively. The html file can be downloaded/ pulled from gitlab and can be run in the browser, which shows that the access control problem has been solved. By clicking on the button "User List", the list of users and passwords can be checked and the result looks for example like this:

[1] https://howtodoinjava.com/servlets/java-cors-filter-example/
[2] https://stackoverflow.com/questions/12383109/access-control-allow-origin-in-tomcat, https://amodernstory.com/2014/12/27/using-cors-headers-with-java-example/

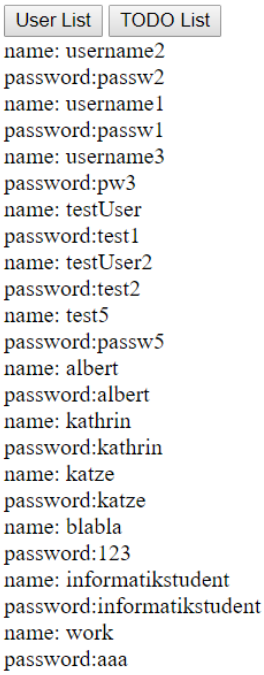


Figure 1 – result of dse.html test userlist service

At the time of the SUPD version, the MS1 was not fully developed and did not include all services needed yet. Also, the services were tested locally, but not all of them were deployed on Tomcat and tested by the means of a client-side program. Therefore, for the final version of MS1, I added further services, and I had to make changes to the existing login service (or rather developed a second version of the login in order to cater to the needs of the MS3 client) best possible. After these changes and additions, I deployed the final version of the MS1, which is the java project called "DSE". As already mentioned, I developed a new login service using POST method and HeaderParam now. The service returns the User to be logged in as a JSON object, as is can be seen from the screenshot below. Requests to the login service can be made to http://tomcat01lab.cs.univie.ac.at:31811/DSE/users/loginMS3, while adding the necessary HeaderParams password and username as a String.

```java
/**
 * Web Service fuer Login (fuer MS3-Client)
 * Es wird POST verwendet und der Aufruf erfolgt ueber den Pfad /users/loginMS3.
 * Uebergeben werden username und password als Parameter.
 * Die Funktion gibt bei erfolgreichem Login einen User in Format JSON zurueck an den Client.
 * @param username
 * @param password
 * @return User
 */
@POST
@Path("/loginMS3")
@Produces(MediaType.APPLICATION_JSON)
public User loginMS3(@HeaderParam("username") String username, @HeaderParam("password") String password) {
   return userManagement.login(username, password);
}
```

Figure 2 – Login Service (MS1)

The two figures below depict the successful login test via ARC (Advanced REST Client) Google Chrome app, showing the request URL, method, and header parameters as well as the response from the server, which is given in the form of a JSON object of the User data:

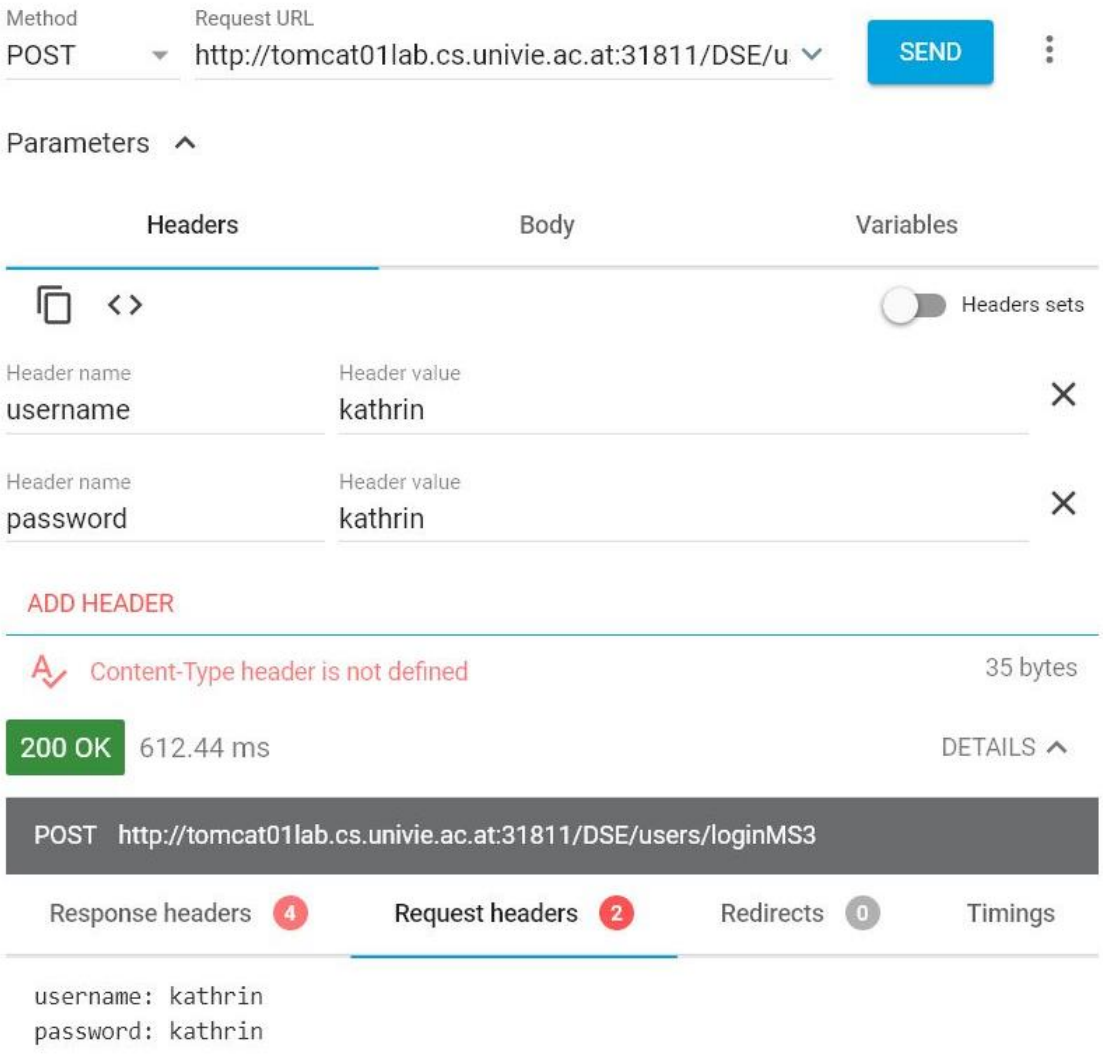


Figure 3 – ARC Login Test Part 1

Source message

```
POST /DSE/users/loginMS3 HTTP/1.1
HOST: tomcat01lab.cs.univie.ac.at:31811
username: kathrin
password: kathrin
content-length: 0
```

```
{
   "id": 11,
   "username": "kathrin",
   "password": "kathrin",
   "firstname": "firstkathrin",
   "lastname": "lastkathrin",
   "city": "wien"
}
```

Figure 4 – ARC Login Test Part 2

Furthermore, the service to delete a user has been created. The service uses the GET method and takes the userId of the user to delete as a QueryParam. It can be accessed for example to delete the user with user ID 2 via http://tomcat01lab.cs.univie.ac.at:31811/DSE/users/delete?userId=2. A Response is returned depending on whether the user could be deleted, which results in response status code 200 and the statement "User wurde geloescht.", or the user could not be deleted, which results in the response status code 400 and the statement "User existiert nicht. Kein Loeschen durchgefuehrt.". The delete service is shown below.

```java
/**
 * Web Service fuer das Loeschen eines Users.
 * Es wird GET verwendet und der Aufruf erfolgt ueber den Pfad /users/delete.
 * Es wird der Parameter userId als QueryParam uebergeben.
 * Retourniert wird Response mit einem Status und einer Nachricht, deren Inhalt
 * davon abhaengt, ob geloescht wurde oder nicht.
 * @param userId
 * @return Response
 */
@GET
@Path("/delete")
public Response delete(@QueryParam("userId") int userId) {
  System.out.println(userId);

  if(userManagement.checkUserNameExists((userManagement.getUserById(userId)).getUsername())){
    userManagement.deleteUser(userId);
    return Response.status(200).entity("User wurde geloescht.").build();
  }
  else {
    return Response.status(400).entity("User existiert nicht. Kein Loeschen durchgefuehrt.").build();
  }
}
```

Figure 5 – Delete user service

The following figure depicts the ARC test for the delete user service.

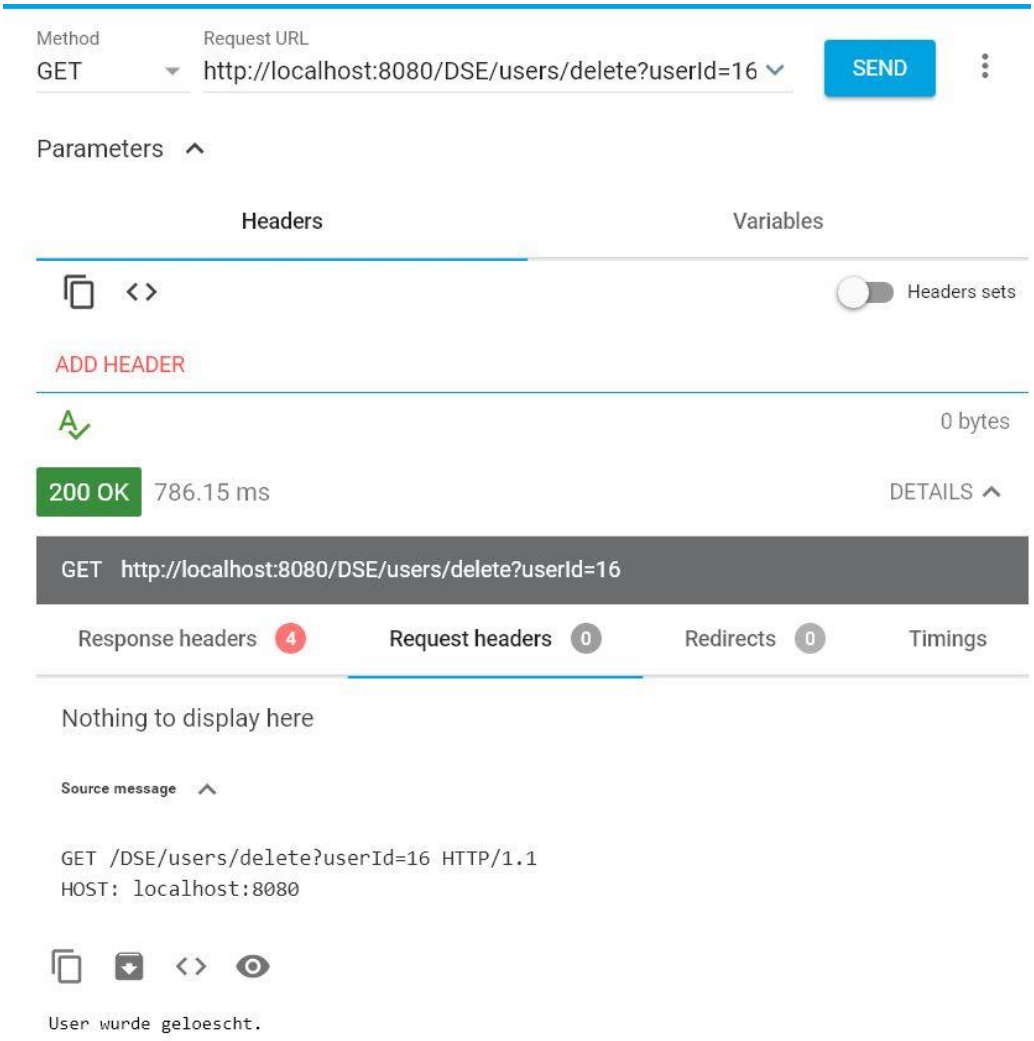


Figure 6 – ARC test of the delete user service

A service to change the password of the user has been implemented in the DSE java project for MS1 as well. In addition to this, the corresponding changes have been made to the JSP file meinprofil.jsp, where the password of the user can actually be changed in the DSETestClient. And the Servlet ChangePasswordServlet has been created. The changePassword service can be requested via http://tomcat01lab.cs.univie.ac.at:31811/DSE/users/changePassword while also sending the header parameters id, username, password, firstname, lastname, and city as Strings. The service returns a Response depending on whether the password could actually be changed or not. If the password could be changed, the response status code is 200, and the message is “Passwort wurde geaendert.” If the password could not be changed, the response

status code is 400, and the message is “User existiert nicht. Keine Aenderung durchgefuehrt.”. The following screenshot shows the changePassword service.

```
/**
 * Web Service fuer das Aendern des password eines Users.
 * Es wird POST verwendet und HeaderParam ueber den Pfad users/changePassword
 * @param username
 * @param password
 * @param firstname
 * @param lastname
 * @param city
 * @return Response
 */
@POST
@Path("/changePassword")
// @Produces(MediaType.APPLICATION_JSON)
public Response changePassword(@HeaderParam("id") String id, @HeaderParam("username") String username,
        @HeaderParam("password") String password, @HeaderParam("firstname") String firstname,
        @HeaderParam("lastname") String lastname, @HeaderParam("city") String city) {

      if(userManagement.checkUserNameExists(username)){
            // System.out.println("in if changePW ok: "+ password);
            User user = new User(Integer.parseInt(id), username, password, firstname, lastname, city);
            userManagement.updateUser(user);
            return Response.status(200).entity("Passwort wurde geaendert.").build();
          }
          else {
            return Response.status(400).entity("User existiert nicht. Keine Aenderung durchgefuehrt.").build();
          }

}
```

Figure 7 – change password service

The next figure shows the ARC test of the change password service, which also includes the header parameters indicated before.

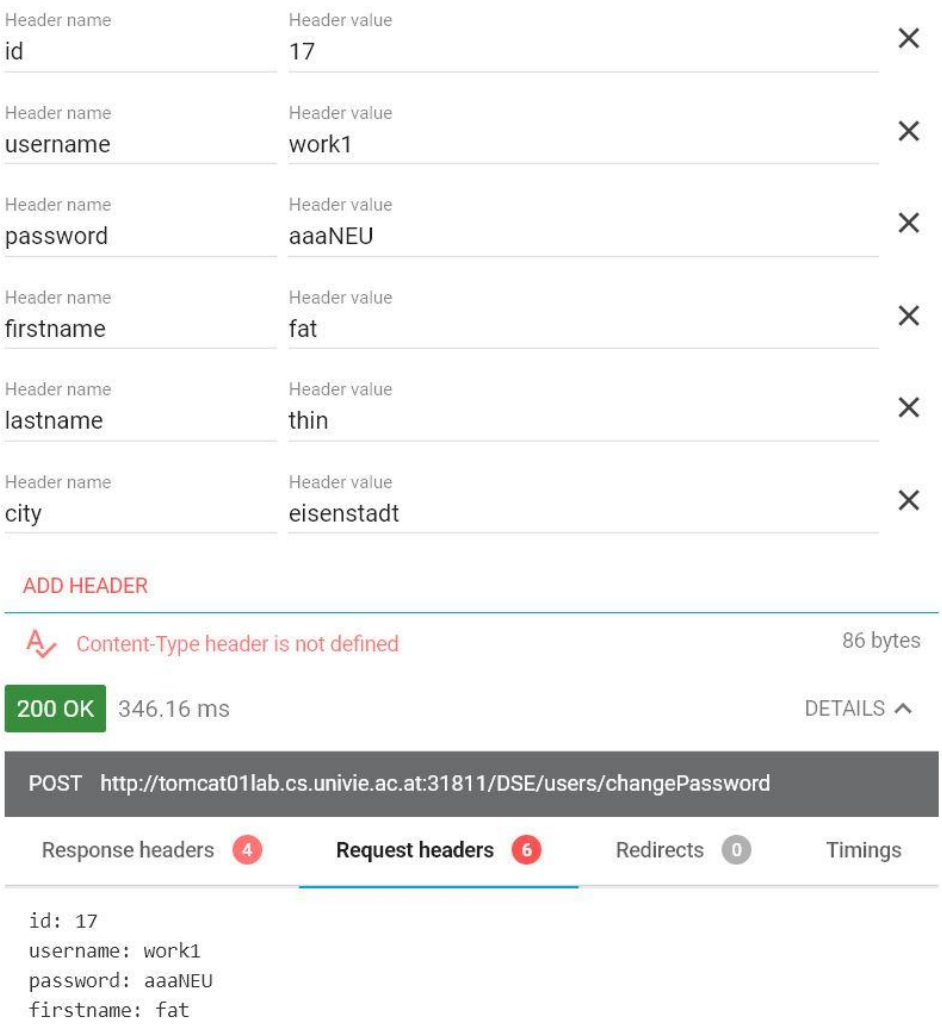


Figure 8 – ARC test of the change password service

2. Description of final service interface

For the implementation of the project, RESTful Webservices using JAX RS with Jersey 2.20 has been chosen. For the user interface in the DSETestClient that I have created to the MS1, I chose JSP (JavaServerPages) Files and Servlets. The test client checks the functionality of MS1 and also simulates the interplay with a mockup ToDoListTestServer in order to enable isolated tests of my service, for the case that MS2 and MS3 are not available. In doing so, the MS1 functionality can be presented independently, if desired. Additional tests have been made using the ARC (Advanced REST Client) App of Google Chrome to check whether the services are functioning correctly. The DSE Project includes the MS1 services, which refers to the user-related services. The MS1 has been deployed on the Tomcat server of the university on port 31811, i.e. http://tomcat01lab.cs.univie.ac.at:31811. To make requests to the server that provides the user-related services, the respective request in terms of URI hast to be sent to the server. In most cases of requests, the server returns data in the form of JSON or text.

The project name is BusyBee, which constitutes an organization that offers a website for the administration of to-do-lists for registered users. My project part comprises three java projects, which are all dynamic web projects that are Maven projects. The project named "DSE" offers the user-related services (according to MS1). "ToDoListTestServer" provides the mockup version of the to-do-list-related services, so that the isolated presentation of MS1 in case of a lack of MS2 and MS3, can be carried out seamlessly. The project "DSETestClient" acts as the mockup client (similar to MS3) and focuses on the presentation and testing of the user-related (MS1) services, and their interplay with the to-do-list-related (MS3) services.

The communication between the project parts takes place via HTTP and by using JAX RS with Jersey for RESTful Web Services. Data persistence is ensured by the use of the university's mySQL database named a0750881mysql3 that I created for the user-related data, and the mySQL database name a0750881mysql4 that I created for the to-do-list-related data (in reference to the mockup to-do-list data)[3].

[3] ***Note:*** The to-do-list services in my project are only implemented for mockup purposes to simulate the interplay with the user-related services and to present the isolated version of MS1, for the case that MS2 and MS3 are not available. The TestClient project contains the related functionality to present MS1 accordingly in its isolated and independent version.

DSE (server-side – user-related services):

In the java project ”DSE“, the services for the registration of users, the login of users, the deletion of users, the userlist, and the change of password, are provided. The server can be contacted by a client, which requests user-related information and/or wants to change or delete user data. I placed the DSE project (containing the required services of MS1) on the university Tomcat server, so that it can be contacted by clients. The server can be reached on: http://tomcat01lab.cs.univie.ac.at:31811/DSE/. Depending on the chosen extension of the link (URI), the requests of the client are serviced. The figure below illustrates the deployed services “DSE” (for the user-related services/MS1) and the “ToDoListTestServer” (for the mockup to-do-list-related services/MS2).

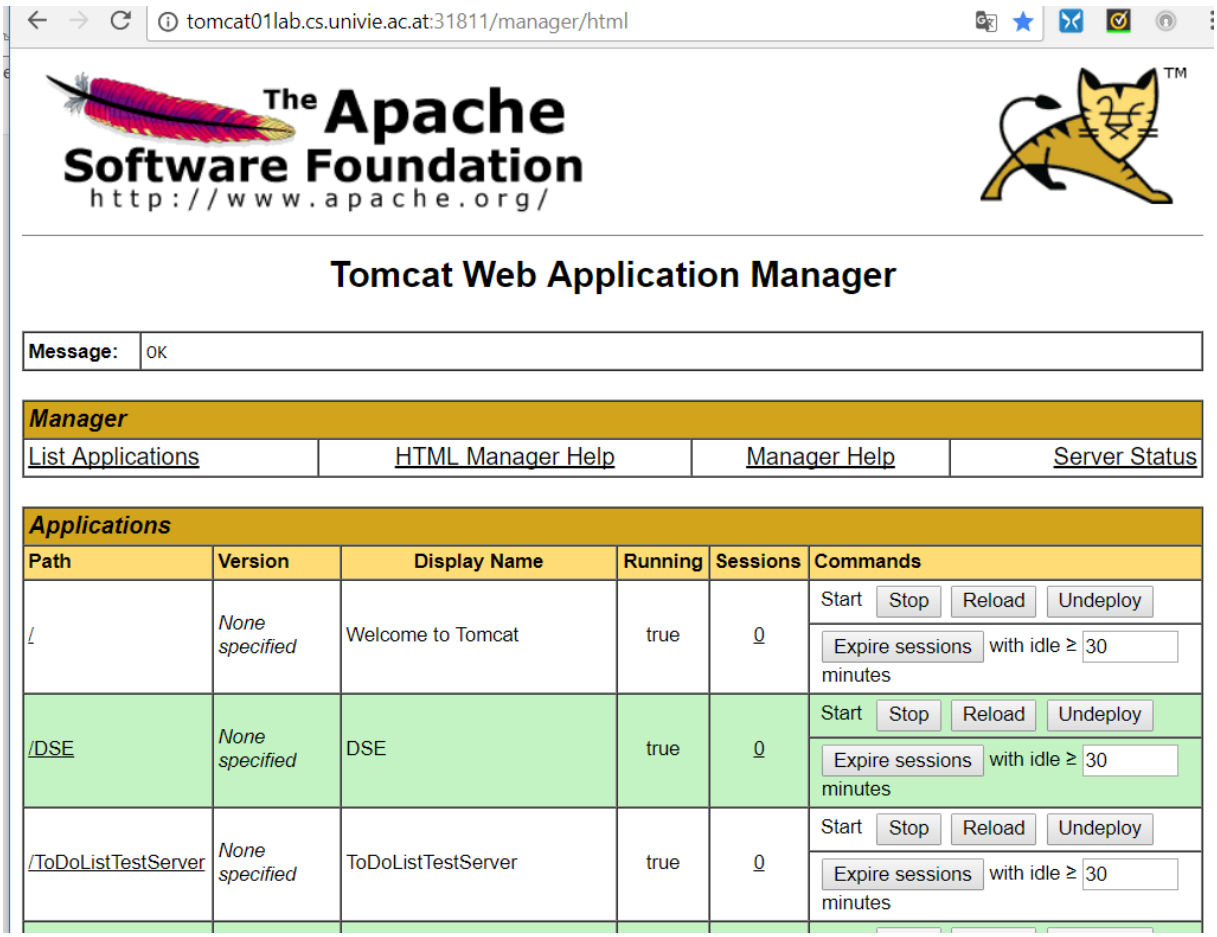


Figure 9 - deployment of services on Tomcat server

The data modifications that are requested by the client, occur in the mySQL database a0750881mysql3, which is provides by the University of Vienna. It should be noted at this point that the user of the services needs to be logged into UniVPN (Big-IP Client/F4) in order to be able to use the database access. The communication occurs via HTTP.

The functioning of the services and the persistent data operations can be checked as well with the help of the ARC (Advanced REST Client) App offered by Google Chrome. For example, if the parameters are given and the extension …/DSE/users/register?username=… is requested, and the HTTP response "200 OK" is obtained[4], and the message "User wurde hinzugefügt" comes back, the user could be successfully registered. The below screenshot shows the request of the registration of a new user and the response that the server sends, once the user could be correctly added to the database, i.e. correctly registered.

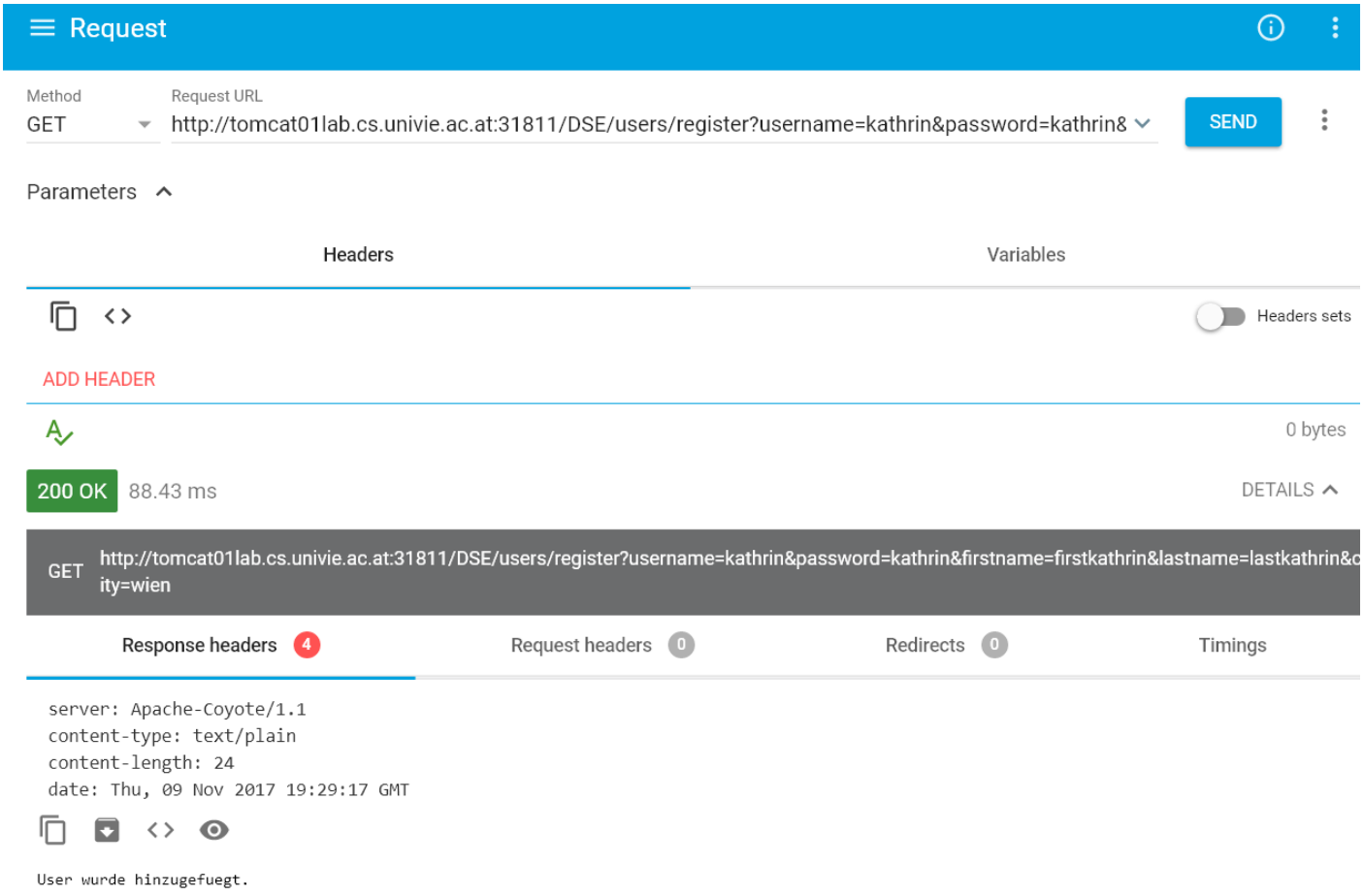
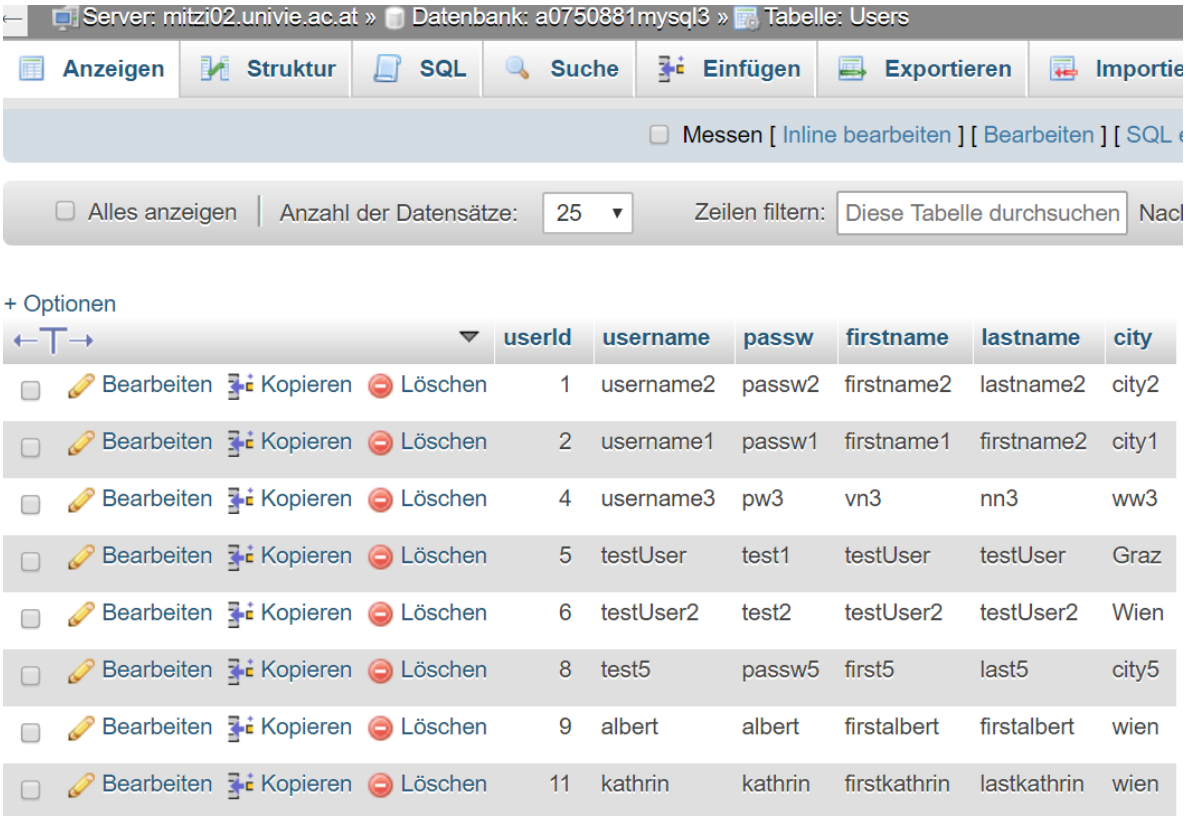


Figure 10 – request from client to DSE server and receipt of response (via HTTP)

On the admin side of the mySQL database of the University of Vienna, the administrator can now check, if the user has been added correctly, as it is shown in the below screenshot – the user was added correctly and can now perform the login on the client website.

[4] The HTTP status code 200 means "success", i.e. the request could successfully be carried out.



	userId	username	passw	firstname	lastname	city
Bearbeiten Kopieren Löschen	1	username2	passw2	firstname2	lastname2	city2
Bearbeiten Kopieren Löschen	2	username1	passw1	firstname1	firstname2	city1
Bearbeiten Kopieren Löschen	4	username3	pw3	vn3	nn3	ww3
Bearbeiten Kopieren Löschen	5	testUser	test1	testUser	testUser	Graz
Bearbeiten Kopieren Löschen	6	testUser2	test2	testUser2	testUser2	Wien
Bearbeiten Kopieren Löschen	8	test5	passw5	first5	last5	city5
Bearbeiten Kopieren Löschen	9	albert	albert	firstalbert	firstalbert	wien
Bearbeiten Kopieren Löschen	11	kathrin	kathrin	firstkathrin	lastkathrin	wien

Figure 11 – excerpt of the mySQL database a0750881mysql3 to check if the user "kathrin" has been registered correctly

The DSE server offers the services to register, log in, delete users and to change their password. Also, it offers the opportunity to retrieve the complete userlist. The services were placed in the java class UserService.java in the package "services". The return of user data and/or messages occurs either in JSON or in text. These data have to be received and processed by the client, and the corresponding actions have to be taken in the user interface on the client side.

ToDoListTestServer (server-side – to-do-list-related services - mockup):

In the java project ToDoListTestServer several server-side services related to the creation, retrieval, and administration of to-do-lists are provided. These services interact with user information, and simulate the interplay of the MS1 and MS2 services. The ToDoListTestServer was also deployed on the university's Tomcat server (see Figure 9), and is available on: http://tomcat01lab.cs.univie.ac.at:31811/ToDoListTestServer/. The ToDoListTestServer can also be contacted by clients. In this case, the DSETestClient wants to retrieve for example the data of the user who is currently logged in. Depending on the chosen link extension (URI), the client can make requests, and the data changes are made in the mySQL database a0750881mysql4. As already mentioned, one has to be logged into the UniVPN in order to use the service of the database. Communication with the

ToDoListTestServer occurs via HTTP. Just as for the user-related services, for ToDoListTestServer, the services can be tested via the mockup user interface and/or via the ARC app. The parameters are handed over using the extension …/ToDoListTestServer/todolist/… and the respective service is called. In the example below, the to-do-list of the user with userId 2 is requested via GET request, for which the URI is http://tomcat01lab.cs.univie.ac.at:31811/ToDoListTestServer/todolist/todolistPerUser/?userId=2. The response status code shown in the figure below is "200 OK", which means that the request was successful. Thus, for the user with userId 2 all to-do-lists that the user has created already can be retrieved from the database and returned to the client in this case. With regard to the response format, the ToDoListServer returns a JSONArray of to-do-list objects which are also mapped in JSONObject format. At this point, it can be checked on the admin page of the mySQL database a0750881mysql4, if the data have been correctly provided to the client (see Figure 12).

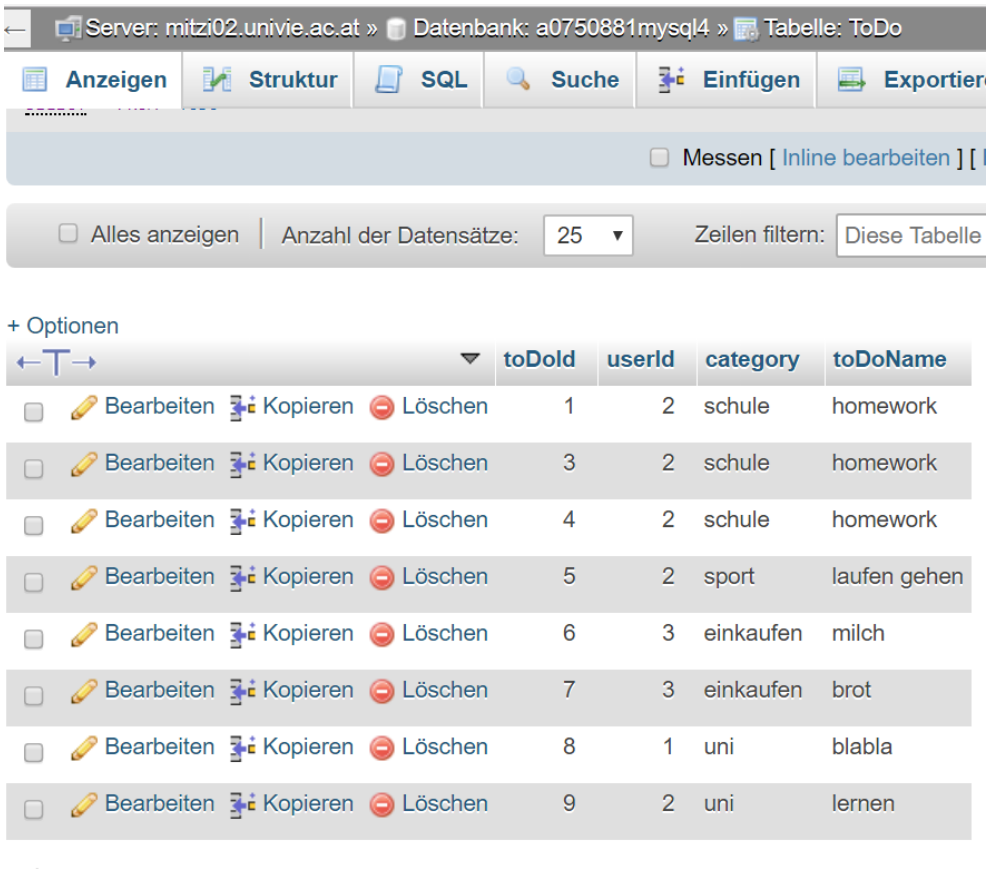


Figure 13 - Ansicht der mySQL-Datenbank a0750881mysql4 bzgl. gespeicherter ToDos von userId 2

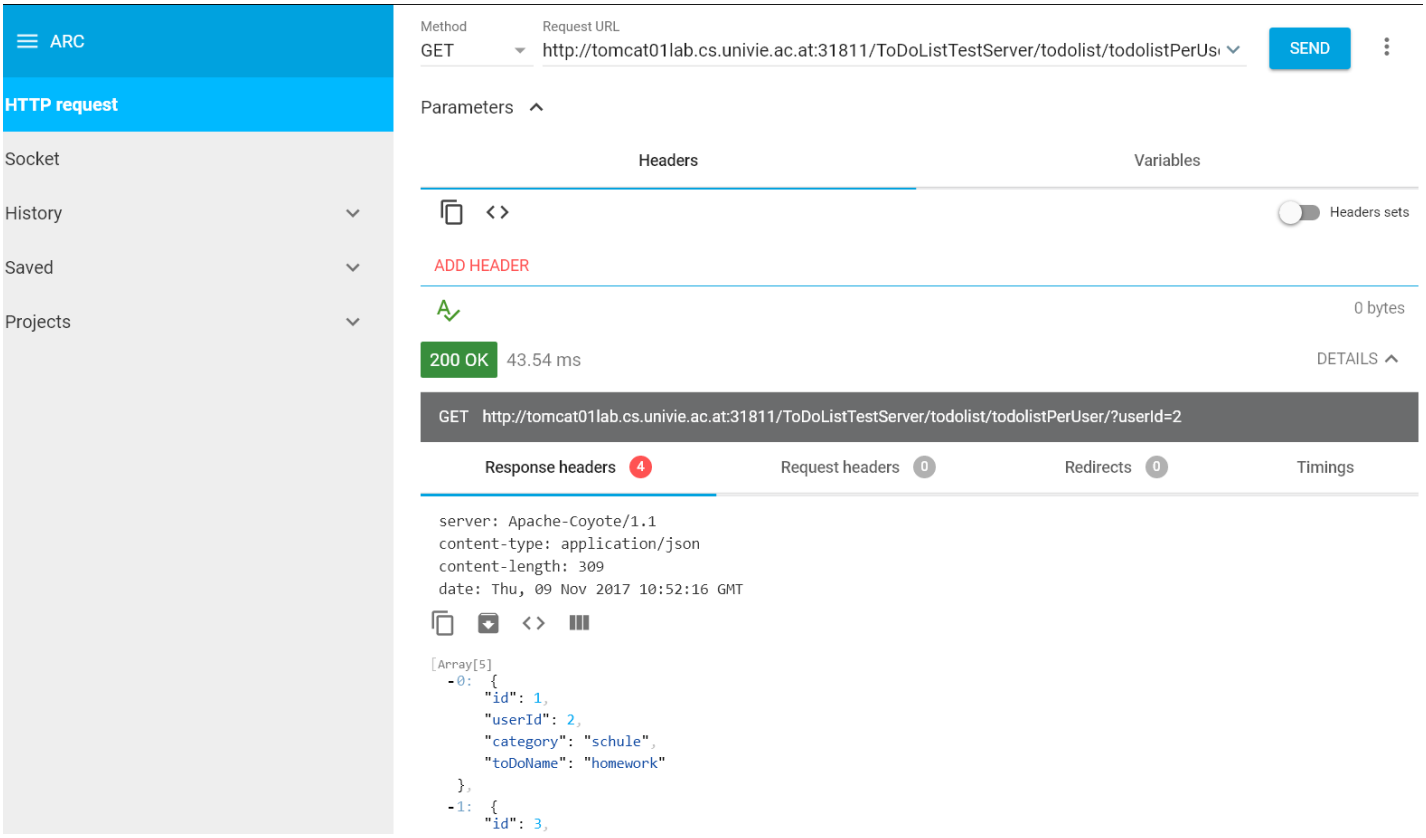


Figure 13 - request from client to ToDoListTestServer and receipt of response (via HTTP)

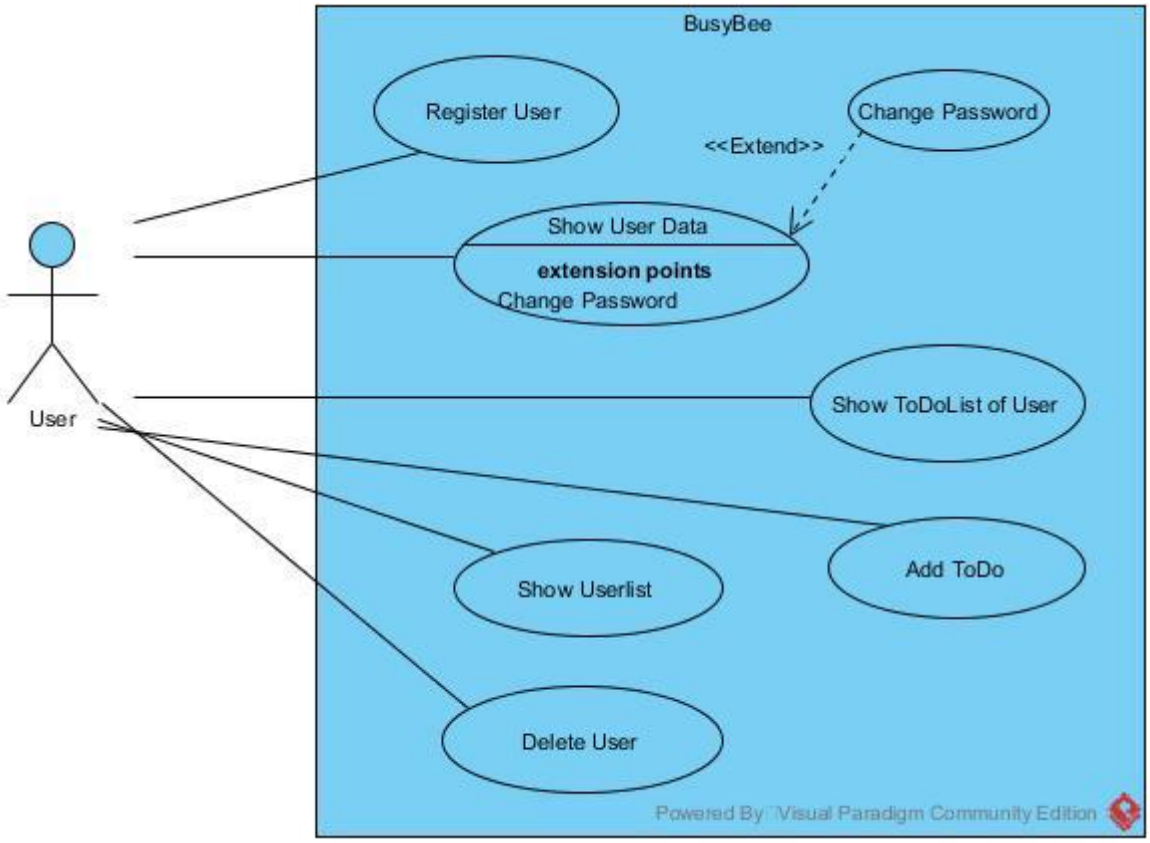
The ToDoListTestServer offers services to add to-dos and to show the to-do-list of the user who is logged in. The services can be found in the class ToDoListService.java in the package "services". Most services return JSON format, or text format, which have to be obtained and processed by the client-side, and needs to be presented in the user interface accordingly.

DSETestClient:

The DSETestClient provides the mockup user interface, which allows new users to register and to subsequently log in if the registration process was successful. Once logged in, the user can make use of the to-do-list services and of the user-related services. The mockup user interface for the isolated presentation of the MS1, consists of HTML/JSP pages, on which the respective functionalities are made available to the user. Upon successful login, the user is set as a session attribute. In the package controller of the DSETestClient project, the servlets, which take over the data and request from the user interface and connect to the server, which queries the database, are placed. The servlets also receive the corresponding response from the server(s) and process the obtained data, and then forward the data and/or message to the user interface (JSP/HTML). The communication takes place via HTTP requests and responses.

Use case diagram of BusyBee:

My part of the implementation comprises the user-related services (MS1) and the mockup of the to-do-list-related services:

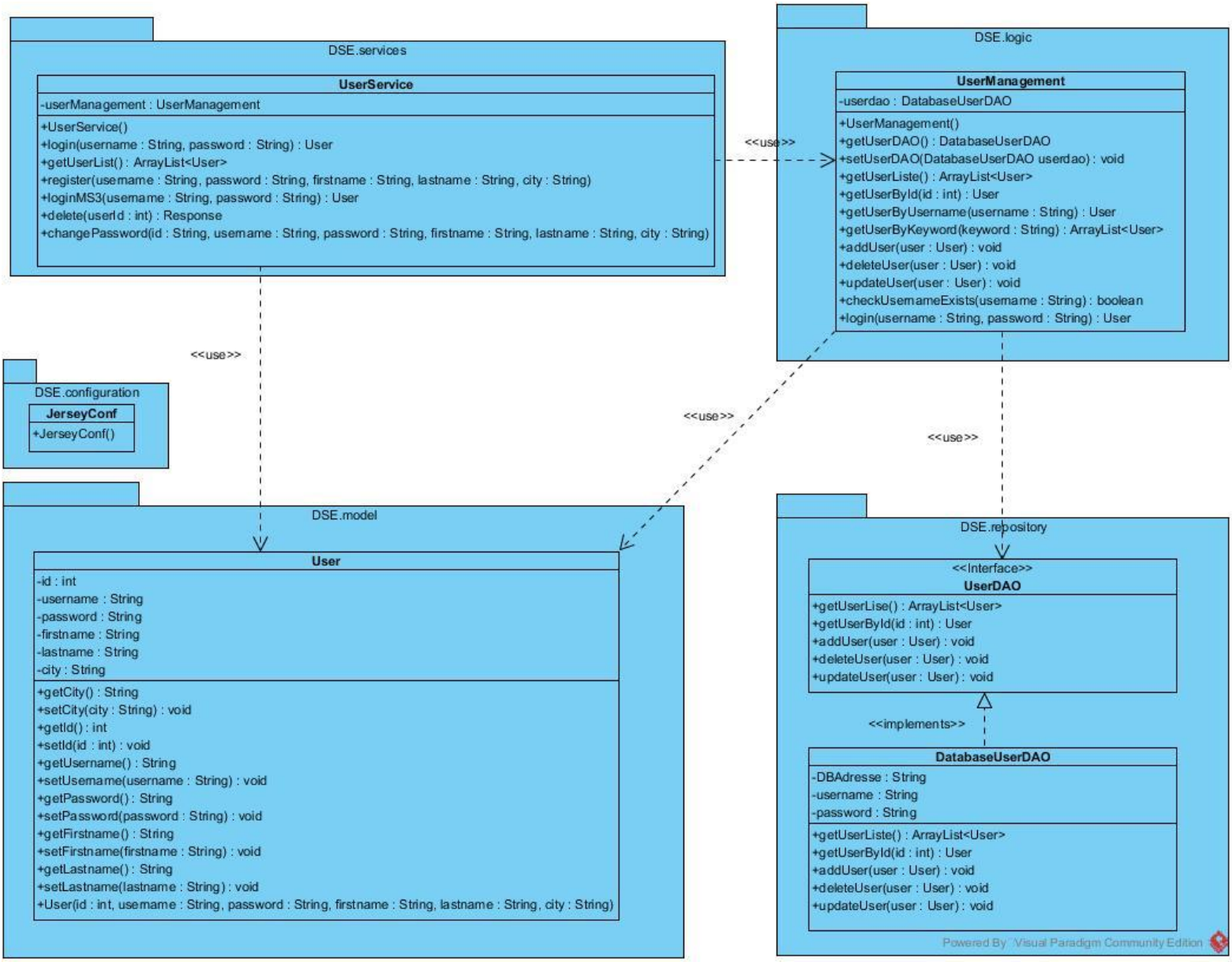


Note:

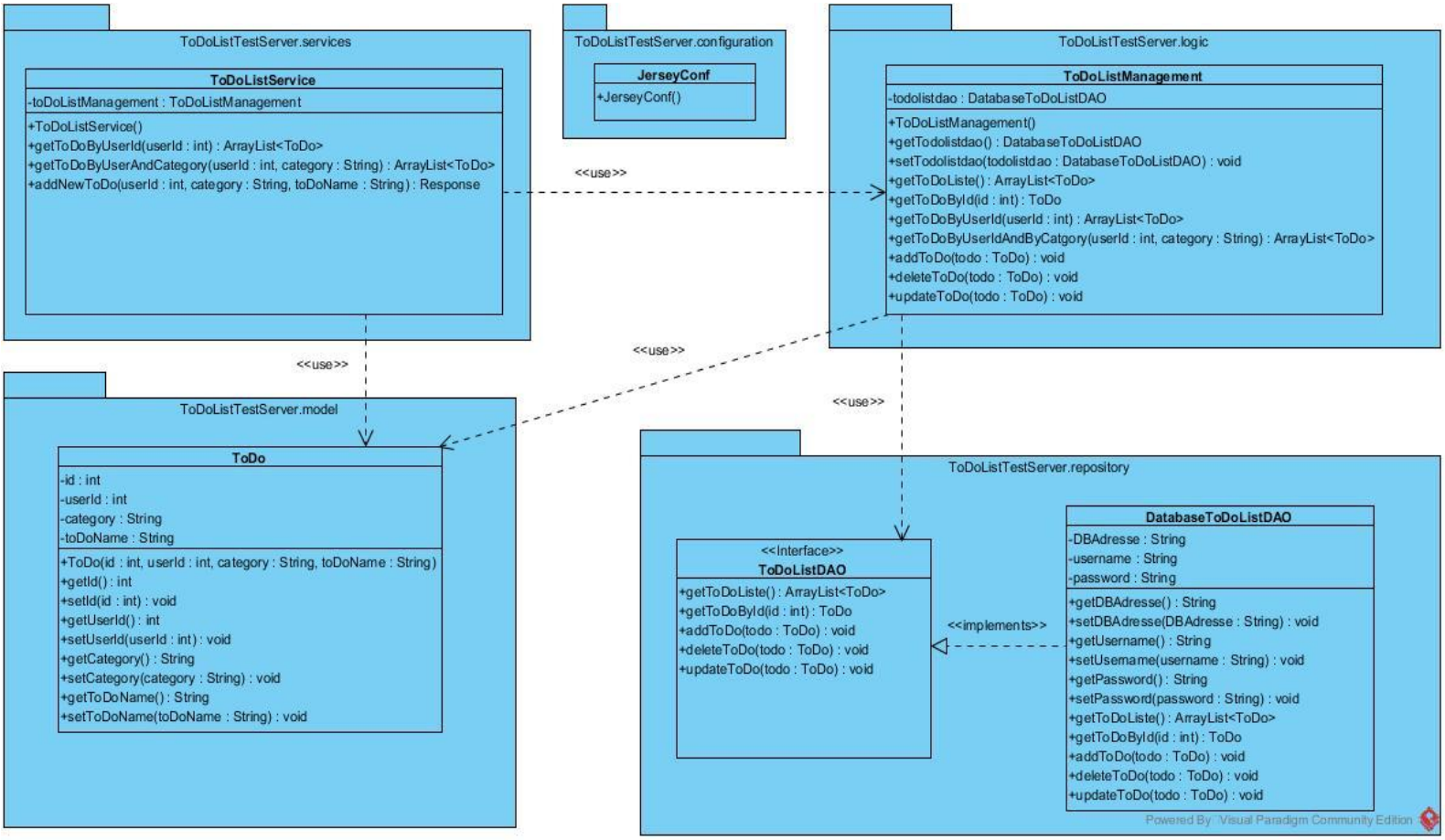
The Login has not been included in the use case diagram, as it is no actual use case according to VO SWE (Prof. Benkner).

The following class diagrams are intended to provide an overview on the structure of my three java projects:

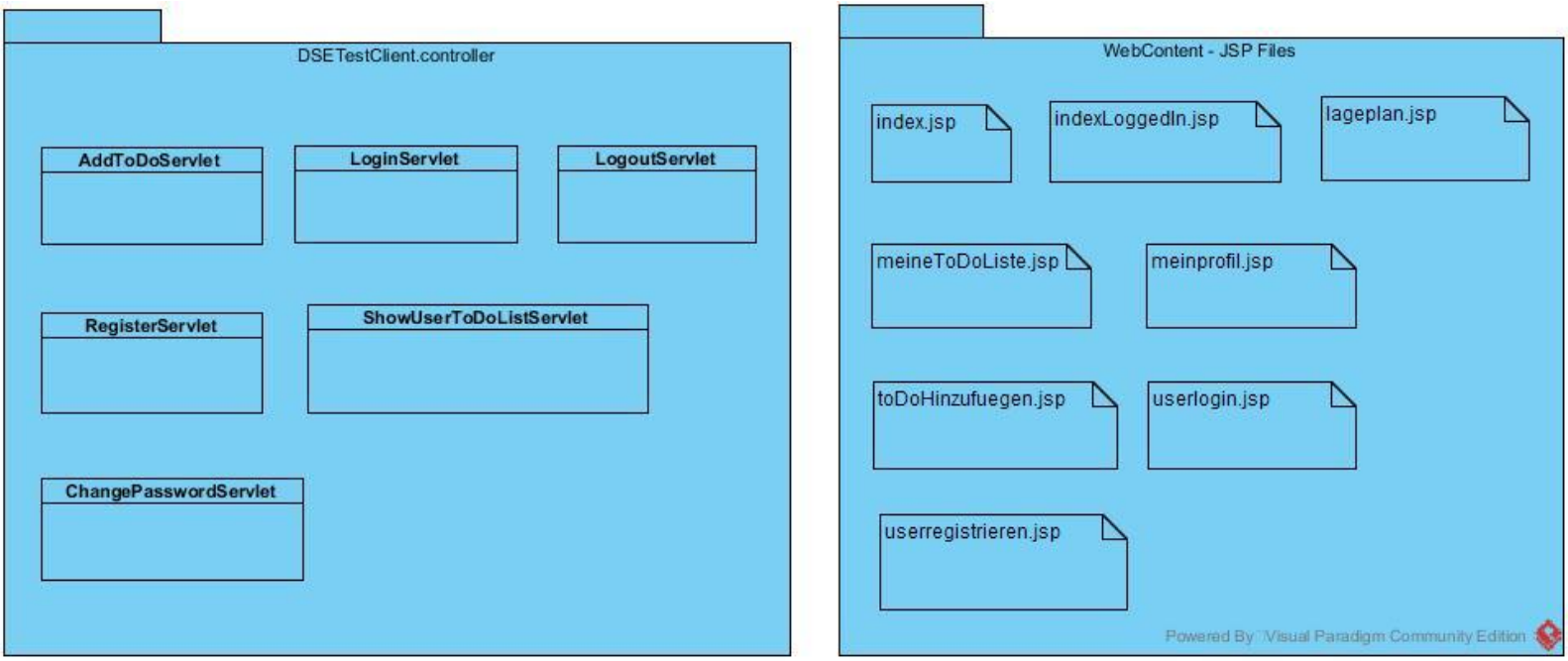
Class diagram of the java project DSE (user-related services / server-side):



Class diagram of the java project ToDoListTestServer (to-do-list-related services / server side):



Class diagram of the java project DSETestClient (client side):



Description of what the service interface looks like and how it can be contacted by the client/by the further MS:

The services that I implemented (i.e. MS1 and as a mockup service MS2) can be requested and contacted via the following service interface using the listed URI and HTTP communication (requests and responses). As return, the server will provide JSON or text format and the respective status code.

This the list of sample requests that can be made to use the user-related and (mockup) to-do-list-related services, which I implemented and deployed on the university's Tomcat server:

Services of the DSE-Server (which offers the user-related services):

- **Register user:**

http://tomcat01lab.cs.univie.ac.at:31811/DSE/users/register?username=informatikstudent&password=informatikstudent&firstname=michael&lastname=test&city=Salzburg

(Parameters username, password, firstname, lastname and city are QueryParam. It uses the get method.)

- **Login user (login):**

http://tomcat01lab.cs.univie.ac.at:31811/DSE/users/login

(Parameters username and password are sent as CookieParam. It uses the get method.)

- **Login user (loginMS3):**

http://tomcat01lab.cs.univie.ac.at:31811/DSE/users/loginMS3

(Parameters username and password are sent as HeaderParam. It uses the post method.)

- **Retrieve user list:**

http://tomcat01lab.cs.univie.ac.at:31811/DSE/users/userlist

(No parameters are sent. It uses the get method.)

- **Change user password:**

http://tomcat01lab.cs.univie.ac.at:31811/DSE/users/login

(Parameters userId, username, password, firstname, lastname and city are sent as CookieParam. It uses the post method.)

- **Delete user:**

http://tomcat01lab.cs.univie.ac.at:31811/DSE/users/login

(Parameter userId is sent as QueryParam. It uses the get method.)

Services of the ToDoListTestServer:

- **Add new to-do:**

http://tomcat01lab.cs.univie.ac.at:31811/ToDoListTestServer/todolist/newToDo?userId=2&category=softwareengineering&toDoName=hausuebung1-machen

(Parameters userId, category and toDoName are QueryParam. It uses the get method.)

- **Show to-dos of the user:**

http://tomcat01lab.cs.univie.ac.at:31811/ToDoListTestServer/todolist/todolistPerUser?userId=2

(Parameter userId is sent as QueryParam. It uses the get method.)

As already mentioned, the services which I implemented can be checked by using the ARC app or the test client for instance. The Figure 14 below show the request of the “userlist” from the DSE server. It returns the list of all registered users in the form of a JSONArray with the users included as JSONObjects that can be obtained and processed by the client.

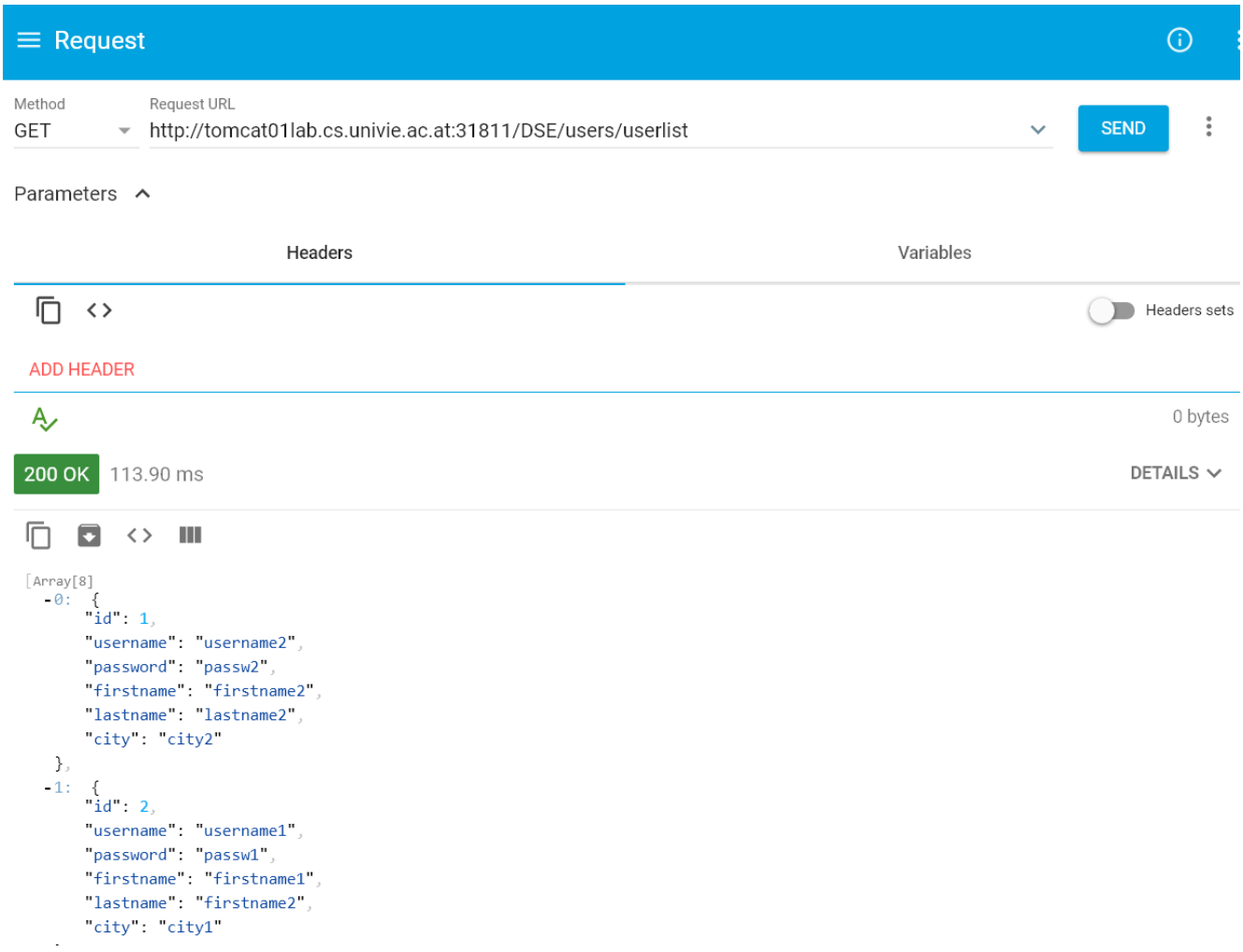


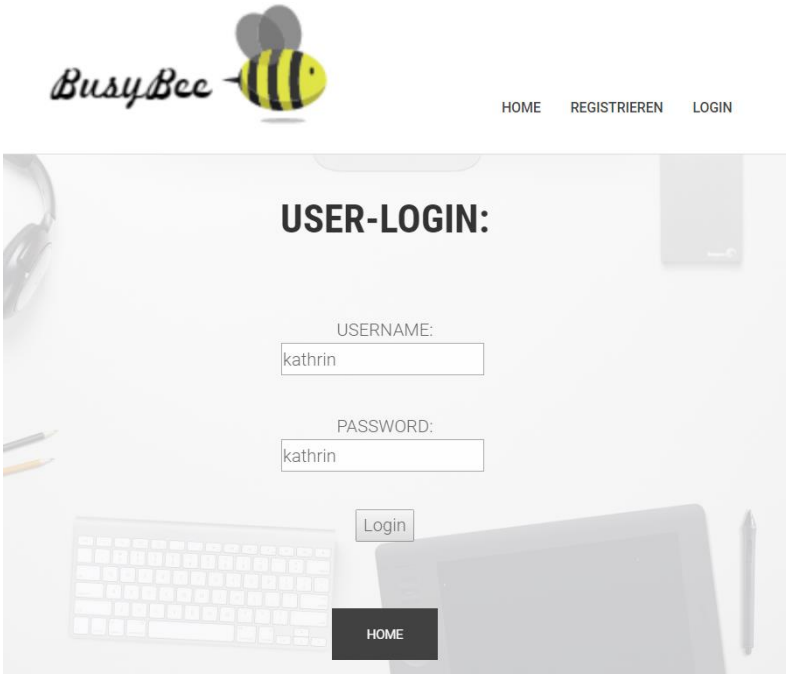
Figure 14 - request "userlist" from DSE Server

As a further opportunity to show the functionalities of my services, I implemented the DSETestClient, which provides a test user interface to make requests to DSE server and ToDoListTestServer to simulate the interplay of all three services in the isolated case that MS2 and MS3 are not available. The communication is carried out via HTTP and JAX-RS is used for the REST service.

User interface of DSETestClient:

1. Index.jsp (Main page – state of the user who is not logged in):

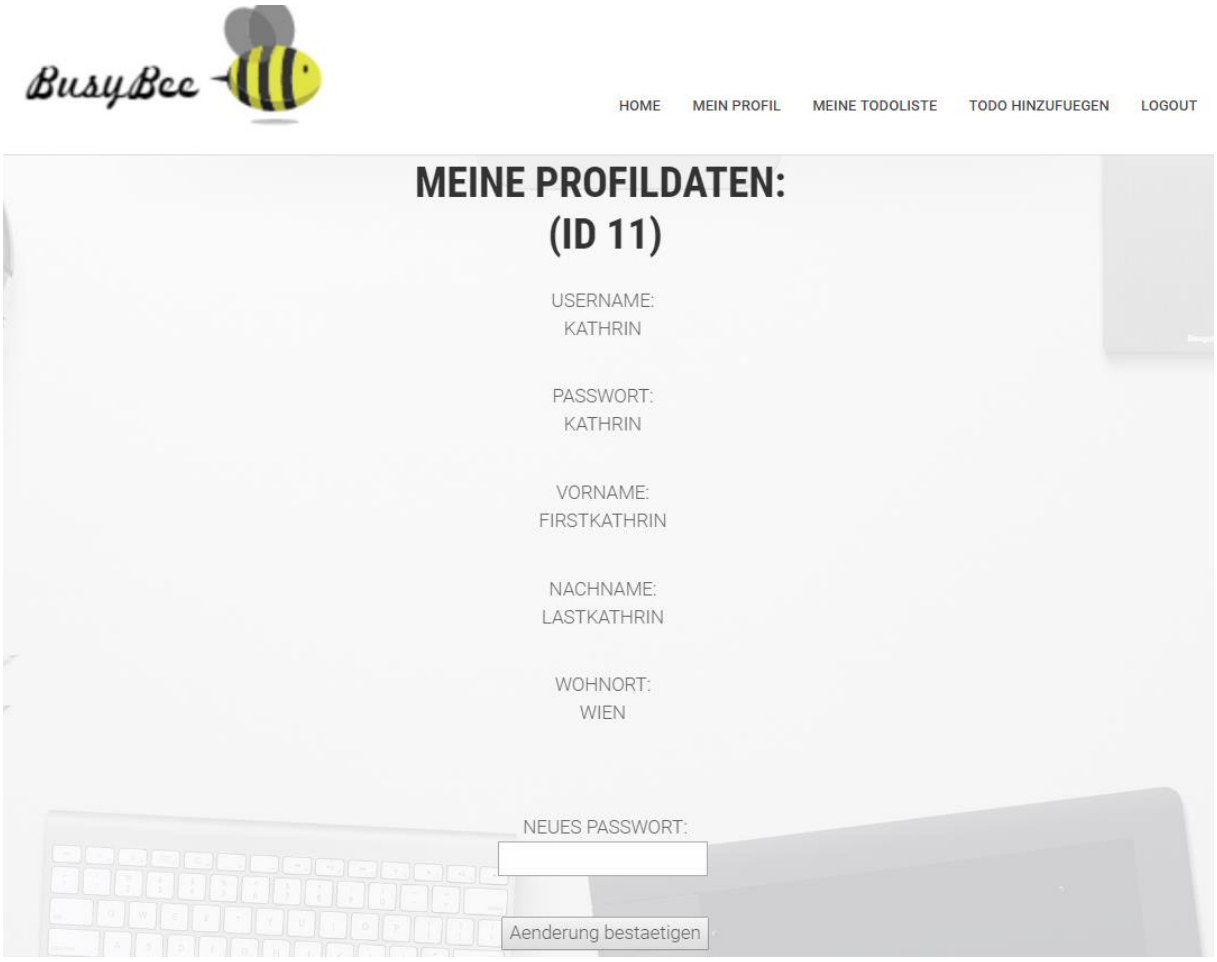
2. User login (entering username „kathrin“ and password „kathrin“):



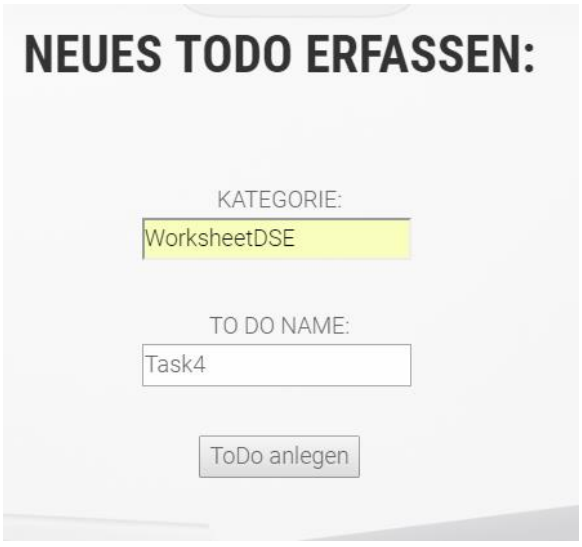
3. Main menu once a registered user is logged in:

4. "My profile" (showing the user data of a user who is logged in):

5. Change password form at the bottom of "My Profile":

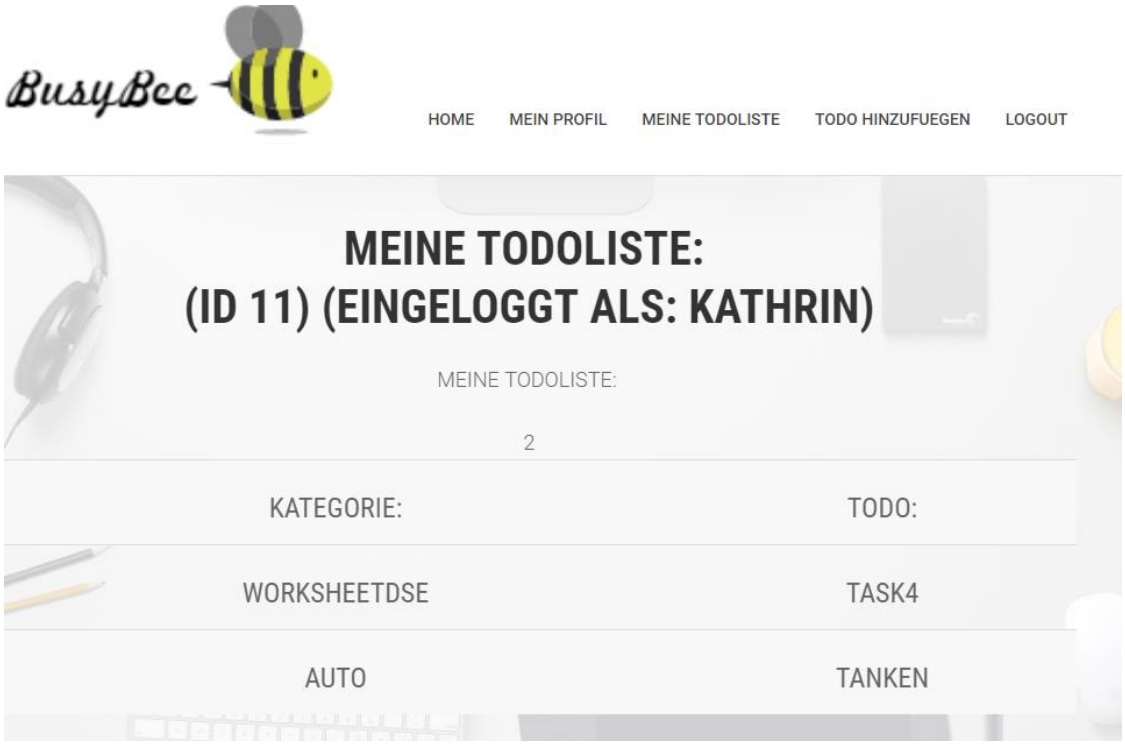


6. Add new to-do:



7. Feedback message, if to-do has been added correctly:

8. Show to-do-list of the user who is logged in ("My ToDoList"):



9. State after logout of the user (again index.jsp):

3. Quick Start Tutorial

This quick start tutorial is intended to describe how to download, install, set up and test the MS1 in the isolated case. The three microservices working together are described separately in the group report.

Firstly, the three java projects, which are all Maven projects, named DSE, ToDoListTestServer, and DSETestClient should be downloaded/pulled from gitlab MS1 folder in the submission system. The projects are all Maven projects, so the dependencies will normally already be available in the dependencies section of the pom.xml. If this is not the case, the dependencies can be added manually. These are the dependencies to add:

DSE: (for the user-related services)

```
<dependencies>
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>6.0.6</version>
</dependency>
<dependency>
  <groupId>org.glassfish.jersey.core</groupId>
  <artifactId>jersey-server</artifactId>
  <version>2.20</version>
 </dependency>

 <dependency>
  <groupId>org.glassfish.jersey.containers</groupId>
  <artifactId>jersey-container-servlet</artifactId>
  <version>2.20</version>
 </dependency>
<!-- https://mvnrepository.com/artifact/org.glassfish.jersey.media/jersey-media-json-jackson -->
<dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-json-jackson</artifactId>
    <version>2.25.1</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.thetransactioncompany/cors-filter -->
<dependency>
    <groupId>com.thetransactioncompany</groupId>
    <artifactId>cors-filter</artifactId>
    <version>2.6</version>
</dependency>

</dependencies>
```

DSETestClient: (for the mockup user interface)

```
    <dependencies>
  <!-- https://mvnrepository.com/artifact/org.json/json -->
<dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20170516</version>
</dependency>
<dependency>
  <groupId>org.glassfish.jersey.core</groupId>
  <artifactId>jersey-server</artifactId>
  <version>2.20</version>
 </dependency>

 <dependency>
  <groupId>org.glassfish.jersey.containers</groupId>
  <artifactId>jersey-container-servlet</artifactId>
  <version>2.20</version>
 </dependency>
  </dependencies>
</project>
```

ToDoListTestServer: (for the mockup to-do-list-related services)

```
 <dependencies>
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>6.0.6</version>
</dependency>
<dependency>
  <groupId>org.glassfish.jersey.core</groupId>
  <artifactId>jersey-server</artifactId>
  <version>2.20</version>
 </dependency>

 <dependency>
  <groupId>org.glassfish.jersey.containers</groupId>
  <artifactId>jersey-container-servlet</artifactId>
  <version>2.20</version>
 </dependency>
<!-- https://mvnrepository.com/artifact/org.glassfish.jersey.media/jersey-media-json-jackson -->
<dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-json-jackson</artifactId>
    <version>2.25.1</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.thetransactioncompany/cors-filter -->
<dependency>
    <groupId>com.thetransactioncompany</groupId>
    <artifactId>cors-filter</artifactId>
    <version>2.6</version>
</dependency>

</dependencies>
```

The DSE and the ToDoListTestServer java projects also use the CORS Filter, which has to be added not only as a dependency to the dependencies section in the pom.xml, but also in the Web Content in the web.xml, where cross-origin requests are not restricted in this case.

For example for the DSE, the web.xml has to be adapted to look like this[5]:

web.xml

```
    <filter>
        <filter-name>CORS</filter-name>
        <filter-class>com.thetransactioncompany.cors.CORSFilter</filter-class>
        <init-param>
            <param-name>cors.allowGenericHttpRequests</param-name>
            <param-value>true</param-value>
        </init-param>
        <init-param>
            <param-name>cors.allowOrigin</param-name>
            <param-value>*</param-value>
        </init-param>
        <init-param>
            <param-name>cors.allowSubdomains</param-name>
            <param-value>true</param-value>
        </init-param>
        <init-param>
            <param-name>cors.supportedMethods</param-name>
            <param-value>GET, HEAD, POST, OPTIONS</param-value>
        </init-param>
        <init-param>
            <param-name>cors.supportedHeaders</param-name>
            <param-value>*</param-value>
        </init-param>
        <init-param>
            <param-name>cors.exposedHeaders</param-name>
            <param-value>*</param-value>
        </init-param>
        <init-param>
            <param-name>cors.supportsCredentials</param-name>
            <param-value>true</param-value>
        </init-param>
        <init-param>
            <param-name>cors.maxAge</param-name>
            <param-value>-1</param-value>
        </init-param>
        </filter>

        <filter-mapping>
        <filter-name>CORS</filter-name>
         <url-pattern>/*</url-pattern>
        </filter-mapping>
```

[5] For the ToDoListTestServer project, the web.xml looks similar.

Before starting the services, one should make sure that the connection with the university VPN is active because the service connects to the university mySQL databases a0750881mysql3 and a0750881mysql4. Otherwise, it might occur that the connection to access the data is refused.

A first test of the service can be carried out via the ARC (Advanced REST Client) App of Google Chrome or other similar apps, as it has already been described in detail above in this report.

Once the connection with the UniVPN is successfully established, you can start the DSE server and the ToDoListTestServer and then run the index.jsp of the DSETestClient java project, which is the main page of the mockup user interface of my part. From this point onwards, you can navigate via the navigation bar on the upper side of the page. You can either use the existing username kathrin with password kathrin, which was already elaborated and shown in the form of a user interface case above in greater detail, or you can create (i.e. register) a new user and test the available services via the menu bar. In the chapter before on the service interface, a detailed description of the link extensions (URI) and the user interface used for the respective services and their necessary parameters and the resulting return types has been given. Please refer to the previous chapter on the service interface to obtain an explanation on how to use it. The user-related services as well as the mockup to-do-list-related services have all been deployed on the university Tomcat server on port 31811 (http://tomcat01lab.cs.univie.ac.at:31811/), as the screenshot below shows.

The Apache Software Foundation
http://www.apache.org/

Tomcat Web Application Manager

Message:	OK

Manager			
List Applications	HTML Manager Help	Manager Help	

Applications					
Path	Version	Display Name	Running	Sessions	Commands
/	*None specified*	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/DSE	*None specified*	DSE	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/ToDoListTestServer	*None specified*	ToDoListTestServer	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes