CA3 – Open Data with AngularJS and REST

Andreas, Christian & Jonas

1. Testing

Façade Testing:

Our strategy for testing our Façade was to create scenarios that would replicate the behavior of the methods used by our Façades, or by creating objects, and compare it to the result of the Façade methods.

For example:

```
@Test
public void authenticateUserTest() throws NoSuchAlgorithmException, InvalidKeySpecException {
    //Create new user
    User u = new User("Hans", PasswordHash.createHash("123"));
    u.AddRole("user");
    uf.saveUser(u);
    //Authenticating user returns List of user roles
    List<String> hansRoles = uf.authenticateUser("Hans", "123");
    //Creating user to compare
    User u1 = new User("Bo", "123");
    u1.AddRole("user");
    System.out.println(hansRoles);
    //Compare u1 roles to the Authenticated list
    assertEquals(u1.getRoles(), hansRoles);
}
```

This jUnit test tests the authenticateUser method in our UserFacade.

public List<String> authenticateUser(String userName, String password)

To test this method, we created a user, gave it the necessary attributes, and persisted it to our Database. We then created a List of Strings “hansRoles”, containing the list returned by our authenticateUser method, which in this case -should- be containing the roles of user “Hans”.

In order to see if our authenticateUser method returns the correct roles, we created another user “Bo”, and gave him the role “user”, since we expect the hansRoles list to only contain the string “user”. Lastly we call assertEquals and expect “u1.getRoles()” to be equal to the list hansRoles.

API Testing:

The strategy for the testing of our API consisted of replicating the correct and wrong way of interacting with our REST API, meaning we would impersonate an actual person trying to use our service, taking the proper route of actions in order to reach the correct endpoints and GET/POST/DELETE/PUT the desired data.

For example:

```
@Test
public void TestGetAllUsersWithLogin() {
    String json = given().
            contentType("application/json").
            body("{'username':'admin','password':'test'}").
            when().
            post("/login").
            then().
            statusCode(200).extract().asString();
    given().
            contentType("application/json").
            header("Authorization", "Bearer " + from(json).get("token")).
            get("/demoadmin/users").
            then().
            statusCode(200).body("username", hasItems("user", "admin", "both"));
}
```

In this screenshot we're attempting to test the endpoint "api/demoadmin/users". A GET call, which returns a json string containing all users in our Database.

In order to reach that end point, we first have to POST to the endpoint "api/login" with "username:"admin" as a part of the body. We do that since the "api/demoadmin/users" is only accessable by users with the admin role. After that we make a GET call to the "api/demoadmin/users" endpoint, which –should- return a StatusCode(200), along with a body containing a json String where "username" is "user", "admin" and "both", since they are all located in our database.

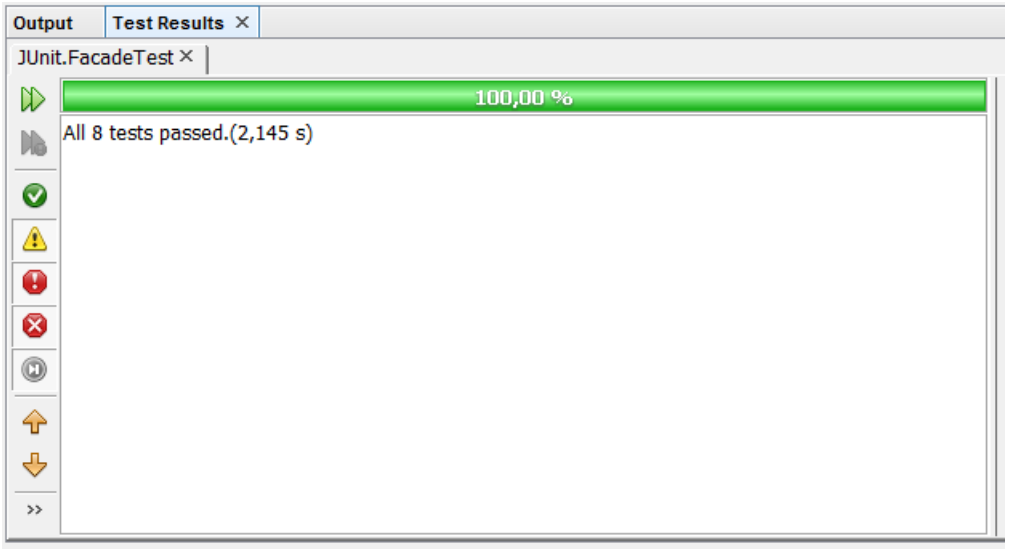
This test will obviously fail if any other users is added to our Database, and is a bit of a flaw, but we wanted to test on more than just the StatusCode(200), and couldn't think of another way.

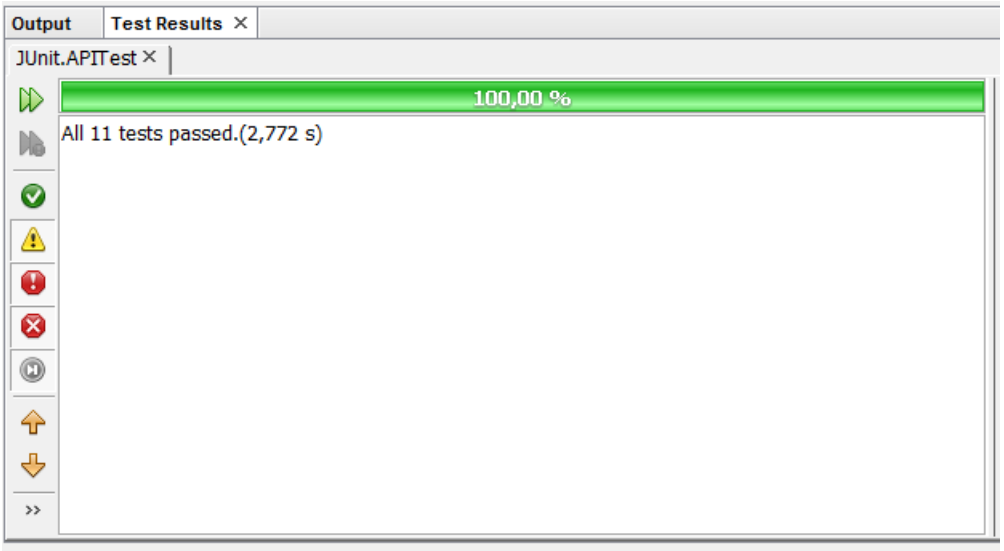
AngularJS Testing:

Sadly we did not feel like we had enough knowledge coding-wise in order to complete this task, and barely enough to interpret what the goal of the tests would be, apart from testing our controllers, functions and $http calls on the different views.

Overall Testing:

We ended up with a total of 19 JUnit tests, where just a few of them test malfunction triggering events, such as attempting to reach an endpoint, where your role is not allowed, or without being logged in.





2. Who did what?

We did not have any strategy in particular in mind when handing out who does what. In school we tried to split up some of the assignment, and help each other whenever an issue arose. We got stuck in the 3rd part of the assignment, and we decided it would be best if Andreas started working on the 4th part, so we that we didn't use all our working power on the 3rd one.

1. Setting up the Project Architecture
   We did everything in this section as a group, but Christian took care of Openshift.
2. Using Open Data, AngularJS and Cool Controls
   We did everything in this section as a group.
3. More Open Data (this time provided as XML)
   Once again, we did everything as a group.
4. The Admin (All Users Page)
   Andreas took care of this one alone.
5. Testing the Application
   Façade testing: Andreas.
   API testing: In school as a group, Jonas had done half of it at home.

3. How to test the system?

Website:
On our website you can create a user yourself, or use the users:
username: "user", password: "test" for user access.
username: "admin", password: "test" for admin access.
Navigate through our panel in the top to use all of our applications' functionalities.

Postman:
Via Postman you can access all of our endpoints with proper authentication (some without). Simply use basic authentication and use the username/password combinations listed in the last website description, or POST one yourself!

http://localhost:8080/AngSeedServer/api/saveUser
This will let you create a user. Simply copy the input below, and replace the empty strings with your own username and password!

```
{
        "username":" ",
        "password":" "

}
```

http://localhost:8080/AngSeedServer/api/demoadmin/users
This will return a json string containing all users (only admins allowed at this endpoint), so use basic authenticating and log in with the admin information above.

For more endpoints, simply look into our REST APIs in the GitHub link.

REST Assured:
Start the AngSeedServer in our project, and start the APITest in the Test Packages folder.
(You will need to run the DBCreator first for some test cases to pass, since the Database will be empty!)

QA Website:

For this CA, we were provided with a webpage, with the sole purpose of asking/answering questions. Both to help our own group, but also to help out others if we had a neat way to handle said problem. In theory, this would handle the "wasted" time it would take the teachers to clear the help list.

It wasn't really used much by our group, we only had the need to ask 2 questions. Although, the answers did pay off. They fixed our problems and much faster compared to waiting for help from a teacher.

We only created one user, "Chris", used by everyone in our group.