

# 2019A.sofer

## Farmground



Date: 07/01/19

Students:

- Linor Dolev - 201619079.
- Shiran Sofer - 308535483.
- Itay Goz - 307920074.

<b>Introduction</b>	<b>4</b>
Purpose of System	4
Scope of System	4
<b>Actors and Goals</b>	<b>5</b>
<b>Functional Requirements</b>	<b>6</b>
Use Case Diagram	6
Use Case Details	7
Use Case: Registration	7
Use Case: Confirmation	8
Use Case: Create Message Board	9
Use Case: Plant seed	10
Use Case: Post Message	11
Use Case: Read Message	12
<b>Non - Functional Requirements</b>	<b>13</b>
<b>Appendix</b>	<b>14</b>
Screenshots	14
Gherkin	17
Feature: farmground server	17
Feature: register user	17
Feature: confirm user	18
Feature: login User	19
Feature: Update user	20
Feature: Update points	20
Feature: create a new element	21
Feature: update an existing element	22
Feature: get specific element by id	23
Feature: show all elements	24
Feature: get elements by distance	25
Feature: search elements	27
Feature: create activity	28
Feature: Plant Plugin	28
Feature: Message Board Plugin	30
Test Status Report	31
Kanban Board	33
Technologies	34

Install instructions	34
List of students	35
Roles In Team	35
General summary of work	35
General summary of project	36
Kanban board - history	37
sprint 5	37
Sprint 4	38
Sprint 3	39
Sprint 2	40
Sprint 1	41

# **Introduction**

'Farmground' is a realistic game platform for simulating the management of a virtual farm.

In order to create a realistic platform, Farmground will provide the experience of dealing with real-life farm tasks as planting seeds and harvest their crops. For each task the players will earn 'Farmpoints'.

Additionally, Farmground will contain the 'Farmground Messages Board', where players can post messages and use it as a forum.

## **Purpose of System**

The purpose of the system would be using the platform as a game for players around the world and gives them opportunity to be a farmers.

## **Scope of System**

The system will allow the players planting vegetables.

The system will allow the players to read messages from the Farmground - Messages Board.

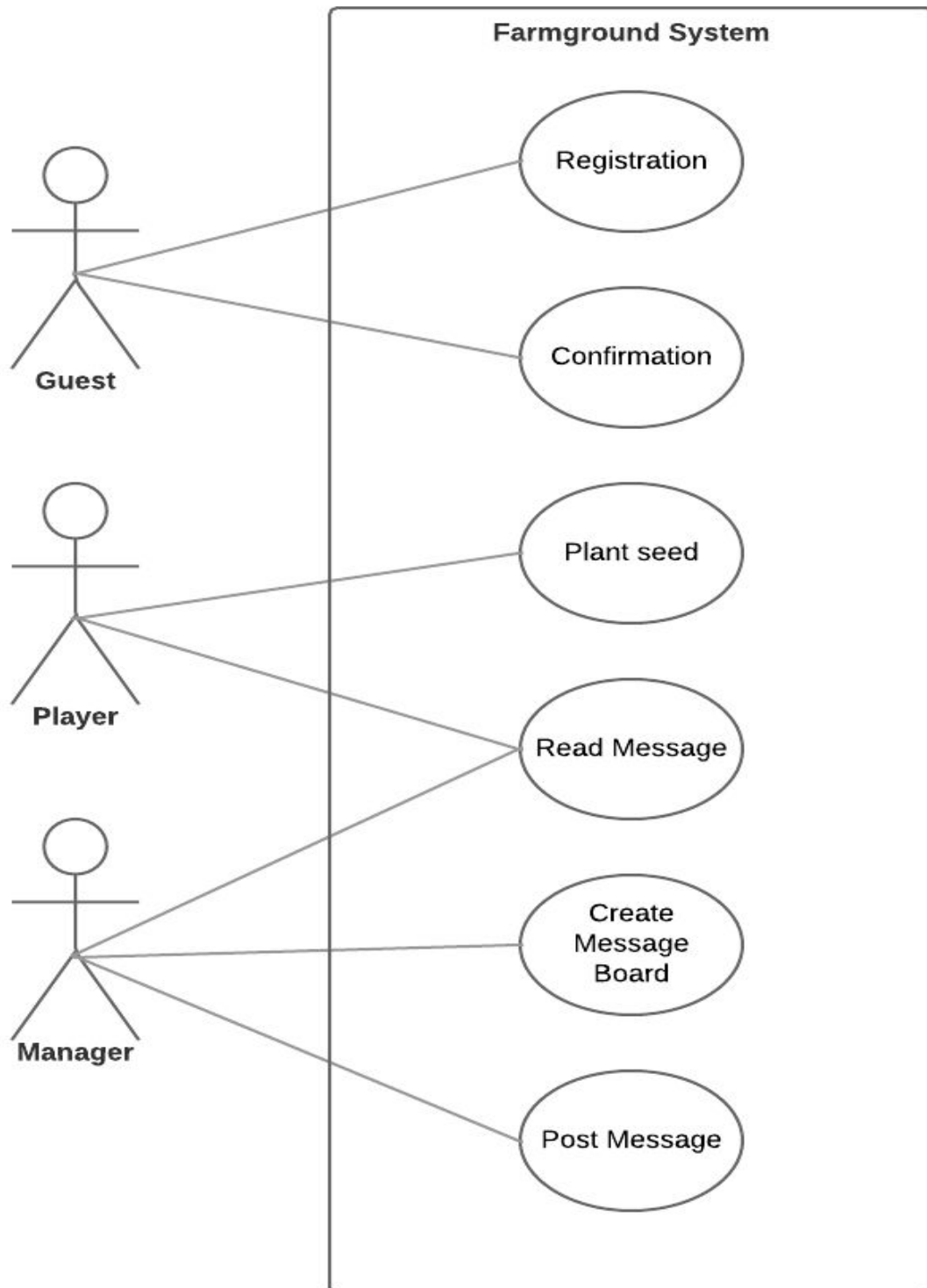
The system will give Farmpoints to players after completing tasks.

## Actors and Goals

Actors Names	Primary/ Support	Description	Goals
Manager	Primary	Can create message board and post messages to the board	Update the players in everything that happens on the farm
Player	Primary	Can plant seeds and read messages from the board	Earn Farmpoints
Guest	Primary	Non-Registered player	To register to Farmground

# Functional Requirements

## Use Case Diagram

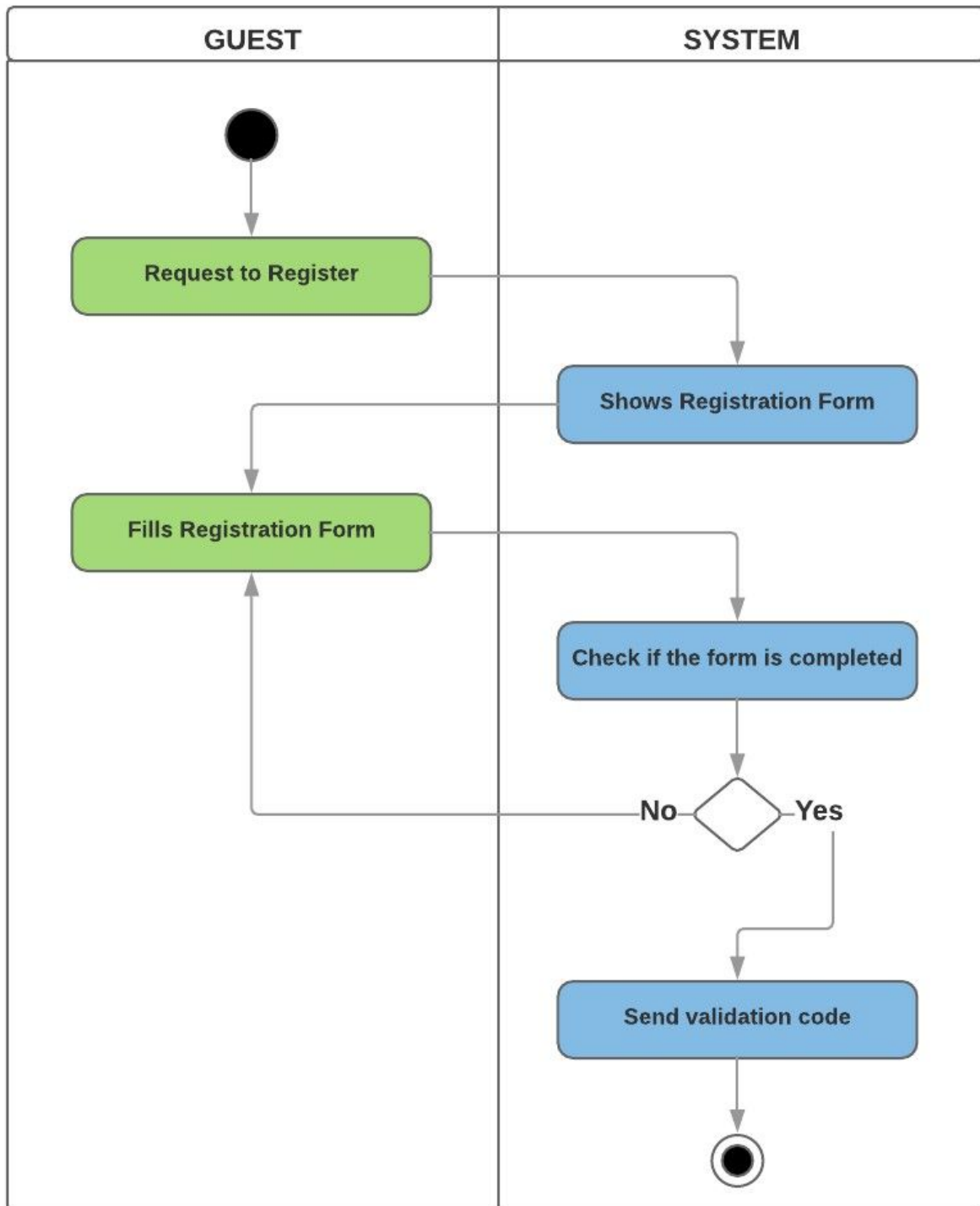


## Use Case Details

### Use Case: Registration

Participating Actors: Guest

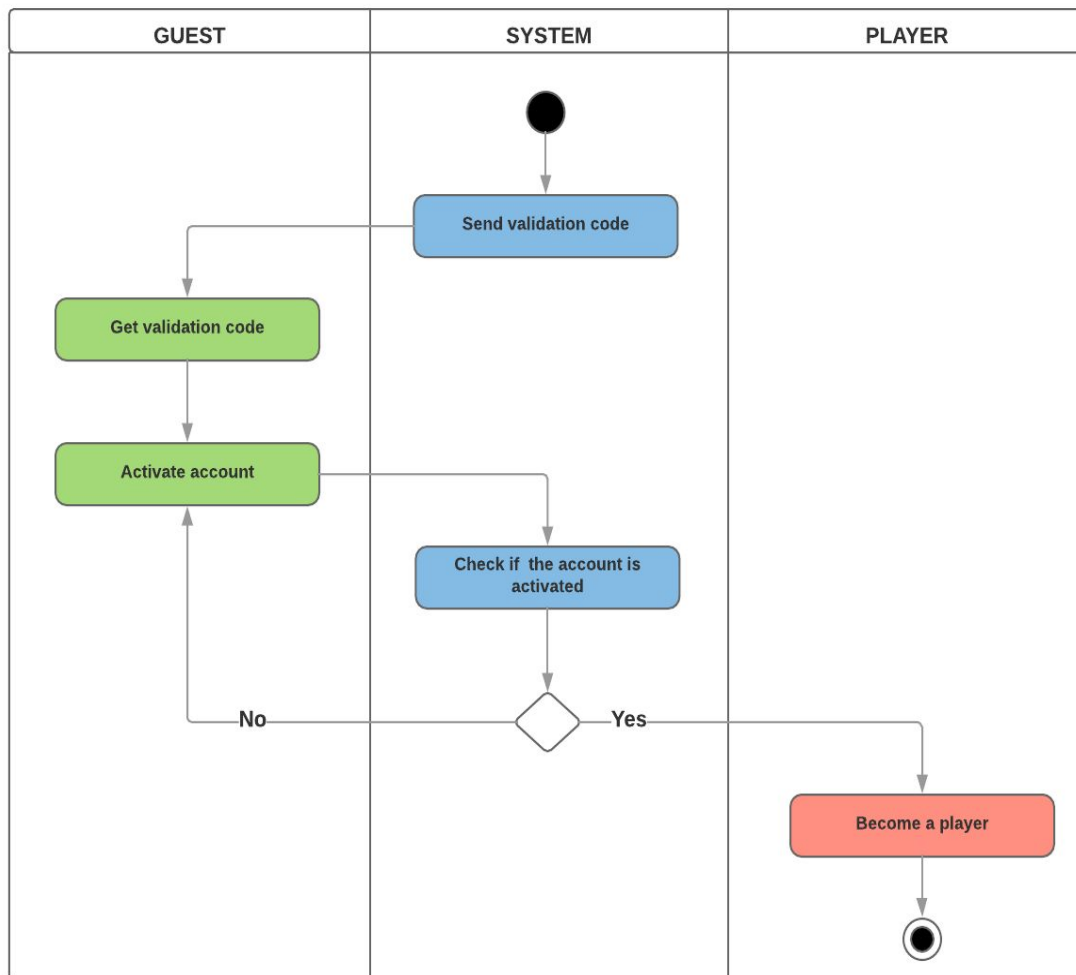
Goal: To register to Farmground



## Use Case: Confirmation

Participating Actors: Guest

Goal: To validate the account and become a player

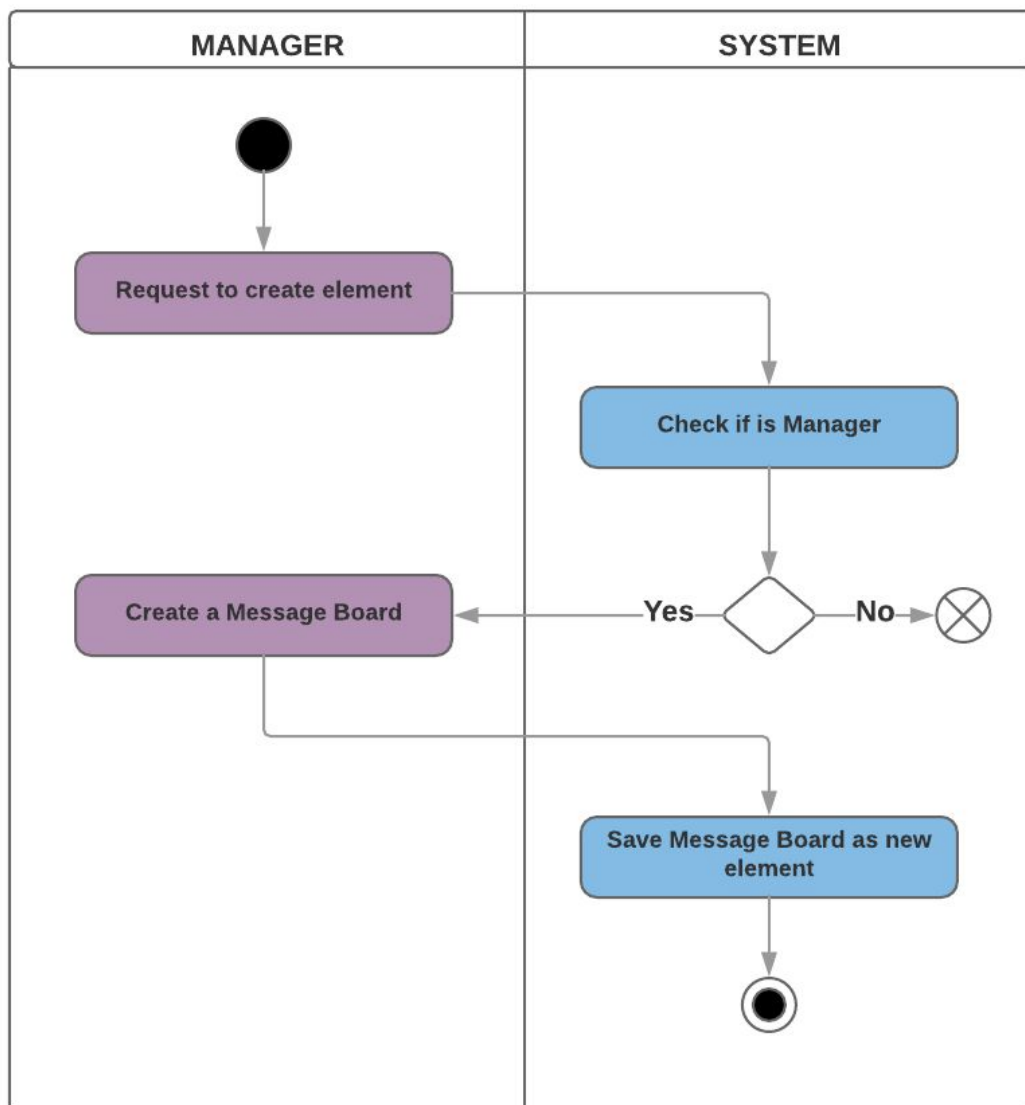




## Use Case: Create Message Board

Participating Actors: Manager

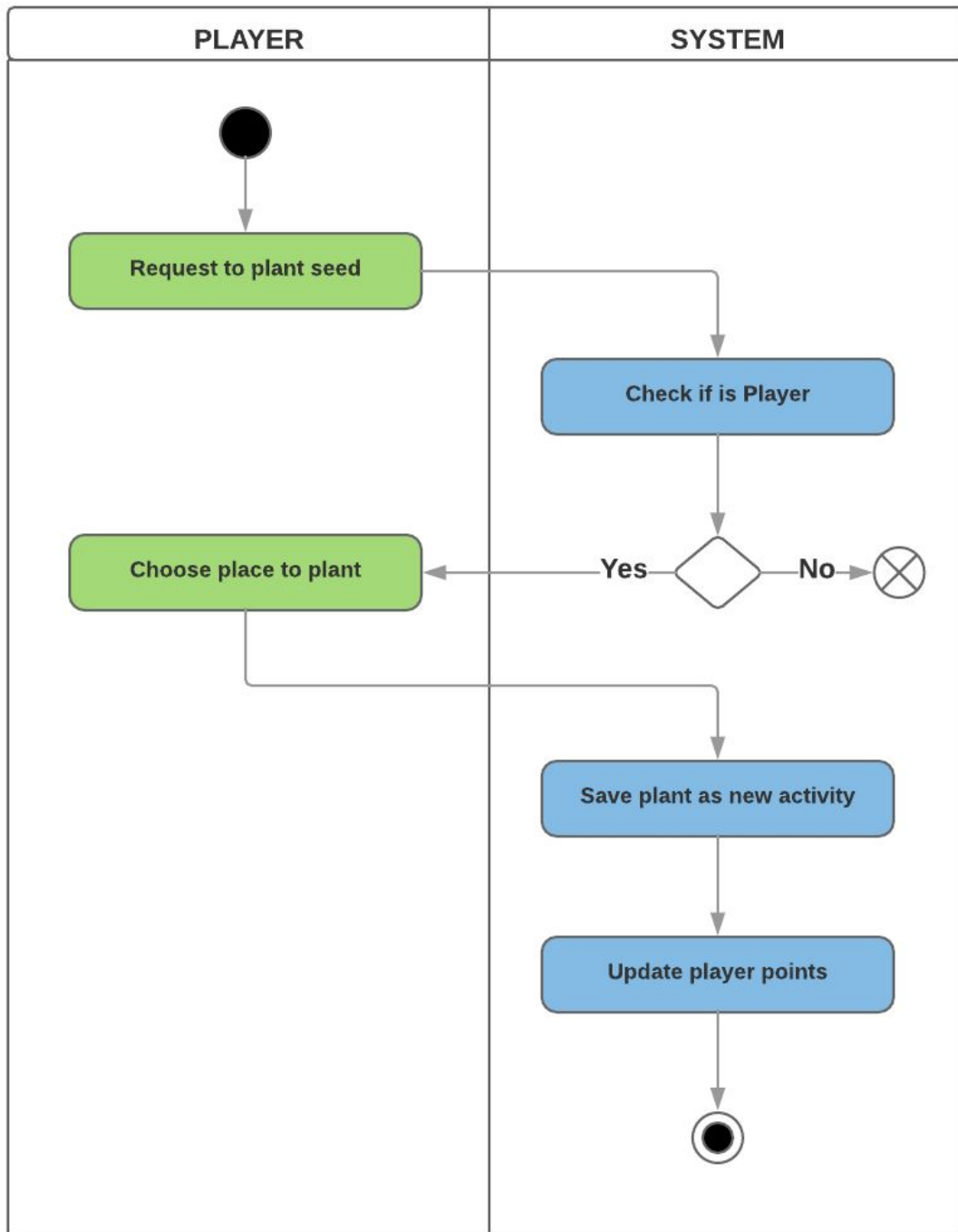
Goal: Managing Messages Board



## Use Case: Plant seed

Participating Actors: Player.

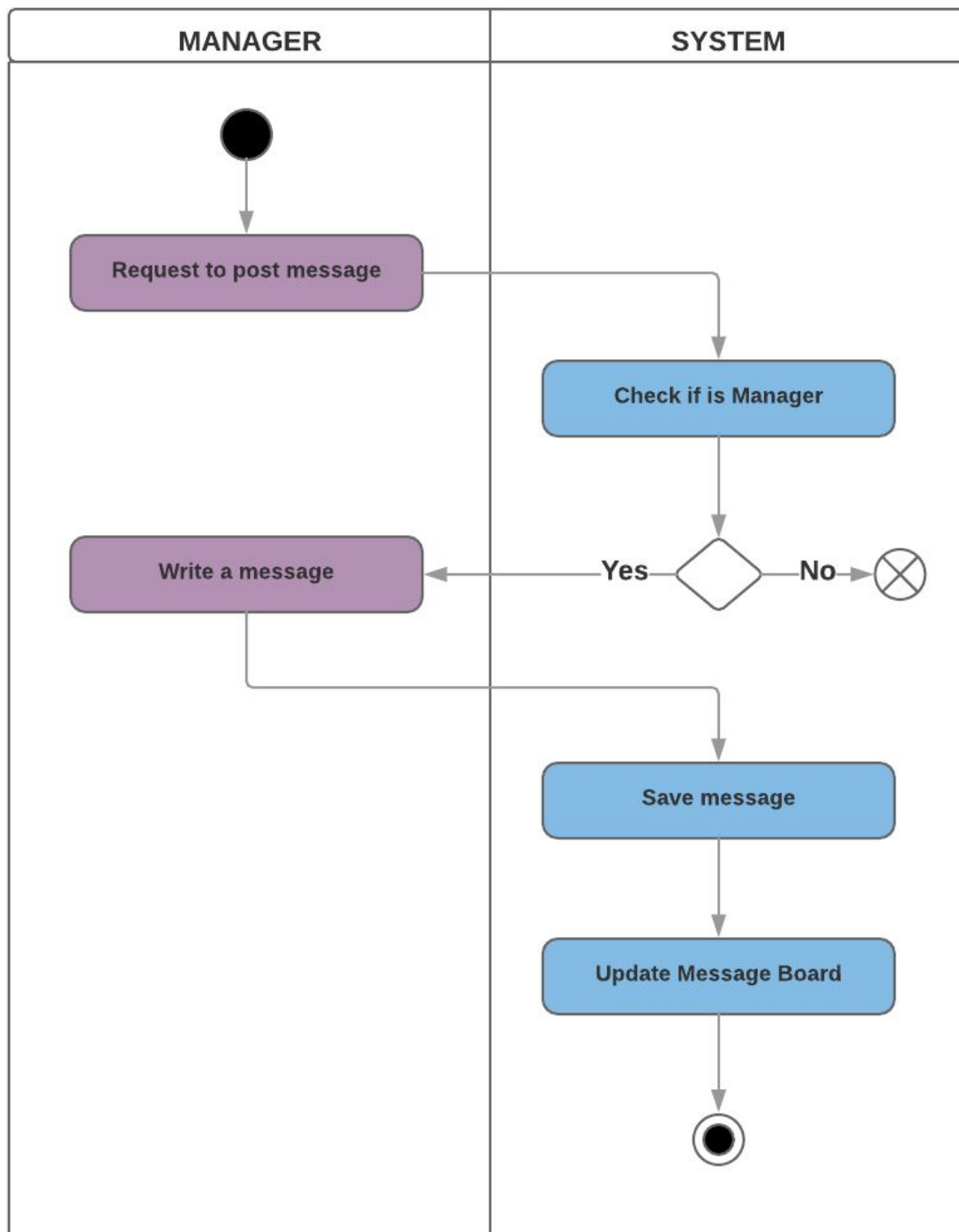
Goal: plant seed and earn farmpoints.



## Use Case: Post Message

Participating Actors: Manager

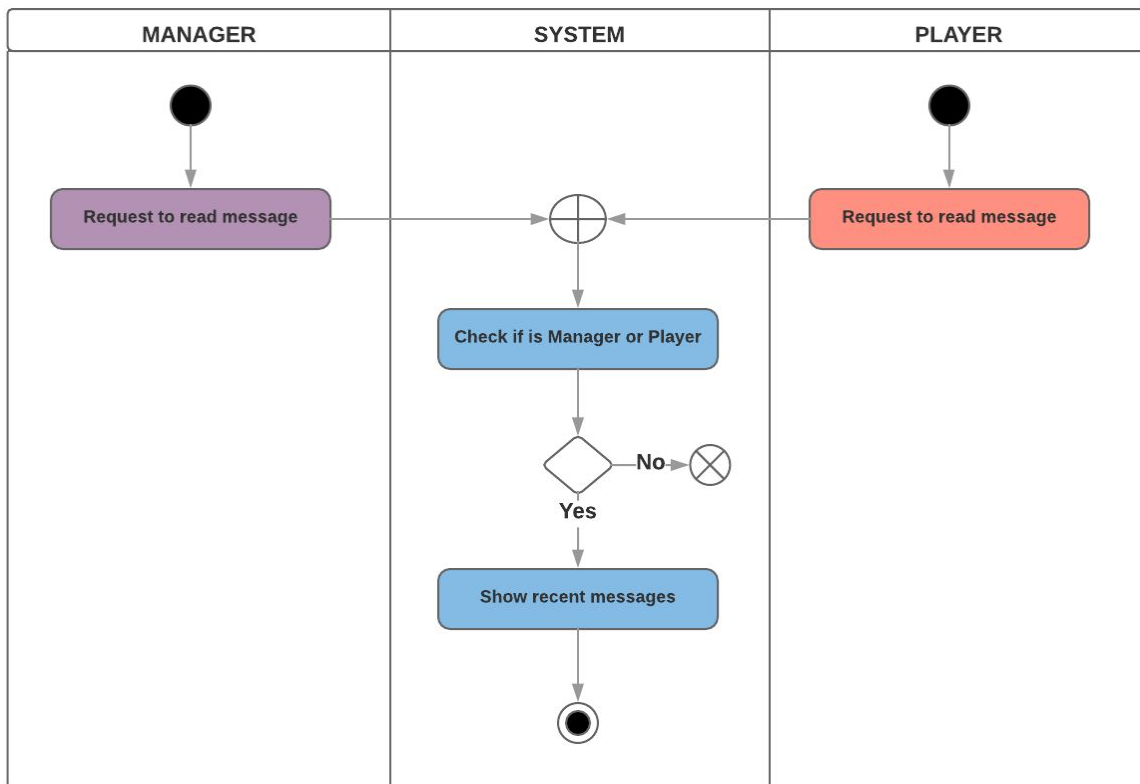
Goal: Inform players about Farmground.



## Use Case: Read Message

Participating Actors: Manager, Player

Goal: Be informed about Farmground.



## Non - Functional Requirements

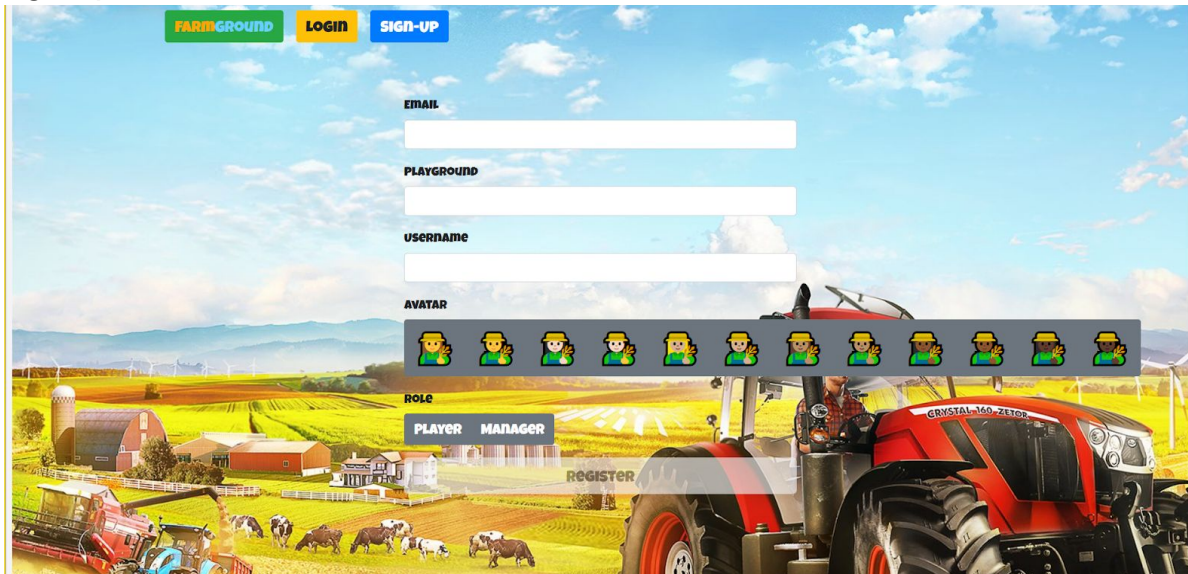
Requirement Number	Requirement Description	Requirement Type
1	The system will be convenient to the user	Usability
2	The system will not allow guests to play the game	Reliability
3	The system will allow three players to play simultaneously	Performance
4	The system will support all operating system	Supportability

- ❑ Test for requirement number 3:  
We created 3 players and played them simultaneously.
- ❑ Test for requirement number 4:  
The system was written in Java language and therefore supported on all operating systems.

# Appendix

## Screenshots

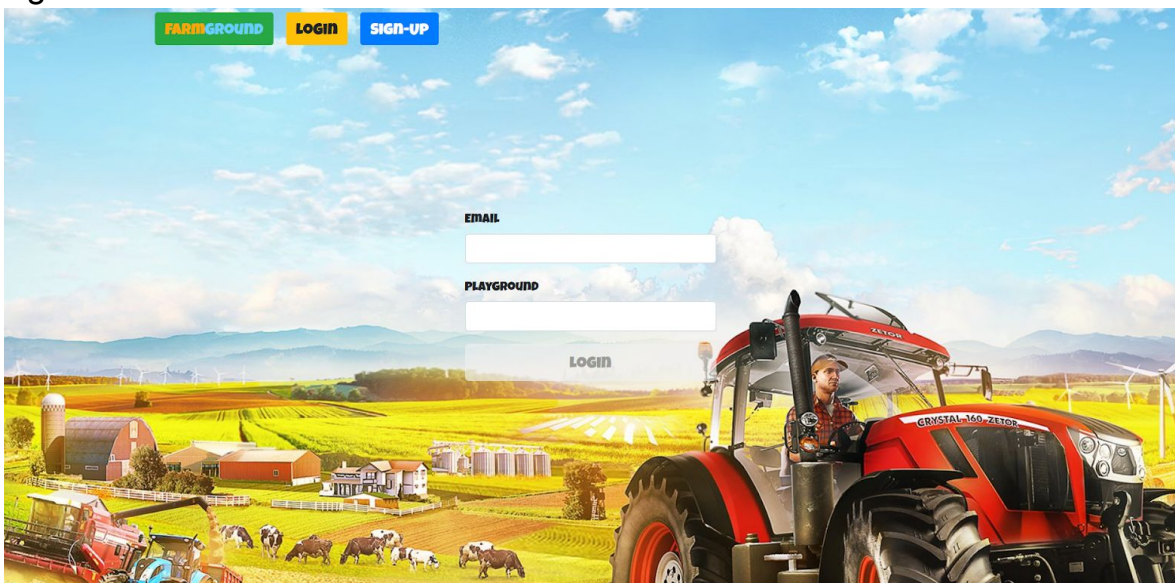
sign-up



The sign-up form is overlaid on a vibrant farm scene featuring a red Zetor tractor, a barn, and rolling hills. At the top, there are three buttons: 'FARMGROUND' (green), 'LOGIN' (yellow), and 'SIGN-UP' (blue). The form fields are as follows:

- EMAIL**: A white text input field.
- PLAYGROUND**: A white text input field.
- USERNAME**: A white text input field.
- AVATAR**: A horizontal row of 12 identical cartoon farmer avatars.
- ROLE**: Two buttons, 'PLAYER' and 'MANAGER', both in grey.
- REGISTER**: A semi-transparent grey button located below the role selection.

login

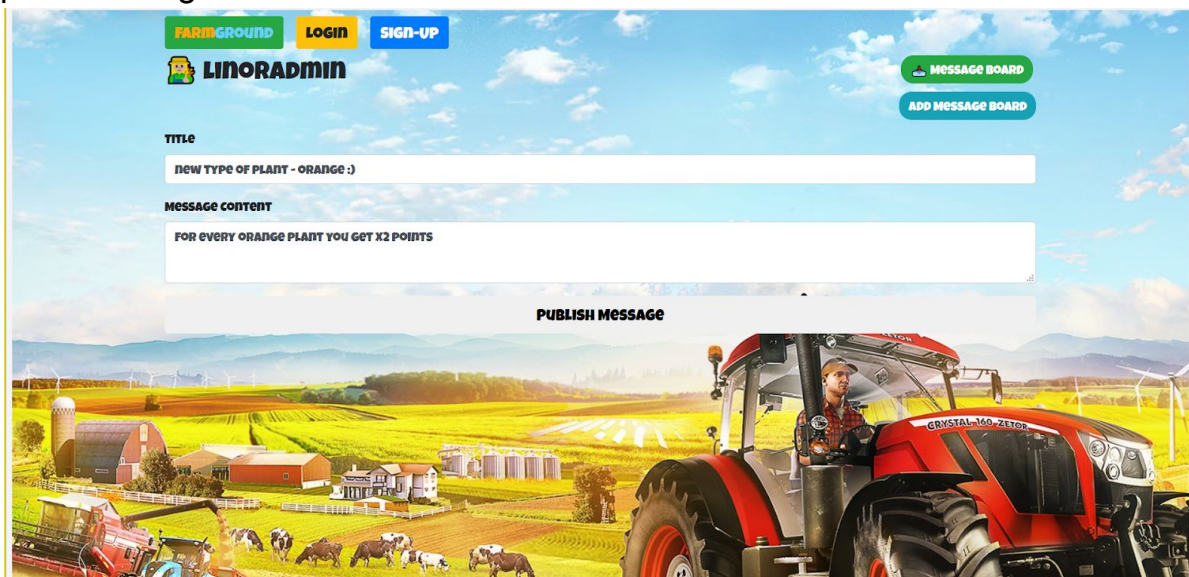


The login form is overlaid on the same farm scene as the sign-up page. At the top, there are three buttons: 'FARMGROUND' (green), 'LOGIN' (yellow), and 'SIGN-UP' (blue). The form fields are as follows:

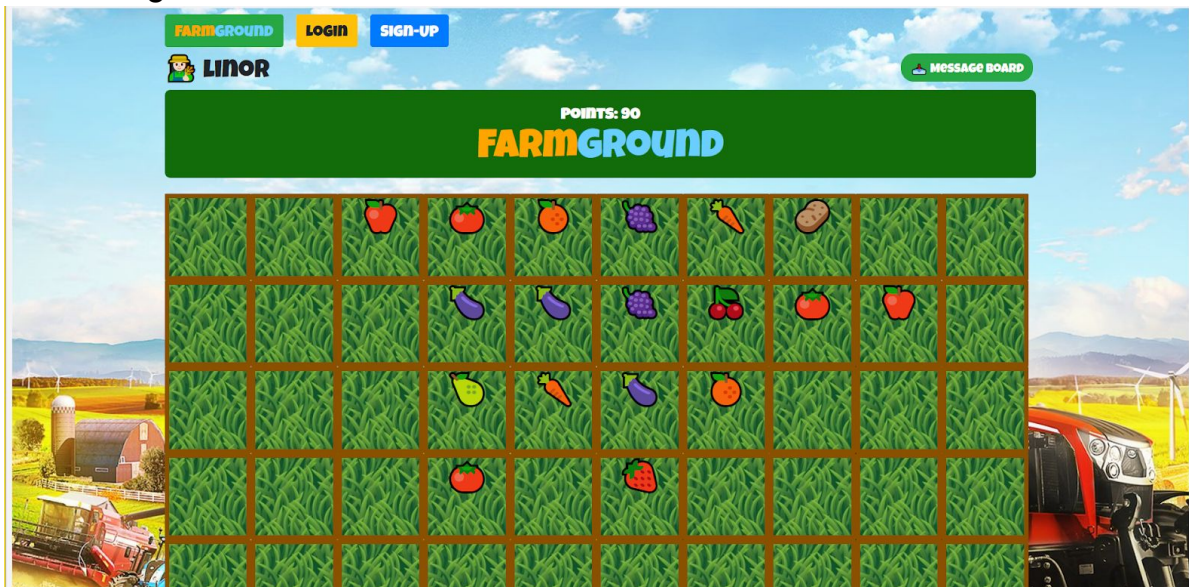
- EMAIL**: A white text input field.
- PLAYGROUND**: A white text input field.
- LOGIN**: A semi-transparent grey button located below the playground field.



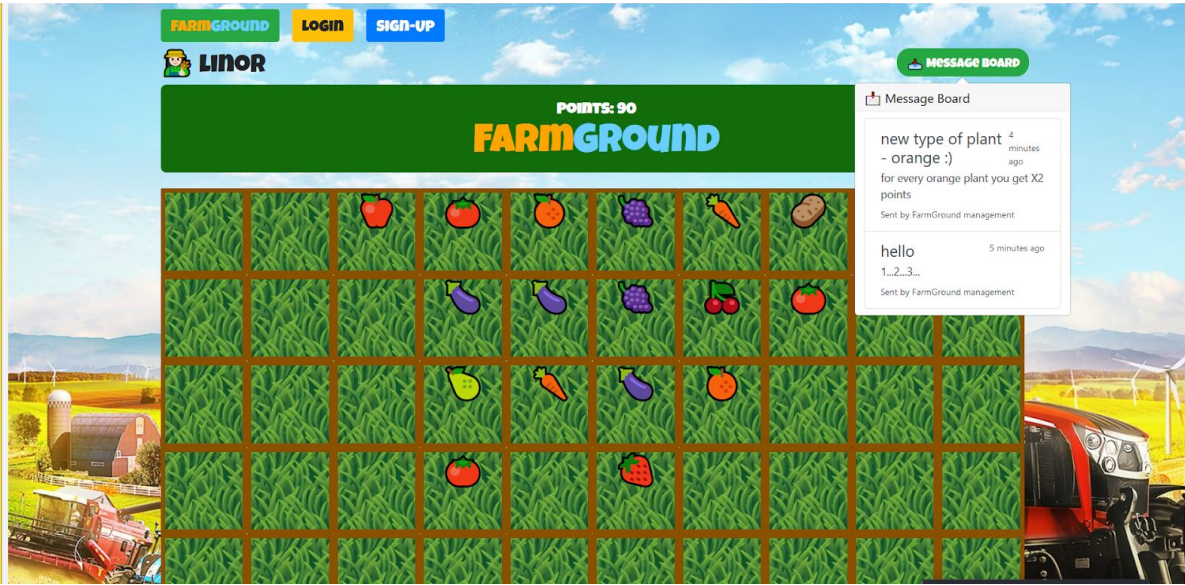
post message to the board



user farmground



message board





## Gherkin

### Feature: farmground server

- ❑ Scenario: test server initialized properly

**Given** nothing

**When** I starts the server

**Then** the server runs without throwing any errors/exception

### Feature: register user

- ❑ Scenario: test register user

**Given** the server is up

**When** I **POST** /playground/users

With { "email": "dummyUser@gmail.com",  
"username": "dummyUser",  
"avatar": "emoji",  
"role": "player" }

and header: **Accept:** application/json

**Content-Type:** application/json

**Then** the return status is 200

And the returned value is:

```
{  
  "email": "dummyUser@gmail.com",  
  "playground": "defaultPlayground",  
  "username": "dummyUser",  
  "avatar": "emoji",  
  "role": "player",  
  "points": 0  
}
```

- ❑ Scenario: register user with email that already registered

**Given** the server is up

And the database contains

[ { "NewUserFormEntity": "registerForm" } ]

**When** I **POST** /playground/users

With { "email": "dummyUser@gmail.com",  
"username": "dummyUser",  
"avatar": "emoji",  
"role": "player" }

and header: **Accept:** application/json

**Content-Type:** application/json

**Then** the return status is <> 2xx

## **Feature: confirm user**

### ❑ Scenario: test confirm user

**Given** the server is up

And the database contains [{"UserEntity": "userEntity"}]

With code: 200

**When I GET**

/playground/users/confirm/dummyPlayground/dummyUser@gmail.com/  
123

and header: **Accept:** application/json

**Content-Type:** application/json

**Then** the return status is 200

And the returned value is:

```
{  
  "email": "dummyUser@gmail.com",  
  "playground": "dummyPlayground",  
  "username": "dummyUser",  
  "avatar": "emoji, the code is:200",  
  "role": "player",  
  "points": 0  
}
```

### ❑ Scenario: test confirm user that does not exist

**Given** the server is up

And the database not contains [{"UserEntity": "userEntity"}]

**When I GET**

/playground/users/confirm/dummyPlayground/dummyUser@gmail.com/  
123

and header: **Accept:** application/json

**Content-Type:** application/json

**Then** the return status is <> 2xx

## Feature: login User

### ❑ Scenario: test login user

**Given** the server is up

And the database contains [{"UserEntity" : "userEntity"}]

**When I GET**

/playground/users/login/dummyPlayground/dummyUser@gmail.com

and header: **Accept:** application/json

**Content-Type:** application/json

**Then** the return status is 200

And the returned value is:

```
{
  "email": "dummyUser@gmail.com",
  "playground": "dummyPlayground",
  "username": "dummyUser",
  "avatar": "emoji",
  "role": "player",
  "points": 0
}
```

### ❑ Scenario: test login user that does not exist

**Given** the server is up

And the database not contains [{"UserEntity" : "userEntity"}]

**When I GET**

/playground/users/login/dummyPlayground/dummyUser@gmail.com

and header: **Accept:** application/json

**Content-Type:** application/json

**Then** the return status is <> 2xx

### ❑ Scenario: test login user that already logged in

**Given** the server is up

And the user is already logged in

**When I GET**

/playground/users/login/dummyPlayground/dummyUser@gmail.com

and header: **Accept:** application/json

**Content-Type:** application/json

**Then** the return status is <> 2xx.

### **Feature: Update user**

#### ❑ Scenario: test update user

**Given** the server is up

And the database contains: [{"UserEntity": "entity"}]

**When I PUT** /playground/users/dummyUser/dummyUser@gmail.com  
and header: **Content-Type:** application/json

**Then** the return status is 200

#### ❑ Scenario: update user that does not exist

**Given** the server is up

And the database not contains: [{"UserEntity": "entity"}]

**When I PUT** /playground/users/dummyUser/dummyUser@gmail.com  
and header: **Content-Type:** application/json

**Then** the return status is <> 2xx

### **Feature: Update points**

#### ❑ Scenario: test update points

**Given** the server is up

And the database contains: [{"UserEntity": "entity"}]

**When I PUT** /playground/users/dummyUser/dummyUser@gmail.com  
and header: **Content-Type:** application/json

**Then** the return status is 200

## Feature: create a new element

### ❑ Scenario: test create element successfully

**Given** the server is up

**When** I POST

```
/playground/elements/playground/elements/{userPlayground
}/{email} with body {      "id":"e1",
                           "name":"dummyElement",
                           "Type":"elementType",
                           "expirationDate":"null",
                           "Attributes":{"attr1":1},
                           "creatorPlayground":"playground",
                           "creatorEmail":"dummyUser@gmail.com"   }
```

**Then** the return status is 200.

### ❑ Scenario: test create element with existing key

**Given** the server is up

And the database already contains an element with id: "e1"

**When** I POST

```
/playground/elements/playground/elements/{userPlayground }/{email}
with body {  "id":"e1",
              "name":"dummyElement",
              "Type":"elementType",
              "expirationDate":"null",
              "Attributes":{"attr1":1},
              "creatorPlayground":"playground",
              "creatorEmail":"dummyUser@gmail.com"   }
```

**Then** the return status is <> 2xx.

## Feature: update an existing element

### ❑ Scenario: test update element successfully

**Given** the server is up

And the database contains { "id": "e1",  
"name": "dummyElement" }

**When** I PUT

/playground/elements/{userPlayground}/{email}/{playground}/{id}

with body { "id": "e1",  
"name": "dummyElement-UPDATED",  
"Type": "elementType",  
"expirationDate": "null",  
"Attributes": {"attr1": 1},  
"creatorPlayground": "playground",  
"creatorEmail": "dummyUser@gmail.com" }

**Then** response status is 200

And the database contains for id "e1" { "id": "e1",  
"name": "dummyElement-UP  
DATED" }

### ❑ Scenario: test update non existing element

**Given** the server is up

**When** I PUT

/playground/elements/{userPlayground}/{email}/{playground}/{id}

with body { "id": "e1",  
"name": "dummyElement-UPDATED",  
"Type": "elementType",  
"expirationDate": "null",  
"Attributes": {"attr1": 1},  
"creatorPlayground": "playground",  
"creatorEmail": "dummyUser@gmail.com" }

**Then** the return status is <> 2xx.

## Feature: get specific element by id

### ❑ Scenario: test get specific element by id successfully

**Given** the server is up

And the database contains an element with id: "e1"

**When** I GET

/playground/elements/{userPlayground}/{email}/{playground}/{id}  
and **Accept**:application/json

**Then** response status is 200

And the return value is { "id":"e1",  
                                  "name":"dummyElement",  
                                  "Type":"elementType",  
                                  "expirationDate":"null",  
                                  "Attributes":{"attr1":1},  
                                  "creatorPlayground":"playground",  
                                  "creatorEmail":"dummyUser@gmail.com"  
                                  }

### ❑ Scenario: test get specific element by invalid id

**Given** the server is up

**When** I GET

/playground/elements/{userPlayground}/{email}/{playground}/"null"  
and **Accept**:application/json

**Then** the return status is <> 2xx.

## Feature: show all elements

- ❑ Scenario: test show all elements using pagination successfully

**Given** the server is up

And the database contains an elements

[{"id": "e1", "id": "e11", "id": "e11"}]

**When** I GET

/playground/elements/{userPlayground}/{email}/all?size={5}&page={1}  
and **Accept**: application/json

**Then** response status is 200

And the return value is { "id": "e1",  
"name": "dummyElement",  
"Type": "elementType",  
"expirationDate": "null",  
"Attributes": {"attr1": 1},  
"creatorPlayground": "playground",  
"creatorEmail": "dummyUser@gmail.com",  
,"id": "e11",  
"name": "dummyElement1",  
"Type": "elementType",  
"expirationDate": "null",  
"Attributes": {"attr1": 1},  
"creatorPlayground": "playground",  
"creatorEmail": "dummyUser@gmail.com",  
,"id": "e111",  
"name": "dummyElement11",  
"Type": "elementType",  
"expirationDate": "null",  
"Attributes": {"attr1": 1},  
"creatorPlayground": "playground",  
"creatorEmail": "dummyUser@gmail.com",  
}



## Feature: get elements by distance

### ❑ Scenario: test get elements by distance

**Given** the server is up

And the database contains elements

```
[{"id": "e1", "x": "3.0", "y": "2.7"},  
 {"id": "e2", "x": "1.5", "y": "4.2"},  
 {"id": "e3", "x": "1D", "y": "100D"}]
```

**When I GET**

/playground/elements/dummyUser/dummyUser@gmail.com/near/3.0/2.  
7/50D

and header: **Accept:**application/json

**Then** the return status is 200

And the returned value is:

```
[{  "id": "e1",  
  "location": { "x": 3, "y": 2.7},  
  "name": "dummyElement"  
    "Type": "elementType",  
    "expirationDate": "null",  
    "Attributes": {"attr1": 1},  
    "creatorPlayground": "playground",  
    "creatorEmail": "dummyUser@gmail.com"  },  
  {  
    "id": "e2",  
    "location": { "x": 1.5, "y": 4.2},  
    "name": "dummyElement"  
      "Type": "elementType",  
      "expirationDate": "null",  
      "Attributes": {"attr1": 1},  
      "creatorPlayground": "playground",  
      "creatorEmail": "dummyUser@gmail.com"  }  
  }  
]
```

### ❑ Scenario: test get elements by distance with illegal x

**Given** the server is up

**When I GET**

/playground/elements/dummyUser/dummyUser@gmail.com/near/2D/2.  
7/50D

and header: **Accept:**application/json

**Then** the return status is <> 2xx

❑ Scenario: test get elements by distance with illegal y

**Given** the server is up

**When I GET**

/playground/elements/dummyUser/dummyUser@gmail.com/near/3.0/2D/50D

and header: **Accept:**application/json

**Then** the return status is <> 2xx

❑ Scenario: test get elements by distance with illegal distance

**Given** the server is up

**When I GET**

/playground/elements/dummyUser/dummyUser@gmail.com/near/3.0/2.7/-2D

and header: **Accept:**application/json

**Then** the return status is <> 2xx

❑ Scenario: test not found distance

**Given** the server is up

And the database contains element

```
[{  "id": "e1",
   "location": { "x": 1D, "y": 1D},
   "name": "dummyElement"
   "Type": "elementType",
   "expirationDate": "null",
   "Attributes": {"attr1": 1},
   "creatorPlayground": "playground",
   "creatorEmail": "dummyUser@gmail.com"  }]
```

**When I GET**

/playground/elements/dummyUser/dummyUser@gmail.com/near/100D/100D/1D

and header: **Accept:**application/json

**Then** the return status is <> 2xx

## Feature: search elements

### ❑ Scenario: test search elements

**Given** the server is up

And the database contains elements

```
{  "id": "e1",
  "name": "dummy"
  "Type": "value",
  "creatorPlayground": "playground",
  "creatorEmail": "dummyUser@gmail.com" },
{  "id": "e2",
  "name": "dummyElement"
  "Type": "value",
  "creatorPlayground": "playground",
  "creatorEmail": "dummyUser@gmail.com" },
{  "id": "e3",
  "name": "dummyElement"
  "Type": "elementType",
  "creatorPlayground": "playground",
  "creatorEmail": "dummyUser@gmail.com" }
```

**When I GET**

/playground/elements/userPlayground/dummyUser@gmail.com/search/  
attributeName/value

and **Accept:** application/json

**Content-Type:** application/json

**Then** the return status is 200

And the returned value is:

```
{  "id": "e1",
  "name": "dummy"
  "Type": "value",
  "creatorPlayground": "playground",
  "creatorEmail": "dummyUser@gmail.com" }
```

## Feature: create activity

### ❑ Scenario: test create activity successfully

**Given** the server is up

**When** I POST /playground/activities/{userPlayground}/{email}  
with body { "playground": "playground",  
              "id": "a1",  
              "elementPlayground": "dummyElement",  
              "elementId": "e1",  
              "type": "type",  
              "playerPlayground": "Farmground",  
              "playerEmail": "dummyUser@gmail.com",  
              "Attributes": { "attr1": 1 } }

**Then** response status is 200

## Feature: Plant Plugin

### ❑ Scenario: test Plant Plugin Successfully

**Given** the server is up

**When** I POST /playground/activities/{userPlayground}/{email}  
with body { "playground": "playground",  
              "id": "a1",  
              "elementPlayground": "dummyElement",  
              "elementId": "e1",  
              "type": "Plant",  
              "playerPlayground": "Farmground",  
              "playerEmail": "dummyUser@gmail.com",  
              "Attributes": { "color": "Purple", "name": "Eggplant" } }

**Then** response status is 200

### ❑ Scenario: test Plant Plugin With User That Is Not Exist

**Given** the server is up

**When** I POST /playground/activities/{userPlayground}/{email}  
with body { "playground": "playground",  
              "id": "a1",  
              "elementPlayground": "dummyElement",  
              "elementId": "e1",

```
    "type": "Plant",
    "playerPlayground": "Farmground",
    "playerEmail": "dummyUser@gmail.com",
    "Attributes": { "color": "Purple", "name": "Eggplant" } }
```

**Then** the return status is <> 2xx

❑ Scenario: test Plant Plugin With Wrong Element Type

**Given** the server is up

**When** I POST /playground/activities/{userPlayground}/{email}

with body { "playground": "playground",  
 "id": "a1",  
 "elementPlayground": "dummyElement",  
 "elementId": "e1",  
 "type": "Plant",  
 "playerPlayground": "Farmground",  
 "playerEmail": "dummyUser@gmail.com",  
 "Attributes": { "color": "Purple", "name": "Eggplant" } }

**Then** the return status is <> 2xx

## Feature: Message Board Plugin

### ❑ Scenario: test Add Messages Board Successfully

**Given** the server is up

**When** I POST /playground/elements/{userPlayground}/{email}

with body {  
    "id": "e1",  
    "name": "Message Board",  
    "Type": "Message Board",  
    "expirationDate": "null",  
    "Attributes": { "name": addMessageBoard },  
    "creatorPlayground": "playground",  
    "creatorEmail": "dummyUser@gmail.com" }

**Then** response status is 200

## Test Status Report

test	status
server initialized properly	V
register user	V
register user with email that already registered	V
confirm user	O
confirm user that does not exist	V
login user	O
login user that does not exist	V
login user that already logged in	V
update user	O
update user that does not exist	V
Update user points	O
create element successfully	V
create element with existing key	V
update element successfully	V
update element with existing key	V
get specific element by id	V
show all elements using pagination successfully	V
get elements by distance	V
get elements by distance with ilegal x	V
get elements by distance with ilegal y	V
get elements by distance with ilegal distance	V
search elements	V
create activity successfully	V
plant plugin successfully	O

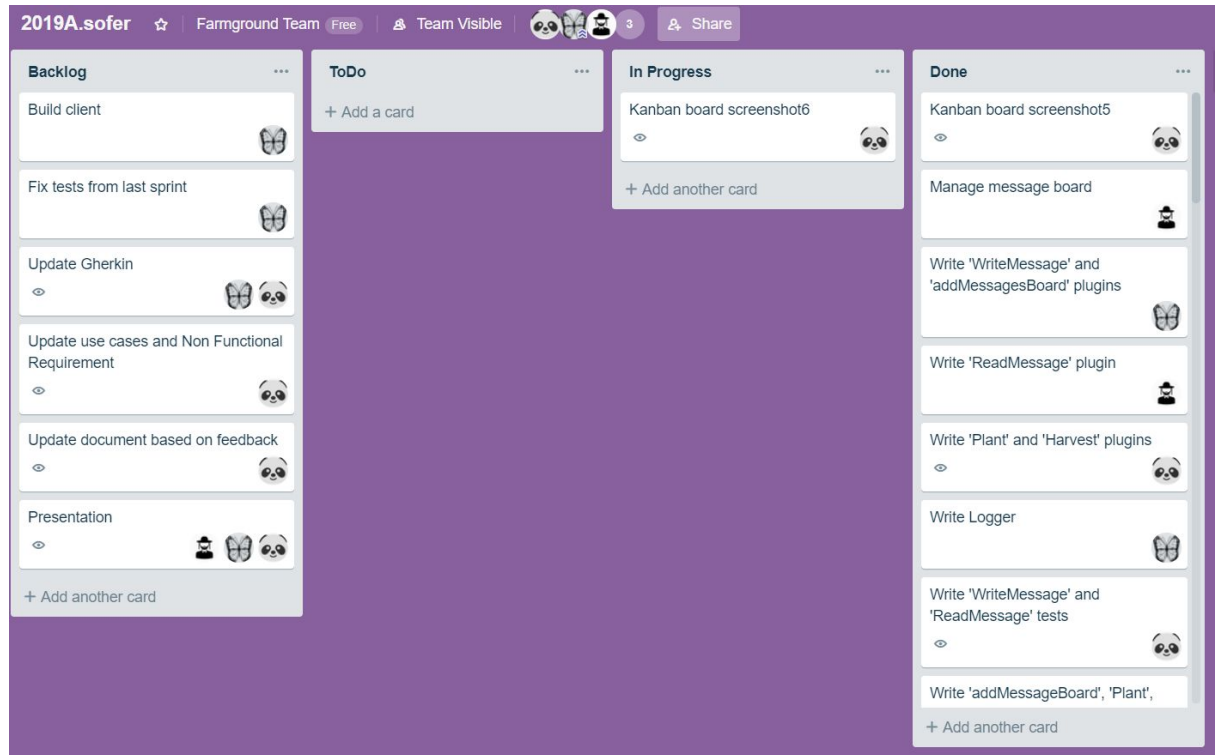
test Plant Plugin With User That Is Not Exist	V
test Plant Plugin With Wrong Element Type	V
test Add Messages Board Successfully	V
test write message to the messages board	V
test read message from the messages board	V

O - Registration test and user authentication are irrelevant when working with real email.

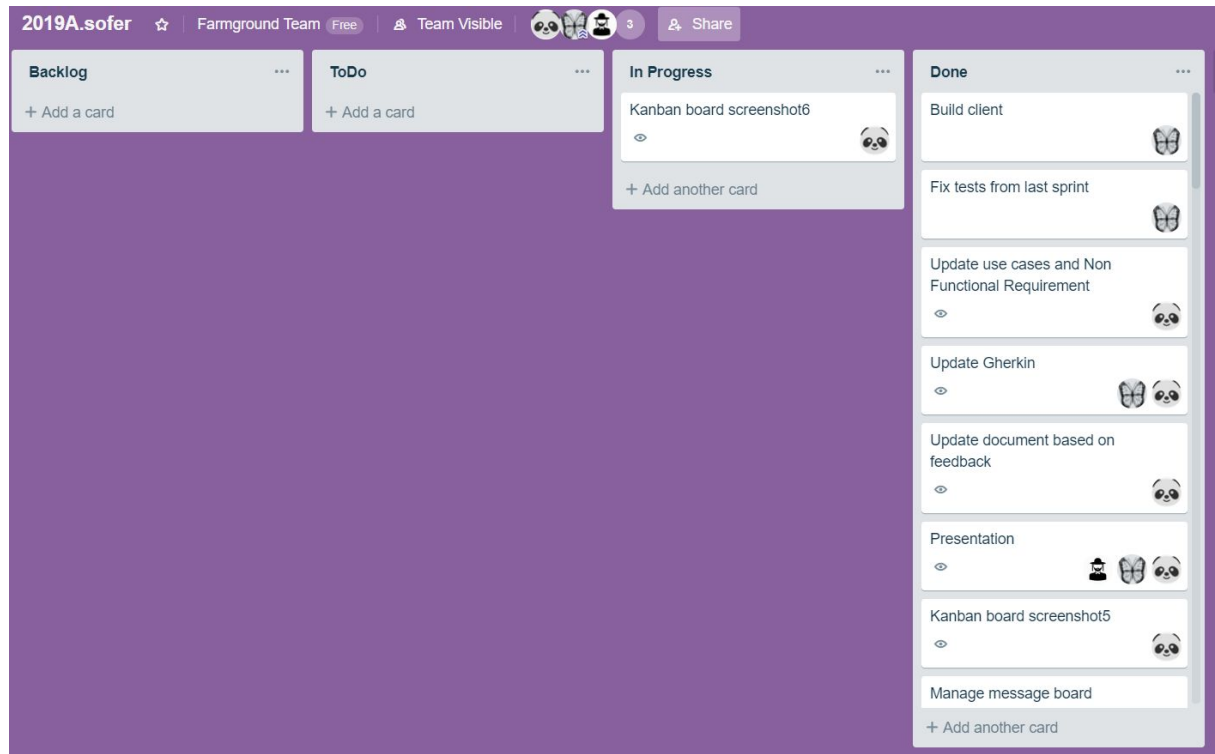


# Kanban Board

24/12/18:



07/01/19:



## Technologies

- Spring Core, Boot, Web, test, Jpa, Transaction, AOP.
- RESTful web.
- HTTP.
- JUnit.
- AssertJ.
- Postman.
- Bitbucket.
- Hibernate.
- H2 Database.
- Apache Common Logging.
- Java Reflection.
- JavaScript
- Reactjs.
- JavaMail API.

## Install instructions

1. Download Eclipse 2018-2019 - from the moodle.
2. Download jar files from web.lib dir - from the moodle.
3. Download JavaMail API jars.  
In Eclipse:
4. Open new java project.
5. Remove source folder by  
right click on your project -> Build Path -> Configure Build Path -> Remove src folder.
6. Import Projects from Git - Copy the Clone URI from BITBUCKET.  
Import as a general project.
7. Switch it again to JAVA Project by - right click on your project -> Project facets -> check Java.
8. Import jar files by  
right click on your project -> Build Path -> Configure Build Path -> Libraries -> Add External JARs -> select jars from stage 2.
9. Run it as a java application.
10. Download WebStorm from <https://www.jetbrains.com/webstorm/>
  - a. Open in WebStorm the client: File -> Open -> choose 'farmground-client' folder.
11. Write in terminal: npm install.
12. Write in terminal: npm start.

## List of students

- Linor Dolev - 201619079



- Shiran Sofer - 308535483



- Itay Goz - 307920074



## Roles In Team

- Linor Dolev - Scrum Master, DBA
- Shiran Sofer - Team Leader, QA
- Itay Goz - Product Owner, DevOps

## General summary of work

- What went well throughout the sprint:
  - We finished our project and we are satisfied from the result
- What should be improved in team work:
  - Planning our time more correctly according to the difficulty of tasks.
- What problems we had throughout the sprint:
  - It was difficult to learn new language in short time.
  - We had problems to connect the client to the server.  
We solved those problems by Google.

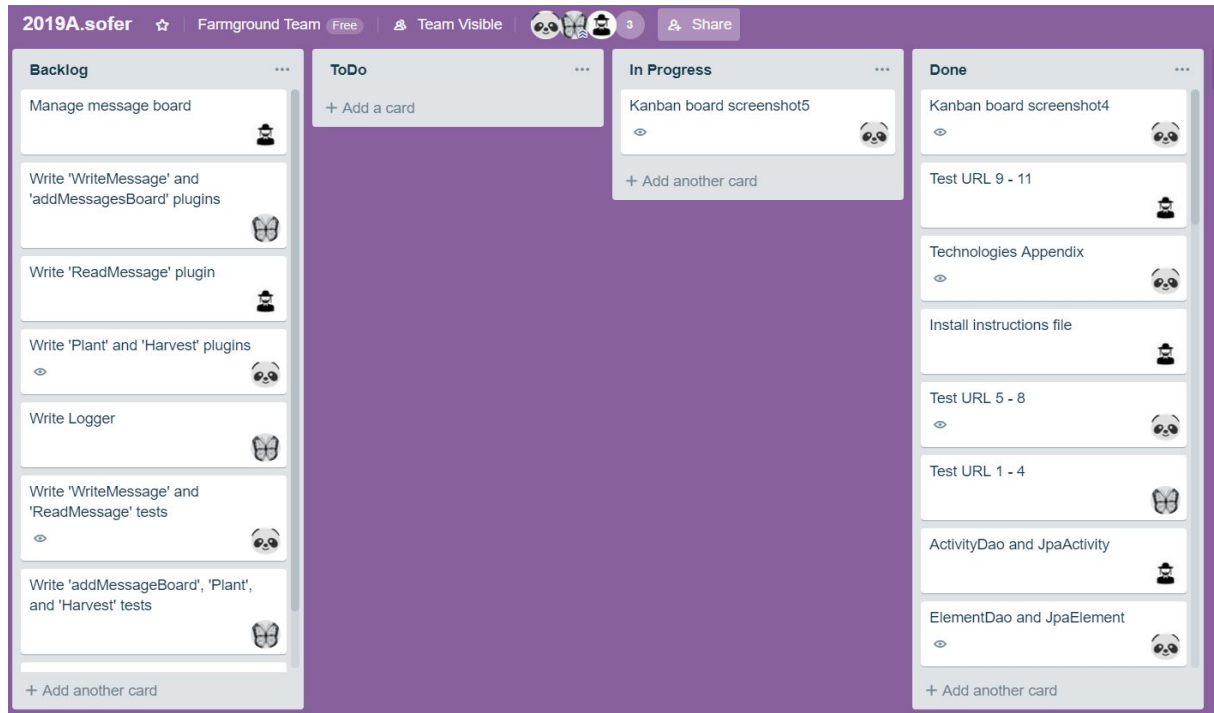
## General summary of project

- What went well throughout the project:  
We learned how to work in a team and a face-to-face meeting is not always necessary, we managed communication via whatsapp.
- What should be improved in team work:  
Divide the work into smaller tasks between members and plan our time more correctly.
- What we enjoyed most about working on the project:  
We enjoyed handling every sprint with new technologies that we did not know before.
- What would we do differently if you started the project now, after the knowledge and experience you accumulated in the semester:  
We would have start from writing the client because in that way the requirements would be more understandable.

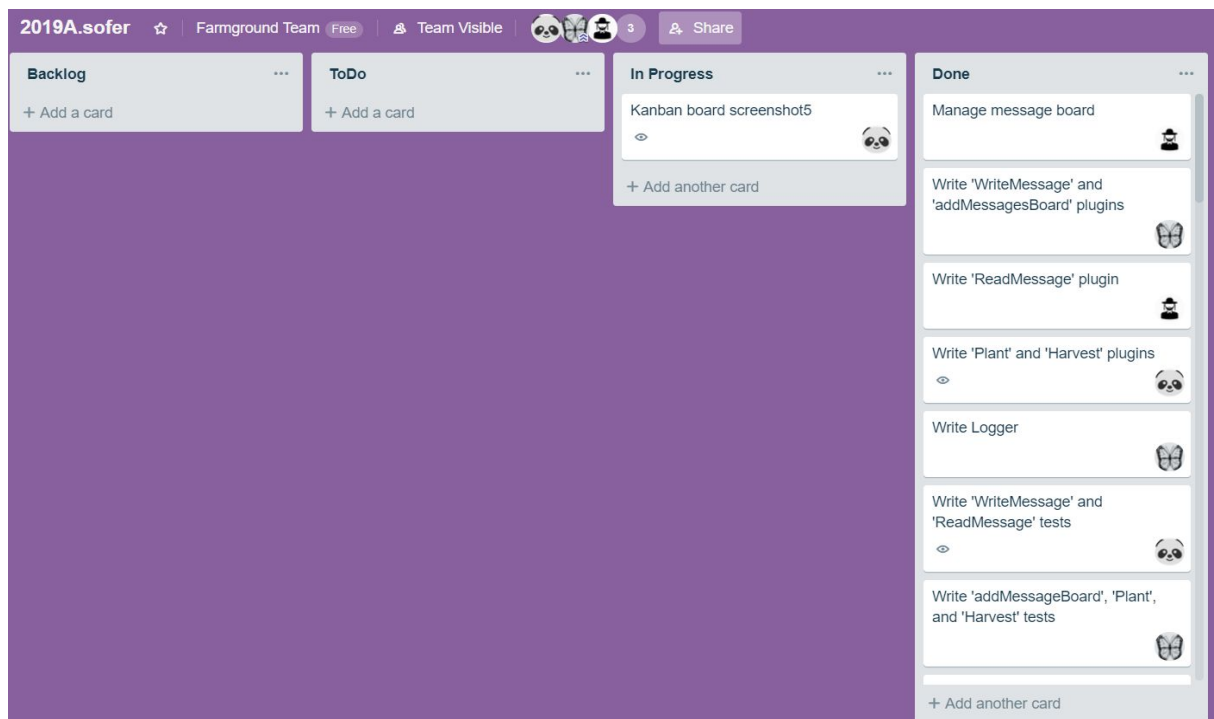
# Kanban board - history

sprint 5

10/12/18:



24/12/18:



## Sprint 4

26/11/18:

2019A.sofer ☆ Farmground Team Free Team Visible 3 Share

Backlog	ToDo	In Progress	Done
UserDao and JpaUser	+ Add a card	Kanban board screenshot4	Kanban board screenshot3
ElementDao and JpaElement			Test URL 9 - 11 Nov 24
ActivityDao and JpaActivity			Test URL 5 - 8 Nov 24
Test URL 1 - 4			Test URL 1 - 4 Nov 24
Test URL 5 - 8			Write technologies appendix
Test URL 9 - 11			ActivityTO - > entity + service Nov 18
Install instructions file			ElementTO - > entity + service Nov 18
Technologies Appendix			UserTO - > entity + service Nov 18
+ Add another card		+ Add another card	Read sprint1 feedback + Add another card

9/12/18:

2019A.sofer ☆ Farmground Team Free Team Visible 3 Share

Backlog	ToDo	In Progress	Done
+ Add a card	+ Add a card	Kanban board screenshot4	Technologies Appendix
			Install instructions file
			Test URL 9 - 11
			Test URL 5 - 8
			Test URL 1 - 4
			ActivityDao and JpaActivity
			ElementDao and JpaElement
			UserDao and JpaUser
		+ Add another card	+ Add another card

<https://trello.com>

## Sprint 3

12/11/18:

The screenshot shows a Kanban board for '2019A.sofer' with a team named 'Farmground Team'. The board is divided into four columns: Backlog, To Do, In Progress, and Done. The 'Backlog' column contains seven cards, including 'Read sprint1 feedback' (due Nov 12), 'UserTO - > entity + service' (due Nov 18), 'ElementTO - > entity + service' (due Nov 18), 'ActivityTO - > entity + service' (due Nov 18), and three 'Test URL' cards (due Nov 24). The 'In Progress' column has one card, 'Kanban board screenshot3'. The 'Done' column contains six cards, including 'Appendix - Project progress report', 'Kanban Board Screenshot2', 'Writes Gherkin documentation', 'Writes ElementTO Class', 'Writes NewUserForm and ActivityTo Classes', and 'Writes Location and UserTo Classes'. The 'To Do' column is empty.

Column	Card Title	Due Date	Assignee
Backlog	Read sprint1 feedback	Nov 12	Assignee 1
	UserTO - > entity + service	Nov 18	Assignee 2
	ElementTO - > entity + service	Nov 18	Assignee 3
	ActivityTO - > entity + service	Nov 18	Assignee 4
	Test URL 1 - 4	Nov 24	Assignee 5
	Test URL 5 - 8	Nov 24	Assignee 6
	Test URL 9 - 11	Nov 24	Assignee 7
In Progress	Kanban board screenshot3		Assignee 8
Done	Appendix - Project progress report		Assignee 9
	Kanban Board Screenshot2		Assignee 10
	Writes Gherkin documentation		Assignee 11
	Writes ElementTO Class		Assignee 12
	Writes NewUserForm and ActivityTo Classes		Assignee 13
	Writes Location and UserTo Classes		Assignee 14

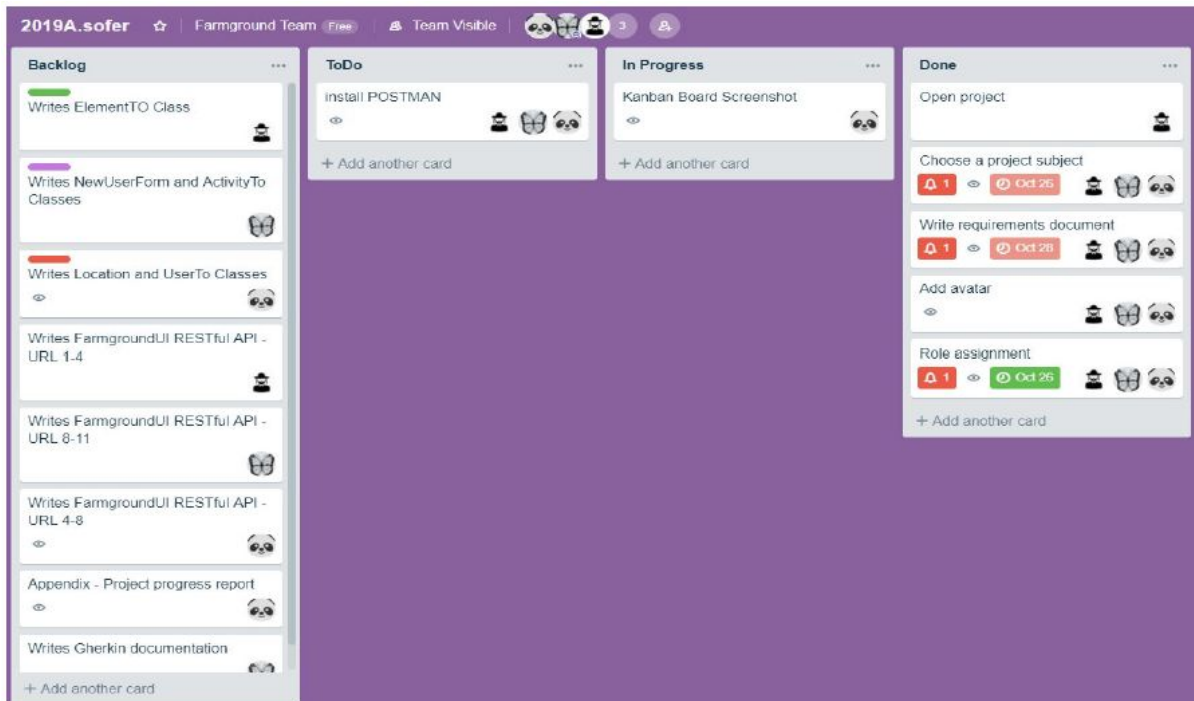
25/11/18:

The screenshot shows the same Kanban board on 25/11/18. The 'Backlog' column is now empty. The 'In Progress' column still has 'Kanban board screenshot3'. The 'Done' column now contains eight cards, including 'Test URL 9 - 11' (due Nov 24), 'Test URL 5 - 8' (due Nov 24), 'Test URL 1 - 4' (due Nov 24), 'Write technologies appendix', 'ActivityTO - > entity + service' (due Nov 18), 'ElementTO - > entity + service' (due Nov 18), 'UserTO - > entity + service' (due Nov 18), and 'Read sprint1 feedback' (due Nov 12). The 'To Do' column is empty.

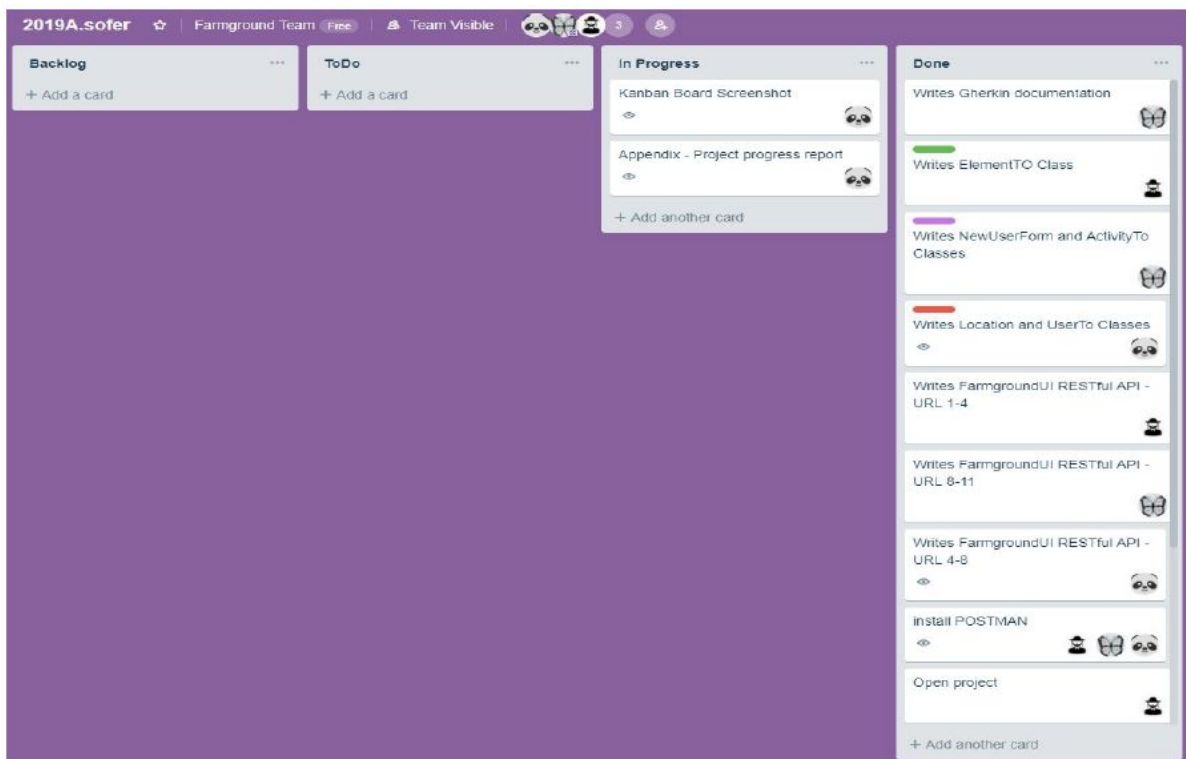
Column	Card Title	Due Date	Assignee
In Progress	Kanban board screenshot3		Assignee 8
Done	Test URL 9 - 11	Nov 24	Assignee 15
	Test URL 5 - 8	Nov 24	Assignee 16
	Test URL 1 - 4	Nov 24	Assignee 17
	Write technologies appendix		Assignee 18
	ActivityTO - > entity + service	Nov 18	Assignee 19
	ElementTO - > entity + service	Nov 18	Assignee 20
	UserTO - > entity + service	Nov 18	Assignee 21
	Read sprint1 feedback	Nov 12	Assignee 22

## Sprint 2

October 29th:



November 12th:





# Sprint 1

