ATD:

Acceptance Testing Document

POLITECNICO
MILANO 1863

AY 2023/2024

Alessandro Fornara Simone Colecchia

Prof. Matteo Rossi

Version 1.0

February 10th, 2024

SUMMARY

1. TESTED PROJECT

- **Authors**
    - Alessandro Masini
    - Davide Grazzani

- **Repository link**
    - https://github.com/d-graz/GrazzaniMasini

- **Considered documents**
    - RASD: Requirements Analysis Specification Document
    - DD: Design Document
    - ITD: Implementation and Testing Document

2. INSTALLATION

For the installation phase, we followed the instructions provided in the ITD document (section 5). First, we set up the testing environment, cloning the GitHub repository with the source code. Then, according to what specified in the dedicated ITD section, we proceeded with the installation of all the components needed for the deployment of the prototype to be tested.
We proceeded as follows.

A. DBMS

- Unfortunately, in the dedicated section, we haven't found any link to MariaDB website for the installation, so we downloaded it from here.

- After that, we have successfully managed to configure the database following the provided instructions. However, we noticed that the password used by the application to access the database (written in the *application.properties*) was different from the one reported in the ITD instructions ("*se2*" instead of "*password*"). Since there was no instruction on this matter, we assumed the password was already correct, resulting in a failure in the first execution of the application.

B. BACKEND

After installing the DBMS, we proceeded with the configuration of the backend.

The installation went smoothly, except for the fact that the instructions are not for all kinds of OS: in fact, there is a step which works only on Linux and is unnecessary on Windows, although this is not specified.

```
#give to the maven file execution permissions
chmod +x ./mvnw
```

C. FRONTEND

Finally, we installed the frontend. The instructions were clear and correct, and following them we succeeded in installing all the necessary modules and load the GUI.

3. ACCEPTANCE TEST CASES

Following the installation and setup phase, we started with the acceptance testing.
In this section, the selected strategy is described, together with all the encountered issues.

A. General issues

After starting the application for the first time and signing up, we noticed that the database was empty: there were no registered users, no examples of tournaments, no battles etc. This represented an issue, since we had no idea of how the data was stored and no instruction on how to use MariaDB and visualize data had been provided. After some research, we managed to solve the problem.

After checking the ITD document, we noticed that in section 2.8 is mentioned that the application was also running on a cloud server which was reported to have a sample of database. However, it was empty too. So, we went back to the local application and performed an initial phase of populating the database with some entities.

Finally, we noticed that there wasn't any log for the application server for almost all of the functions, making it hard to understand what was going on, especially for a testing process.

B. Test strategy

To test the application, we analyzed the use cases provided in the RASD and, for the most significant ones, tried to emulate the flow of events verifying the results and how exceptions were handled.

We noticed that the "*actually implemented*" requirements in the ITD are marked with a ✓ and some functionalities were marked with a ~ and others were not marked. We assumed that ✓ means implemented, no mark means not implemented and ~ means partially implemented. However, after testing, we noticed that almost all ~ requirements were actually not implemented correctly or at all.

C. Test scenarios

For all the selected test scenarios, we took as references the relative use case (from the RASD) and the mark (✓ or ~) in the ITD for all the involved requirements. On this base, we tested some special cases and observed the application reactions.

Steps, cases, and results performed are reported below. In particular, results are marked as “SUCCESS”, meaning that the application responded correctly to the test case, or as “FAILED”, meaning that the application does not behave as described in the relative use case, allowing the user to perform forbidden actions, possibly corrupting the database or the application itself.

N.B.: NOTIFICATION

For all the test scenarios involving some kind of notification, we checked the correct delivery in the “notification center”. All kinds of notification are delivered correctly but, unfortunately, the dedicated section of the GUI doesn’t work well: the “dismiss” or “accept” commands are not linked to the correct notification but instead they always apply changes on the last one. Moreover, sometimes the “accept” option brings to an error page of the frontend.

➢ LOGIN/REGISTER

Test steps:

i. Provide requested information
ii. Click on “Login” / “Register”

Test cases:

1) Signing up a user without providing any information
2) Signing up a user twice with the same email (marked as an exception in the use case reported in the RASD)

Results:

1) FAILED: it was possible to register a phantom user
2) FAILED: it was possible to register a already registered user (even the phantom one)

➢ CREATE TOURNAMENT

Test steps:

i. Login
ii. Click on “Create a new tournament”
iii. Insert tournament name, description, subscription deadline and upload an image.
iv. Click on “CREATE A NEW TOURNAMENT”

Test cases:

1) Creation of a tournament with blank name
2) Creation of two tournaments with the same name (R8-R9 marked with ~ and exception in the use case reported in the RASD)
3) Creation of a tournament with invalid subscription deadline (set in the past)

4) Check if the tournament is visible to other users

Results:

1) FAILED: it was possible to create a tournament without a name
2) FAILED: it was possible to create two tournaments with the same name
3) FAILED: it was possible to create a tournament starting in the past (which is non-sense)
4) SUCCESS

➢ INVITE OTHER EDUCATORS TO MANAGE A TOURNAMENT

Test steps:

i. Create a tournament
ii. Click on the tournament and on "MANAGE"
iii. Click on "Invite a new educator"
iv. Provide an email address

Test cases:

1) Leaving the e-mail field blank
2) Trying to invite a non-registered user (marked as an exception in the use case reported in the RASD)
3) Trying to invite oneself

Results:

1) The server responds with 500 status code, the exception is not handled
2) The server responds with 500 status code, the exception is not handled
3) FAILED: it is possible to invite oneself and if the user clicks on "Accept" the frontend displays an error page with an uncaught runtime error and the server displays an unhandled exception

➢ CREATE BATTLE

Test steps:

i. Click on a tournament where the user is allowed to create a battle and on "MANAGE"
ii. Click on "Create new battle"
iii. Fill the fields

Test cases:

1) Creating a battle with blank name
2) Creating a battle with invalid start and end date
3) Creating a battle without code kata
4) Creating a battle without executable name

All are marked as an exception in the use case reported in the RASD

Results:

1) FAILED: it was possible to create a battle without a name
2) FAILED: it was possible to create a battle starting in the past or with an end date before the start date. A string on the server is displayed, but the battle is still saved in the database.
3) SUCCESS: the battle was not created but the frontend displays an error page with an uncaught runtime error
4) FAILED: it was possible to create a battle without executable name

In general, there is no explanation or instruction on what should be uploaded as a code kata and how the automated evaluation works (see below).

➢ SUBSCRIBE TO A TOURNAMENT

Test steps:

i. Click on a tournament
ii. Click on "SUBSCRIBE"

Test cases:

1) Trying to subscribe to a tournament with an expired subscription deadline
2) Trying to subscribe to a tournament the user has created or managing

Results:

1) SUCCESS: it wasn't possible to subscribe
2) SUCCESS: it wasn't possible to subscribe

➢ CREATE GROUP

Test steps:

i. Click on battle in a tournament the user is subscribed to
ii. Click on an incoming battle
iii. Click "ENROLL"
iv. Click "CREATE GROUP"
v. Provide a name

Test cases:

1) Creating two groups with the same name in the same battle (marked as an exception in the use case in the RASD)
2) Trying to create a group for an ongoing battle

Results:

1) FAILED: it was possible to create two groups with the same name in the same battle
2) SUCCESS: it wasn't possible

➢ JOIN AN EXISTING GROUP

Test steps:

i. User that is already in a group for a battle invites another user
ii. User that was invited click on the notification and clicks "ACCEPT"

Test cases:

1) A user which is not subscribed to the relative tournament tries to join the group by accepting the invitation
2) A user which is subscribed to the relative tournament but is not enrolled in the battle tries to join the group by accepting the invitation
3) A user tries to join a full team

Results:

1) SUCCESS: it wasn't possible to join the group, however trying to do so displays an error page of the frontend with an uncaught runtime exception
2) SUCCESS: it wasn't possible to join the group, however trying to do so displays an error page of the frontend with an uncaught runtime exception.
3) SUCCESS: it wasn't possible to join the group, however trying to do so displays an error page of the frontend with an uncaught runtime exception.

➢ AUTOMATED AND MANUAL EVALUATION

No instructions were provided to test the "automated evaluation" function: there are no guidelines on how to set up a code kata to be uploaded and we had no idea on where the automated repository creation was intended to happen. This made the testing process a bit difficult.

Another important issue was that, while creating a battle, it wasn't possible to set the time for deadlines: this means that we could create a battle that started either immediately (making it impossible to enroll and create groups) or after 24 hours, further increasing the difficulty of testing.

After asking for clarification, we tried to set up a workflow that interacted properly with the suggested form of code kata but, nonetheless, this functionality didn't seem to work: the process returned a 0 score to all the provided solutions. Moreover, by inspecting the source code, we noticed that the final score is computed multiplying the passed test ratio by 10, instead of 100, but we don't understand why.

➢ CLOSE A TOURNAMENT

Test steps:

i. Click on a tournament the user is allowed to close and on “MANAGE”
ii. Click on “Close Tournament”

Test cases:

1. Trying to close a tournament with an ongoing battle (marked as an exception in the use case reported in the RASD)

Results:

1. SUCCESS: it wasn’t possible to close that tournament, but the frontend displays an error page with an uncaught runtime error

After testing this functionality, we noticed that once the tournament has been closed, it is no longer visible in the application, so it’s impossible for the user to see the tournament final rank.

4. PROJECT INSPECTION

A. Quality of documentation

A.1. RASD

We analyzed the Requirement Analysis and Specification Document to have a reference in particular on the use cases (section 3.2.3). As specified in the previous chapter, we focused the attention on the events flow and which exceptions have been intended to be handled.

We noticed that, in our opinion, some important exceptions have not been considered at all or given for granted.

On the other hand, the most significant exceptions that have been specified have not been implemented or managed, neither backend-side nor frontend-side. Very few checks on data have been implemented and the application allows the user to perform many unconventional operations.

A.2. ITD

We employed the Implementation and Testing Document mostly as a guidance to the installation and usage of the application, but also as a summary for the implemented functionalities.

- Installation instructions:
  For what concerns section 5, they provided good and quite complete instructions on all the phases of the installation process. Except for low accuracy in some details, the provided steps and explanations seemed to us to be well elaborated and sufficient to properly set up the testing environment.

- Usage instructions:
  There is no sign of any instructions on how to use the application properly, either built into the GUI or outside of the platform (for example in the delivery folder). For most of the simpler functions, the interface appeared intuitive and allowed us to test them. On the other hand, as mentioned in section 2.C under "AUTOMATED AND MANUAL EVALUATION", we were not given any instructions on how we needed to upload a code kata as the developers had intended it to be, making it necessary to analyze the source code or to ask for information to understand.

B. Quality of code

For what concerns source code documentation, it was realized as in-line comments but overall, most of the methods are explained.

Although the code is pretty clear, it lacks in robustness: when subjected to erroneous inputs, most of the time, the system does not behave correctly. Moreover, these situations generate many exceptions, both on the frontend and backend, that have not been taken into consideration by the developers.

5. EFFORT SPENT

We tested the application together (for a total of 5 hours). For what concerns the redaction of this document, section 3 has been written together (2 hours). The rest has been written by Simone Colecchia (2 hours) and revised together (1 hour).