WAES HEROES APIs

AUTOMATION TEST

REQUIREMENTS 4
INSTRUCTIONS 4
1) GET SOLUTION 4
2) EXECUTE TESTS 4
3) RESULTS 5
ABOUT SOLUTION 6
FINDINGS 7
FINDING #1: 7
FINDING #2: 8
FINDING #3: 8
FINDING #4: 10
PROVIDED APP: UNIT TESTS COMMENTS 11

This document provides the details of test solution for proposed WAES heroes web site APIs, as well as instructions to run the implemented automation tests. Besides some other details regarding the assignment are included.

REQUIREMENTS

- Java 8 + JDK
- Maven
- **backend-for-testers** application provided by Waes, must be running.

INSTRUCTIONS

1) GET SOLUTION

Clone following repository -> **https://github.com/carpmatias/waes-test.git**

2) EXECUTE TESTS

Tests can be executed in 3 different optional groups.
In command line go to project root folder and execute the following, depending on the type of test you want to run:

➔ **Smoke:** Execute the main positive test case for each provided API.

```
mvn -PSmoke test
```

Execution time: *~10 seconds*.*

➔ **Regression:** Execute the main positive test case for each provided API, plus some other functional and integration scenarios.

```
mvn -PRegression test
```

Execution time: *~15 seconds*.*

➔ **Full Regression:** Execute the main positive test case for each provided API, plus some other functional and integration scenarios, and different combination of positive/negative scenarios for APIs with payloads in their request (**Sing Up and Update user tests**).

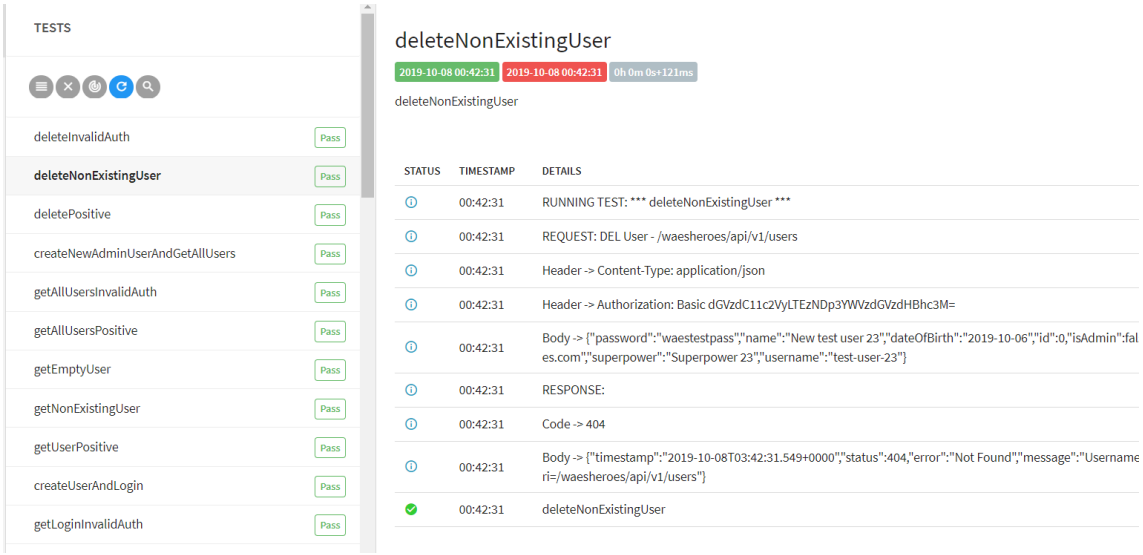
```
mvn -PFullRegression test
```

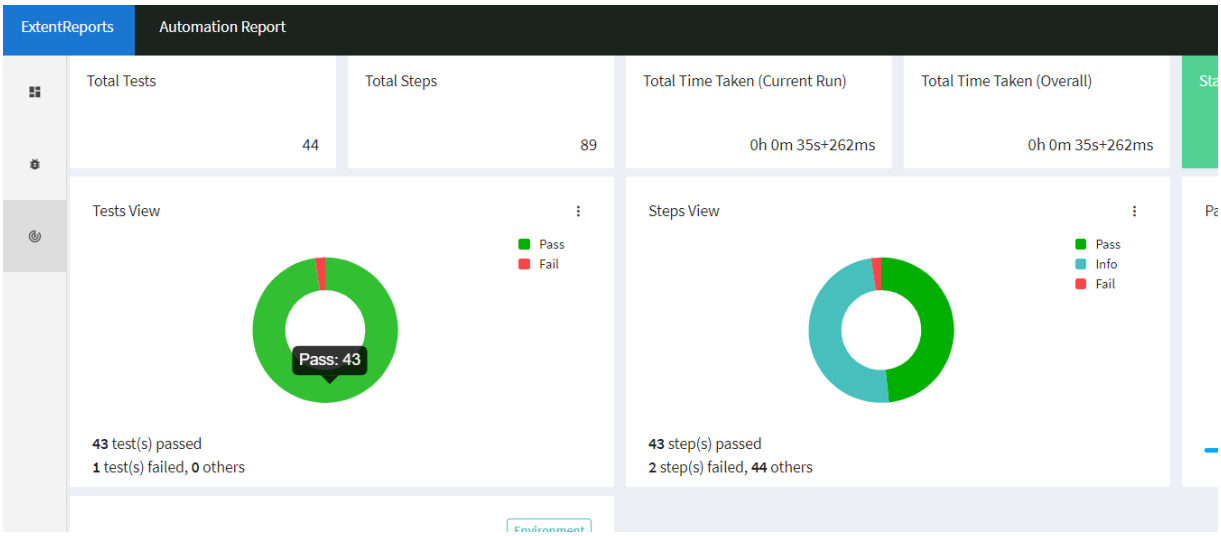
Execution time: *~1 min*.*

*Execution times are estimated when all libraries for the app has already been downloaded. If running in a system without the libraries, first execution will download them so it might take more time.

3) RESULTS

Once executed, results are logged in runtime in the console. Besides a report is generated with the details for each test and charts at *\target\surfire-reports\ExtentReport.html*





ABOUT SOLUTION

Solution was implemented in Java using:

- **TestNG**
- **RestAssured** – for APIs testing
- **ExtentReports** – for reporting
- **Log4j** – Logging

➔ For different full regression scenarios for ***Sign Up and Update user*** tests (which have a payload in their requests), an Excel file is used as data entry, allowing to configure custom scenarios specifying for each field in you want it to be:

- **FILLED** -> default valid value.
- **EMPTY** -> Present in the request, but with no value.
- **MISSING** -> Not present in the request.

And additionally, the **expected response code** to validate with, after performing each call.

Scenario	id	name	username	email	superpower	dateOfBirth	isAdmin	password	Expected code
WithId	FILLED	FILLED	MISSING	FILLED	FILLED	FILLED	FILLED	MISSING	200
EmptyName	FILLED	EMPTY	MISSING	FILLED	FILLED	FILLED	FILLED	MISSING	200
EmptyUsername	FILLED	FILLED	MISSING	FILLED	FILLED	FILLED	FILLED	MISSING	200
EmptyEmail	FILLED	FILLED	MISSING	EMPTY	FILLED	FILLED	FILLED	MISSING	200
EmptySuperpower	FILLED	FILLED	MISSING	FILLED	EMPTY	FILLED	FILLED	MISSING	200
EmptyDateOfBirth	FILLED	FILLED	MISSING	FILLED	FILLED	EMPTY	FILLED	MISSING	200
EmptyIsAdmin	FILLED	FILLED	MISSING	FILLED	FILLED	FILLED	EMPTY	MISSING	200
EmptyPassword	FILLED	FILLED	MISSING	FILLED	FILLED	FILLED	FILLED	MISSING	200
MissingName	FILLED	MISSING	MISSING	FILLED	FILLED	FILLED	FILLED	MISSING	200
MissingUsername	FILLED	FILLED	MISSING	FILLED	FILLED	FILLED	FILLED	MISSING	200
MissingEmail	FILLED	FILLED	MISSING	MISSING	FILLED	FILLED	FILLED	MISSING	400
MissingSuperpower	FILLED	FILLED	MISSING	FILLED	MISSING	FILLED	FILLED	MISSING	200
MissingDateOfBirth	FILLED	FILLED	MISSING	FILLED	FILLED	MISSING	FILLED	MISSING	200
MissingIsAdmin	FILLED	FILLED	MISSING	FILLED	FILLED	FILLED	MISSING	MISSING	200
MissingPassword	FILLED	FILLED	MISSING	FILLED	FILLED	FILLED	FILLED	MISSING	200
MissingNameAndUsername	FILLED	MISSING	MISSING	FILLED	FILLED	FILLED	FILLED	FILLED	200
MissingUsernameAndEmail	FILLED	FILLED	MISSING	MISSING	FILLED	FILLED	FILLED	FILLED	400
MissingEmailAndSuperpower	FILLED	FILLED	FILLED	MISSING	MISSING	FILLED	FILLED	FILLED	400
MissingSuperpowerAndDoB	FILLED	FILLED	FILLED	FILLED	MISSING	MISSING	FILLED	FILLED	200
MissingDoBAndAdmin	FILLED	FILLED	FILLED	FILLED	FILLED	MISSING	MISSING	FILLED	200
MissingAdminAndPassword	FILLED	FILLED	FILLED	FILLED	FILLED	FILLED	MISSING	MISSING	200

Scenarios can be added (if you're in a system with Microsoft Office) as pleased in the document and will run and be reported after the execution. File is located in *\src\main\java\data\waesdata.xlsx*

Similar solution can be implemented, for instance, with xml files, but for visual aspects and simplicity Excel was chosen.

A couple of tests in each Excel Sheet have wrong expected codes **in purpose** so this way they fail and it can be observed in the reports (just as a sample of failed test case).

FINDINGS

When developing solution, creating scenarios and performing tests, following issues/suggestions were found:

FINDING #1:

Summary: **New admin user created can't retrieve all users**

Description:

GET All Users endpoint -> http://localhost:8081/waesheroes/api/v1/users/all does not work for a new user created with *admin* field set as *true*.
(This can be thought as expected, considering the provided user to perform this call is the only with that condition).
Besides, its returned code and message for that call doesn't give much information about the error.

Step 1) CREATE USER

```
curl -X POST http://localhost:8081/waesheroes/api/v1/users -H 'Accept: */*' -H 'Content-Type: application/json'
    -d '{
            "username": "newadminuser",
            "isAdmin": true,
            "dateOfBirth": "1985-09-18",
            "email": "newadminuser@wearewaes.com",
            "name": "New Admin User",
            "password": "admin",
            "superpower": "Kaioken"
}'
```

RESPONSE:

CODE: **201**
RESPONSE BODY:

```
{
    "id": 4,
    "name": "New Admin User",
    "username": "newadminuser",
    "email": "newadminuser@wearewaes.com",
    "superpower": "Kaioken",
    "dateOfBirth": "1985-09-18",
    "isAdmin": true
}
```

Step 2) ATTEMPT TO GET ALL USERS (with created admin user)

```
curl -X GET http://localhost:8081/waesheroes/api/v1/users/all -H 'Accept: */*' -H 'Authorization: Basic bmV3YWRtaW51c2VyOmFkbWlu'
```

RESPONSE:

CODE: **403**
RESPONSE BODY:


```
{
    "timestamp": "2019-10-08T00:59:07.013+0000",
    "status": 403,
    "error": "Forbidden",
    "message": "Forbidden",
    "path": "/waesheroes/api/v1/users/all"
}
```


FINDING #2:

Summary: **When retrieving single user, empty value is not validated.**

Description:

GET User endpoint -> http://localhost:8081/waesheroes/api/v1/users/details, does not validate "empty" users. When sending an empty user response is as follow:

```
curl -X GET 'http://localhost:8081/waesheroes/api/v1/users/details?username=
```

RESPONSE:

CODE: **404**
RESPONSE BODY:


```
{
    "timestamp": "2019-10-08T03:04:22.427+0000",
    "status": 404,
    "error": "Not Found",
    "message": "Username  does not exist.",
    "path": "uri=/waesheroes/api/v1/users/details"
}
```


A validation and response notifying that username field can't be empty would be advisable.

FINDING #3:

Summary: **Sign up response messages are not very representative/informative for the user when sending mandatory fields empty or not sending them (missing fields).**

Description:

When performing calls in different negative scenarios to POST Sign up user -> http://localhost:8081/waesheroes/api/v1/users, response messages are not very informative:

Example 1) -> Empty username

curl -X POST http://localhost:8081/waesheroes/api/v1/users -H 'Accept: */*' -H 'Content-Type: application/json'

```
-d ' {
  "password":"waestestpass",
  "name":"test user632",
  "dateOfBirth":"2018-12-12",
  "isAdmin":false,
  "email":"newtestuser889@wearewaes.com",
  "superpower":"Kaoiken 72",
  "username":""
}'
```

RESPONSE:

CODE: **400**
RESPONSE BODY:

```
{
   "timestamp":"2019-10-08T19:08:35.284+0000",
   "status":400,
   "error":"Bad Request",
   "message":"Cannot pass null or empty values to constructor",
   "path":"uri=/waesheroes/api/v1/users"
}
```

Example 2) -> Missing username

curl -X POST http://localhost:8081/waesheroes/api/v1/users -H 'Accept: */*' -H 'Content-Type: application/json'

```
-d ' {
  "password":"waestestpass",
  "name":"test user333",
  "dateOfBirth":"2018-12-12",
  "isAdmin":false,
  "email":"newtestuser855@wearewaes.com",
  "superpower":"Kaoiken 610"
}'
```

RESPONSE:

CODE: **400**
RESPONSE BODY:

```
{
   "timestamp":"2019-10-08T19:08:45.188+0000",
   "status":400,
   "error":"Bad Request",
   "message":"Validation failed for classes [com.waes.backend.model.User] during persist time for
groups [javax.validation.groups.Default, ]\nList of constraint
violations:[\n\tConstraintViolationImpl{interpolatedMessage='must not be null',
propertyPath=username, rootBeanClass=class com.waes.backend.model.User,
messageTemplate='{javax.validation.constraints.NotNull.message}'}\n]",
   "path":"uri=/waesheroes/api/v1/users"
}
```

FINDING #4:

Summary: **Update user response messages are not very representative/informative for the user when sending mandatory fields empty or not sending them (missing fields).**

Description:

When performing calls in different negative scenarios to PUT user -> http://localhost:8081/waesheroes/api/v1/users, response messages are not very informative:

➔ Missing email

```
curl -X PUT http://localhost:8081/waesheroes/api/v1/users -H 'Authorization: Basic ZGV2OndpemFyZA==' 
  -d '{
   "password":"waestestpass",
   "name":"New test user 108",
   "dateOfBirth":"2019-10-06",
   "id":5,
   "isAdmin":false,
   "superpower":"Henki dama",
   "username":"test-user-108"
}'
```

RESPONSE:

CODE: 400
RESPONSE BODY:

```
{
   "timestamp":"2019-10-08T23:32:25.299+0000",
   "status":400,
   "error":"Bad Request",
   "message":"Could not commit JPA transaction; nested exception is 
javax.persistence.RollbackException: Error while committing the transaction",
   "path":"uri=/waesheroes/api/v1/users"
}
```

PROVIDED APP: UNIT TESTS COMMENTS

To give a brief paragraph with an opinion about the unit tests present in the given solution, on the one hand, the **positive** things I see about them is that they look very simple, easy to read and to maintain, as every unit test should be. They also focus at very component level as well (at least most of them).
On the other hand, as few **negative** things I could mention are that maybe not ALL the components are included, even *POST or PUT User* test with a single call could be there. Also, as mentioned before, the idea of unit tests is to be very focused in just ONE component, simple and easy to maintain, so I would say that in the case of TC *deleteByUsernameAndEmail*, it's not necessary to perform the DELETE call and then the GET to validate, validating that each component respond as expected should be enough for the unit tests suite, then in our functional tests we will validate multiple integration scenarios.