POLITECNICO
MILANO 1863

DREAM project Buttiglione M.D.,
Caffagnini M., Faouzi D.

Implementation and Test Deliverable

Deliverable:	ITD
Title:	Implementation and Test Deliverable
Authors:	Marco Domenico Buttiglione, Martina Caffagnini, Dounia Faouzi
Version:	1.0
Date:	6-February-2022
Download page:	https://github.com/marticaffa/CaffagniniFaouziButtiglione.git
Copyright:	Copyright © 2022, Buttiglione M.D., Caffagnini M., Faouzi D. – All rights reserved

Contents

1 Introduction

1.1 Purpose

The current problems caused by climate change and Covid 19 in the Indian agricultural and economic sector have led to the need for renewal of the entire food supply chain. The goal is to create a versatile and resilient system that adopts innovative methodologies.
DREAM wants to acquire and combine data concerning meteorological short-term and long-term forecasts, farmers production, water irrigation, humidity of soil and data provided by the experts in order to help the main stakeholders.
On one hand, the application wants to monitor and determine who are the best farmers to resist to adverse situations and who are the others to be rewarded. On the other hand, it also defines who are the farmers that need help. Finally, the application is useful to understand if the initiatives carried out by the cooperation of agronomists and farmers bring significant results.
DREAM also aims to be a place for farmers where it is possible to view data on agriculture such as weather forecasts and information about the sowing time of specific plants. DREAM is a place to upload data regarding your production or problems and to ask for help and suggestions from the local agronomists and farmers by creating forums for discussion with colleagues.
DREAM sets to give agronomists the chance to share their knowledge and to answer questions from local farmers. The system allows agronomists to view their daily farm visit plans and may possibly change it at the end of the day.

1.2 Scope

The DREAM service wants to acquire and combine data to manage and to keep under control the farmers' production in Telangana with the goal of building a resilient food system.
DREAM should provide the following main features:

- **Request for Help**: allows farmers to send requests for help to their specific agronomist of the area. The local agronomist will reply by giving some advice.

- **Forum**: allows farmers to communicate with other farmers and to share information and problems they face. It is possible to post a topic, to ask a question, to leave a comment, to open conversations.

- **Calendar for Meetings**: allows farmers and agronomists to check their schedule, to modify and to cancel appointments. The agronomist is supposed to visit the farmer at least twice each year to check how the farmer is managing the farm. The agronomist will perform some routine tests such as retrieving a sample of soil to perform composition soil tests. During the first meeting, the agronomist have to install the sensors to obtain soil data.

- **Display Knowledge and Weather Forecasts**: allows farmers to visualize relevant data as weather forecasts, personalized suggestions, such as specific crops to plant or fertilizers to use. The latter is shared by the agronomists considering his studies and experience.

- **Enter Production Data**: after having registered and logged in, this feature gives farmers the possibility to enter the system data concerning their production. They will be able to enter data such as the type of products they cultivate and the correspondent production.

- **Grading System**: The information added by the farmers and the data retrieved by the agronomists will be used to evaluate farmers resilience and how well they are performing. Finally, the grades are shared with the Policy Makers to allow them to understand whether a farmer is performing well and needs to be rewarded or is performing badly and deserves to be helped by the government.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **Farmer**: the "average" customer of the application that shares their progress and visualizes data relevant to him;
- **Agronomist**: the customer that receives requests for help and answers them, visualizes data concerning weather forecasts and farmers in the area and organizes meetings with farmers;
- **Policy Maker**: the customer that visualizes data uploaded by farmers and agronomists;
- **Customer**: general DREAM customer, can be a farmer or an agronomist or a policy maker;
- **Notification**: a message shown to the customer when he needs to be notified about something;
- **Request**: request for help made by a farmer to a single agronomist;
- **Topic**: anything that farmers want to share with other farmers. They can either ask for help or share something they know that might help others;
- **Knowledge**: information shared by the agronomists. It has the goal to inform the farmers and to increase their awareness to improve their performance;

1.3.2 Acronyms

Acronyms	Description
RASD	Requirements Analysis and Specification Document
UI	User interface
DREAM	Data-dRiven prEdictive fArMing
MVC	Model View Controller
RDBMS	Relational DBMS
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UI	User Interface

1.3.3 Abbreviations

Abbreviation	Description
Rn	n-th Requirement

1.4 Revision history

Date	Description
25/01/2022	Creation of the document
29/01/2022	Introduction
02/02/2022	Source Code Structure
02/02/2022	Implementation Requirement
02/02/2022	Development Details
03/02/2022	Test
03/02/2022	Installation Instructions
06/02/2022	First Release

1.5 Reference Documents

- Slides of the course Software Engineering 2
- Requirement Engineering and Design Project: goal, schedule, and rules
- I&T assignment goal, schedule, and rules

1.6 Document Structure

The document is composed of the following chapter:

Chapter 1: This chapter provides an introduction to the purpose and the scope of the software as already specified in the previous documents. At fist there will be a general description of the system and of the features it should provide. Also, the lists of abbreviations, acronyms, and definitions used in the document, the revision history, and the reference documents are shown.

Chapter 2: This chapter specifies the traceability of the requirements with the details which ones are covered by functionalities already implemented at the current state of the software system development.

Chapter 3: This chapter specifies the framework, programming languages and other details about the development phase. It expands what had already been discussed in the DD.

Chapter 4: This chapter includes and overview of the structure of the source code and its organization.

Chapter 5: This chapter includes the main test cases and the approach to the test phase. It expands the previously selected section in the DD.

Chapter 6: This chapter shows the phases to install the web application and the needed prerequisites.

Chapter 7: This chapter shows the amount of time that each member spent on write down this document.

Chapter 8: This chapter shows the documents and online resources used to produce this document.

2 Implementation Requirement

2.1 Overview

During the implementation of the software, the main focus has been on the main functionalities of the software.
At the current state of development, the system should:

- The **farmer** should be able to register to the platform and then login in it. The farmer can also create the profile concerning his own farm and eventually update it. The farmer can view the calendar of the meetings with the agronomist and change their status. Farmers can also send requests for help to the agronomist, modify, delete them and view the agronomist's answer. The farmer can also view the weather forecasts for the area and the data concerning of the Soil of the farm. Each farmer is also responsible for their production and the sharing of it with the agronomist who is responsible to check on the farmer. The farmer can also view al the Knowledges shared by the agronomist with his farmers and express a judgment on them by liking them or not. Finally, the farmer can post in the Forum with all farmers to share information or ask questions to other farmers, modify created topics and delete them, and comment on topics posted by other farmers.
- The **agronomist** can login into the system and view all the farmers he is responsible for. The agronomist can share Knowledge with the farmers, modify and delete them. The agronomist can plan meetings with the farmers, modify them, close them and delete them. The agronomist has also access to his farmers' production data. Agronomists have the possibility to view all help request sent by the farmers and answer them. Other than that, the agronomist has access to the weather forecasts of the area and can handle the publication of soil data for each farmer's farm. After each meeting, the agronomist can also grade the farmer to keep track of their conditions and improvements.

2.2 Traceability

R.#	Description	Implemented
R.1	The system allows the farmer to add data concerning his production	YES
R.2	The system allows the agronomist to view the data concerning their farmers' production	YES
R.3	The system allows the agronomist to book an appointment with the farmer by showing all available slots	YES
R.4	The system allows the agronomist to modify a booked appointment with the farmer	YES
R.5	The system allows the agronomist to cancel a booked appointment with a farmer	YES
R.6	The system check that the agronomist booked at least two appointments per year with each farmer	YES
R.7	The system alerts the farmer before the appointment	YES
R.8	The system alerts the agronomist before the appointment	YES
R.9	The system allows the agronomist to view the calendar	YES
R.10	The system allows the farmer to view the calendar	YES
R.11	The system allows the farmer to send a help request to the agronomist	YES
R.12	The system allows the agronomist to answer to the help requests that are sent to him	YES
R.13	The system allows the agronomist to view all the help requests sent to him	YES
R.14	The system allows the farmer to view all the help requests sent by him	YES
R.15	The system allows the farmer to publish a topic on the forum	YES
R.16	The system allows all farmers to view the forum	YES
R.17	The system allows all farmers to publish an answer to a topic on the forum	YES
R.18	The system allows the farmers to check the weather forecasts for their area	YES
R.19	The system allows the agronomists to share knowledge with the farmer	YES
R.20	The system allows the farmers to visualize the knowledges shared by their agronomist	YES
R.21	The system allows all agronomists to check the weather forecasts for their area	YES
R.22	The system allows the agronomists to publish a grade for each farmer after each visit	YES
R.23	The system allows the policy makers to view the grades of all farmers of Telangana	NO
R.24	The system requires a sign up and a login to access to the data	YES

R.#	Description	Implemented
R.25	The system shows to each agronomist the list of all farmers they are responsible for	YES
R.26	The system notifies the agronomists when a new farmer has registered under their area	YES
R.27	The system allows the farmer to insert their position when they first register	YES

3 Development Details

3.1 Adopted Development Frameworks

As previously specified in the DD, the system has been implemented using SpringBoot. SpringBoot is an open source, enterprise-level framework which is used to create standalone applications that run on the Java Virtual Machine or JVM.
We chose to use SpringBoot since it allows to develop web applications, by using the Spring Framework, in a much easier and faster way. It offers a "dependency injection" feature that lets objects define their own dependencies and are then later injected into the objects themselves.
Therefore, this mechanism makes it possible to create modular applications made of loosely coupled components that are ideal for micro-services and distributed network applications.
Other than that, Spring Frameworks offers built-in support for some typical tasks that an application needs to perform such as data binding, validation, exception handling, resource and event management. Some of those we have exploited also in the development of our system.
SpringBoot is the perfect tool avoid issues that are met using Spring Framework such as the time and the knowledge required to configure, set up and deploy the applications through auto-configuration, opinionated approach and standalone applications.

1. **Auto-configuration:** the application is initialized with some pre-defined dependencies so that no manual configuration is needed making it possible to save time and avoid possible human errors. It is still possible to override them.
2. **Opinionated approach:** through it, SpringBoot adds and configures the correct starter dependencies without requiring the developer to take all these decisions.
3. **Standalone applications:** applications that do not rely on any external web server by embedding a web server such as Tomcat or Netty during the initialization process. More simply, the application can be run just by pressing the run button.

Another technology we have adopted is "Firebase". Firebase is a Google-backed software used to develop applications that helps in developing not only Web applications as in our case, but also other applications such as IOS or Android applications.
The reason why we choose this hosting system is that it is powered by Google which provides an exceptionally powerful platform.
The front-end part of our system, the Web Applications, have been developed using ReactJS. ReactJS is a open-source JavaScript library used to build User Interfaces (or UI). It is very useful since it allows to design views for each state of the application and then React will take care of updating them in a efficient way and rendering just the right components when data change. ReactJS has the peculiarity of being component-based. Thus, it is made of different encapsulated components managing their own state, then composed together to result in the final UI by ReactJs. Being a component-based approach, ReactJS is very simple but it still guarantees high performance and easy testability.
Finally, the system has been developed in Java because of its many advantages that make it suitable for any programming task such as its robustness, ease of use, cross-platform capabilities and security features. It is also a object-oriented programming language which allows to create modular programs and to eventually reuse the code, and because it is platform independent either at source and binary level. This means that it easy to move from a computer system to another and that it is possible to run the same program on many different systems, fundamental aspects in developing Word Wide Web softwares.

4 Source Code Structure

The source code of the application is divided into two main parts: the back-end and the front-end.
The back-end is divided into seven packages, each of which refers to a specific type of component of the application. In addition there are a couple Java Classes.
A description of the different packages is provided. For what concerns to Controllers, Repositories and Services they are better explained one by one in the DD under the "Component View" section.

1. controller
   In this folder there are all the Controllers which contain all the requests used in the application. We have implemented the following controllers:

   1.1 AgronomistController
   1.2 CommentController
   1.3 FarmController
   1.4 FarmerController
   1.5 KnowledgeController
   1.6 LocationController
   1.7 LoginController
   1.8 MeetingController
   1.9 NotificationController
   1.10 ProductController
   1.11 ProductionController
   1.12 RequestController
   1.13 SoilDataController
   1.14 TopicController

2. entity
   In this folder there are all the entities used by the application:

   2.1 Agronomist
   2.2 Comment
   2.3 Farm
   2.4 Farmer
   2.5 Knowledge
   2.6 LikeComment
   2.7 LikeKnowledge
   2.8 Location
   2.9 Meeting
   2.10 NotificationAgronomist
   2.11 NotificationFarmer
   2.12 Product
   2.13 Production
   2.14 Request

2.15 SoilData

2.16 Topic

3. extraClass
   In this folder there are some external classes we have created to perform some additional functions used in the implementation of the system, respectively

   3.1 Liked: to handle the liking process to Comments and Knowledges.

   3.2 Response: to handle the different responses given by the application as results coming from different methods.

   3.3 Allert: to handle the liking process to Comments and Knowledges.

4. filter
   In this folder there are two filters used to implement the Authentication-Authorization processes:

   4.1 CustomAuthenticationFilter

   4.2 CustomAuthorizationFilter

5. repository
   In this folder there are all the repositories which implement the queries used to fetch the database:

   5.1 AgronomistRepository

   5.2 CommentRepository

   5.3 FarmerRepository

   5.4 FarmRepository

   5.5 KnowledgeRepository

   5.6 LikeCommentRepository

   5.7 LikeKnowledgeRepository

   5.8 LocationRepository

   5.9 MeetingRepository

   5.10 NotificationAgronomistRepository

   5.11 NotificationFarmerRepository

   5.12 ProductionRepository

   5.13 ProductRepository

   5.14 RequestRepository

   5.15 SOilDataRepository

   5.16 TopicRepository

6. security
   In this folder there is the class SecurityConfig which handles the security of all the requests:

   6.1 SecurityConfig

7. service
   In this folder there are the services in which the methods used are implemented:

   7.1 AgronomistService

   7.2 CommentService

   7.3 FarmerService
   7.4 FarmService
   7.5 KnowledgeService
   7.6 LikeCommentService
   7.7 LikeKnowledgeService
   7.8 LocationService
   7.9 LoginService
   7.10 MeetingService
   7.11 NotificationAgronomistService
   7.12 NotificationFarmerService
   7.13 ProductionService
   7.14 ProductService
   7.15 RequestService
   7.16 SoilDataService
   7.17 TopicService

8. DreamApplication
   This Class is used to start the Server for the application.

9. WebConfig
   This Class is used to implement the method to avoid Cors issues during implementation.

The front-end is divided into two main repositories: the Farmer's Web Application and the Agronomist's Web Application.
Agronomist's Web Application:

1. commons
   All the main functions used by the rest of the web app.

   1.1 apiClient.js
   1.2 appGlobal.js

2. components
   All React component used in the web application, such as footer, header and the body of pages.

   2.1 footer
   2.2 header
   2.3 pages
   2.4 RoutesAgronomist.js

3. App.js

4. App.test.js

5. index.js

6. reportWebVitals.js

7. setupTests.js

Farmer's Web Application: Agronomist's Web Application:

1. commons
   All the main functions used by the rest of the web app.
   1.1 apiClient.js
   1.2 appGlobal.js
   1.3 withRouter.js
2. components
   All React component used in the web application, such as footer, header and the body of pages.
   2.1 footer
   2.2 header
   2.3 pages
   2.4 RoutesFarmer.js
3. images
   3.1 government_logo-png
4. App.js
5. App.test.js
6. index.js
7. reportWebVitals.js
8. setupTests.js

5 Test

5.1 Integration Test

As previously discussed in the DD, the goal is to perform the testing of each component once it gets integrated in the system with all the others already integrated.
Eventually, the whole system will be tested in order to verify the functioning of the whole system all together.
For this purpose, the technology we have used is Postman, an application used fro API testing. Postman is an HTTP client that tests HTTP requests through the use of a graphical interface that provides different responses that need to be subsequently validated.
Postman provides many interaction methods, the most used are: GET to obtain information, POST to add information, PUT to replace information and DELETE to delete information.
Postman was used for both integration and performance testing.

5.1.1 Authorization and authentication test

Please note that we have included only two Login tests (successful and error) because the functions are the same for both the Farmer and the Agronomist. The same thing has been done with the SignUp process.

1. Success Login
   This test simulates a well formed login request with valid credentials that the system is supposed to accept.

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 200 (request correctly received, understood and accepted)	Status Code is 200
Response Format	Checks if the response of the request is in JSON format	Response is in JSON format
Schema Validation	Checks if the format of the JSON and its content are coherent with what excepted	The content and the format are coherent

2. Error Login
   This test simulates a login request performed with invalid credentials that should not be accepted by the system.

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 403 (the server understands the request but refuses to authorize it)	Status Code is 403
Response Format	Checks if the response of the request is empty	Response is empty
Schema Validation	Checks if the format of the response and its content are coherent with what excepted	The content and the format are coherent

3. Success SignUp Farmer
   This test simulates a signUp request well formed that should be accepted.

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 200 (request correctly received, understood and accepted)	Status Code is 200
Response Format	Checks if the response of the request is in Response Format	Response is in Response format
Schema Validation	Checks if the format of the Response and its content are coherent with what excepted	The content and the format are coherent

4. SignUp error
   This test simulates a signUp request not well formed that should not be accepted.

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 400 (Bad request, the server cannot or will not process the request due to a client error)	Status Code is 400
Response Format	Checks if the response of the request is the Response format	Response is in the Response format
Schema Validation	Checks if the format of the response and its content are coherent with what excepted	The content and the format are coherent

5. Success Refresh Token
   This test simulates a refresh Token request well formed that should be accepted.

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 200 (request correctly received, understood and accepted)	Status Code is 200
Response Format	Checks if the response of the request is in JSON Format	Response is in JSON format
Schema Validation	Checks if the format of the Response and its content are coherent with what excepted	The content and the format are coherent

6. Refresh Token error
   This test simulates a refresh Token request not well formed that should not be accepted.

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 403 (the server understands the request but refuses to authorize it)	Status Code is 403
Response Format	Checks if the response of the request is in JSON format	Response is in JSON format
Schema Validation	Checks if the format of the response and its content are coherent with what excepted	The content and the format are coherent

5.1.2 Other tests

Due to the way we have implemented our system, each group of requests in a Controller follows more or less the same structure. In each Controller, we handle a request to get all instances of an Object (GET), a request to extract a specific object (for instance GetLocationById) by passing an Id or another unique identifier (GET), a request to create a new instance of an object (POST), a method to update a given instance of an object (PUT) and, eventually, a request to delete a given instance of an object (DELETE). In order to slim down the document and avoid unnecessary repetitions, we will present only one example for each type of request considering either the successful case and the error generating case. This is possible because we handled all equal errors and successful in the same way.

1. Success GetAllFarmById
   This test simulates a GetAllFarmByLocation request well formed that should be accepted.

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 200 (request correctly received, understood and accepted)	Status Code is 200
Response Format	Checks if the response of the request is in Response Format	Response is in Response format
Schema Validation	Checks if the format of the Response and its content are coherent with what excepted	The content and the format are coherent

2. GetAllFarmById Error
   This test simulates a GetAllFarmByLocation request not well formed that should not be accepted.

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 404 (the server cannot find a response status, possible broken link)	Status Code is 404
Response Format	Checks if the response of the request is in Response format	Response is in Response format
Schema Validation	Checks if the format of the response and its content are coherent with what excepted	The content and the format are coherent

This test captures errors such as a bad formulated request but we also handle another error which refers to the Authentication issue. If a user who is not supposed to perform a specific request tries to to it anyway, the status code of the error would be 403 which means that the server understands the request but refuses to authorize it.
Here an example follows:

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 403 (the server understands the request but refuses to authorize it)	Status Code is 403
Response Format	Checks if the response of the request is in JSON format	Response is in JSON format
Schema Validation	Checks if the format of the response and its content are coherent with what excepted	The content and the format are coherent

3. Success GetAllLocation
This test simulates a GetLocationById well formed that should be accepted.

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 200 (request correctly received, understood and accepted)	Status Code is 200
Response Format	Checks if the response of the request is in Response Format	Response is in Response format
Schema Validation	Checks if the format of the Response and its content are coherent with what excepted	The content and the format are coherent

4. GetAllLocation Error
This test simulates a GetLocationById request not well formed that should not be accepted.

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 404 (the server cannot find a response status, possible broken link)	Status Code is 404
Response Format	Checks if the response of the request is in Response format	Response is in Response format
Schema Validation	Checks if the format of the response and its content are coherent with what excepted	The content and the format are coherent

This test captures errors such as a bad formulated request but we also handle another error which refers to the Authentication issue. If a user who is not supposed to perform a specific request tries to to it anyway, the status code of the error would be 403 which means that the server understands the request but refuses to authorize it.
Here an example follows:

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 403 (the server understands the request but refuses to authorize it)	Status Code is 403
Response Format	Checks if the response of the request is in JSON format	Response is in JSON format
Schema Validation	Checks if the format of the response and its content are coherent with what excepted	The content and the format are coherent

5. Success PostLocation
   This test simulates a PostLocation request well formed that should be accepted.

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 200 (request correctly received, understood and accepted)	Status Code is 200
Response Format	Checks if the response of the request is in Response Format	Response is in Response format
Schema Validation	Checks if the format of the Response and its content are coherent with what excepted	The content and the format are coherent

6. PostLocation error
   This test simulates a PostLocation request not well formed that should not be accepted.

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 404 (the server cannot find a response status, possible broken link)	Status Code is 404
Response Format	Checks if the response of the request is in Response format	Response is in Response format
Schema Validation	Checks if the format of the response and its content are coherent with what excepted	The content and the format are coherent

   This test captures errors such as a bad formulated request but we also handle another error which refers to the Authentication issue. If a user who is not supposed to perform a specific request tries to to it anyway, the status code of the error would be 403 which means that the server understands the request but refuses to authorize it.
   Here an example follows:

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 403 (the server understands the request but refuses to authorize it)	Status Code is 403
Response Format	Checks if the response of the request is in JSON format	Response is in JSON format
Schema Validation	Checks if the format of the response and its content are coherent with what excepted	The content and the format are coherent

7. Success PutAgronomist
This test simulates a PutLocation request well formed that should be accepted.

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 200 (request correctly received, understood and accepted)	Status Code is 200
Response Format	Checks if the response of the request is in Response Format	Response is in Response format
Schema Validation	Checks if the format of the Response and its content are coherent with what excepted	The content and the format are coherent

8. PutAgronomist Error
This test simulates a PutLocation request not well formed that should not be accepted.

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 404 (the server cannot find a response status, possible broken link)	Status Code is 404
Response Format	Checks if the response of the request is in Response format	Response is in Response format
Schema Validation	Checks if the format of the response and its content are coherent with what excepted	The content and the format are coherent

This test captures errors such as a bad formulated request but we also handle another error which refers to the Authentication issue. If a user who is not supposed to perform a specific request tries to to it anyway, the status code of the error would be 403 which means that the server understands the request but refuses to authorize it.
Here an example follows:

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 403 (the server understands the request but refuses to authorize it)	Status Code is 403
Response Format	Checks if the response of the request is in JSON format	Response is in JSON format
Schema Validation	Checks if the format of the response and its content are coherent with what excepted	The content and the format are coherent

9. Success DeleteMeeting
This test simulates a DeleteMeeting request well formed that should be accepted.

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 200 (request correctly received, understood and accepted)	Status Code is 200
Response Format	Checks if the response of the request is in Response Format	Response is in Response format
Schema Validation	Checks if the format of the Response and its content are coherent with what excepted	The content and the format are coherent

10. DeleteMeeting error
This test simulates a DeleteMeeting request not well formed that should not be accepted.

Test	Description	Expected Result
Status Code	Checks if the response status of the HTTP request is equal to 404 (the server cannot find a response status, possible broken link)	Status Code is 404
Response Format	Checks if the response of the request is in Response format	Response is in Response format
Schema Validation	Checks if the format of the response and its content are coherent with what excepted	The content and the format are coherent

This test captures errors such as a bad formulated request but we also handle another error which refers to the Authentication issue. If a user who is not supposed to perform a specific request tries to to it anyway, the status code of the error would be 403 which means that the server understands the request but refuses to authorize it.
Here an example follows:

Test	**Description**	**Expected Result**
Status Code	Checks if the response status of the HTTP request is equal to 403 (the server understands the request but refuses to authorize it)	Status Code is 403
Response Format	Checks if the response of the request is in JSON format	Response is in JSON format
Schema Validation	Checks if the format of the response and its content are coherent with what excepted	The content and the format are coherent

5.2 Functional System Test

This phase of the testing has been performed n the whole system to test the behavior of it. For each test, a description of the actions executed during the test is found and also the final expected output is described.
Please, note that only the main functions are depicted.

1. SignUp Farmer
   This test checks the correctness of the SignUp process performed by a new user.

Test	Description	Expected Result
SignUp Success	Signup with correct data	Dialog with a confirmation message
SignUp Error	Signup with alreay present data or wrong data	Dialog with an error message

2. Login This test checks the correctness of the Login process performed by a registered user.

Test	Description	Expected Result
Login Success	Login with valid credentials	Redirect to the HomePage of the user
Login Error	Login with wrong data or incomplete data	Dialog with an error message

3. Farmer Home Activity This test checks the process of accessing all the different sections of the Farmer Home Page.

Test	Description	Expected Result
Home Tab	Click on the HomePage button to check the retrived data and the information thar are shown	The HomePage is correctly shown
Forum Page	Click on the Forum button to access the Forum Page	The Forum Page is correctly shown
Knowledge Page	Click on the Knowledge button to access the Knowledge Page	The Knwoledge Page is correctly shown
Help Page	Click on the Help button to access the Help Request Page	The Halep Page is correctly shown
Production Page	Click on the Production button to access the Production Page	The Production page is correctly shown

4. Agronomist Home Activity
   This test checks the process of accessing all the different sections of the Agronomist Home Page.

Test	Description	Expected Result
Home Tab	Click on the HomePage button to check the retrived data and the information thar are shown	The HomePage is correctly shown
Meeting Page	Click on the Meeting button to access the Meeting Page	The Meeting Page is correctly shown
Knowledge Page	Click on the Knowledge button to access the Knowledge Page	The Knwoledge Page is correctly shown
Help Page	Click on the Help button to access the Help Request Page	The Halep Page is correctly shown
Farmers Page	Click on the Farmers button to access the Farmers Page	The Farmers page is correctly shown

5. Create Meeting This test checks the process of creating a Meeting from the Agronomist's WebApplication.

Test	Description	Expected Result
New Meeting Button	Click on the New Meeting button to access the page	The page to create the Meeting is visualized
Plan Meeting	Select the Farmer to meet and the date to plan the meeting on	The system shows all available times
Create Meeting	Click on the Create Meeting button to create a new meeting	A dialog with a confirmation message is shown

6. Create Knowledge This test checks the process of creating a Knowledge from the Agronomist's WebApplication.

Test	Description	Expected Result
New Knowledge Button	Click on the New Knowledge button to access the page	The page to create the Knowledge is visualized
Write Knowledge	Write the post and fill all required fields	The system provides a form page to perform the actions
Create Knowledge	Click on the Create Knowledge button to publish the Knowledge	A dialog with a confirmation message is shown

7. Create Topic This test checks the process of creating a Topic from the Farmer's WebApplication.

Test	Description	Expected Result
New Topic Button	Click on the New Topic button to access the page	The page to create the Topic is visualized
Write Topic	Write the post and fill all required fields	The system provides a form page to perform the actions
Create Topic	Click on the Create Topic button to publish the Topic	A dialog with a confirmation message is shown

8. Post Comment This test checks the process of posting a Comment to a Topic in the Forum from the Farmer's WebApplication.

Test	Description	Expected Result
New Comment Button	Click on the New Comment button to access the page	The page to create the Comment is visualized
Write Comment	Write the post and fill all required fields	The system provides a form page to perform the actions
Create Comment	Click on the Comment button to publish the Comment	A dialog with a confirmation message is shown

9. Send Help Request This test checks the process of sending a Help Request to the Agronomist from the Farmer's WebApplication.

Test	Description	Expected Result
New Help Request Button	Click on the New Help Request button to access the page	The page to create the Help Request is visualized
Write Help Request	Write the post and fill all required fields	The system provides a form page to perform the actions
Send Help Request	Click on the Send Help Request button to publish the Knowledge	A dialog with a confirmation message is shown

This test checks the process of answering a Help Request sent by a Farmer from the Agronomist's WebApplication.

Test	Description	Expected Result
Answer Help Request Button	Click on the Answer Help Request button to access the page	The page to answer to the request is visualized
Write Answer	Write the answer and fill all required fields	The system provides a form page to perform the actions
Send Answer	Click on the Send button to Send the Answer to the Help Request	A dialog with a confirmation message is shown

5.3 Performance System Test

In this part of the document, the purpose is to give an idea of the performance of the system. This is done by evaluating the most frequent requests in terms of time to be performed: the time elapsing from the moment when the request is called to the moment when the results are available.
The results are obtained exploiting the previously discussed Postman Client.
A summary of the results obtained is provided:

Request	**TIme Elapsed**
Login Success	800ms
Login Error	400ms
SignUp Success	1100ms
SignUp Error	300ms
Success Refresh Token	300ms
Refresh Token Error	100ms
Success GetAllFarmById	550ms
GetAllFarmById Error	100ms
GetAllFarmById Error Auth	100ms
Success GetAllLocation	800ms
GetAllLocation Error	100ms
GetAllLocation Error Auth	100ms
PostLocation Success	750ms
PostLocation Error	100ms
PostLocation Error Auth	100ms
PutAgronomist Success	800ms
PutAgronomist Error	100ms
PutAgronomist Error Auth	100ms
DeleteMeeting Success	600ms
DeleteMeeting Error	100ms
DeleteMeeting Error Auth	100ms

6 Installation Instructions

6.1 Installation Requirement

The installation requires:

- browser web, including Internet Explorer 9 and above,
- available internet connection.

6.2 Installation

6.2.1 Back-end

1. Download the back-end directory from the following link
2. Install an IDE for the Java programming language. It is suggested to use the Ultimate version of Intellij downloadable from the following link
3. Open the newly installed directory in the IDE and perform the automatic dependency installation procedure
4. Install now MySQL Server & Workbench the back-end directory from the following link
5. Start the MySQL Installer and install MySQL Community Server and MySQL Workbench, accepting all the default configurations and executing the steps necessary for installing possibly missing prerequisite packages
6. Default username and password for MySQL has been set as "root". If you choose a different password, enter it in application.properties in "back-end/src/main/resources"
7. Create a schema called "db_dream"
8. In the directory "back-end/src/main/java/com/se2project/dream", open the DreamApplication.java file and run the project
9. If JDK is not already installed, perform the automatic installation procedure in IntelliJ.
10. As a default start up, use "db_initialization.sql" file tu insert some default data in the data base.

6.2.2 Front-end

The two web apps can be reached from the following URLs: `dreamfarmerse2.firebaseapp.com` and `dreamagronomistse2.firebaseapp.com`. To use the agronomist web application there will already be an account with the following username and password. Username: agro@gmail.com; password: password

7 Effort Spent

This section shows the amount of time that each member has spent to produce the document. Please notice that each part of the document is the result of coordinated work. The column Member specifies only the main contributor for each topic but it should not be interpreted as a lack of participation by other team member for that topic.

Topic	Member	Hours
General initial brainstorming and creation of the document and First chapter	Buttiglione, Caffagnini, Faouzi	4
Structure of the second chapter and definition	Buttiglione, Caffagnini, Faouzi	4
Implemented Requirement	Caffagnini	4
Description of the Adopted Development Frameworks and Solutions	Caffagnini	4
Source Code Structure	Buttiglione, Caffagnini	5
Test Cases explanation	Caffagnini	6
Server - Back-End (Authentication & Authorization)	Caffagnini	150
Server - Back-End (Entities & Controllers)	Faouzi	200
Applications Front-End	Buttiglione	200
Installation Instructions	Buttiglione	4

8 References

- Postman -https://www.postman.com
- SpringBoot -https://spring.io
- React -https://reactjs.org

