Minister of Education, Culture and Research of Moldova

Technical University of The Republic of Moldova

Software Engineering and Automatics

**REPORT**

Laboratory Work No.3 Software Testing

Performed by: A. Chirosca

st. gr. FAF-161

Checked by: M.Catruc

a.univ.

Chisinau 2018-2019

**Objectives:**

1. Training skills to test product’s functionalities
2. Forming the partitioning skills in the input data equivalence classes
3. Use decision tables and state and transition diagrams to create test cases

**Tasks:**

1. Develop a test scenario for 2-3 functionalities of a software application.

2. Determining the criteria for organizing equivalence classes and decision tables.

3. To make a conclusive test according to these criteria including testing the limits.

4. To highlight test cases where erroneous results can be obtained.

Task 1

In this laboratory work I chose an application that I developed during a month of a internship program. Main idea is an application used by a hospital, where we have doctors and patients. Patients register and for them will be created their own personal medical book with diagnosis, symptoms, analysis results and treatment, also here all the diagnosis will be categorized, if it is chronical or not. Patients can see doctor’s pages, leave a comment and make an appointment according to doctor’s schedule. Doctors profiles can be made only by admin and here they set their info page, see all the appointments that are not expired, and change their schedule (if they already have an appointment in a day, they can’t cancel it)

1. Test case 1 – Registration functionality

* Email has a correct format
* Password is at least 6 characters long
* Password and confirmation password match
* Password have at least one non letter digit
* Password have at least one uppercase digit
* Password have at least one symbol

1. Test case 2 – Doctor appointment

* Only patients can make an appointment to the doctor
* Patients can book an appointment according to doctor schedule
* Appointment fields are completed automatically by the system
* On appointment submit check all validations
* Doctors and patients should see in appointment list that they have a new appointment
* Patient can cancel the appointment

1. Test case 3 – Comment system

* Only patient can make a comment on doctor’s profile
* Doctors and patients can see them

Task 2 (Testing registration process)

*Equivalent classes:*

1. Nonvalid password and email and user was already registered
2. Valid password and email and user was not registered yet

*Decision table*

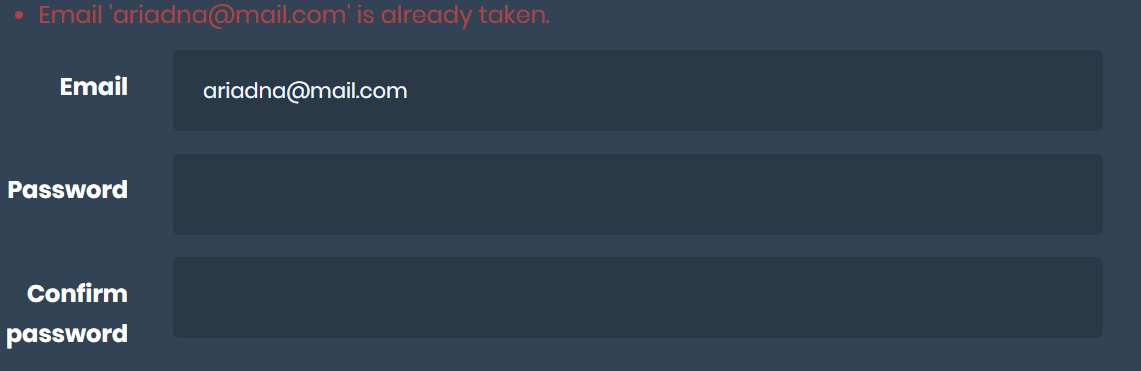
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | R1 | R2 | R3 | R4 |
| Valid mail | T | T | T | F |
| Valid password | T | T | T | F |
| Confirmation and password match | T | T | F | F |
| Non-registered user | T | F | T | F |
| **Result** | **✓** | **×** | **×** | **×** |

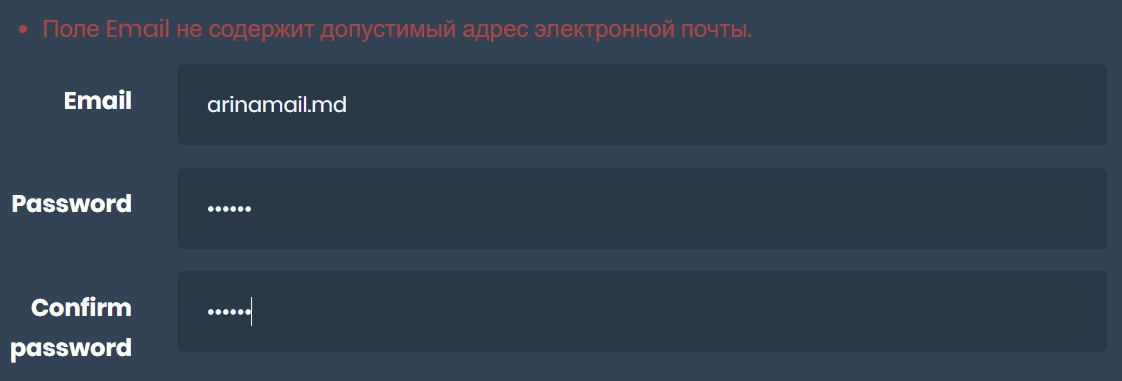
*Boundary values*

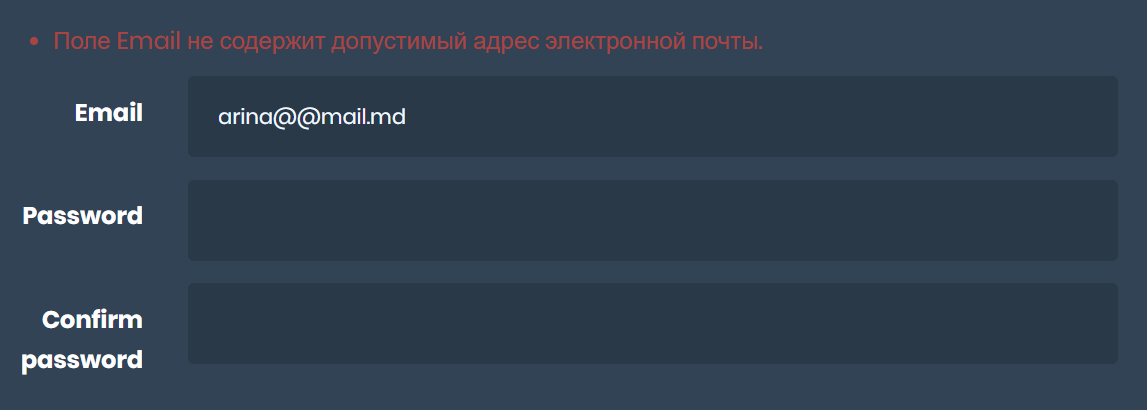
|  |  |
| --- | --- |
| Boundary | Value |
| Mail and password are not valid, email is already registered and password and confirmation doesn’t match | Validation message that user already exist |
| One field is not valid | Validation message for that field |
| All field are valid and user were not registered | Redirect to form for profile info. |

*Results:*

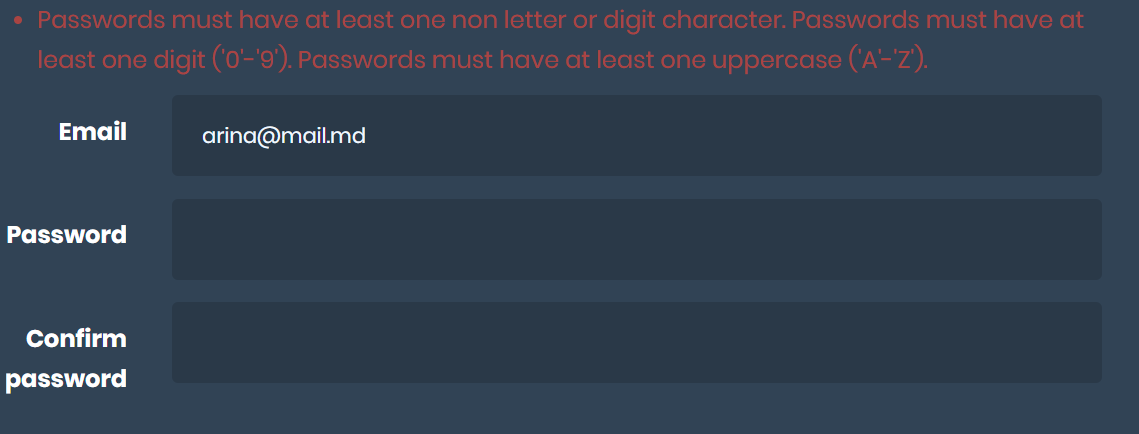
***Error cases***



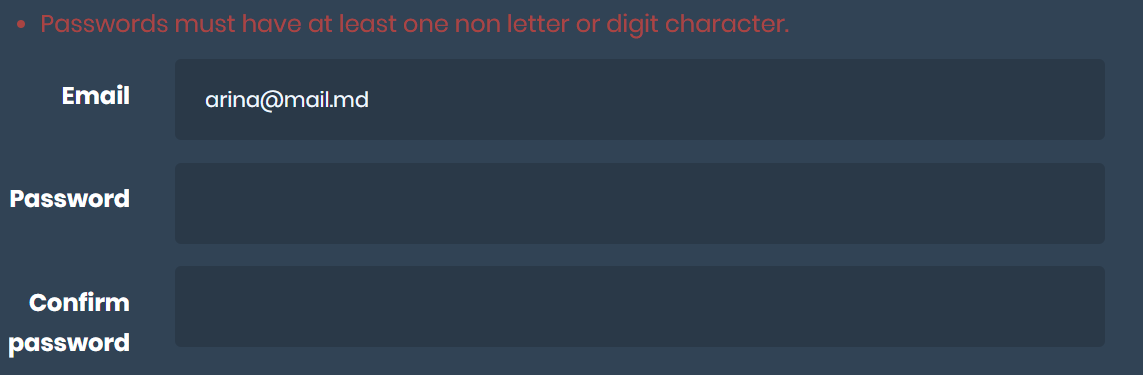




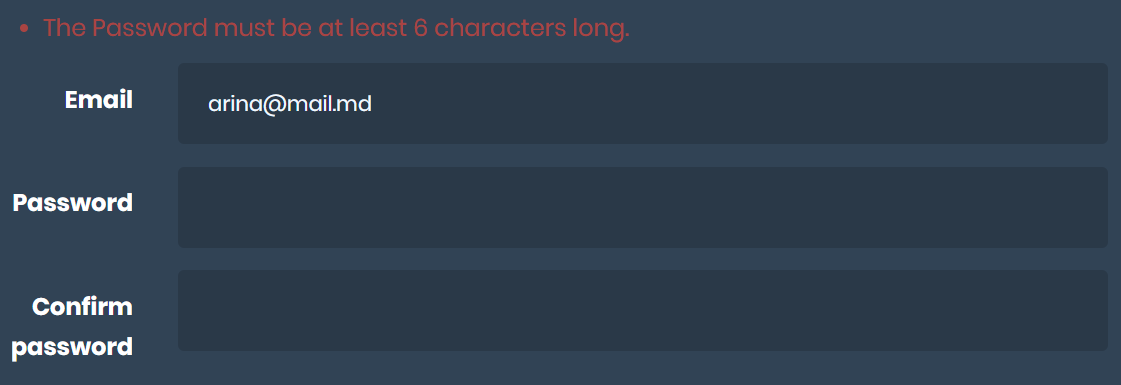
Introduced password: testtest

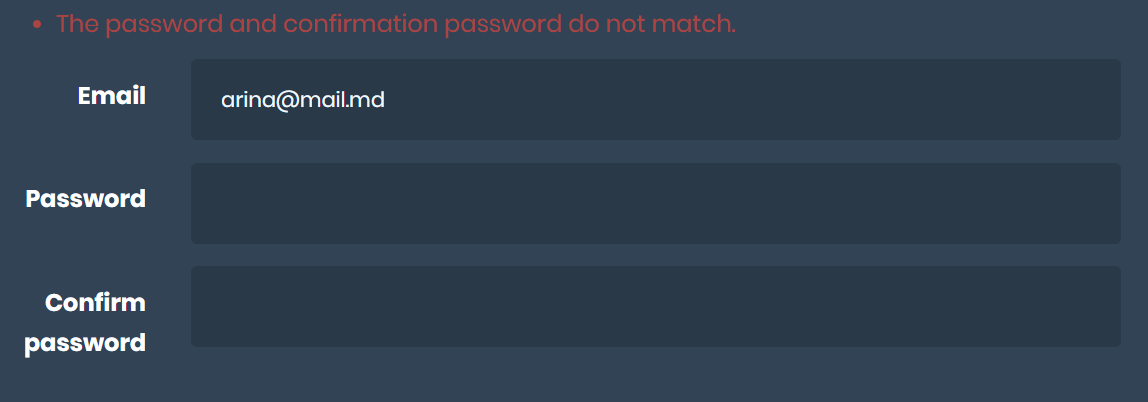


Introduced password: Passw111

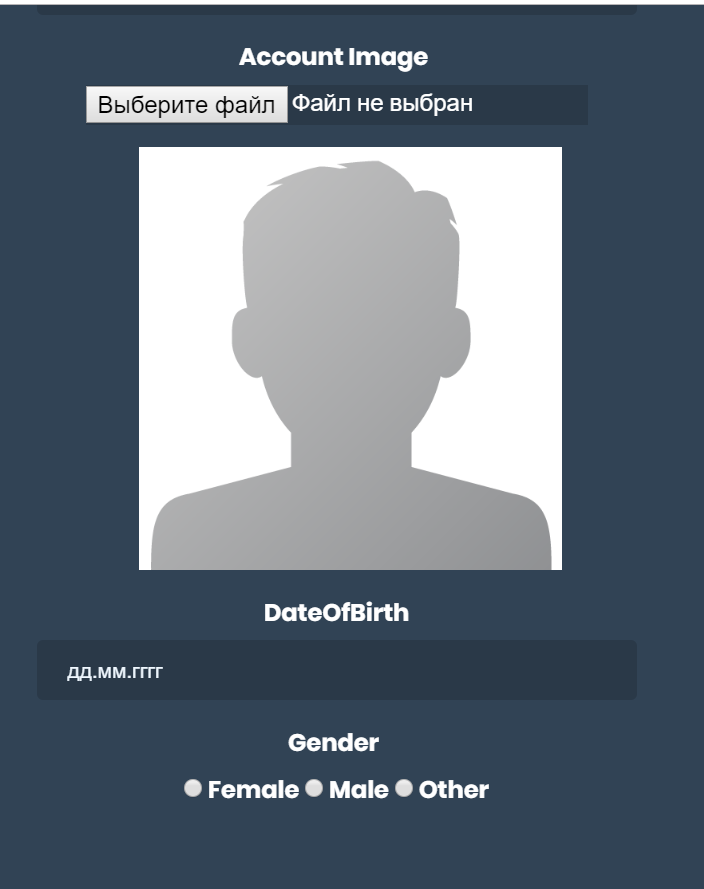


Introduced password: Pa1!





***Success case***



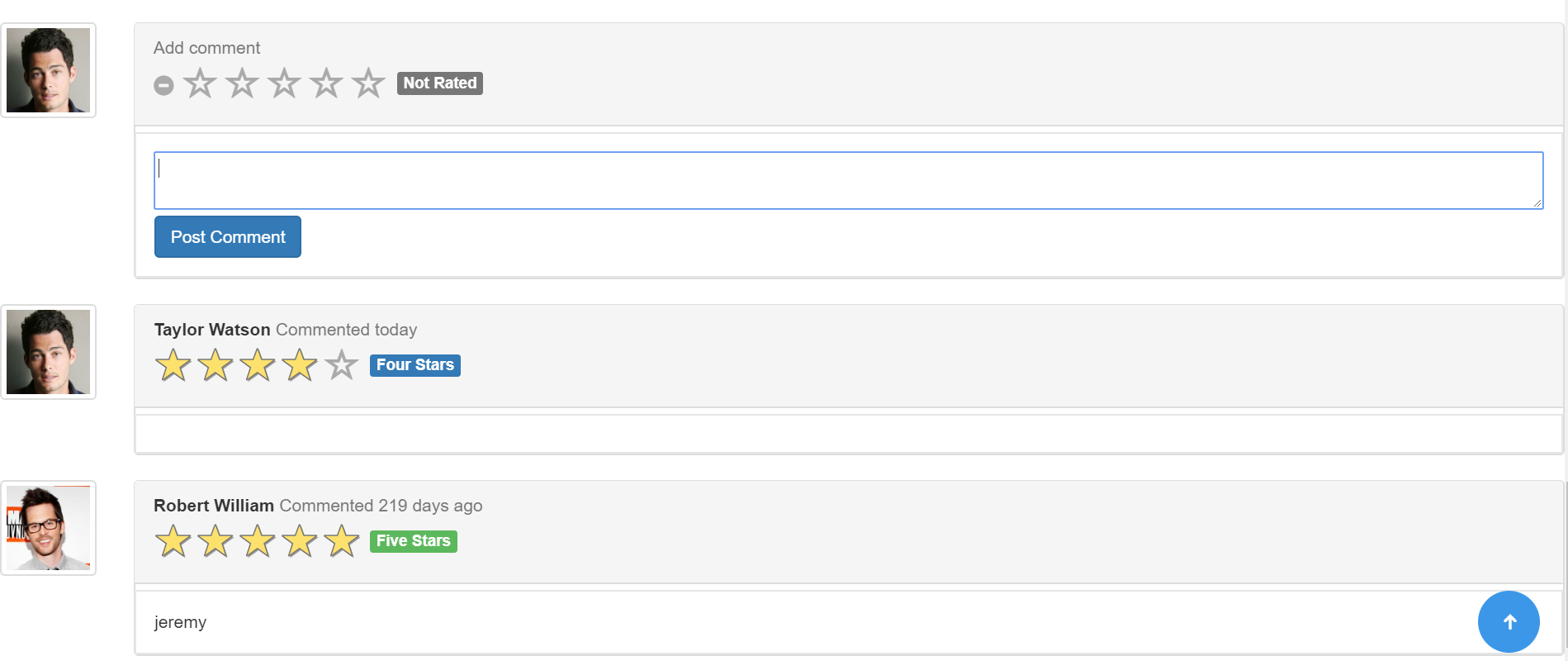
Testing the comment system

Equivalence classes:

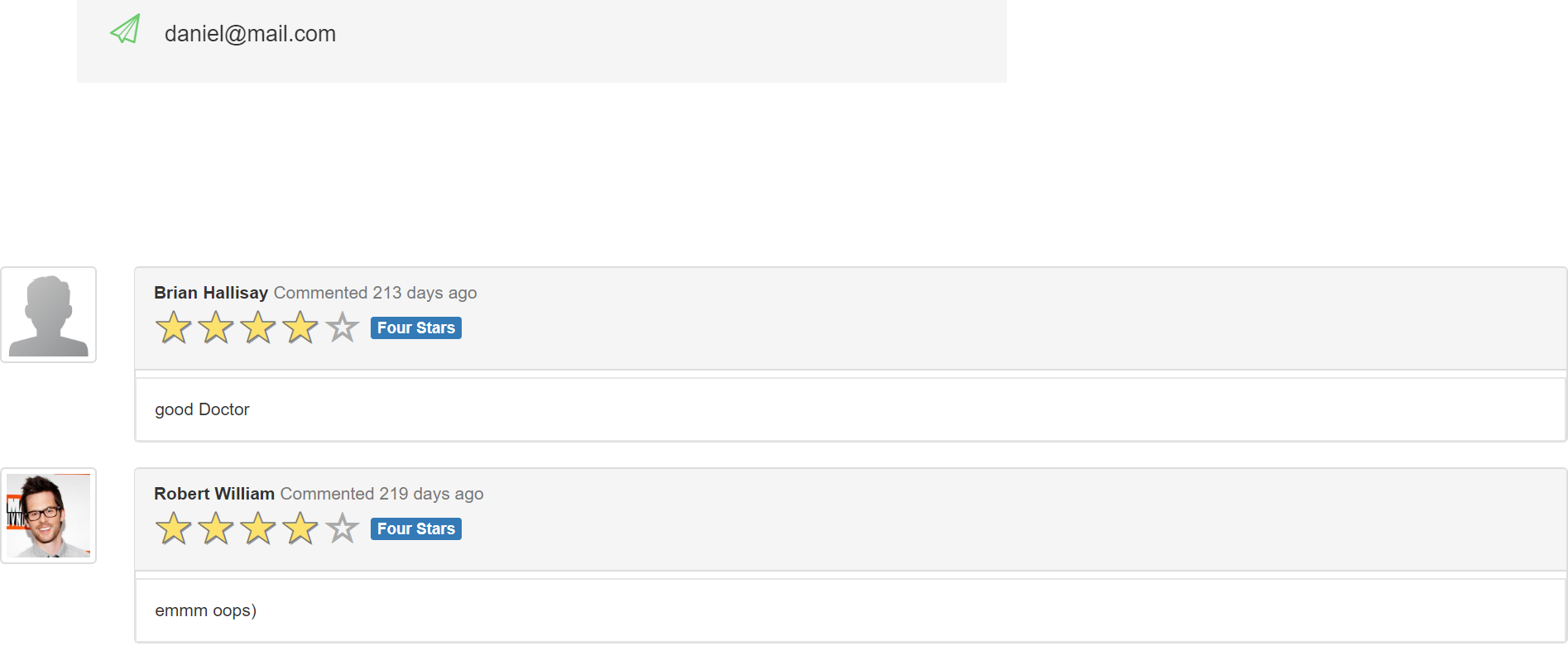
* Patient
* Doctor
* Unauthenticated user

Since here we

* Patient (can leave a comment with text and rating or only rating, also he can see other users rating)



* Doctor or unauthenticated user (can see comments)



***Conclusion***

During this laboratory work, we applied our theoretical knowledge about black box testing technique in practice. I chose to make some test cases for my app and to verify the output. My opinion about this technique is that we can apply it, when we use manual testing and we don’t have many dependencies between different functionalities.

If a lot of our functionalities will be dependent one upon another, then every time the developer will make a change, we should check one more time all our functionalities to verify if something doesn’t break in another place. This type of testing is time consuming.

If the system is big then we should sketch some boundary values and test them, instead of testing all the possible values. In order to make our test more structured, a good practice will be to make a decision table for each functionality, since in that way you will think about all the possible error and success cases, but There are chances of having unidentified paths during the testing process.