

Rating Playground

Integrative Software Engineering

Course 10141

Ran
Noam
Ron
Lior

Table of Contents

| | |
|--|-----------|
| Table of Contents | 2 |
| 1. Project Requirements Document | 3 |
| Introduction | 3 |
| Purpose of the System | 3 |
| Scope of the System | 3 |
| Actors and goals | 4 |
| Functional Requirements | 6 |
| Use Case Diagram | 6 |
| Use Case Details | 7 |
| Non-Functional Requirements | 14 |
| 2. Gherkin Tests | 15 |
| Feature: System initialization | 15 |
| Feature: Register user | 15 |
| Feature: Get specific user | 16 |
| Feature: Update an existing user | 18 |
| Feature: Verify new user | 19 |
| Feature: Create and store new element | 20 |
| Feature: Update an existing element | 22 |
| Feature: Get specific element | 24 |
| Feature: Get all elements | 26 |
| Feature: Get all elements in given radius | 27 |
| Feature: Search for elements by attribute value | 29 |
| Feature: Start an activity | 31 |
| 3. Appendix: Project progress report - last sprint | 37 |
| 4. Appendix: Project progress report - summary of all sprints | 42 |
| 5. Technologies | 53 |
| 6. Setup | 54 |
| Server | 54 |
| Android Client | 54 |
| React Client | 55 |

1. Project Requirements Document

Introduction

The projects theme is a playground. The playground has elements. Users can take actions on each element. There is a score system in the playground- when a user takes action they earn/lose points depending on the rules for each action.

Purpose of the System

The initial purpose is to rate things.

At first, movies and tv shows.

Also, another possible ratings is for lectures, music, books, etc.

Scope of the System

The scope of the system includes the elements native to the playground (created in it) and the actions taken by users in the playground.

Every action taken by the users outside of the playground is out of the projects scope.

Actors and goals

Player (**Primary**)- the players take actions on the elements in the playground.

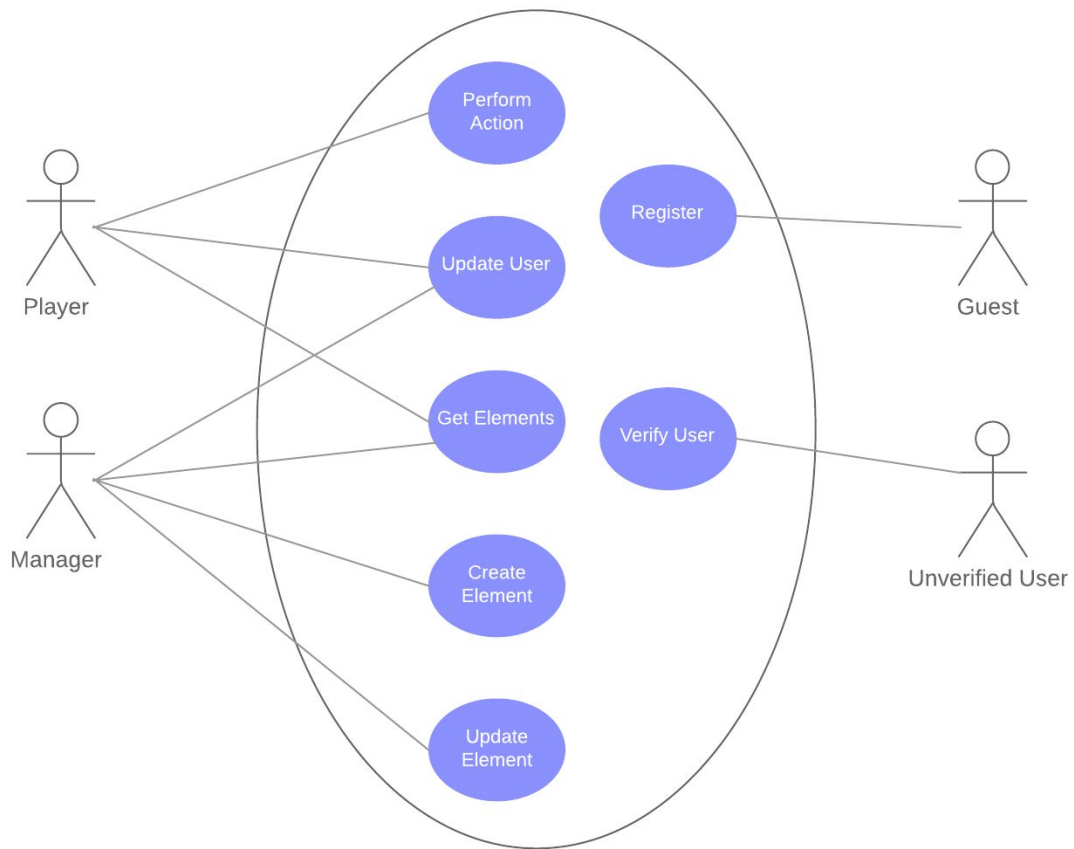
Depending on the action taken and the rules of the playground the players gain or lose points.

Manager (**Primary**)- the manager has his home playground which he can modify, add and remove elements. The manager can create rules in the playground and modify them. The manager does not have a score.

Guest (**Secondary**)- actor that just created an account in order to become a player/manager. They sign up to the playground and their account awaits for an approval by mail.

Functional Requirements

Use Case Diagram



Use Case Details

U.C. Name: Register

Goal: Register a user to the playground

Actors: Guest

Workflow:

1. The guest enters his email address, choses a role - player or manager, username and an avatar.
2. The system creates a waiting to confirmation user and sends an email for account verification.

Alternate flow A:

- 1.A. On step 1, the guest already has an account in a different playground.
- 2.A. The guest enters his home playgrounds id
- 3.A. The system gets the their information and registers them as a player in this playground.

Alternate flow B:

- 1.B. On step 2, the entered email address doesn't exist.
- 2.B. The system shows an error starts again from step 1.

U.C. Name: Verify User

Goal: Verify registration of a user to the playground

Actors: Unverified User

Workflow:

1. The unverified user enters his email address and verification code
2. The system verifies the confirmation code and change his state to verified

Alternate flow A:

- 1.A. On step 1, the unverified user enters wrong confirmation code.
- 2.A. The system returns an error message and the use case starts again from step 1.

U.C Name: Update user

Goal: update user in the playground

Actors: User

Workflow:

1. The user selects the option to update his profile.
2. The System shows a list of user attributes that can be changed.
3. The user modify the wanted attributes and send it to system.
4. The System updates the user details.

Alternate flow:

- 3.A. The user fill invalid details or not filling all details.
- 4.A. The system declined the changes.

U.C Name: Create element

Goal: Create element in the playground

Actors: Manager

Workflow:

1. The manager selects to create new element
2. The system shows new element form
3. The manger fills and sends the element details - location, rules, visibility and status(active/expired).
4. The system creates a new element and send it back to the

manager.

Alternate flow:

- 3.A. The manager fills invalid details or not filling all details.
- 4.A. The system declined the request

U.C Name: Update element

Goal: Update element in the playground

Actors: Manager

Workflow:

1. The manager selects the option to show all elements in the playground.
2. The System shows a list of the playground elements that can be modified.
3. The manager chooses the element he wishes to modify.
4. The System shows update element form.
5. The manager sets the details - location, rules, visibility and status (active/expired).
6. The system approves the new changes.

Alternate flow:

- 5.A. The manager fill invalid details or not filling all details.
- 6.A. The system declined the changes.

U.C Name: Get elements

Goal: Get element in the playground

Actors: User

Workflow:

1. The user selects the option to show all the elements in the playground.
2. The System shows a list of the playground elements.

Alternate flow A:

- 1.A. The user selects the option to retrieve an element by identifiers.
- 2.A The System shows the requested element.

Alternate flow B:

- 1.B. The user selects the option to retrieve an element by identifiers.
- 2.B The element isn't exists, the System returns an error message.

Alternate flow C:

- 1.C. The user selects the option to retrieve elements by distance.
- 2.C The System shows the requested list of elements.

Alternate flow D:

- 1.D. The user selects the option to retrieve elements by element attribute.
- 2.D The System shows the requested list of elements.

Alternate flow E:

- 1.E. The user selects the option to retrieve elements by element attribute.
- 2.E The selected attribute is not valid to be searched by or not exists, the System returns an error message.

Alternate flow F:

- 1.F. The user selects the option to retrieve a list of elements according to main flow or alternate flows C or D.
- 2.F There aren't existing elements that match user request, the system return an empty list to the user.

U.C. Name: Perform action

Goal: Perform action on an element

Actors: Player

Workflow:

- 1.The player chooses the desired element and choose an action to be performed.
2. The system performs the action and updates the player's score accordingly.

Alternate flow A:

- 1.A. The player's action is against the rules.
- 2.A. The system displays an error message and the action isn't performed.

Alternate flow B:

- 1.B. The player's action is to publish a message on a message board element.
- 2.B. The system creates and saves a new activity and returns it to the player.

Alternate flow C:

- 1.C. The player's action is to view all messages on a message board element.
- 2.C. The system creates and saves a new activity and returns it to the player.

Alternate flow D:

- 1.D. The player's action is to rate a rateable element (e.g. movie, book tv show...).
- 2.D. The system creates and saves a new activity, with calculation of average rating for that element, and returns it to the player.

Alternate flow E:

- 1.D. The player's action is to show a rating of a rateable element (e.g. movie, book, tv show...).
- 2.D. If the player already rated this element, the system will create and save a new activity, with calculation of that element average rating, retrieve the rating of the player and return it to the player.

Non-Functional Requirements

| Number | Requirement Description | Type |
|--------|--|------|
| 1 | Performing actions in the system are easy to do by all users that understand english. | U |
| 2 | To use the system utilities, you must send the identifiers of a user in every request (except from register and verify). | U |
| 3 | A user must verify his email with a verification code before he can use the system utilities | R |
| 4 | A user with role different than 'player' and 'manager' won't be able to use the system | R |
| 5 | Posting an activity will be performed in less than 1 sec | P |
| 6 | 2 Users or more can use the system without a significant loss of performance | P |
| 7 | New users would be able to register and verify themselves to the system in less than 2 minutes | P |
| 8 | The system sends an email with confirmation code at registration event | S |
| 9 | The system is written in Java, and can run on any computer with Java installed in version 1.8 and above | S |

2. Gherkin Tests

Feature: System initialization

Scenario: Test Server Is Booting Correctly - Succeeded

Given nothing

When The Server starts up

Then no error occurs

Feature: Register user

Scenario: test register user successfully - Succeeded

Given – the server is up

When – I **POST** /playground/users

With headers:

Accept: application/json

Content-Type: application/json

And request body:

```
{
  "email": "ran@gmail.com",
  "playground": "ratingplayground",
  "username": "ran",
  "code": any number [1000,9999]
}
```

And I **POST** /playground/users

With headers:

Accept: application/json

Content-Type: application/json

And request body:

```
{
  "email": "ran@gmail.com",
  "playground": "ratingplayground",
  "username": "ran",
  "code": generated code
}
```

Then - The response body contains user with name "ran" and the unique key: "ratingplayground@@ran@gmail.com".

the database retrieves for the requested user key the user:

```
{ "email": "ran@gmail.com", "playground": "ratingplayground", "username": "ran",
  "code": -1 }
```

Scenario: test register already existing user - Succeeded

Given – the server is up
and the database contains user with the
id:"ratingplayground@@ran@gmail.com"

When – I **POST** /playground/users

With headers:

Accept: application/json

Content-Type: application/json

And request body:

```
{  
  "email": "ran@gmail.com",  
  "playground": "ratingplayground",  
}
```

Then - The response status <>2xx

Feature: Get specific user

Scenario: test get verified specific user successfully - Succeeded

Given – the server is up

and database contains {

```
"ratingplayground@@ran@gmail.com" : {  
  "email": "ran@gmail.com", "playground": "ratingplayground",  
  "verificationCode": -1  
}}
```

When – I **GET** /playground/users/login/ratingplayground/ran@gmail.com

with Headers:

Accept: application/json

Content-Type: application/json

Then – the response is

```
{  
  "email": "ran@gmail.com",  
  "playground": "ratingplayground"  
}
```


Scenario: test get specific user with invalid email and playground parameters

- **Succeeded**

Given – the server is up
and database is empty

When – I **GET** /playground/users/login/ratingplayground/ran@gmail.com
with Headers:

Accept: application/json

Content-Type: application/json

Then – the response is
Status : <> 2xx

Scenario: test get specific user that not been verified yet - Succeeded

Given – the server is up
and database contains {
"ratingplayground@@ran@gmail.com" : {
"email": "ran@gmail.com", "playground": "ratingplayground",
"code": any number [1000,9999]
}}
}}

When – I **GET** /playground/users/login/ratingplayground/ran@gmail.com
with Headers:

Accept: application/json

Content-Type: application/json

Then – the response is
Status : <> 2xx

Feature: Update an existing user

Scenario: test update user successfully - Succeeded

Given – the server is up

and database contains contains the user:

```
{“ratingplayground@@ran@gmail.com”:  
  {“email”:“ran@gmail.com”,“playground”:“ratingplayground”,“username”:“ran”,“avatar  
”:“avatar” ,“role”:“player”,“code”: -1}}
```

When - I **PUT** /playground/users/ratingplayground/ran@gmail.com

With headers:

Accept: application/json

Content-Type: application/json

And request body:

```
{  
  “email”:“ran@gmail.com”,  
  “playground”: “ratingplayground”,  
  “username”:“changedUsername”,  
  “avatar”:“changedAvatar”,  
  “role”:“manager”  
}
```

Then - database retrieves for the unique id : “ratingplayground@@ran@gmail.com”
the user { “email”:“ran@gmail.com”,“ratingplayground”,
“username”:“changedUsername”, “avatar”:“changedAvatar” , “role”:“manager”}

Scenario: test update non existing user - Succeeded

Given – the server is up

and the database is empty

When - I **PUT** /playground/users/ratingplayground/nonexisting@gmail.com

With headers:

Accept: application/json

Content-Type: application/json

And request body:

```
{  
  “email”:“nonexisting@gmail.com”,  
  “playground”: “ratingplayground”,  
  “username”:“nonexisting”  
}
```

Then - the response is: Status 2xx

Feature: Verify new user

Scenario: Verify new user successfully - Succeeded

Given – the server is up

and database contains contains the user:

```
{“ratingplayground@@ran@gmail.com”:  
{"email":“ran@gmail.com”,“playground”:“ratingplayground”,  
“code”:any number [1000,9999]}
```

And after retrieving the {code} from the user “ratingplayground@@ran@gmail.com”

When – I **GET** /playground/users/confirm/ratingplayground/ran@gmail.com/{code}
with Headers:

Accept: application/json

Content-Type: application/json

Then – the response is

```
{  
  "email": "ran@gmail.com",  
  "playground": "ratingplayground",  
  "code": -1  
}
```

Scenario: Verify new user with wrong code - Succeeded

Given – the server is up

and database contains contains the user:

```
{“ratingplayground@@ran@gmail.com”:  
{"email":“ran@gmail.com”,“playground”:“ratingplayground” ,  
“code”:any number [1000,9999]}}
```

When – I **GET** /playground/users/confirm/ratingplayground/ran@gmail.com/-1
with Headers:

Accept: application/json

Content-Type: application/json

Then – the response is

Status : <> 2xx

Feature: Create and store new element

Scenario: test create element successfully - Succeeded

Given the server is up

And the elements table in database is empty

And the users table in database contains {

```
"playground": "ratingplayground",  
"email": "demo_manager@gmail.com",  
"role": "manager"  
}
```

When I POST /playground/elements/ratingplayground/demo_manager@gmail.com

With headers:

Accept: application/json

Content-Type: application/json

With Body {
 "name": "element",
 "playground": "ratingplayground",
 "location": {"x": 10, "y": 10},
}

Then

The response body contains an element with
id (generated from server, in our case "1")
playground "ratingplayground"
and name "element"

And the database retrieves for the id="1" and
playground "ratingplayground" the element

```
{  
  "id": "1",  
  "name": "element",  
  "playground": "ratingplayground",  
  "location": {"x": 10, "y": 10},  
}
```

Scenario: test create element with player role - Succeeded

Given the server is up

And the elements table in database is empty

And the users table in database contains {

```
"playground": "ratingplayground",  
"email": "demo_player@gmail.com",  
"role": "player"  
}
```

When I POST /playground/elements/ratingplayground/demo_player@gmail.com

With headers:

Accept: application/json

Content-Type: application/json

With Body {

```
"name": "element",  
"playground": "ratingplayground",  
"location": {"x": 10, "y": 10},  
}
```

Then

The response status <> 2xx

Scenario: test create element with id in body - Succeeded

Given the server is up

And the elements table in database is empty

And the users table in database contains {

```
"playground": "ratingplayground",  
"email": "demo_manager@gmail.com",  
"role": "manager"  
}
```

When I POST /playground/elements/ratingplayground/demo_manager@gmail.com

With headers:

Accept: application/json

Content-Type: application/json

With Body

```
{  
  "id": "element",  
  "playground": "ratingplayground",  
  "location": {"x": 10, "y": 10},  
}
```

Then The response status <> 2xx

Feature: Update an existing element

Scenario: test update element successfully - Succeeded

Given the server is up

And the elements table in database contains {
 "id": "1",
 "playground": "ratingplayground",
 "x": 10,
 "y": 10
}

And the users table in database contains {
 "playground": "ratingplayground",
 "email": "demo_manager@gmail.com",
 "role": "manager"
}

When | PUT

/playground/elements/ratingplayground/demo_manager@gmail.com/ratingplayground/test

With headers:

Content-Type: application/json

And request body is

```
{  
    "id": "1",  
    "playground": "ratingplayground",  
    "location": {"x": 5, "y": 5}  
}
```

Then -

Database retrieves for the id="test" and playground "ratingplayground" the element

```
{  
    "id": "test",  
    "playground": "ratingplayground",  
    "location": {"x": 5, "y": 5}  
}
```

Scenario: test update element with player role - Succeeded

Given the server is up

And the elements table in database contains {

```
"id": "1",
"playground": "ratingplayground",
"x": 10,
"y": 10
}
```

And the users table in database contains {

```
"playground": "ratingplayground",
"email": "demo_player@gmail.com",
"role": "player"
}
```

When I PUT

/playground/elements/ratingplayground/demo_player@gmail.com/ratingplayground/test

With headers:

Content-Type: application/json

And request body is

```
{
  "id": "1",
  "playground": "ratingplayground",
  "location": {"x": 5, "y": 5}
}
```

Then the response status is <> 2xx

Scenario: test update element with non existing element - Succeeded

Given the server is up

And the elements table in database is empty

When I PUT

/playground/elements/ratingplayground/demo_manager@gmail.com/ratingplayground/test

With headers:

Accept: application/json

Content-Type: application/json

With Body

```
{
  "id": "test",
  "playground": "ratingplayground",
  "location": {"x": 5, "y": 5}
}
```

Then the response status is <> 2xx

Feature: Get specific element

Scenario: test get specific element with manager role successfully -

Succeeded

Given the server is up

And the elements table in database contains {

```
"playground": "ratingplayground",  
"d": "1",  
"name": "demo"  
}
```

And the users table in database contains {

```
"playground": "ratingplayground",  
"email": "demo_manager@gmail.com",  
"role": "manager"  
}
```

When I GET

/playground/elements/ratingplayground/demo_manager@gmail.com/ratingplayground/demo

With headers:

Accept: application/json

Content-Type: application/json

Then the response is: {

```
"playground": "ratingplayground",  
"id": "1",  
"name": "demo"  
}
```


Scenario: test get specific expired element with player role - Succeeded

Given the server is up

And the elements table in database contains {
 "playground": "ratingplayground",
 "d": "1",
 "name": "demo",
 "expirationDate": some expired date
}

And the users table in database contains {
 "playground": "ratingplayground",
 "email": "demo_player@gmail.com",
 "role": "player"
}

When I GET

/playground/elements/ratingplayground/demo_player@gmail.com/ratingplayground/demo

With headers:

Accept: application/json

Content-Type: application/json

Then the response status <> 2xx

Scenario: test get specific message with invalid name - Succeeded

Given the server is up

And elements table in database is empty

When I GET

/playground/elements/ratingplayground/demo@gmail.com/ratingplayground/demo

With headers:

Accept: application/json

Content-Type: application/json

Then the response status <> 2xx

Feature: Get all elements

Scenario: test get all elements successfully with no pagination parameters - Succeeded

Given the server is up

And the elements table in database contains [
 {"name": "name"}, {"name": "demo2"}, {"name": "demo3"}
]

And the users table in database contains {
 "playground": "ratingplayground",
 "email": "demo_manager@gmail.com",
 "role": "manager"
}

When I GET

/playground/elements/ratingplayground/demo_manager@gmail.com/all

With headers:

Accept: application/json

Content-Type: application/json

Then the response is:

[
 {"name": "demo1"}, {"name": "demo2"}, {"name": "demo3"}
]

Scenario: test get all elements with fault pagination parameters - Succeeded

Given the server is up

And the elements table in database is empty

And the users table in database contains {
 "playground": "ratingplayground",
 "email": "demo_manager@gmail.com",
 "role": "manager"
}

When I GET

/playground/elements/ratingplayground/demo_manager@gmail.com/all?page=-1

With headers:

Accept: application/json

Content-Type: application/json

Then the response is: status <> 2xx

Feature: Get all elements in given radius

Scenario: test get all elements in given radius successfully with no pagination parameters - Succeeded

Given the server is up

And the elements table in database contains [

 {"x":0, "y":0},

 {"x":5, "y":5},

 {"x":2, "y":2}

]

And the users table in database contains {

 "playground": "ratingplayground",

 "email": "demo_manager@gmail.com",

 "role": "manager"

}

When I GET

/playground/elements/ratingplayground/demo_manager@gmail.com/near/1/1/3

With headers:

Accept: application/json

Content-Type: application/json

Then the response is:

[

 {"location": {"x":0, "y":0}},

 {"location": {"x":2, "y":2}}

]

Scenario: test get all elements in given radius with fault pagination parameters

- Succeeded

Given the server is up

And the users table in database contains {

```
"playground": "ratingplayground",  
"email": "demo_manager@gmail.com",  
"role": "manager"  
}
```

When I GET

/playground/elements/ratingplayground/demo_manager@gmail.com/near/1/1/3?page=-1

With headers:

Accept: application/json

Content-Type: application/json

Then the response is: status <> 2xx

Scenario: test get all elements in given radius with string instead double parameters - Succeeded

Given the server is up

And the users table in database contains {

```
"playground": "ratingplayground",  
"email": "demo_manager@gmail.com",  
"role": "manager"  
}
```

When I GET

/playground/elements/ratingplayground/demo_manager@gmail.com/near/1/test/3

With headers:

Accept: application/json

Content-Type: application/json

Then the response is: status <> 2xx

Feature: Search for elements by attribute value

Scenario: test get all searched elements successfully with no pagination parameters - **Succeeded**

Given the server is up

And the elements table in database contains [

 {"name": "movie1"},

 {"name": "movie2"},

 {"name": "movie3"}]

And the users table in database contains {

 "playground": "ratingplayground",

 "email": "demo_manager@gmail.com",

 "role": "manager"

}

When I GET

/playground/elements/ratingplayground/demo_manager@gmail.com/search/name/movie2

With headers:

Accept: application/json

Content-Type: application/json

Then the response is:

[

 {"name": "movie2"}

]

Scenario: test get all searched elements with fault pagination parameters -

Succeeded

Given the server is up

And the users table in database contains {

```
"playground": "ratingplayground",  
"email": "demo_manager@gmail.com",  
"role": "manager"  
}
```

When I GET

/playground/elements/ratingplayground/demo_manager@gmail.com/search/name/movie2?page=-1

With headers:

Accept: application/json

Content-Type: application/json

Then the response is: status <> 2xx

Scenario: test get all searched elements with fault attribute parameter -

Succeeded

Given the server is up

And the users table in database contains {

```
"playground": "ratingplayground",  
"email": "demo_manager@gmail.com",  
"role": "manager"  
}
```

When I GET

/playground/elements/ratingplayground/demo_manager@gmail.com/search/id/test

With headers:

Accept: application/json

Content-Type: application/json

Then the response is: status <> 2xx

Feature: Start an activity

Scenario: Post Message Successfully - Succeeded

Given the server is up

When I POST /playground/activities/userPlayground1/email@gmail.com
with:

```
{
  "playground": "playground",
  "id": "id",
  "elementPlayground": "ePlayground",
  "elementId": "eld",
  "type": "type",
  "playerPlayground": "userPlayground1",
  "playerEmail": "email@gmail.com"
  "attributes": {"message": "Hello"}
}
```

With headers:

```
Accept: application/json
Content-Type: application/json
```

Then the response is:

```
Status: 200
and the response body is:
{
  "playground": "playground",
  "id": "id",
  "elementPlayground": "ePlayground",
  "elementId": "eld",
  "type": "type",
  "playerPlayground": "userPlayground1",
  "playerEmail": "email@gmail.com"
  "attributes": {
    "message": "Hello"
  }
}
```

Scenario: invalid activity type - Succeeded

Given the server is up

When I POST /playground/activities/userPlayground1/email@gmail.com

with:

```
{  
  "playground": "playground",  
  "id": "id",  
  "elementPlayground": "ePlayground",  
  "elementId": "eld",  
  "type": "no_type",  
  "playerPlayground": "userPlayground1",  
  "playerEmail": "email@gmail.com"  
}
```

With headers:

Accept: application/json

Content-Type: application/json

Then the response is:

Status: <> 2xx

Scenario: Rate Movie Successfully - Succeeded

Given the server is up

When I POST /playground/activities/userPlayground1/email@gmail.com

with:

```
{
  "playground": "playground",
  "id": null,
  "elementPlayground": "playground",
  "elementId": "2",
  "type": "Rating",
  "playerPlayground": "playground",
  "playerEmail": "playerEmail@gmail.com",
  "attributes": {"rating": 7.33}
}
```

With headers:

Accept:application/json

Content-Type: application/json

Then the response is:

Status: 200

and the response body is:

```
"playground": "playground",
"id": "5",
"elementPlayground": "playground",
"elementId": "2",
"type": "Rating",
"playerPlayground": "playground",
"playerEmail": "playerEmail@gmail.com",
"attributes": {
  "rating": 7.33,
  "content": {
    "Points": 10,
    "Rating": 7,
    "Average": 7
  }
}
```

Scenario: String rating instead of float - Succeeded

Given the server is up

When I POST /playground/activities/userPlayground1/email@gmail.com

with:

```
{
  "playground": "playground",
  "id": "id",
  "elementPlayground": "ePlayground",
  "elementId": "eld",
  "type": "no_type",
  "playerPlayground": "userPlayground1",
  "playerEmail": "email@gmail.com"
}
```

With headers:

Accept: application/json

Content-Type: application/json

Then the response is:

Status: 500

Scenario: Rate the same movie twice - Succeeded

Given the server is up

and the user with email email@gmail.com has already rated the element with id "eld"

When I POST /playground/activities/userPlayground1/email@gmail.com

with:

```
{
  "playground": "playground",
  "id": "id",
  "elementPlayground": "ePlayground",
  "elementId": "eld",
  "type": "Rating",
  "playerPlayground": "userPlayground1",
  "playerEmail": "email@gmail.com"
}
```

With headers:

Accept: application/json

Content-Type: application/json

Then the response is:

Status: 500

Scenario: Read Messages Successfully with pagination - Succeeded

Given the server is up and 1 message already posted

When I POST /playground/activities/userPlayground1/email@gmail.com

with:

```
{
  "playground": "playground",
  "elementPlayground": "playground",
  "elementId": "1",
  "type": "ReadMessages",
  "playerPlayground": "playground",
  "playerEmail": "playerEmail@gmail.com",
  "attributes": {"page": 0, "size": 5}
}
```

With headers:

Accept:application/json

Content-Type: application/json

Then the response is:

Status: 200

and the response body is:

```
{ "playground": "playground",
  "id": "5",
  "elementPlayground": "playground",
  "elementId": "1",
  "type": "ReadMessages",
  "playerPlayground": "playground",
  "playerEmail": "playerEmail@gmail.com",
  "attributes": {
    "page": 0,
    "size": 5,
    "messages": [
      {
        "playground": "playground",
        "id": "4",
        "elementPlayground": "playground",
        "elementId": "1",
        "type": "PostMessage",
        "playerPlayground": "playground",
        "playerEmail": "playerEmail@gmail.com",
        "attributes": {
          "message": "Hello Integrative World!"
        }
      }
    ]
  }
}
```

Scenario: Read Messages Successfully with pagination - Succeeded

Given the server is up and no messages posted

When I POST /playground/activities/userPlayground1/email@gmail.com

with:

```
{
  "playground": "playground",
  "elementPlayground": "playground",
  "elementId": "1",
  "type": "ReadMessages",
  "playerPlayground": "playground",
  "playerEmail": "playerEmail@gmail.com",
  "attributes": {"page": 0, "size": 5}
}
```

With headers:

Accept:application/json

Content-Type: application/json

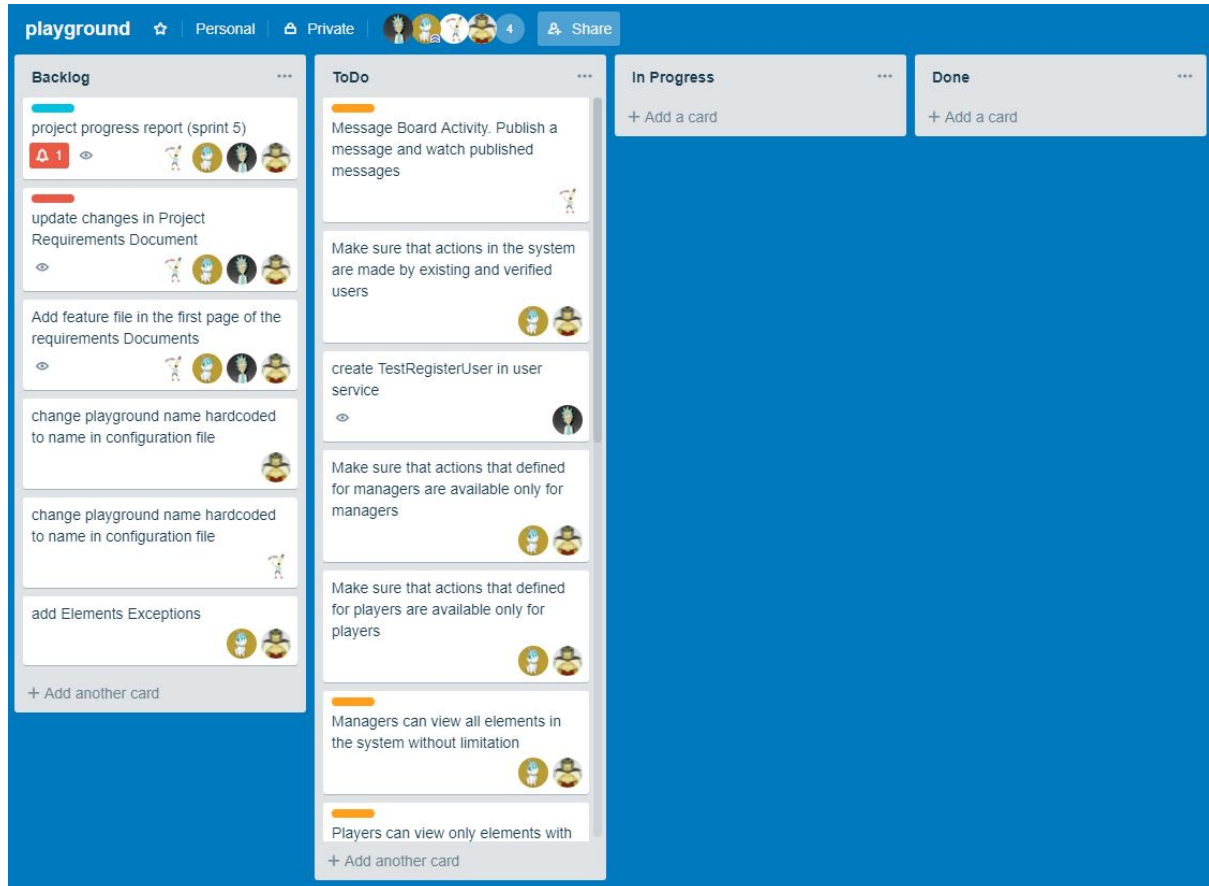
Then the response is:

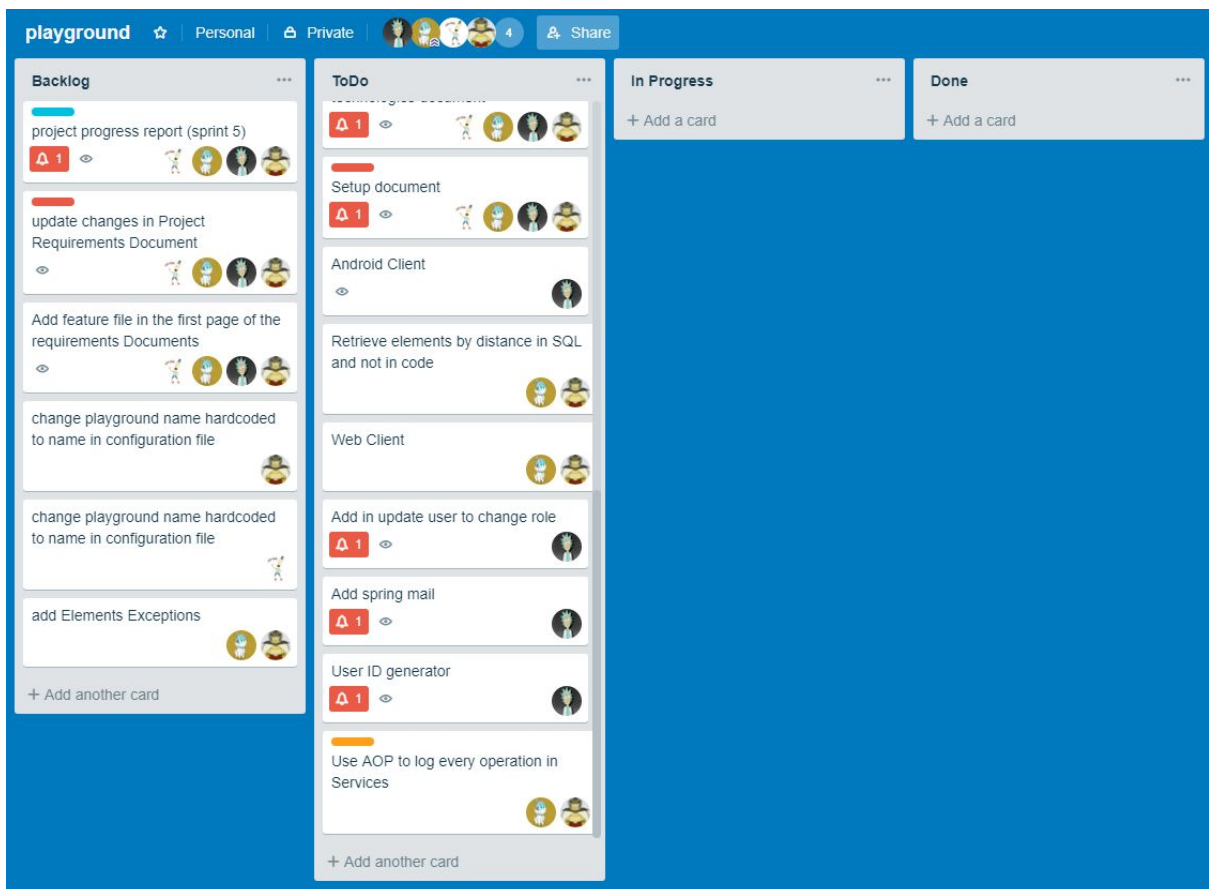
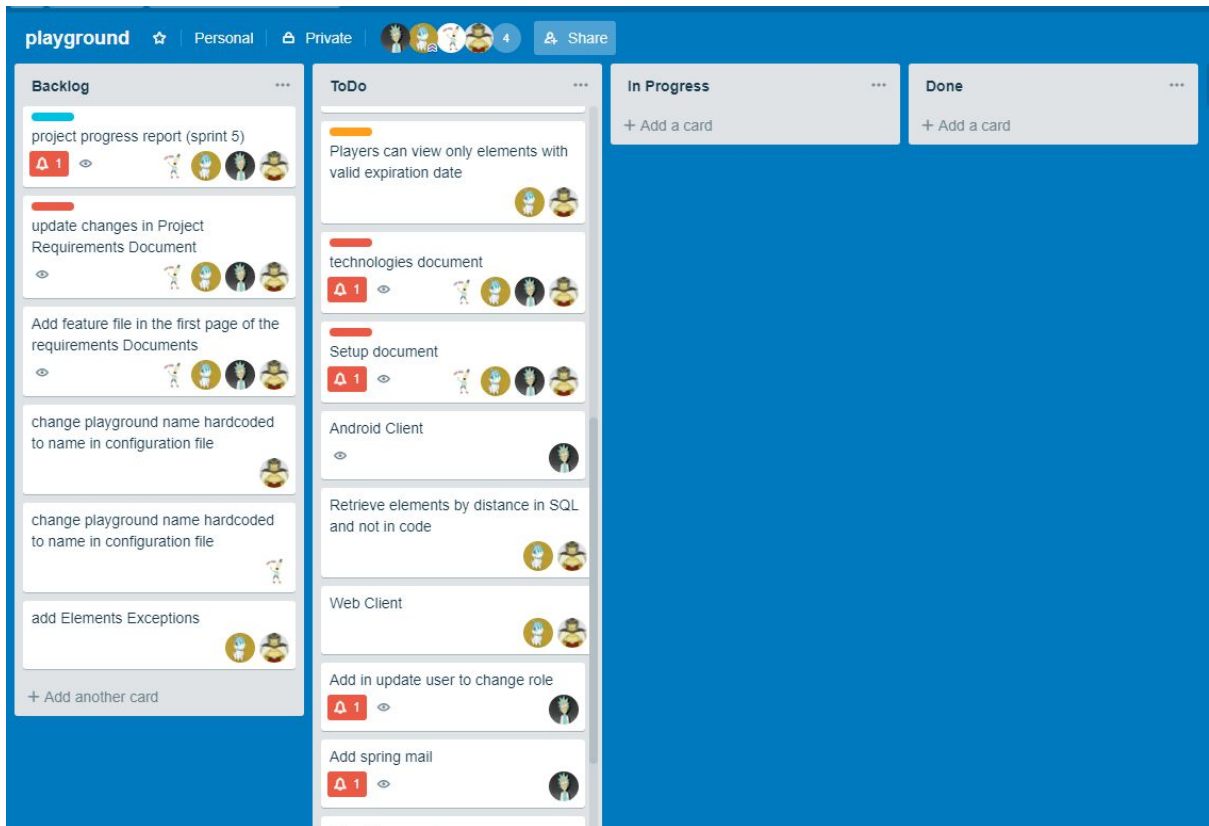
Status: 200

```
{
  "playground": "playground",
  "id": "3",
  "elementPlayground": "playground",
  "elementId": "1",
  "type": "ReadMessages",
  "playerPlayground": "playground",
  "playerEmail": "playerEmail@gmail.com",
  "attributes": {
    "page": 0,
    "size": 5,
    "messages": []
  }
}
```

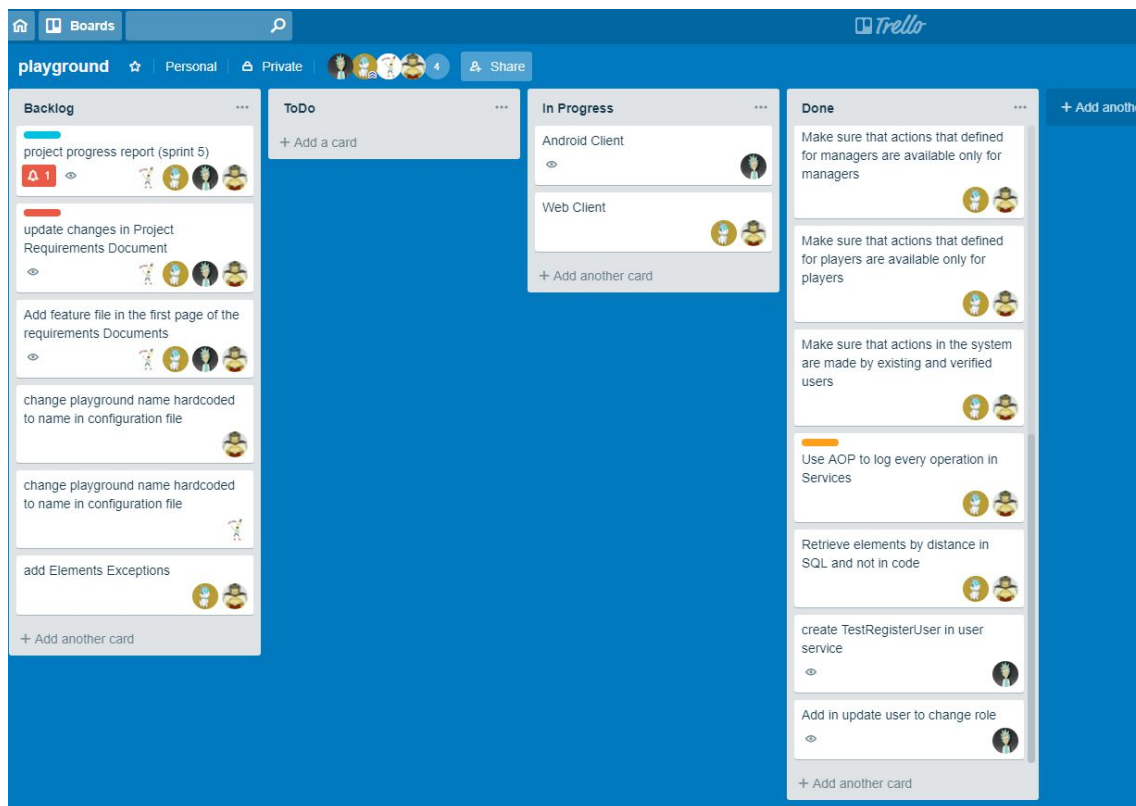
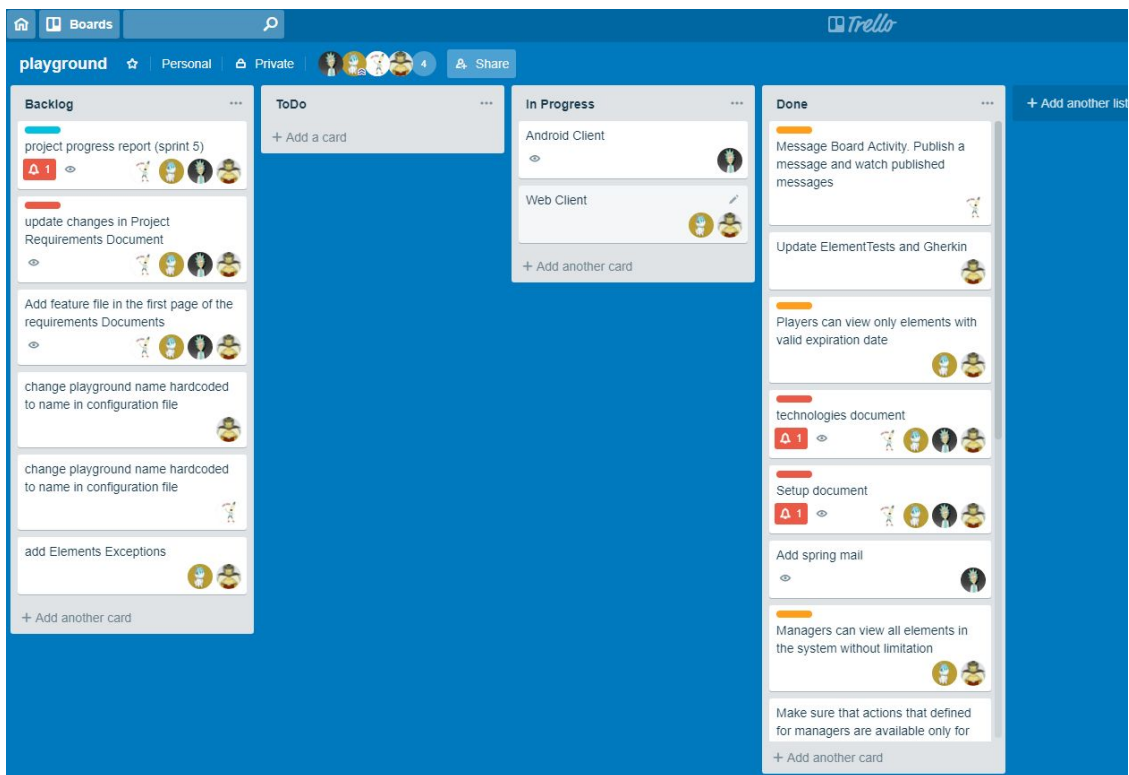
3. Appendix: Project progress report - last sprint

- Kanban Board of project from sprint 5 initiation (December 16th)







Kanban Board of project at the end of sprint 5 (December 30th)




- List of Students




Lior Gal, 208969923, Product Owner, DB



Ron Ginat, 209035450, DevOps, QA



Noam Tayri, 305435372, Scrum Master



Ran Weiner, 203701289, Team Leader

- General summary of work

everyone helped each other when they ask for it and we achieved the results we wanted.

- What should be improved in team work

all the team members must stay in touch with each other more often and not only in the end.

sharing problems that each member is facing would be much more efficient and less time consuming than facing the problem alone. we spent a lot of time by doing that.

- What problems did the team encountered through the sprint

We did not write as much tests as we would have wanted to write.

we had some difficulties in understanding the code that each member write and it was not easy to integrate with it.

- Why did we not complete all planned work

We finished all planned work

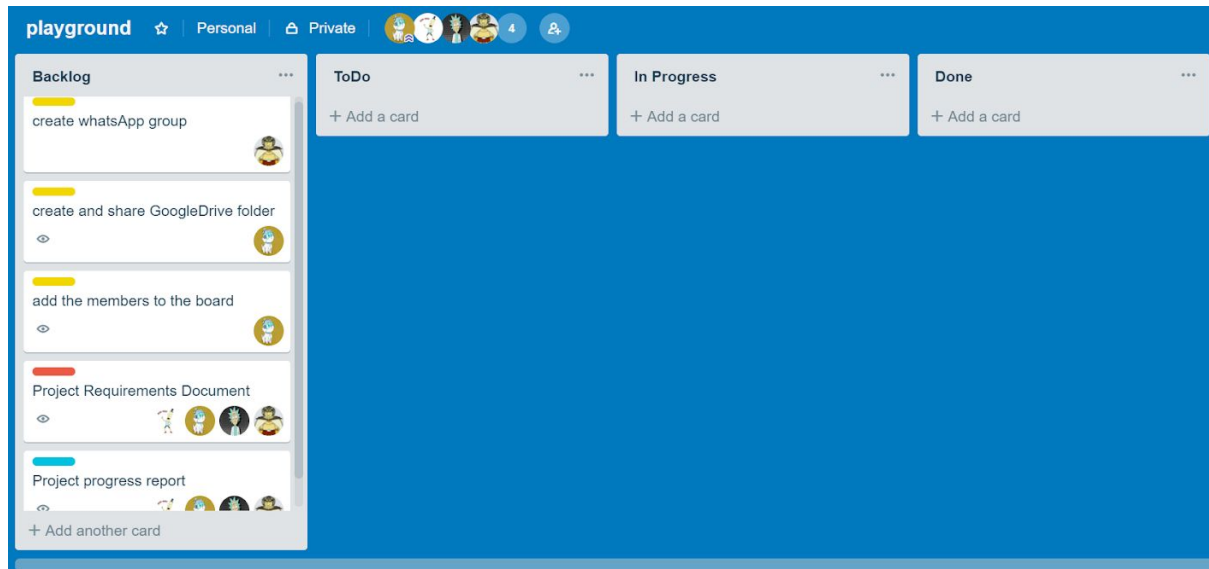
- What is expected for the next sprint

there is no sprint scheduled

4. Appendix: Project progress report - summary of all sprints

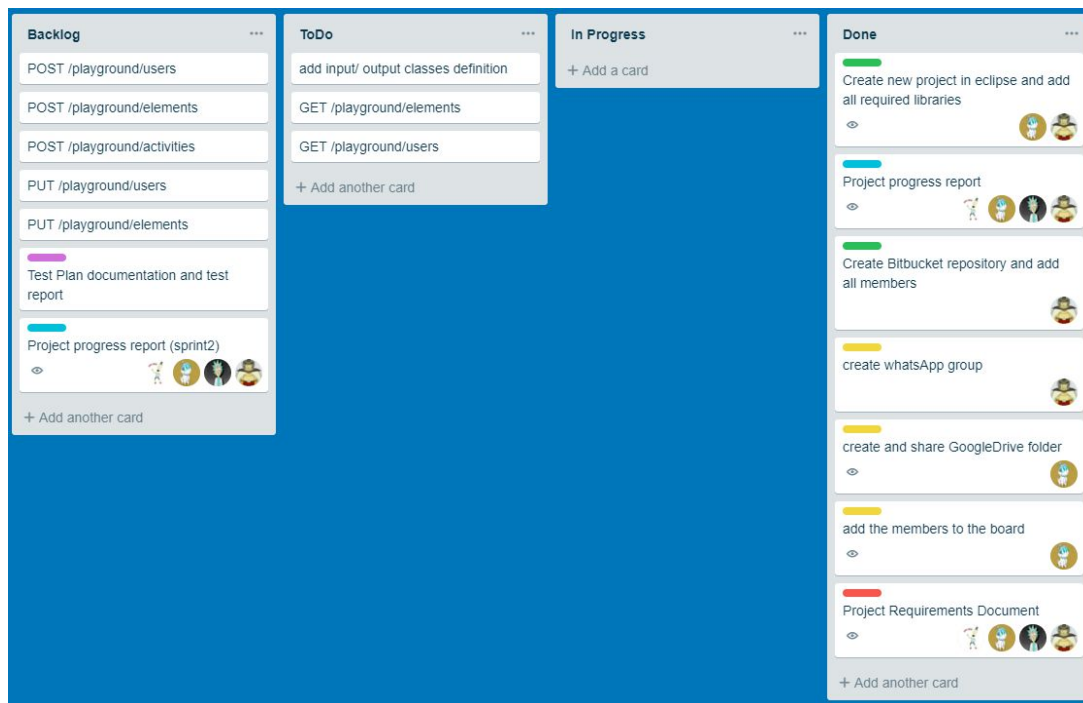
4.1

- Kanban Board of project from sprint 2 initiation (October 14th)

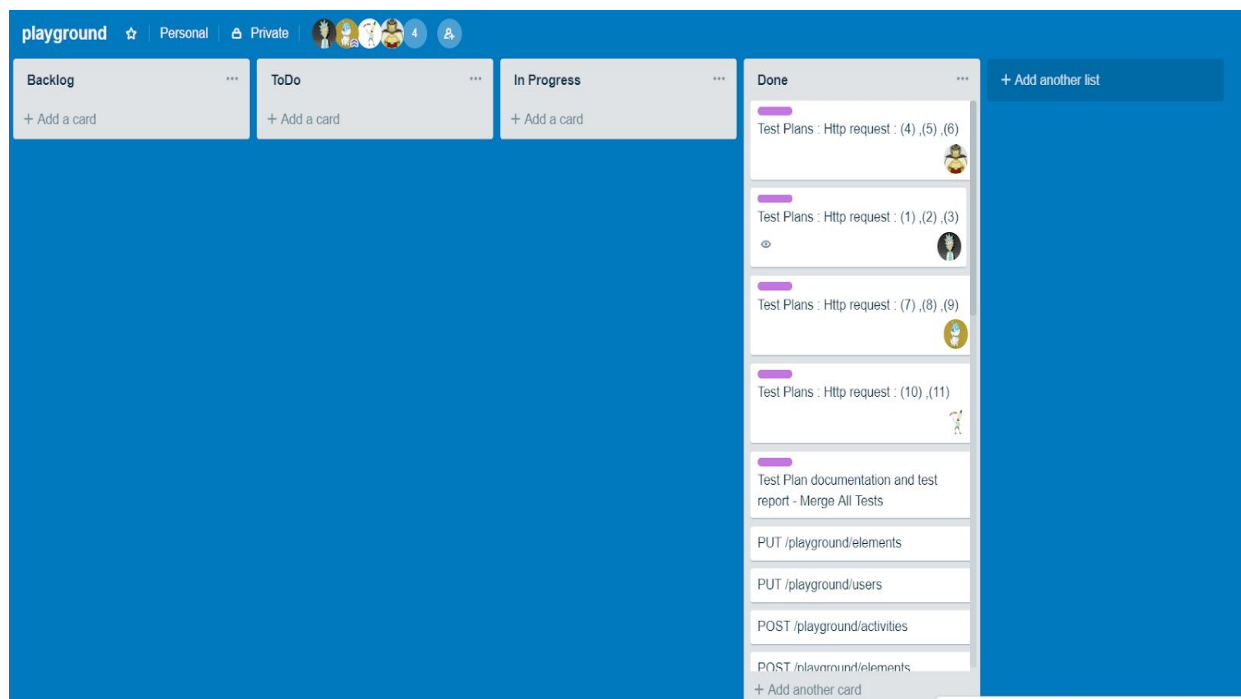


4.2

- Kanban Board of project from sprint 2 initiation (October 28th)

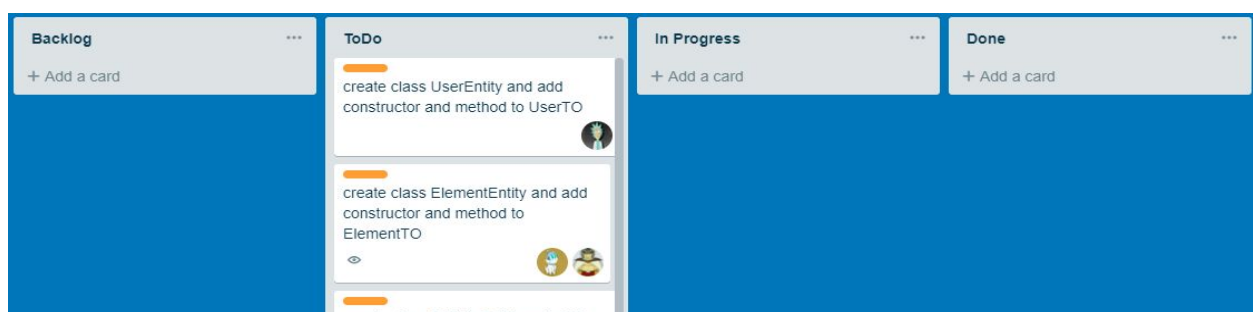


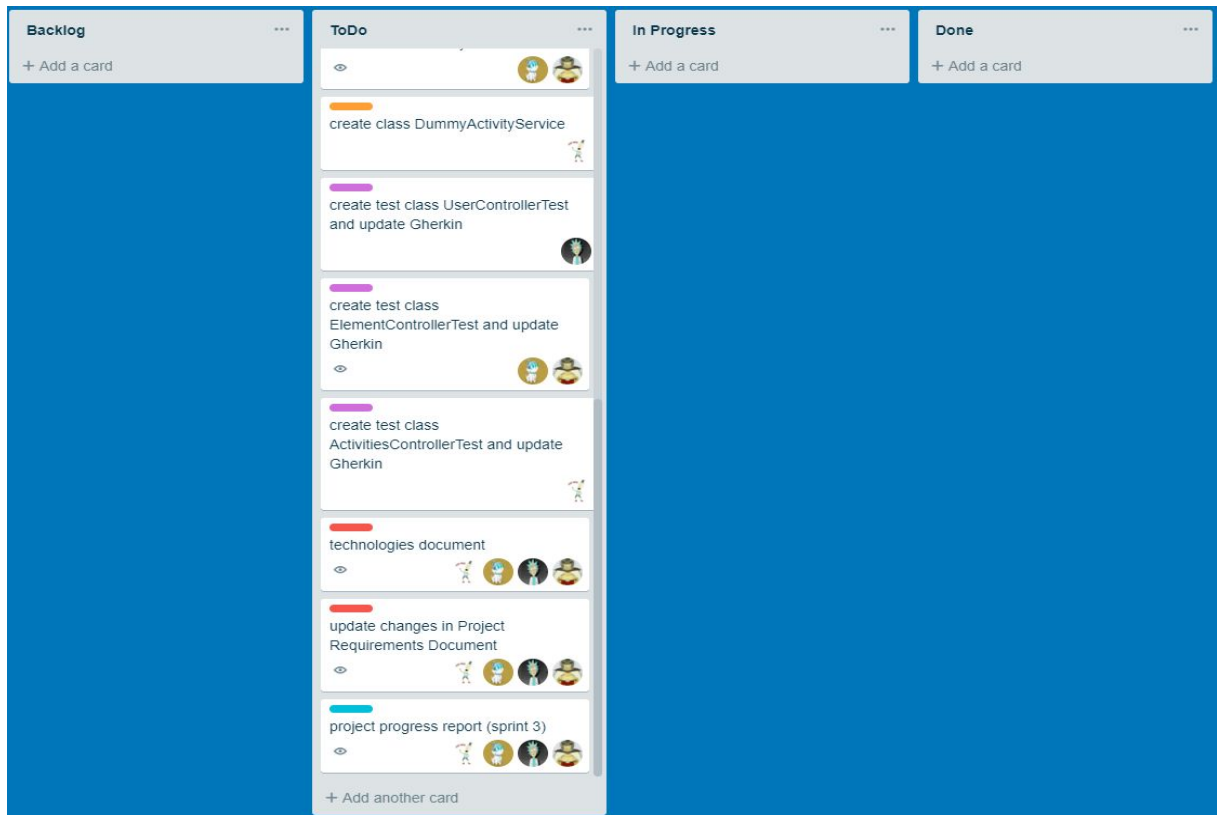
- Kanban Board of project at the end of sprint 2 (November 11th)



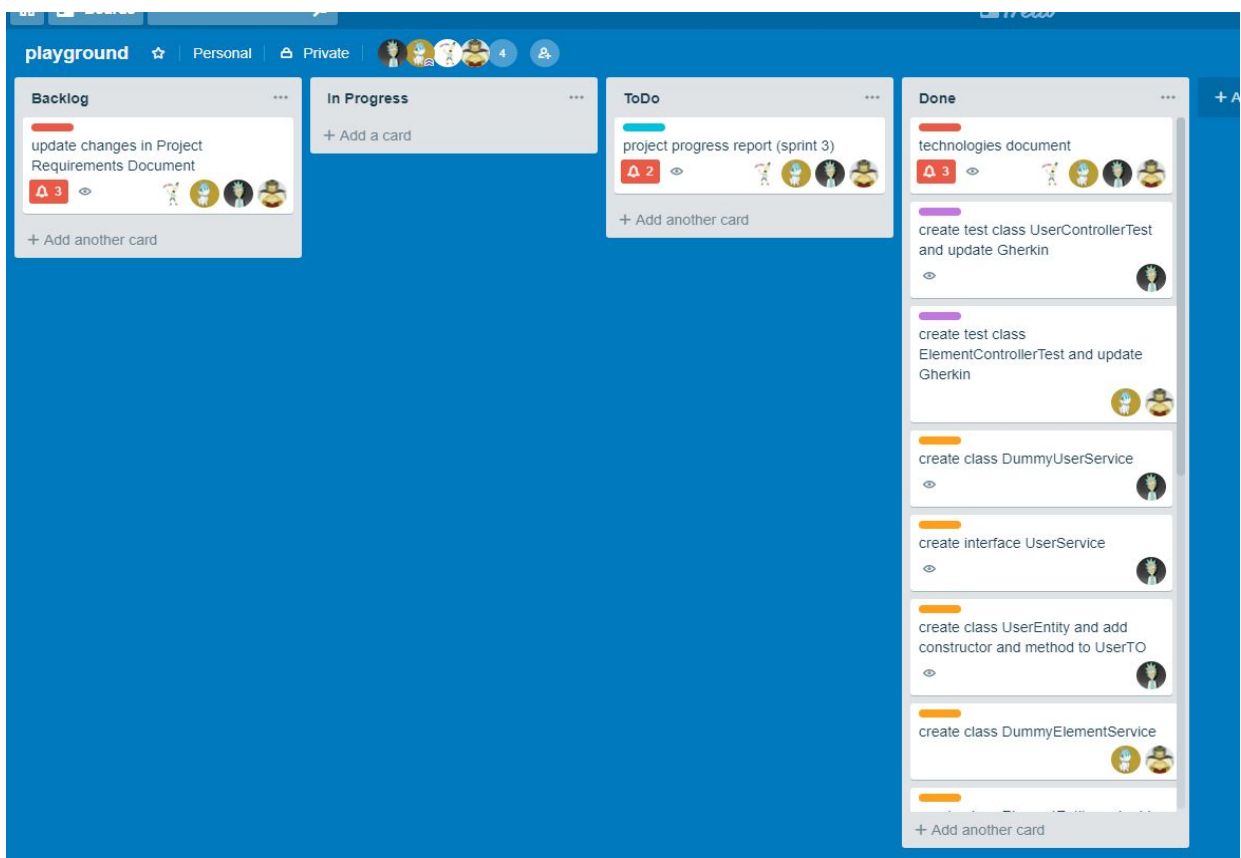
4.3

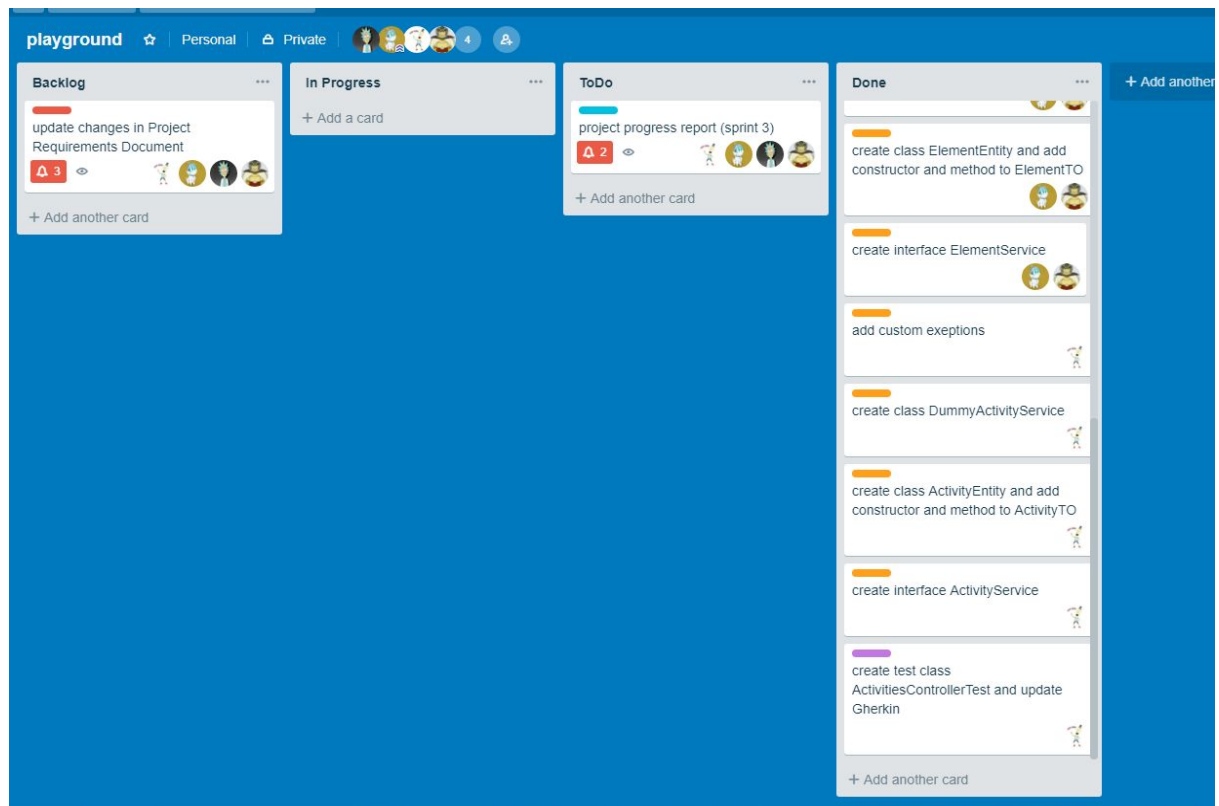
- Kanban Board of project from sprint 3 initiation (November 11th)





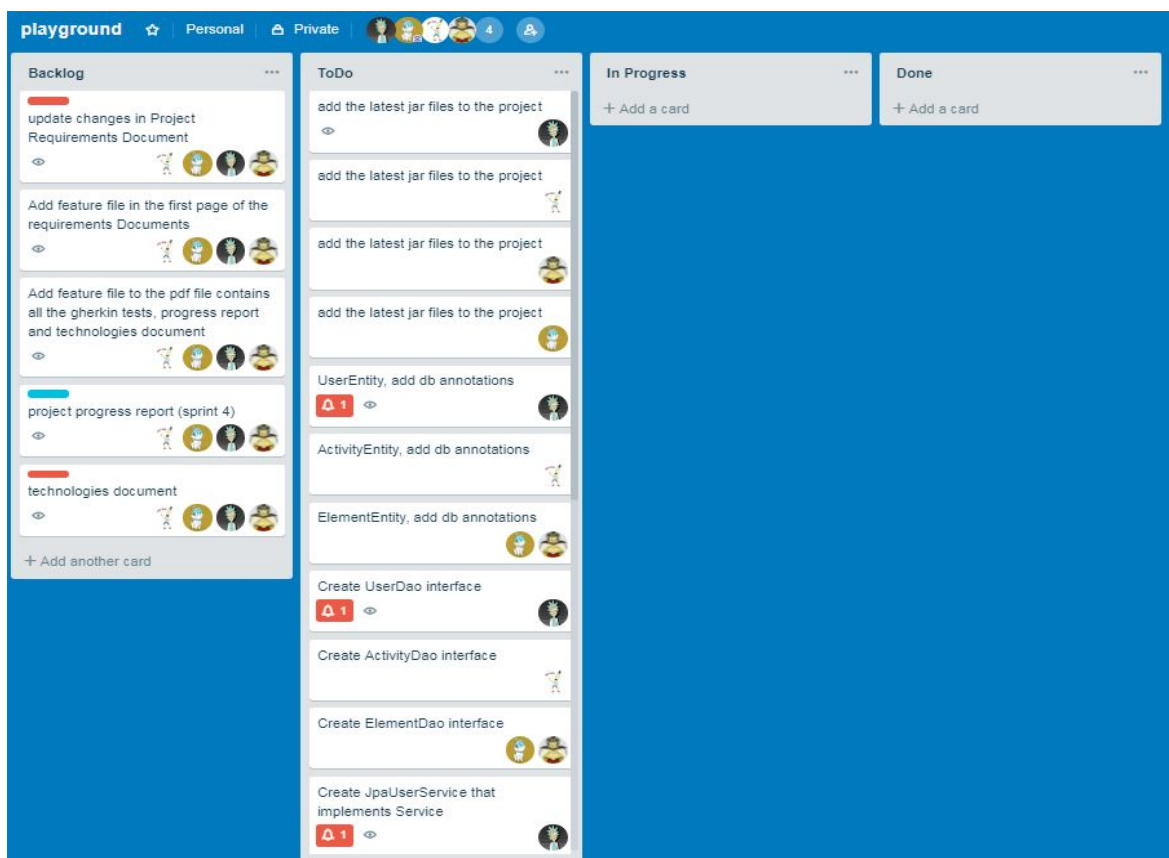
- Kanban Board of project at the end of sprint 3 (November 25th)

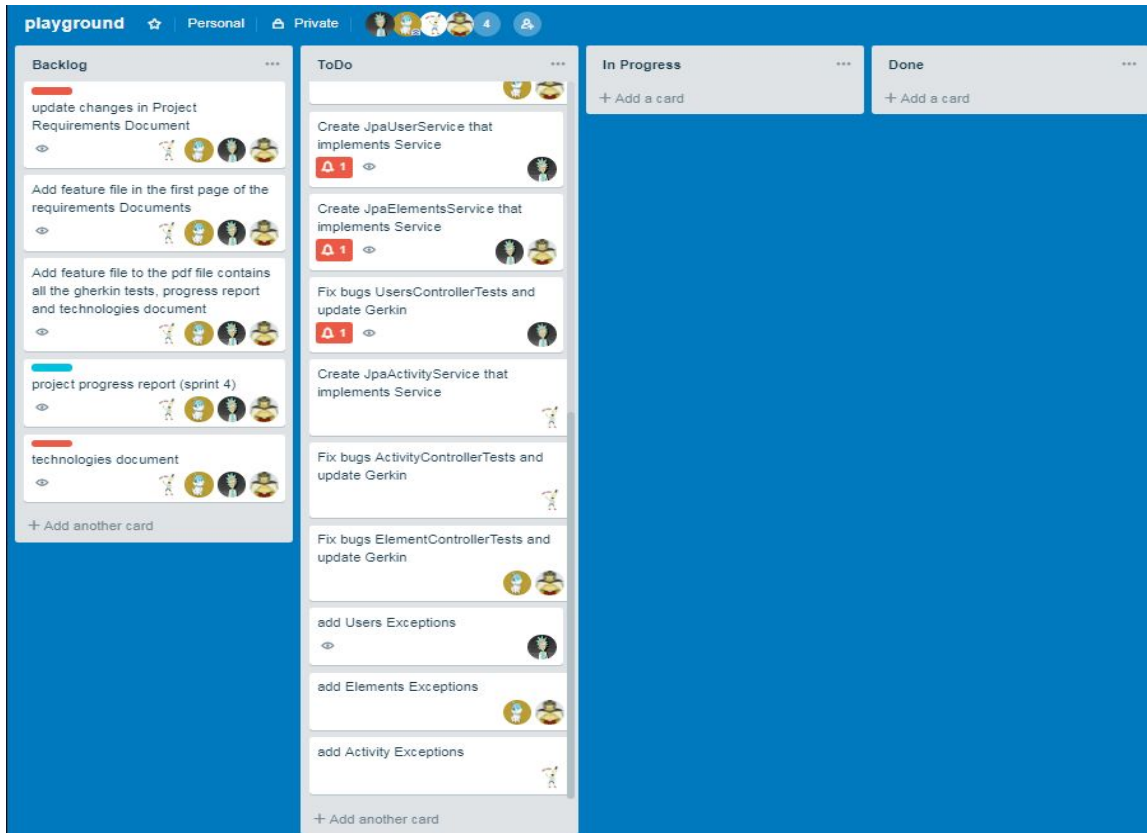




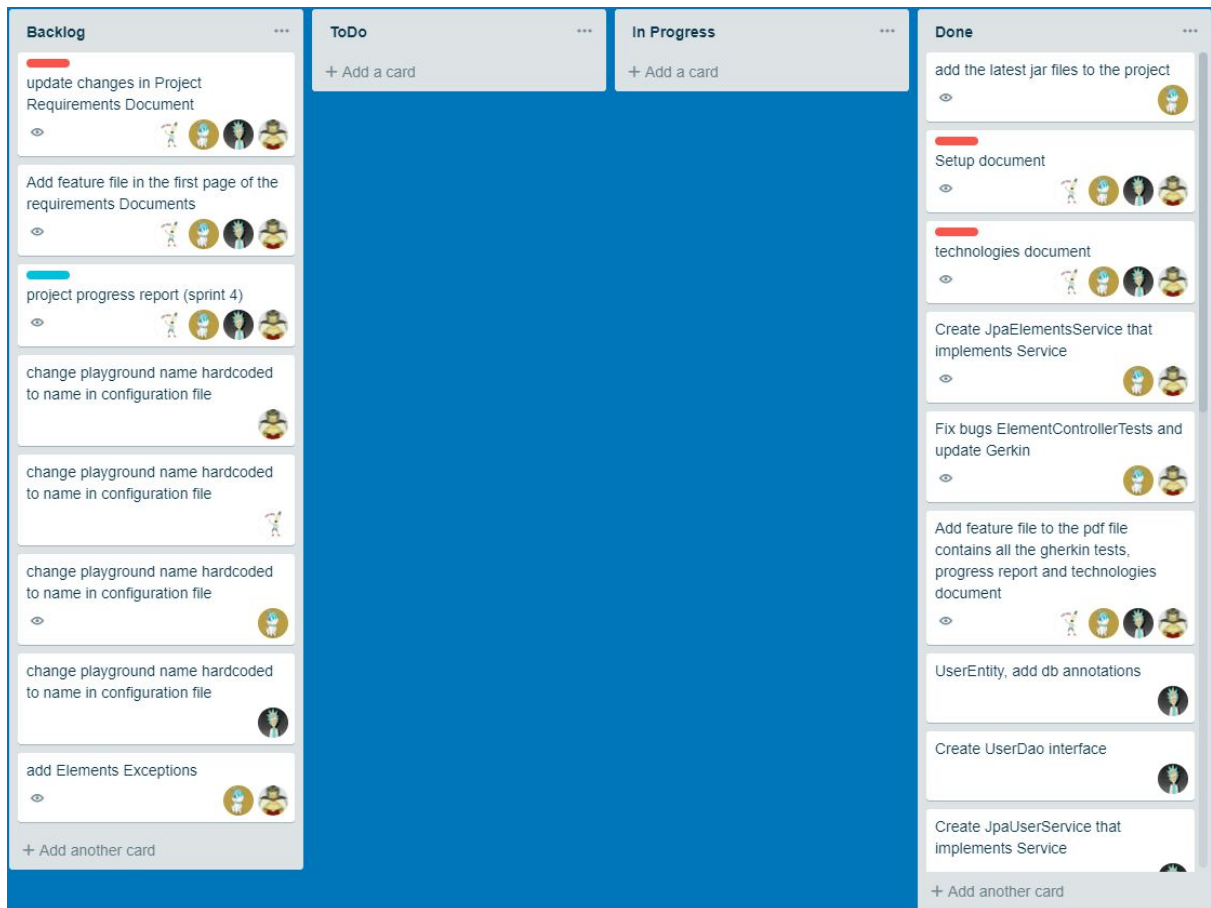
4.4

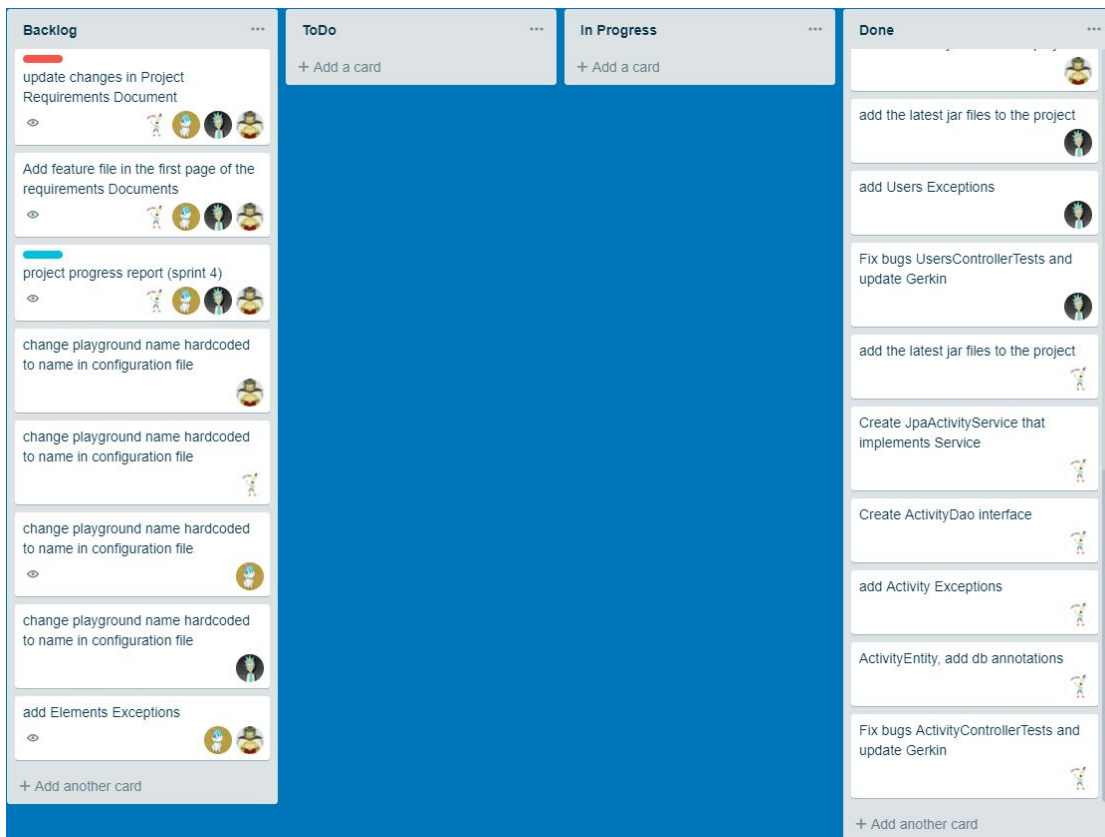
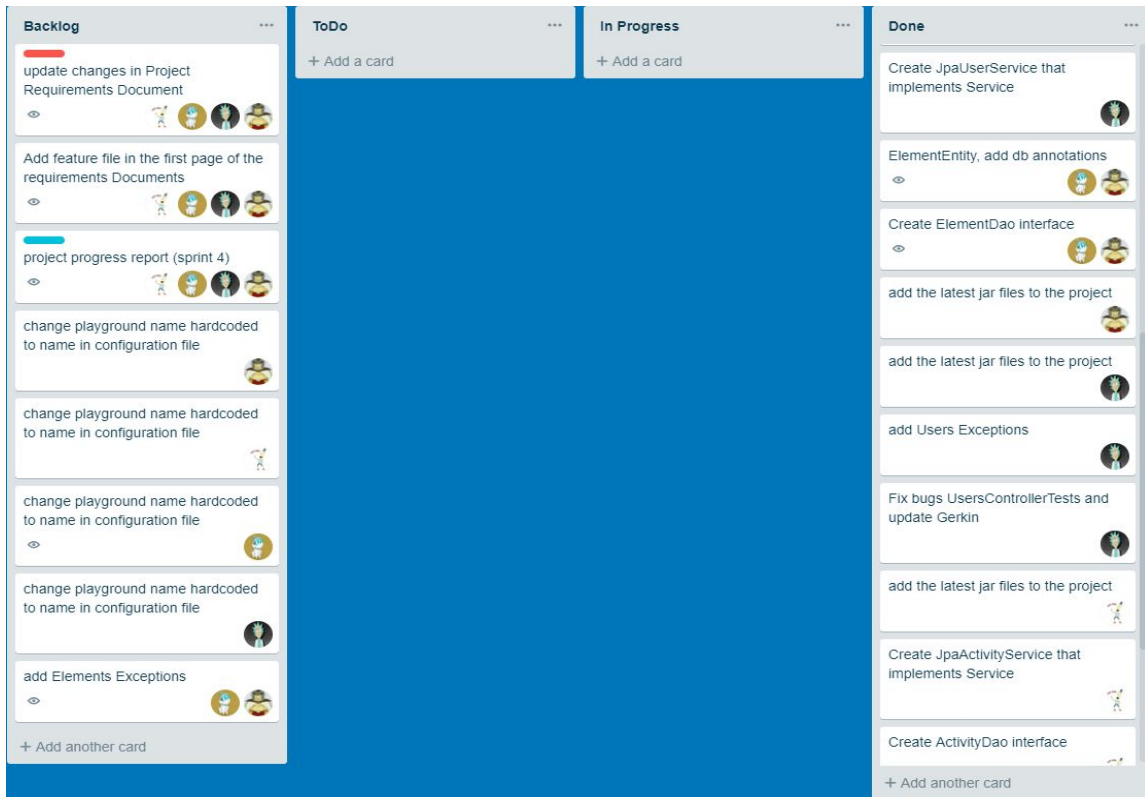
- Kanban Board of project from sprint 4 initiation (November 25th)





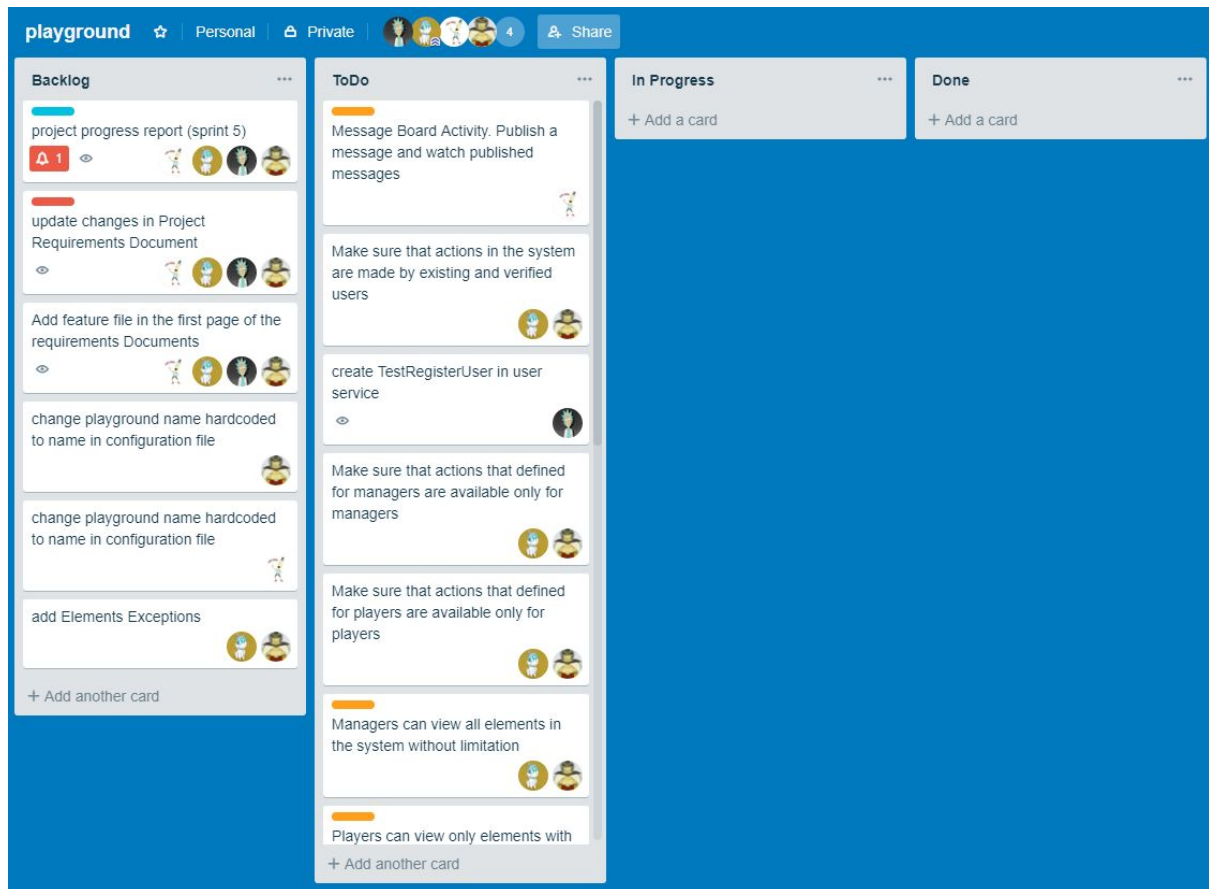
- Kanban Board of project at the end of sprint 4 (December 9th)

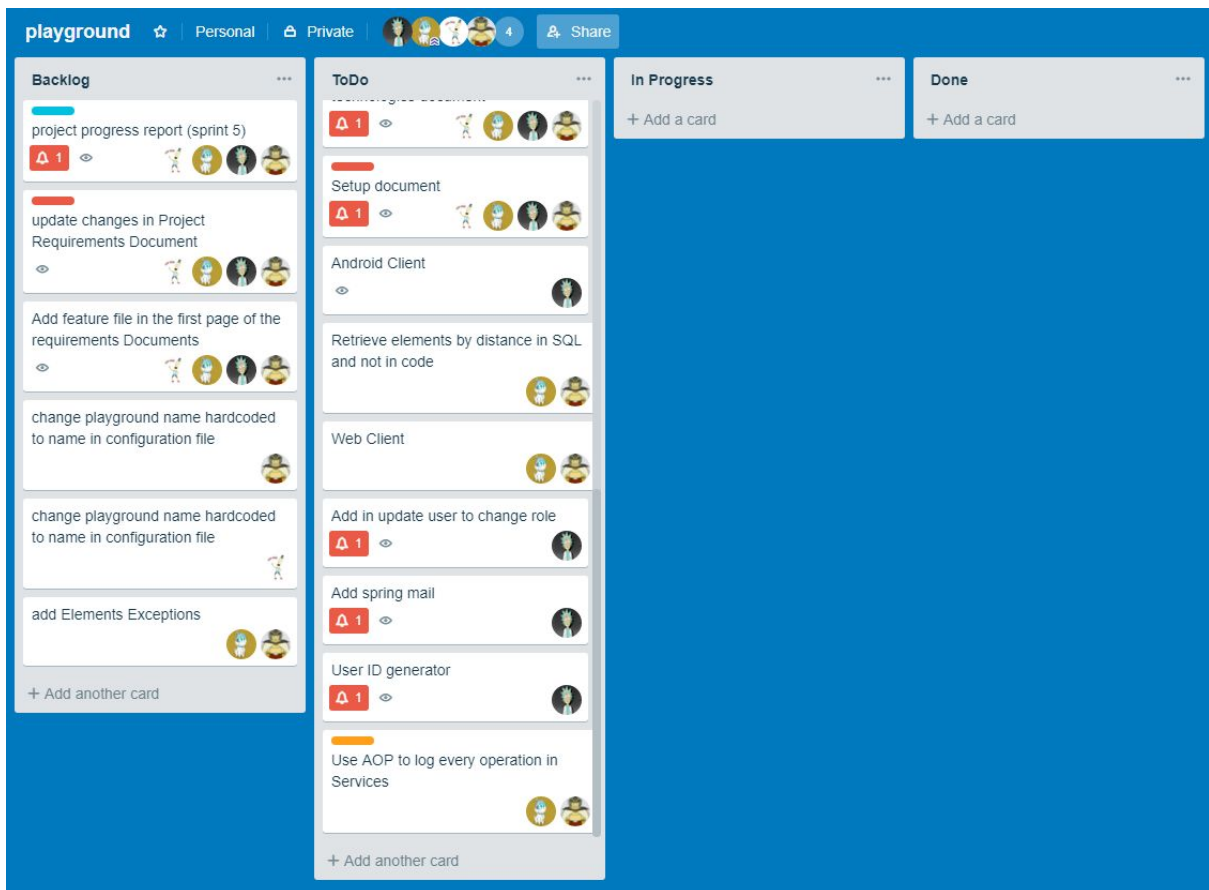
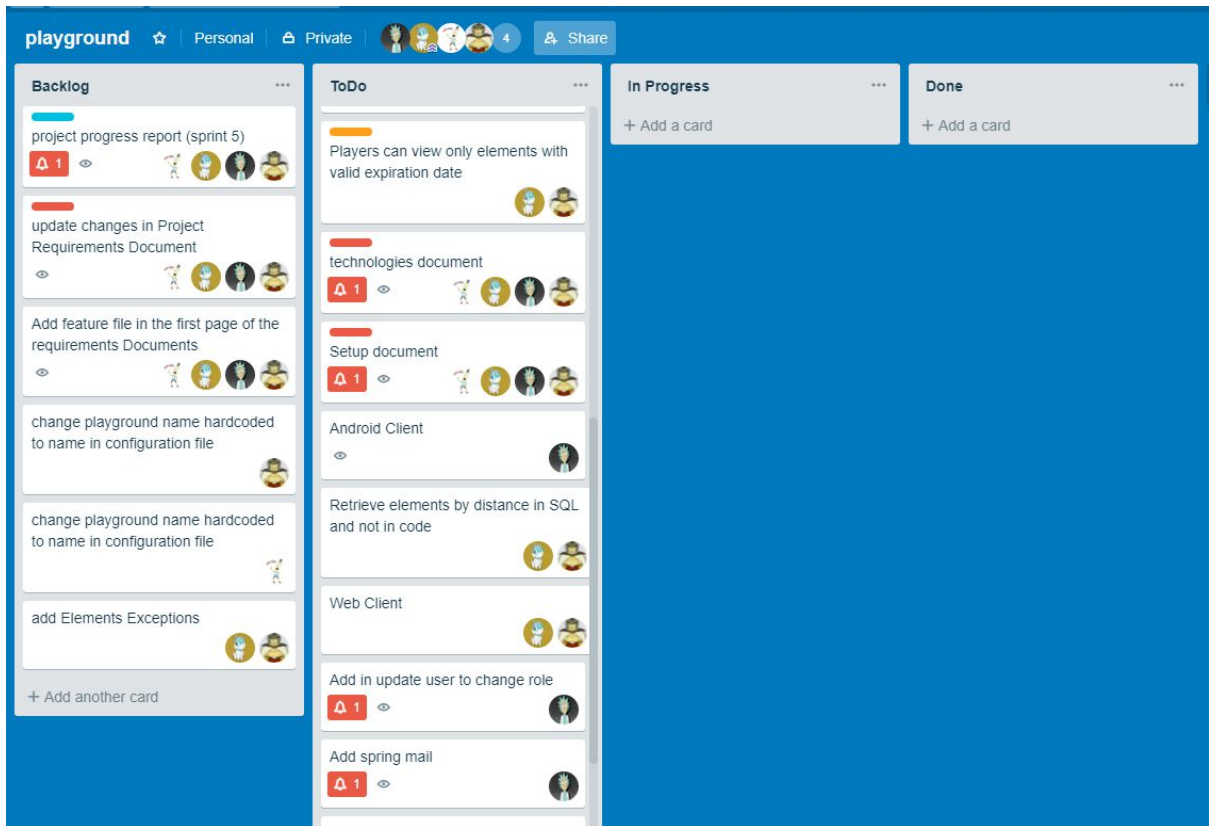




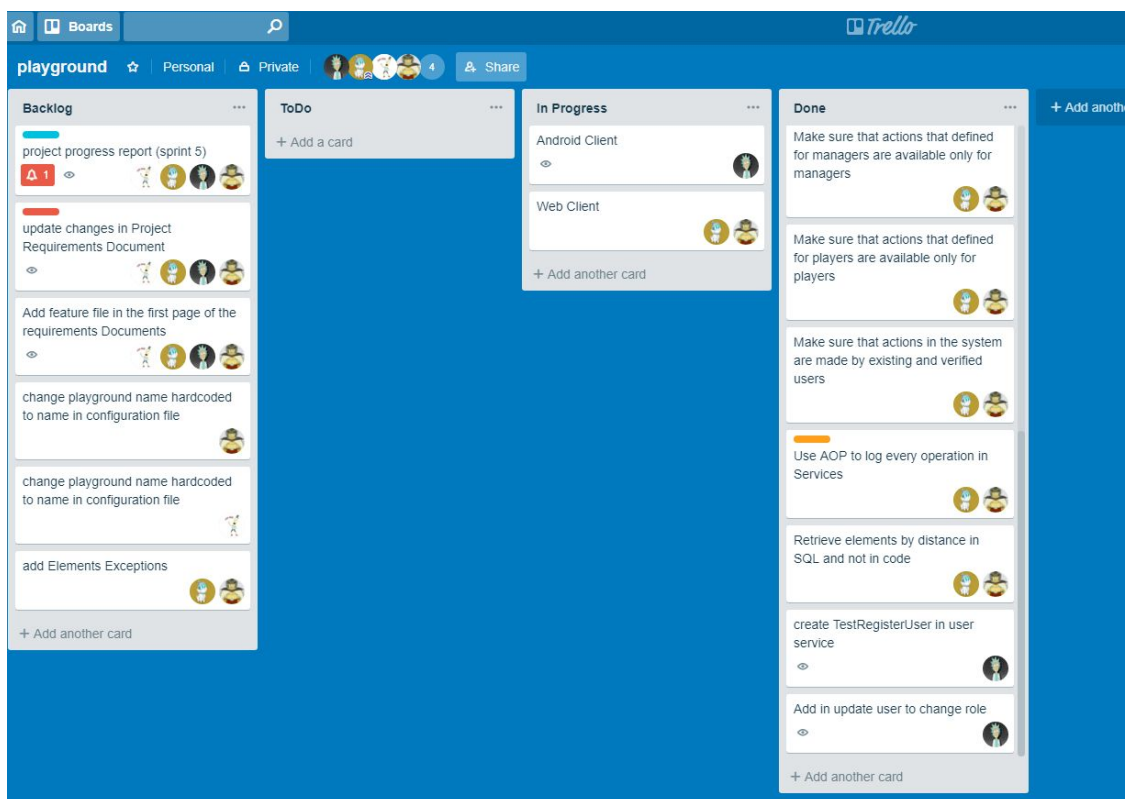
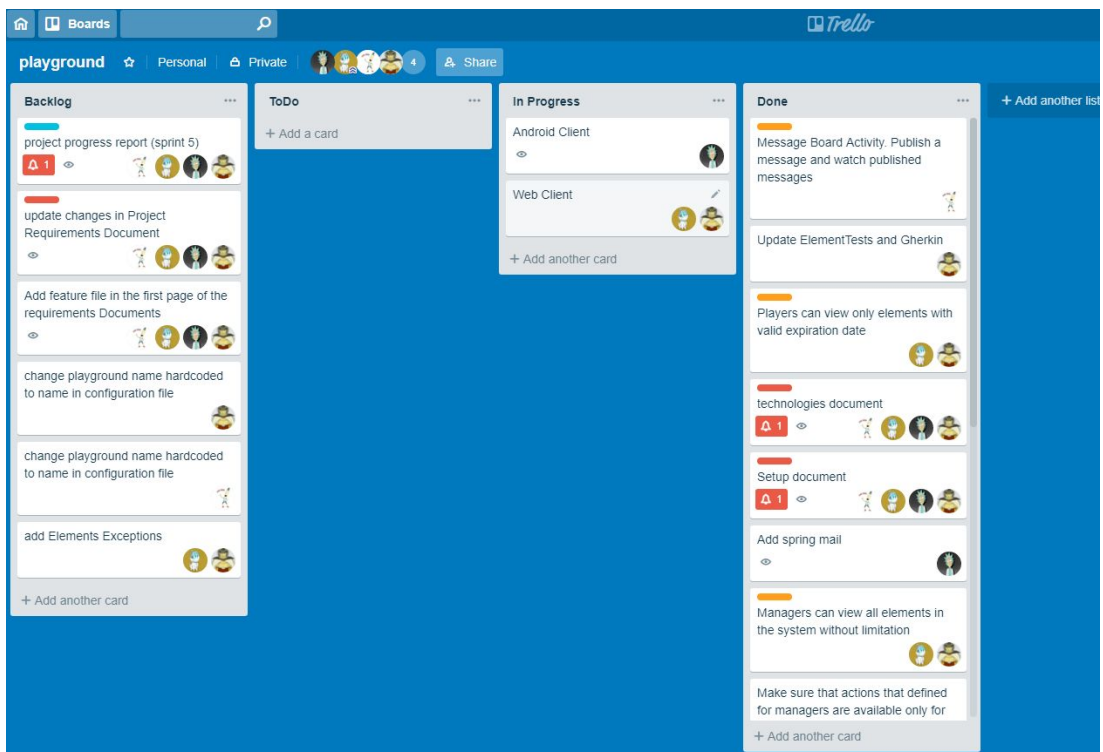
4.4

- Kanban Board of project from sprint 5 initiation (December 16th)





Kanban Board of project at the end of sprint 5 (December 30th)



- **What worked well for your team that you will preserve for future projects?**

Everyone helped each other when they ask for help and we achieved the results we wanted. Also Task management was organized and divided well.

- **How could you improve your work for other projects?**

We could meet more in person, a thing that was tough to arrange because of our different schedules.

- **What did you enjoy most working on this project?**

Seeing the products at the end of each sprint and their evolution into a complete, rich system.

- **What would you have done differently if you had to start this project again now, after the experiences of this semester?**

We would write more tests.

5. Technologies

- Spring
 - SpringBootApplication
 - SpringBootTest
 - SpringTransaction
- Junit
- JavaMail API
- Java Activation Framework (JAF)
- Google Hangouts
- Jackson
- Tomcat
- Stream api
- Git
- BitBucket
- Eclipse
- Trello
- H2
- Hibernate
- Android
- Android Studio
- ReactJS
 - axios
- NodeJS
- Visual Studio Code

6. Setup

Server

In order to run the project with eclipse, please follow these steps:

Install JDK 8 and eclipse.

Configure the 3 packages as sources for the project:

src/main/java

src/main/resources

src/test/java

Add libraries (external jars) from web.db.lib

you should have JavaMail API and Java Activation Framework (JAF) installed on your machine

-You can download latest version of [JavaMail \(Version 1.2\)](#) from Java's standard website.

-You can download latest version of [JAF \(Version 1.1.1\)](#) from Java's standard website.

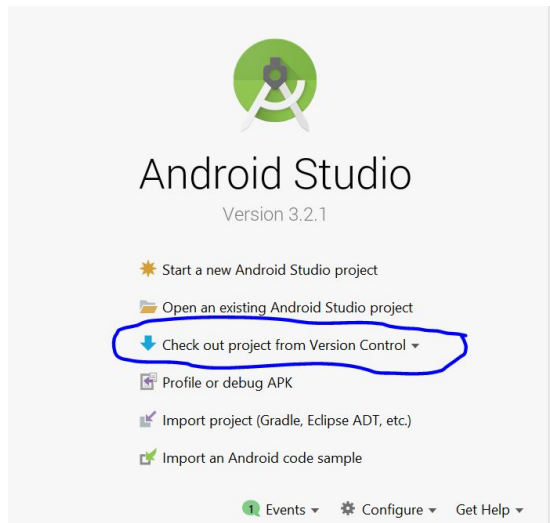
add this jars to your build path

Run the project as Java Application

you will find our server running at PORT 8084.

Android Client

Open new Project on Android studio, Select open and choose from version control



enter the following git repository : <https://github.com/RanWeiner/PlaygroundClient.git>
the project should be open up.

In the project explorer go to :

app/src/main/java/com/example/ran/ratingplayground_client/utils/AppConstants.java
change the variable - "HOST" to the ip of the host machine that runs the server + the
port number (8084 by default).

React Client

Install [node with npm](#).

Make sure that node and npm added to [environment PATH](#)

Extract the react_playground.zip file from the moodle.

open cmd and change to the directory with the extracted files.

In the root directory of the project, there is an .env file, containing the url of the
server, current defined as localhost at port 8084, update it if necessary.

Write 'npm install' and wait for the required packages to be installed.

When installation finished, write 'npm start'. the client should be opened
automatically in your default browser. In any case, it will be available at your
localhost, port 3000.