POLITECNICO
MILANO 1863

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

Design Document

Deliverable:	DD
Title:	Design Document
Authors:	Marco Domenico Buttiglione, Martina Caffagnini, Dounia Faouzi
Version:	2.0
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

List of Figures

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
JWT	JSON Web Tokens
UI	User Interface

1.3.3 Abbreviations

Abbreviation	Description
Gn	n-th Goal
Rn	n-th Requirement
Cn	n-th Component
RFH	Request For Help

1.4 Revision history

Date	Description
20/12/2021	Introduction and Architectural Design : Overview
21/12/2021	Component view and Deployment view
23/12/2021	Fist version of Runtime view
30/12/2021	Implementation and Integration
31/12/2021	Selected architectural styles and patterns
01/01/2022	Requirement traceability
02/01/2022	Mock-ups and User interface design
03/01/2022	Component interfaces
04/01/2022	Other design decision
05/01/2022	Testing
07/01/2022	Revision and corrections
08/01/2022	First release
06/02/2022	Second release: Architectural design, RuntimeView, Component interfaces, Algorithms, Requirements traceability, Implementation and Integration

1.5 Reference Documents

- Slides of the course Software Engineering 2
- Requirement Engineering and Design Project: 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 RASD document. 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, to be considered the main one, includes the Component View, the Deployment View, and the Runtime View. The chapter ends with the presentation of the Component Interfaces and a detailed explanation of all chosen patterns and architectural styles.

Chapter 3: This chapter includes the mock-ups of the several User Interfaces (UI) and their description in considering what was already specified in the RASD document.

Chapter 4: This chapter includes the traceability between the requirements shown in the RASD document and the components presented in this document.

Chapter 5: This last chapter includes the plan for the implementation, integration, and test phases. The strategies embraced in order to achieve these goals are explained.

Chapter 6: This chapter shows the amount of time that each member spent on write down this document.

Chapter 7: This chapter shows the documents and online resources used to produce this document.

2 Architectural Design

2.1 Overview

The SpringBoot Architecture has been chosen for the system since it seems to be the most appropriate to model the system. The SpringBoot Architecture is made of four layers and it is structured as follows:

- **Presentation layer:** The presentation layer provides the interaction with the user through the UI and the appropriate communication layers. The purpose of this layer is to either show and acquire information respectively shown to the user and used by the system.
- **Business Logic Layer:** Also known as Application layer, the Business Logic Layer is used to add, modify, and delete data in the data layer. Also, it is used to process information obtained from the presentation layer through some well defined business rules. Lastly, it handles all the application's functionalities.
- **Persistence Layer:** The persistence layer contains all the storage logic and translates business objects from and to database rows.
- **DataBase Layer:** The database layer is responsible for the CRUD operations: create, retrieve, update and delete operations. This layer is simply the actual database used to build the application.

It is important to highlight how the communication between presentation layer and persistence layer can occur only through the business layer.
In the following schema the architecture of th system is represented.

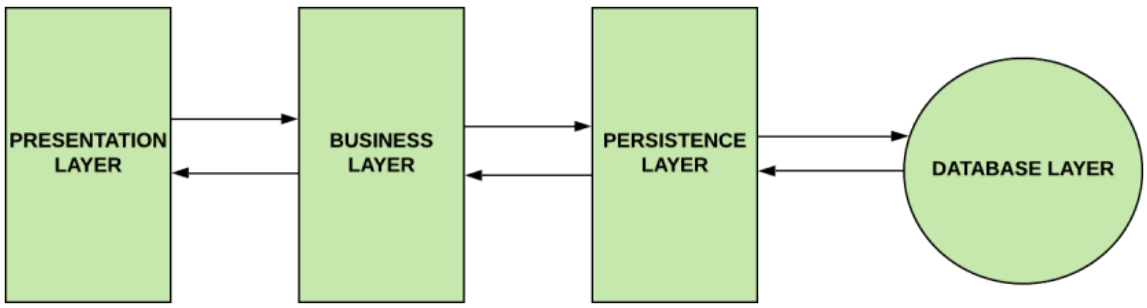


Figure 1: SpringBoot Architecture Diagram

For what concerns our system, the following representation better represents the architecture:

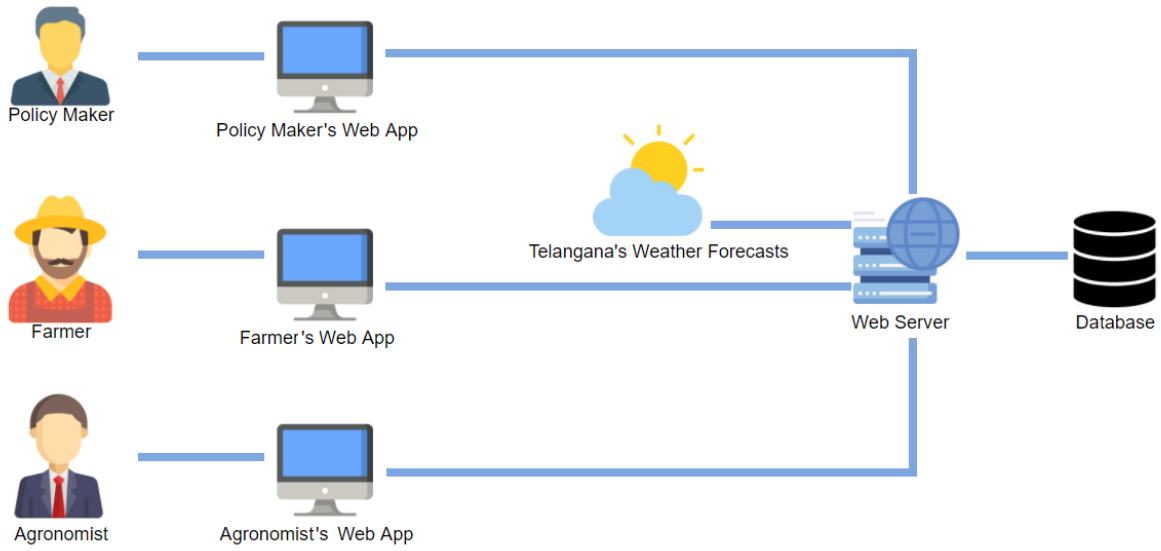


Figure 2: Architecture Diagram

The architecture previously presented provides many advantages. Most importantly, the following strenghts have been considered:

- logical and physical separation of functionalities;
- execution of each tier on a dedicated operating system and on a dedicated virtual hardware component;
- high scalability: the separation of the tiers makes it possible to scale each tier independently to the others;
- high reliability: it is very improbable to that the failure on a specific tier has an impact on the others;
- high security: the separation of the tiers makes it very improbable to undergo any malicious attacks also considering that the presentation and data tier are not directly connected;
- the development time is reduced due to the independence of each tier on the others.

2.2 Component view

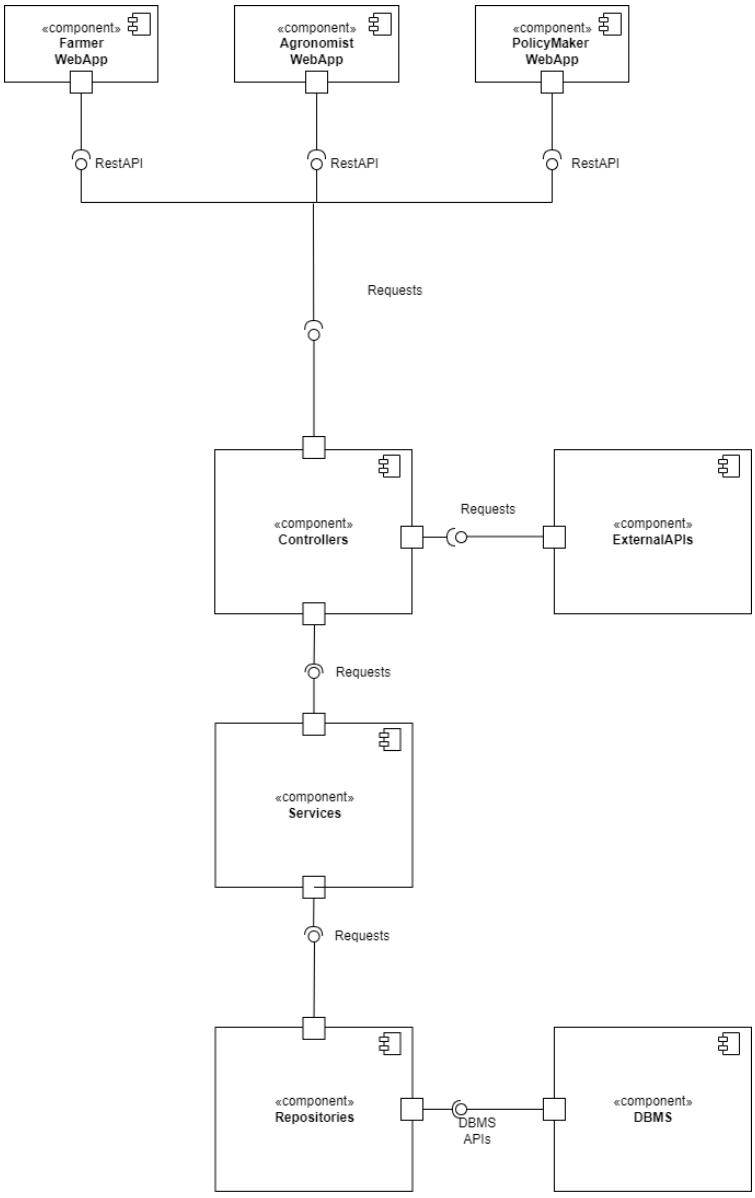


Figure 3: Component Diagram

The Component Diagrams shows the structure of the software system, which describes the software main components, their interfaces, and their dependencies. It shows also how some components are wired together to create larger components. It allows to specify the software components needed before having them developed and implemented. The component diagram provides a view of components and makes it easier to design, develop, and maintain components.
We decided to only include one figure in the diagram for all Controllers, Services, Repositories and External APIs instead of showing all of them to make the diagram more clear and easier to read. The depicted components are:

- **Farmer WebApp** : web application used by the farmer to access the system and its functionalities.
- **Agronomist WebApp**: web application used by the agronomist to access the system and its functionalities.

- **PolicyMaker WebApp**: web application used by the PM to access the system and its functionalities.
- **AgronomistController**: defines all the requests to get all or a specified agronomist by location, id or Aadhaar, to update an existing agroomist and to allow to signUp an agronomist.
- **CommentController**: defines all the requests necessary to create, delete, update, like, unlike and select a specified comment or a list of comments.
- **FarmController**: defines all the requests used to create, update and select a farm or more farms.
- **LocationController**: defines requests that are used to get all locations present, to get a specific location given the Id, to update existing locations and to create a new location.
- **LoginController**: checks if the users that use the application are correctly authenticated. it manages all the logic inherent to the authentication of the customers. It interacts with the DBMS, passing thought the Model, to check that the authentication parameters match the stored ones. It is also responsible for the process of refresh of the token obtained at login.
- **ProductController**: defines all the requests necessary to select a specified product or list of products, to create a new product, to update a product and to delete a selected product.
- **ProductionController**: defines all the requests necessary to select a specified production or list of productions, to create a new production, to update a production and to delete a selected production.
- **SoilDataController**: defines the requests used to retrieve soil data concerning the farm (such as humidity, temperature, and water consumption), to update them, to delete them and to create them.
- **TopicController**: defines requests that view, create, comment, and delete a Topic.
- **KnowledgeController**: defines requests that allow agronomists to create, update, and delete a Knowledge. It also allow the farmer to view the Knowledge that better suit him.
- **RequestController**: defines requests that allow the farmer to send a request for help and view the answer and leave a feedback to the agronomist. It also allow agronomist to view the request from local farmer and response to them.
- **MeetingController**: defines requests that allow agronomists to create, change, and delete a Meeting with a farmer. It allow to the farmer to view and confirm meetings with local agronomist. Lastly, it's responsible for managing the controls about the amount of time that an agronomist meets local farmer, as they should meet twice in a year.
- **FarmerController**: defines requests that allow Farmer to view their data about Production and FarmSoil. This component manage the Production data with request that allow farmers to insert, update and delete production data, it also show them to agronomist and policy maker. On the other side, it allow agronomist to insert, update and delete SoilData. Lastly, it allows to all the authorized customers to view the data from the sensor installed and the data inserted by the agronomist.
- **NotificationController**: define request that allow to send a notification to inform about something a costumer.
- **AgronomistService**: contains and implements all the methods called by the AgronomistController.
- **CommentService**: contains and implements all the methods called by the CommentController.
- **FarmerService**: contains and implements all the methods called by the FarmerController.

- **FarmService**: contains and implements all the methods called by the FarmController.
- **KnowledgeService**: contains and implements all the methods called by the KnowledgeController.
- **LikeCommentService**: contains and implements all the methods called by the LikeCommentController.
- **LikeKnowledgeService**: contains and implements all the methods called by the LikeKnowledgeController.
- **LocationService**: contains and implements all the methods called by the LocationController.
- **LoginService**: contains and implements all the methods called by the LoginController.
- **MeetingService**: contains and implements all the methods called by MeetingController.
- **NotificationAgronomistService**: contains and implements all the methods called by the NotificationAgronomistController,
- **NotificationFarmerService**: contains and implements all the methods called by the NotificationFarmerController.
- **ProductionService**: contains and implements all the methods called by the ProductionController.
- **ProductService**: contains and implements all the methods called by the ProductController.
- **RequestService**: contains and implements all the methods called by the RequestController.
- **SoilDataService**: contains and implements all the methods called by the SoilDataController.
- **TopicService**: contains and implements all the methods called by the TopicController.
- **AgronomistRepository**: defines the queries to retrieve data from the DataBase necessary to perform the methods implemented in the corresponding AgronomistService.
- **CommentRepository**: defines the queries to retrieve data from the DataBase necessary to perform the methods implemented in the corresponding CommentService.
- **FarmerRepository**: defines the queries to retrieve data from the DataBase necessary to perform the methods implemented in the corresponding FarmerService.
- **FarmRepository**: defines the queries to retrieve data from the DataBase necessary to perform the methods implemented in the corresponding FarmService.
- **KnowledgeRepository**: defines the queries to retrieve data from the DataBase necessary to perform the methods implemented in the corresponding KnowledgeService.
- **LikeCommentRepository**: defines the queries to retrieve data from the DataBase necessary to perform the methods implemented in the corresponding LikeCommentService.
- **LikeKnowledgeRepository**: defines the queries to retrieve data from the DataBase necessary to perform the methods implemented in the corresponding LikeKnowledgeService.
- **LocationRepository**: defines the queries to retrieve data from the DataBase necessary to perform the methods implemented in the corresponding LocationService.
- **MeetingRepository**: defines the queries to retrieve data from the DataBase necessary to perform the methods implemented in the corresponding MeetingService.

- **NotificationAgronomistRepository**: defines the queries to retrieve data from the DataBase necessary to perform the methods implemented in the corresponding NotificationAgronomistService
- **NotificationFarmerRepository**: defines the queries to retrieve data from the DataBase necessary to perform the methods implemented in the corresponding NotificationFarmerService.
- **ProductionRepository**: defines the queries to retrieve data from the DataBase necessary to perform the methods implemented in the corresponding ProductionService.
- **ProductRepository**: defines the queries to retrieve data from the DataBase necessary to perform the methods implemented in the corresponding ProductService.
- **RequestRepository**: defines the queries to retrieve data from the DataBase necessary to perform the methods implemented in the corresponding RequestService.
- **SoilDataRepository**: defines the queries to retrieve data from the DataBase necessary to perform the methods implemented in the corresponding SoilDataService.
- **TopicRepository**: defines the queries to retrieve data from the DataBase necessary to perform the methods implemented in the corresponding TopicService.
- **DBMS**: manages the creation, manipulation, and querying of the database.

Furthermore, the system uses some external APIs. In particular:

- **Sensors Service**: it is used to acquire soil data such as humidity and water consumption.
- **Weather Service**: helps to acquire local weather. This information is used by the farmer to improve their resilience, and by the agronomist to help farmers with Knowledge regarding a specific weather condition.

2.3 Deployment view

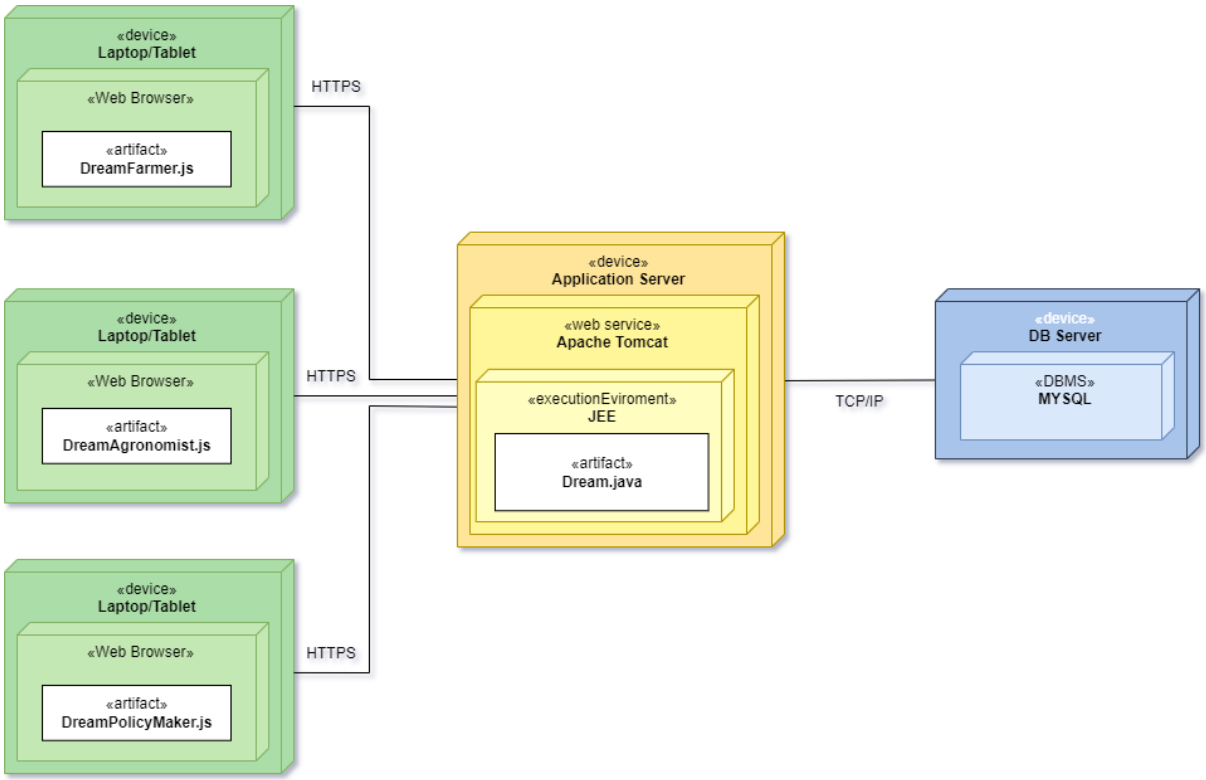


Figure 4: Deployment View

The deployment diagram shows the execution architecture of a system, including nodes such as hardware or software execution environments, and the middleware connecting them. It allow to visualize the physical hardware and software of a system to understand how the system will be physically deployed on the hardware. In the figure are depicted, respectively, the components of the presentation tier (the green ones), the component of the application tier (the yellow one), and the component of the data tier (the blue one).

- **Laptop/Tablet**: are the suggested devices to use, due to the presence of numerous elements on the screen. The only constraint is to have an adequate internet connection.
- **Application Server**: it contains most of the application logic of the system.
- **DBMS**: MySQL. It is a Relational Database Management System (RDBMS) with high availability and performance, and good scalability. It allows dealing with the common CRUD (Create Read Update Delete) operations.

2.4 Runtime view

1. Login for Farmer

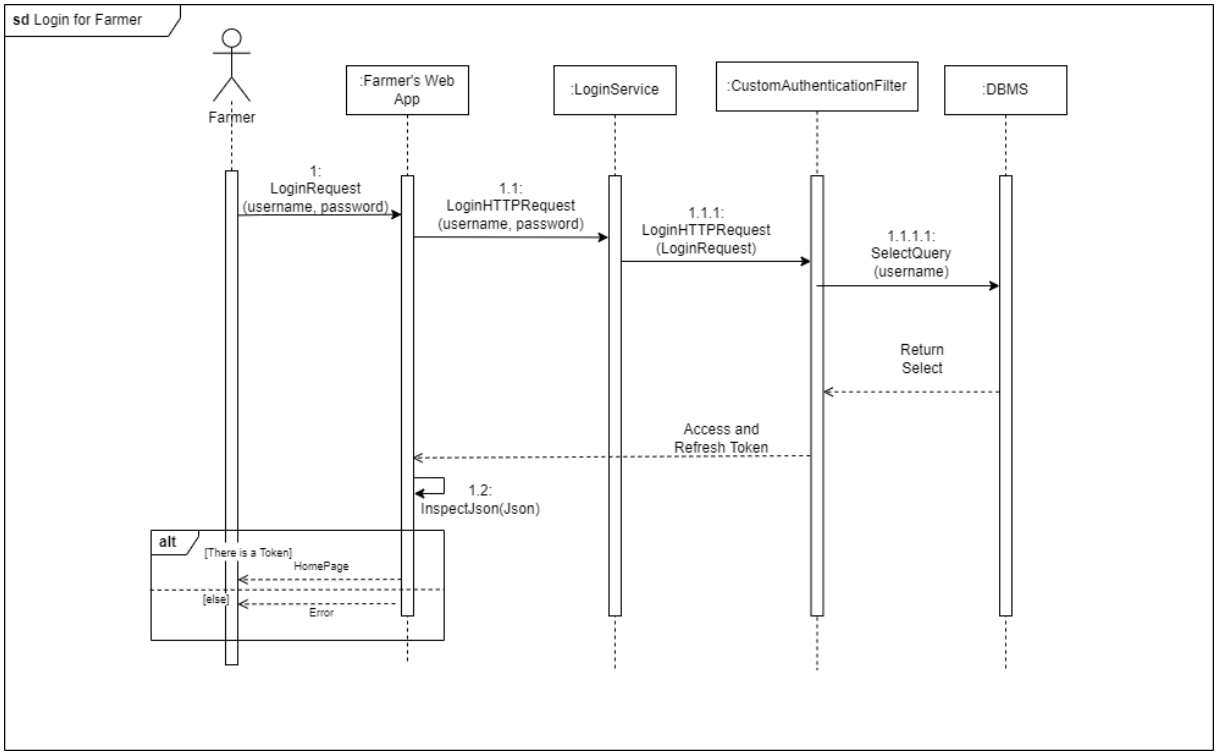


Figure 5: Runtime View - Login Farmer

The sequence diagram just shown represents the login process for any Farmer.
First, the Farmer inserts the credentials chosen during the SignUp process and these are sent, through the Farmer's Web Application, to be authenticated.
The WebApplication sends a POST HTTP request to the LoginService that is passed to the CustomAuthenticationFilter. Here, the authentication is attempted by verifying the credential the Farmer passed in the DBMS. If the result is positive, a pair of tokens (access and refresh) are created by the CustomAuthenticationFilter and are then sent through a JSON file to the WebApp. Otherwise, a JSON containing an error is sent back to the WebApp.

2. Login for Agronomist

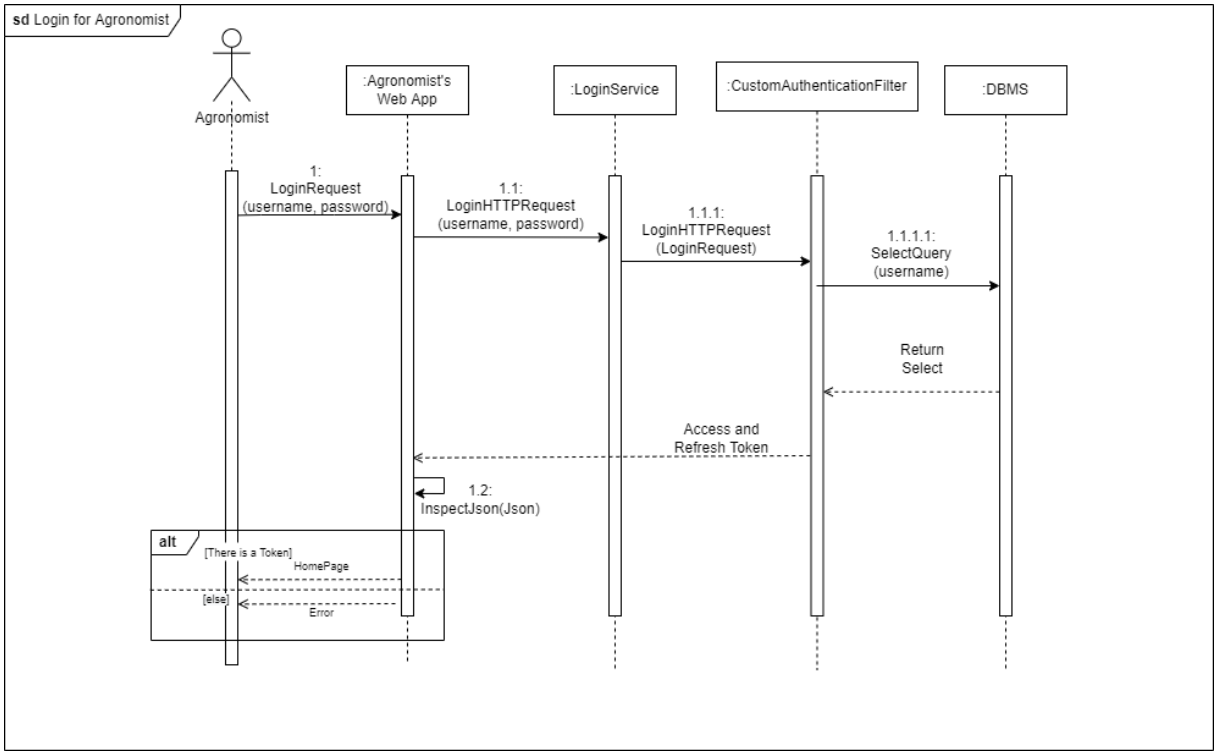


Figure 6: Runtime View - Login Agronomist

The sequence diagram just shown represents the login process for any Agronomist.
First, the Agronomist inserts the credentials chosen during the SignUp process and these are sent, through the Agronomist's Web Application, to be authenticated.
The WebApplication sends a POST HTTP request to the LoginService that is passed to the CustomAuthenticationFilter. Here, the authentication is attempted by verifying the credential the Agronomist passed in the DBMS. If the result is positive, a pair of tokens (access and refresh) are created by the CustomAuthenticationFilter and are then sent through a JSON file to the WebApp. Otherwise, a JSON containing an error is sent back to the WebApp.

3. Login for Policy Maker

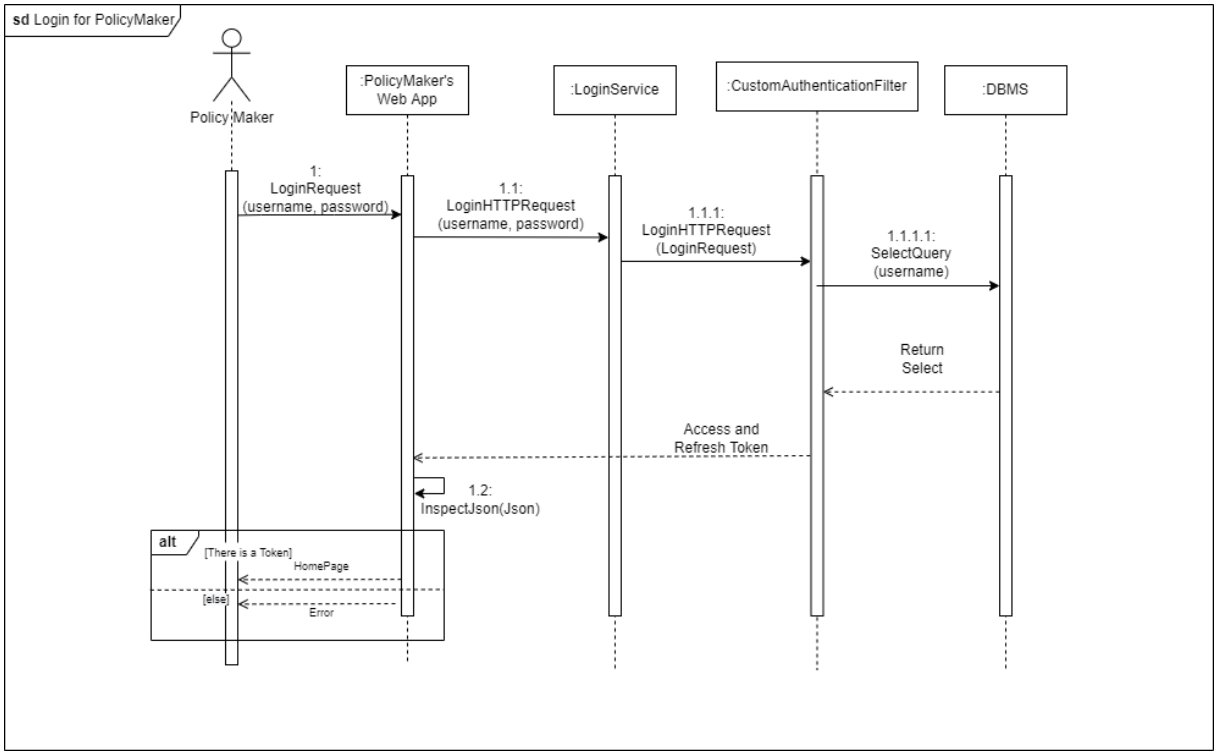


Figure 7: Runtime View - Login Policy Maker

The sequence diagram just shown represents the login process for any Policy Maker.
First, the Policy Maker inserts the credentials chosen during the SignUp process and these are sent, through the Policy Maker's Web Application, to be authenticated.
The WebApplication sends a POST HTTP request to the LoginService that is passed to the CustomAuthenticationFilter. Here, the authentication is attempted by verifying the credential the Policy Maker passed in the DBMS. If the result is positive, a pair of tokens (access and refresh) are created by the CustomAuthenticationFilter and are then sent through a JSON file to the WebApp. Otherwise, a JSON containing an error is sent back to the WebApp.

4. SignUp for Farmer

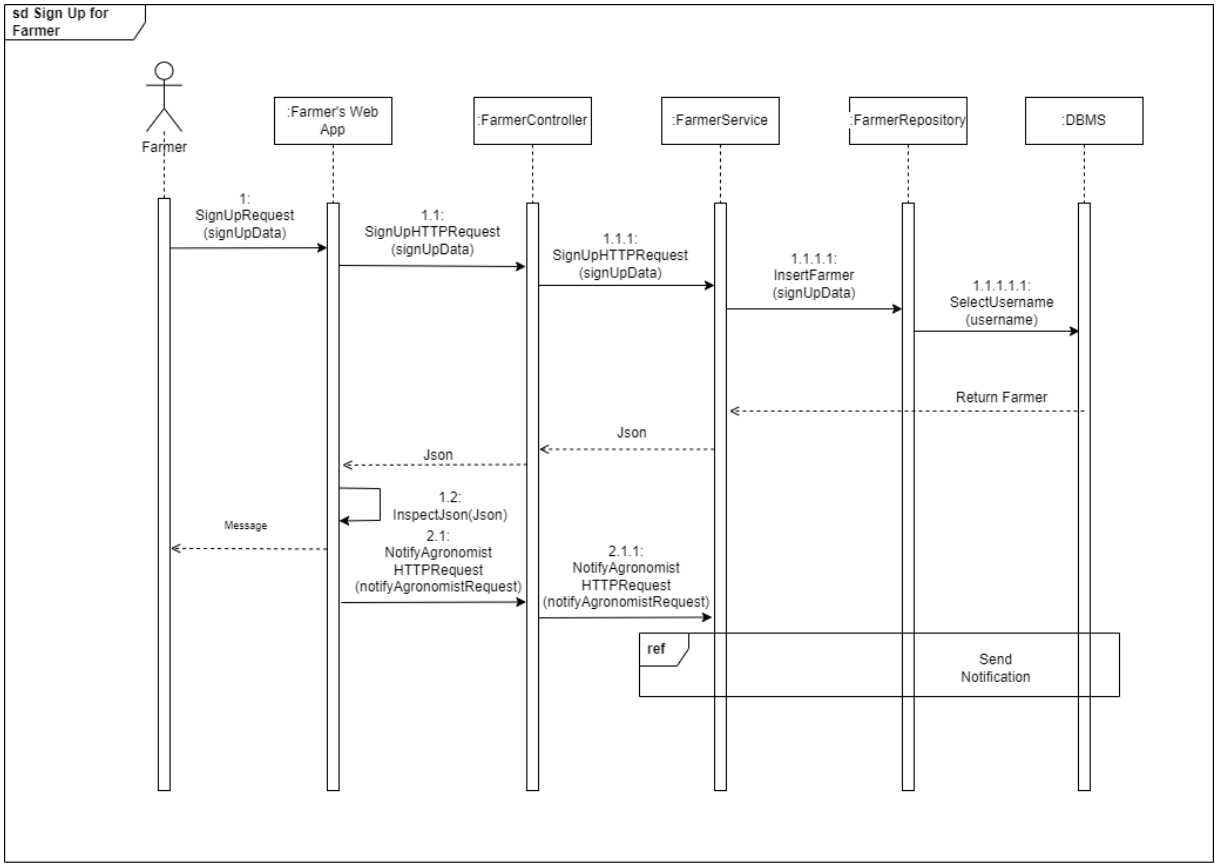


Figure 8: Runtime View - SignUp

The sequence diagram just shown represents the SignUp process for any Farmer.
It is important to highlight how only the Farmer is thought to be able to sign up through the system. Indeed, either the Agronomists and the Policy Makers, being employees, are supposed to be given the credentials to access the system once they are employed. This is the reason why there are only login sequence diagrams concerning Agronomists and Policy Makers.
First, the Farmer inserts the data to sign up through their web application, then they go on with the registration by requesting the registration.
Then, the application sends a POST HTTP request that is sent through the FarmerController. The FarmerService exploiting the Farmer Repository queries the DBMS.
The DBMS, if the SignUp is successful, answers back with a JSON file containing the new Farmer just registered and this message is sent to the Farmer's WebApp through the FarmerController. If the data inserted by the farmer are not valid, an error message is put in the JSON file and then it is shown to the farmer.
Once the registration of the Farmer is successful, a Notification is sent to the corresponding Farmer.

5. Check Token

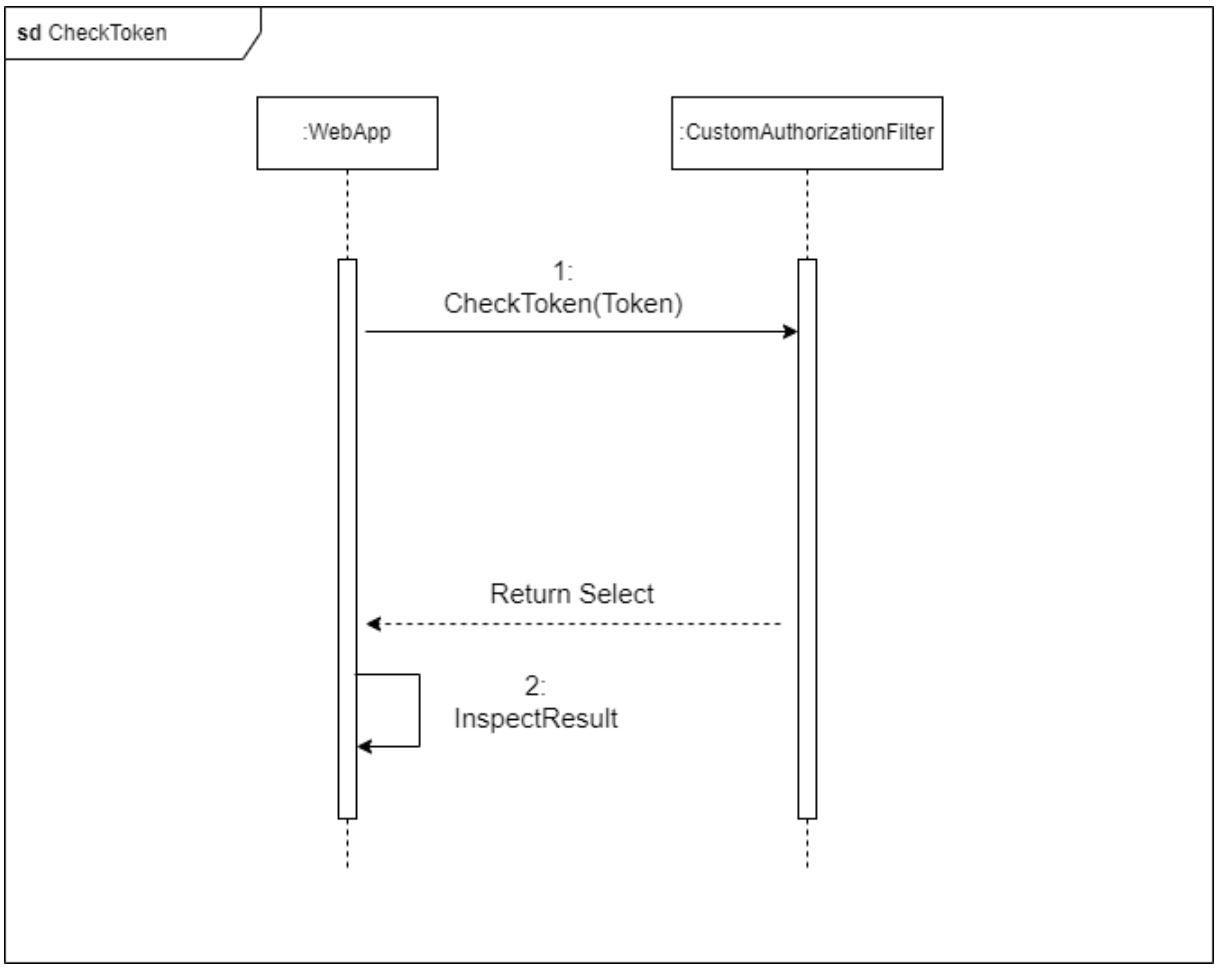


Figure 9: Runtime View - Check Token

The next diagrams will utilize this sequence. The graphic depicts the process of verifying the token in order to approve the operations requested by the users. When the WebApp sends any HTTP request (except for the Login, SignUp and RefreshToken) to the Controller, the system checks if the Token of the user allows to perform that specific HTTP request sent. This is done to Authorize user and deny access to those who do not have the permit to do it. The Token that comes with the request is sent to the CustomAuthorizationFilter that de-crypts it and verifies the authorities of the user. The result is then sent back to the user.

6. Show Farmer Home Page

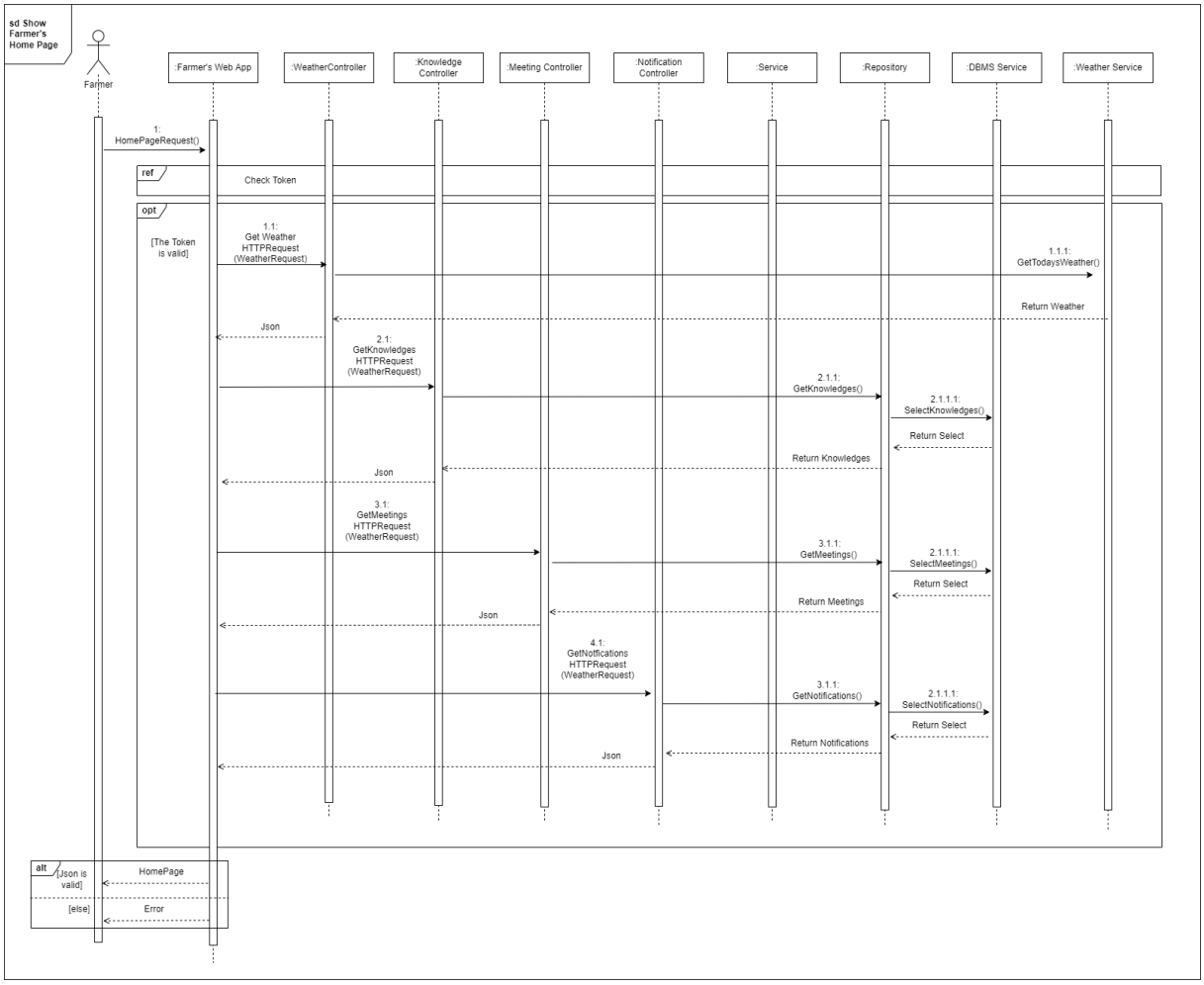


Figure 10: Runtime View - Show Farmer's Home Page

The sequence diagram just shown represents the process that allows to visualize the Farmer's Home Page and the information contained in it.
The Farmer selects the Home Page's section in the Farmer's Web Application. As previously specified, the Home Page contains the weather forecasts, the knowledges shared by the agronomist, the notifications received by the farmer, and the list with all upcoming meetings with the agronomist.
Before sending the HTTP requests to the corresponding controller, the authentication token is checked. If the token is valid the Web application send first a GET HTTP Request to the weather controller witch forward the request to the External Weather Service that return the weather to the Weather Controller that serialize the data in a JSON file. Then a sequence of GET HTTP Request are sent by the Web App to the corresponding controller (Knowledge, Meeting and Notification). The controllers then forward the request to the corresponding Service that will call the find method of the Repository. The Repository will send a select query to the DBMS Service that will perform it on the database. The query results then is brought back to the Controllers and finally serialized in a JSON file, brought back to the Farmer's Web Application.
The JSON file then reaches the Farmer's Web Application where it is checked. The Farmer's Web Application then shows the Home Page. If the token is not valid, the Web App does not forward the request. Then, an error message is serialized in the JSON file, sent to the Farmer's Web Application and eventually the application displays an error message instead of the Home Page.

7. Show Agronomist Home Page

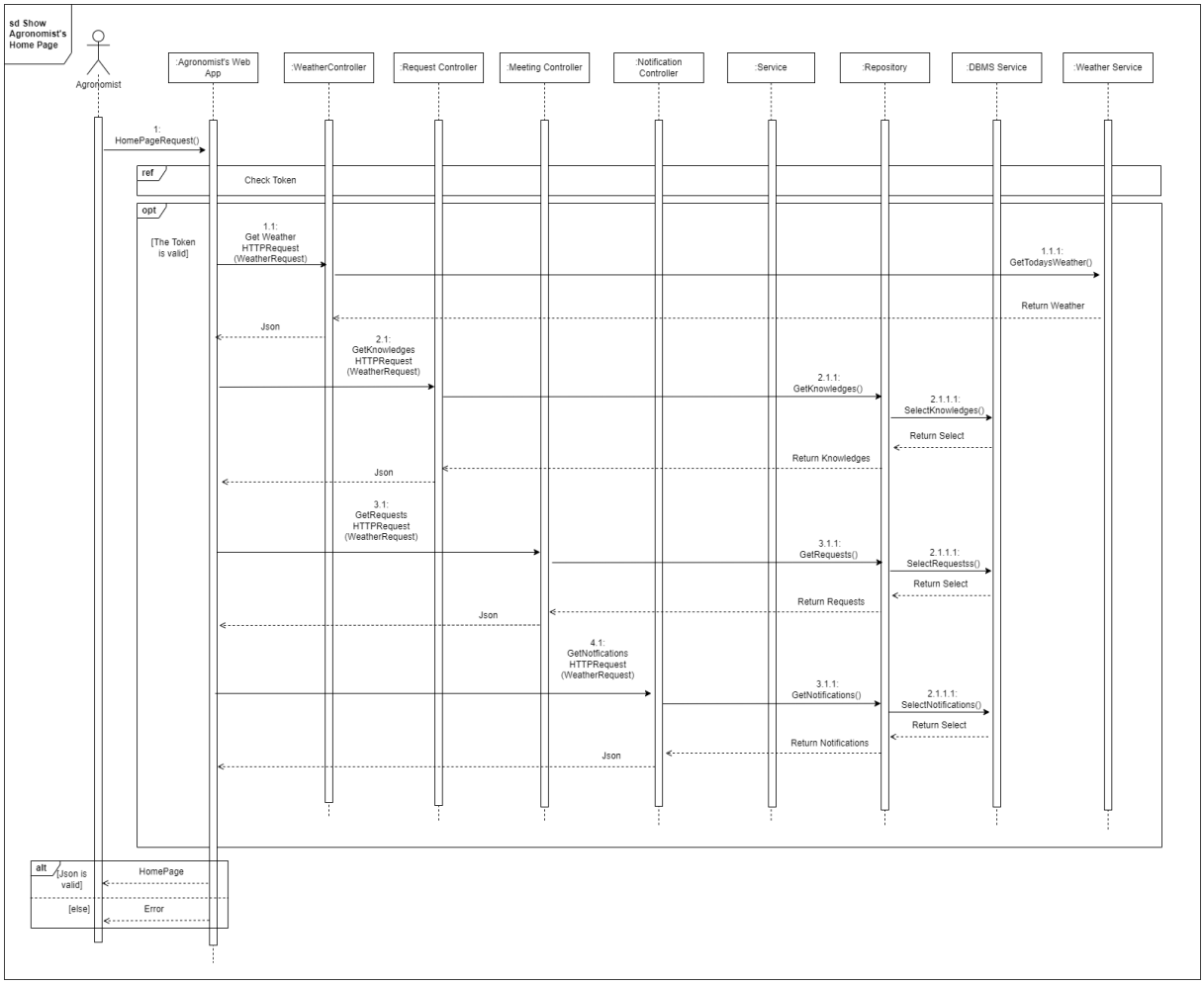


Figure 11: Runtime View - Show Agronomist's Home Page

The sequence diagram just shown represents the process that allows to visualize the Agronomist's Home Page and the information contained in it.
The Agronomist selects the Home Page's section in the Agronomist's Web Application. As previously specified, the Home Page contains the weather forecasts, the request received from farmers, the notifications received by the farmer, and the list with all upcoming meetings with the agronomist.
Before sending the HTTP requests to the corresponding controller, the authentication token is checked. If the token is valid the Web application send first a GET HTTP Request to the weather controller witch forward the request to the External Weather Service that return the weather to the Weather Controller that serialize the data in a JSON file. Then a sequence of GET HTTP Request are sent by the Web App to the corresponding controller (Request, Meeting and Notification). The controllers then forward the request to the corresponding Service that will call the find method of the Repository. The Repository will send a select query to the DBMS Service that will perform it on the database. The query results then is brought back to the Controllers and finally serialized in a JSON file, brought back to the Agronomist's Web Application.
The JSON file then reaches the Agronomist's Web Application where it is checked. The Agronomist's Web Application then shows the Home Page. If the token is not valid, the Web App does not forward the request. Then, an error message is serialized in the JSON file, sent to the Agronomist's Web Application and eventually the application displays an error message instead of the Home Page.

8. Show Policy Maker Home Page

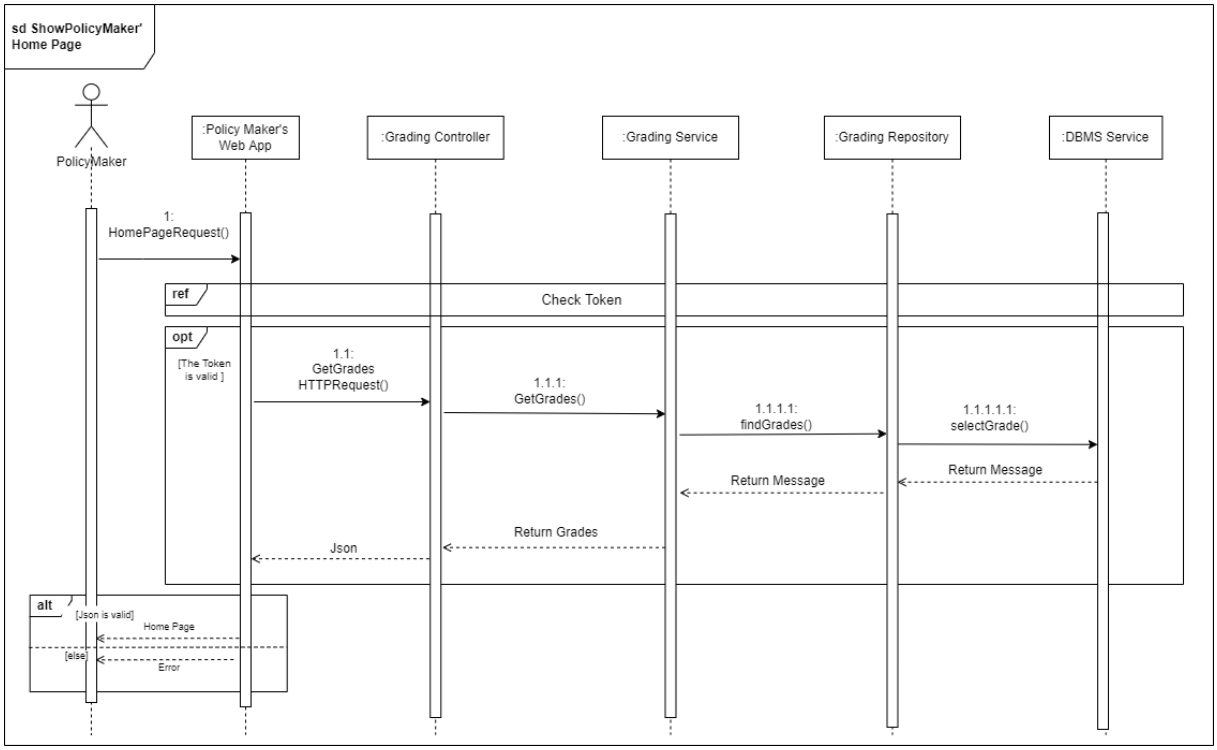


Figure 12: Runtime View - Show Policy Maker's Home Page

The sequence diagram just shown represents the process that allows to visualize the Policy Maker's Home Page and the information contained in it.
The Policy Maker selects the Home Page's section in the Policy Maker's Web Application. As previously specified, the Home Page contains the list with all farmers in the state of Telangana and the correspondent grades given to them by the agronomists.
First, the token is checked and if valid the GET HTTP request is passed to the Grading Service passing thought the Grading Controller.
The Grading service then send a find request to the Grading Repository that forward a select query to the DBMS Service where is performed on the database. The query result then is brought back to the Grading Controller. Finally it is serialized in a JSON file, brought back to the the Policy Maker's Web Application.
If the token is not valid, an error message is serialized and the request is not propagated. The Policy Maker's Web Application will show an error message as well.

9. Check Data

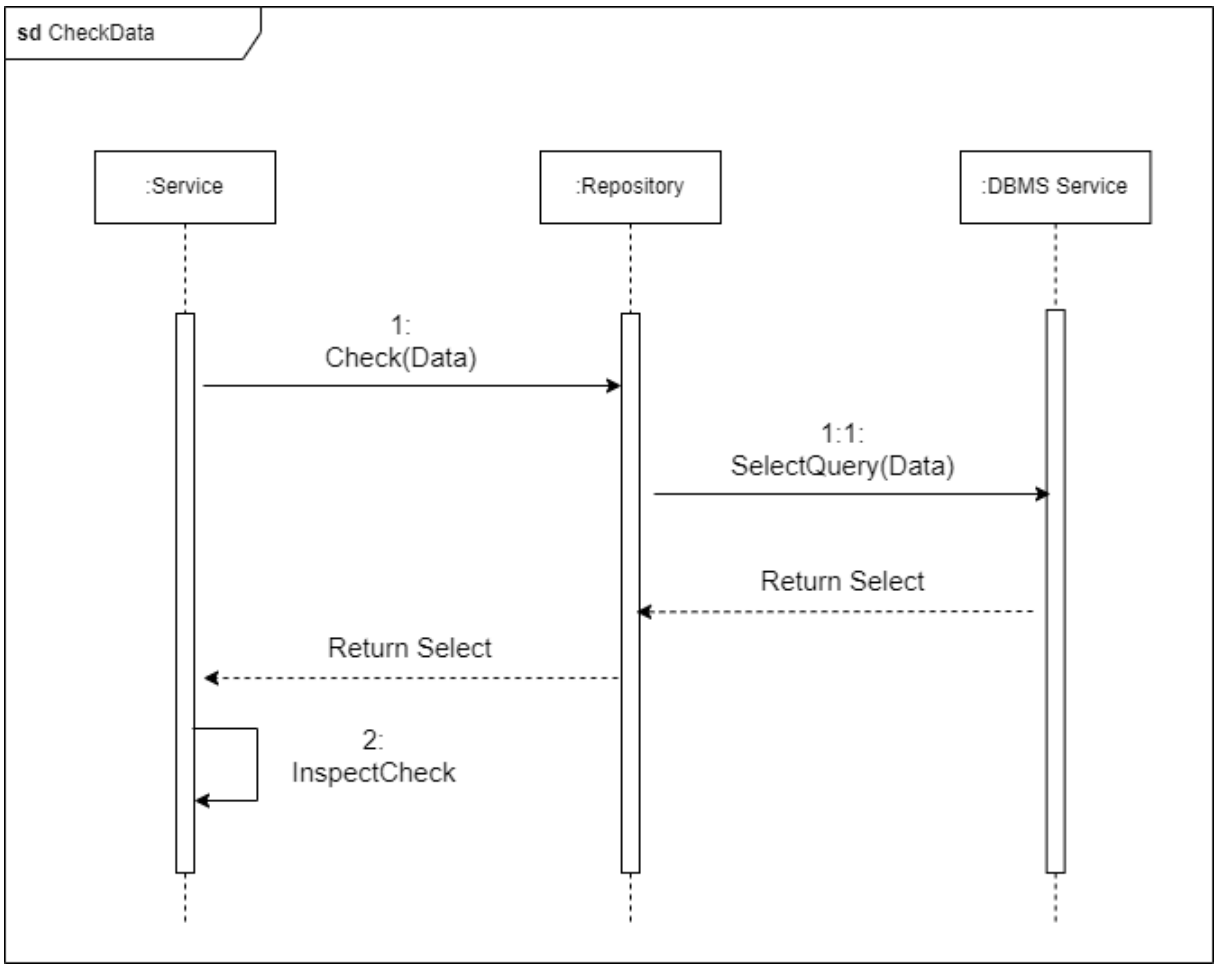


Figure 13: Runtime View - Check Data

The next diagrams will utilize this sequence. The graphic depicts the process of verifying the the input data in order to continue the insert operations requested by the users. First, a component of the business layer (service component) send a get request to evaluate the data (for example if there's any duplicate) to the corresponding Repository. This component will then send an select query that will be executed in the DBMS Service. The result arrives at the service component, that will examine the results to decide if the request by the user can be transmitted or it should appear an error message.

10. Send Notification Farmer

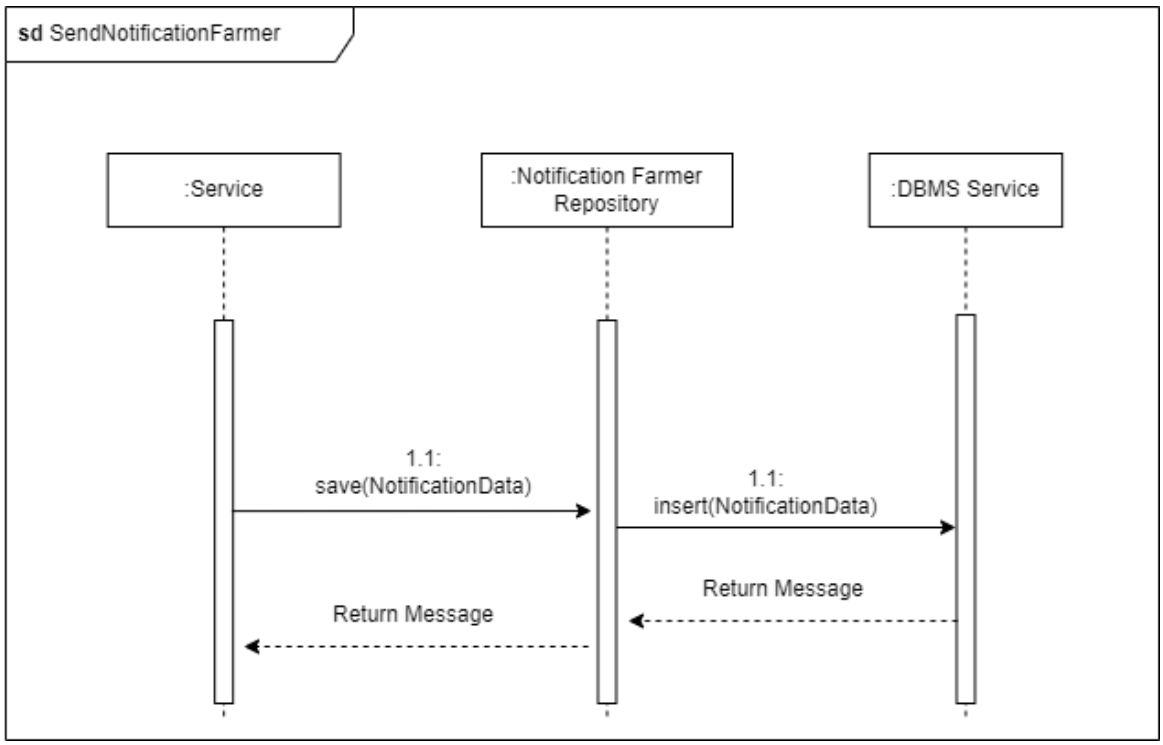


Figure 14: Runtime View - Send Notification Farmer

The next diagrams will utilize this sequence. The graphic describe the process of sending a notification to a farmer of the application. First, a component of the business layer (service component) forward the notification data, the farmer that should will receive it and the notification text, to the Notification Farmer Repository. This component will then send an insert query that will be executed in the DBMS Service. The result arrives at the service component, that will continue the previously interrupted process.

11. Send Notification Agronomist

The next diagrams will utilize this sequence. The graphic describe the process of sending a notification to a agronomist of the application. First, a component of the business layer (service component) forward the notification data, the agronomist that should will receive it and the notification text, to the Notification Agronomist Repository. This component will then send an insert query that will be executed in the DBMS Service. The result arrives at the service component, that will continue the previously interrupted process.

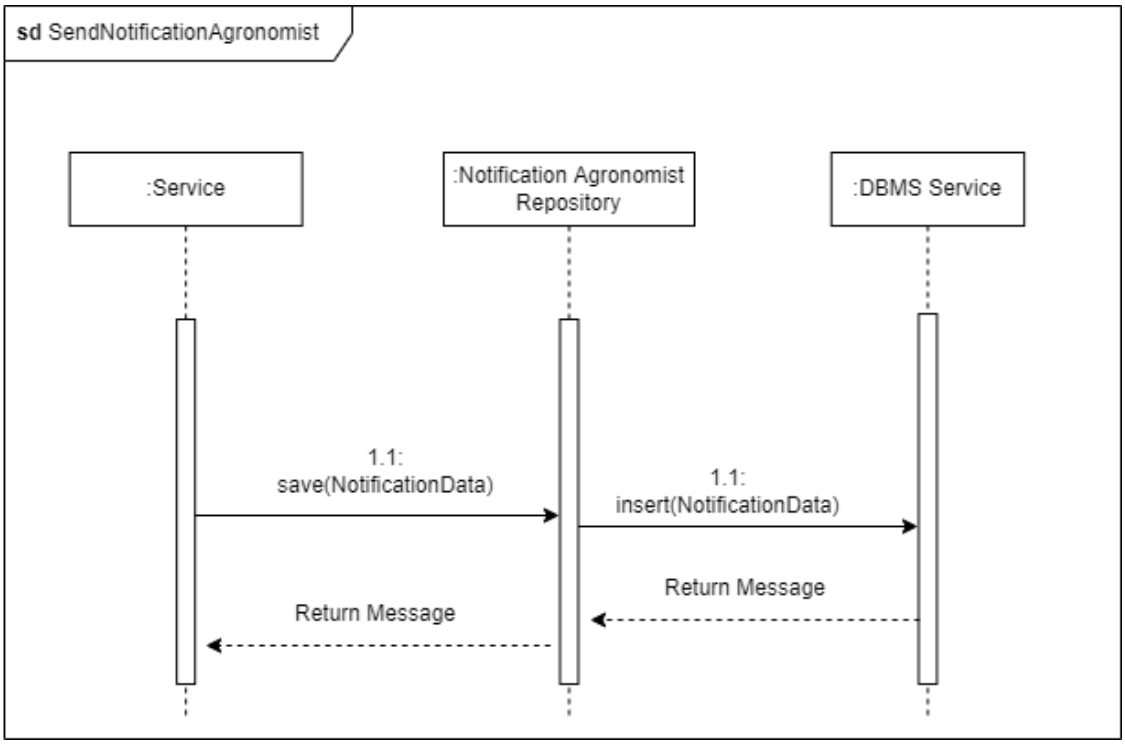


Figure 15: Runtime View - SendNotificationAgronomist

12. Create Knowledge

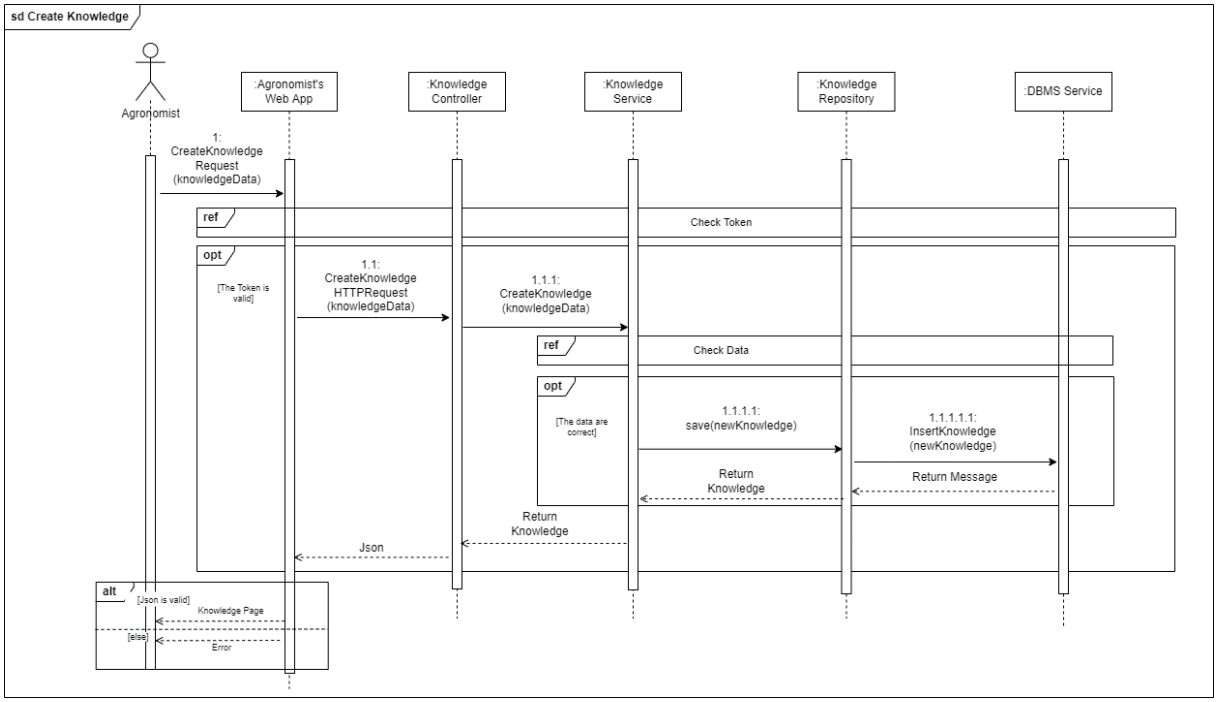


Figure 16: Runtime View - Create Knowledge

The sequence diagram just shown represents the Create Knowledge sequence that is only allowed to Agronomists.
The Agronomist uses the Agronomist's Web Application to publish a new Knowledge. Before forwarding the HTTP Request to the controller, the authentication token is checked. If the token in valid a POST HTTP Request is send to the Knowledge Service, passing throught the KnowledgeController. The Knowledge service checks then the data that should be inserted in the DB with the CheckData sequence. If the data are valid, the service calls a save request to the knowledge repository to save the new Knowledge. The repository send an insert query that is executed in the DBMSService. The new Knowledge is sent back to the Knowledge Controller and it is serialized in a JSON file. The JSON file then reaches the Agronomist's Web Application where it is checked. The Agronomist's Web Application then shows the Knwoledge Page. If the token is not valid, the Web App does not forward the request. Then, an error message is serialized in the JSON file, sent to the Agronomist's Web Application and eventually the application displays an error message instead of the Knowledge Page.

13. Like Knowledge

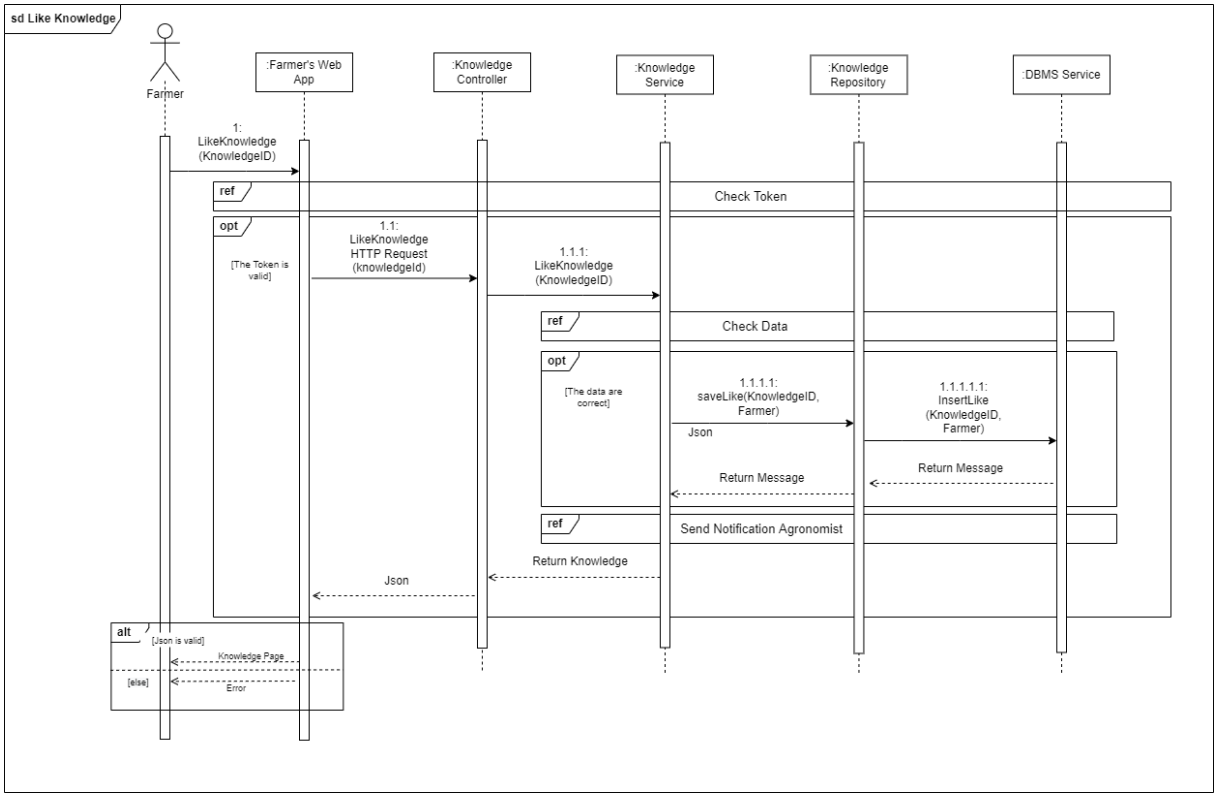


Figure 17: Runtime View - Like Knowledge

The sequence diagram just shown represents the Like process that any Farmer can perform on any knowledge shared by the Agronomist.
It is important to highlight how only the Farmer is thought to be able to give like through the system, being the only customer having access to knowledge shared by the agronomist.
First, the Farmer press the heart icon through their web application, Then, the application check the authentication token, and only if valid it sends a PUT HTTP request that reaches the Knowledge Controller. The KnowledgeController calls the like method of the Knowledge Service, that send a save request to the Knowledge Repository and an insert query gets executed in the DBMSService.
To inform the creator of the knowledge it get sent a notification to allow him to stay updated about its post. Lastly, the information liked Knowledge is brought back to the KnowledgeController and then it is serialized in a JSON file. This file then reaches the Farmer's Web Application where it gets checked. Eventually, the Farmer's Web Application show the updated Knowledge Page.
If the data inserted by the farmer are not valid, an error message is put in the JSON file and then it is shown to the farmer.

14. Create Meeting

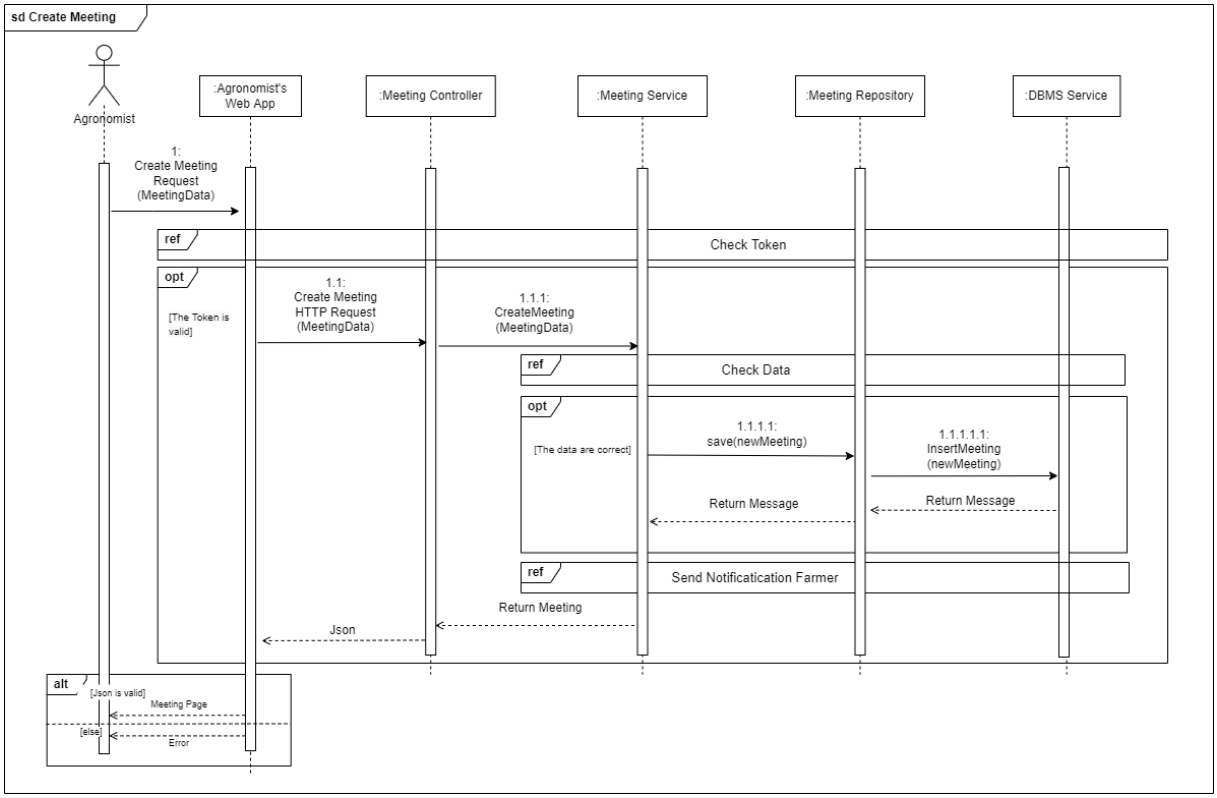


Figure 18: Runtime View - Create Meeting

The sequence diagram just shown represents the Create Meeting sequence that is only allowed to Agronomists. The Agronomist uses the Agronomist's Web Application to create a new Meeting. Before sending the HTTP Request to the controller, the token is checked. If the token is valid the POST HTTP Request is forwarded to the Meeting controller. The Meeting Controller calls the create Meeting method of the Meeting Service, that fisrt check the data in case there's a meeting in the same slot time, or is selected a wrong slot time. If the data are valid, the new meeting is created and the Meeting Service send a save request to the Meeting Repository witch send an insert query that is executed in the DBMSService. Note that the Meeting is saved with a pending status that will be changed when the Farmer confirm the Meeting. To inform the Farmer of the new Meeting a notification is sent to him by the Send Notification Farmer sequence, in that way the farmer will by able to know that there's a new meeting to accept or reject. The information about the Meeting are then sent back to the Meeting Controller and it is serialized in a JSON file. Lastly, the JSON file reaches the Agronomist's Web Application where it is checked. The Agronomist's Web Application then shows the Meeting Page with the updated Meetings.
If a token is not valid, the Web App does not forward the request. Then, an error message is serialized in the JSON file, by the check Token and sent to the Agronomist's Web Application and eventually the application displays an error message instead of the Meeting Page.

15. Modify Meeting

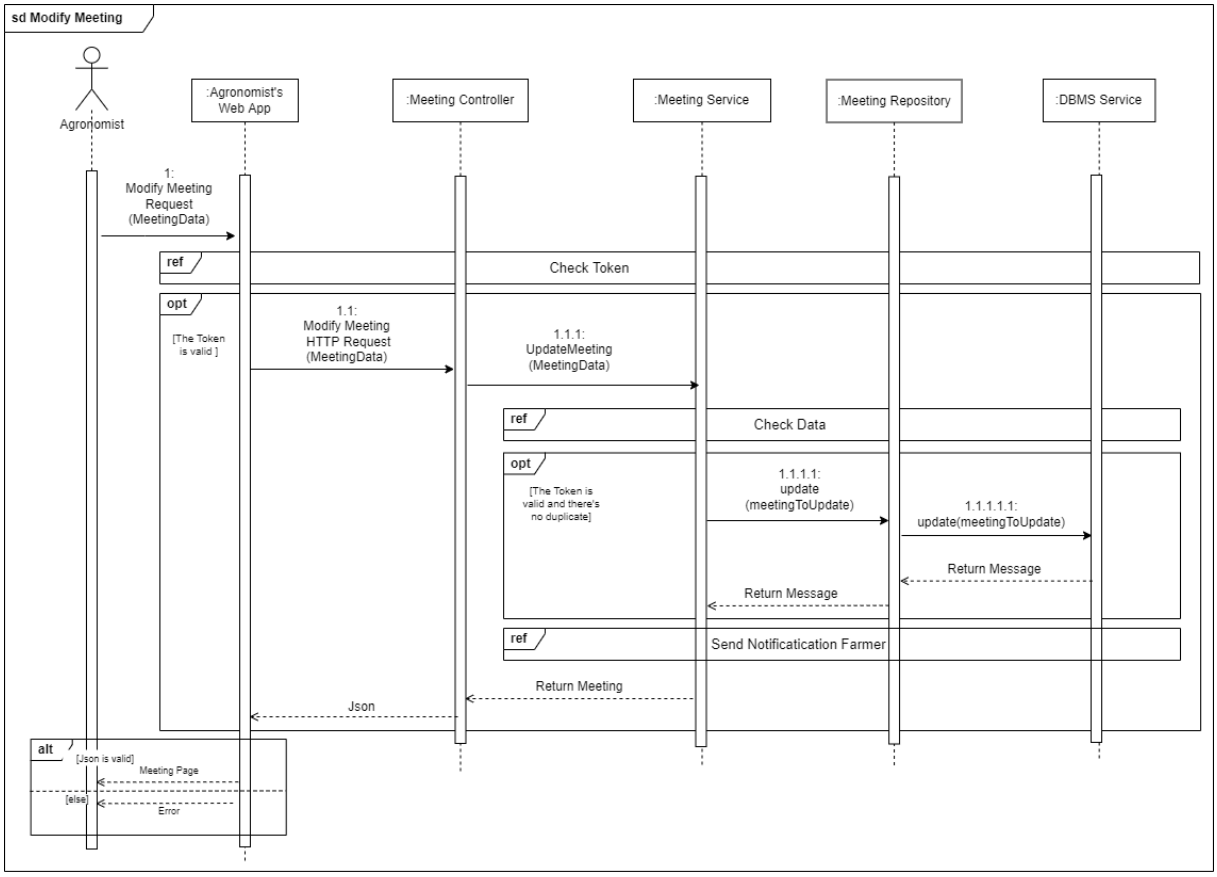


Figure 19: Runtime View - Modify Meeting

The sequence diagram just shown represents the modification of a Meeting sequence that is only allowed to Agronomists.
The Agronomist uses the Agronomist's Web Application to modify an existing Meeting. Before sending the HTTP Request to the Controller the token is checked with the CheckToken sequence.
If the token in valid the Agronomist's Web App send the PUT HTTP Request to the Meeting Controller, the Modify Meeting Request is propagated until the Meeting Service. The Meeting Data are checked by the Check Data and if there no other meeting in that slot time and the data are correct, the Service create the new Meeting and calls an update request to the Meeting Repository that send an update query is executed in the DBMSService. Note that the Meeting will be updated with a pending status that will be changed when the Farmer confirm the Meeting. To inform the Farmer of the changing of the meeting a notification is sent to the farmer with the information about the old meeting and the updated meeting. The information about the Meeting are sent back to the Meeting Controller and it is serialized in a JSON file. Lastly, the JSON file reaches the Agronomist's Web Application where it is checked. The Agronomist's Web Application then shows the Meeting Page with the updated Meetings. If a token is not valid, the Web App does not forward the request. Then, an error message is serialized in the JSON file by the check token, sent to the Agronomist's Web Application and eventually the application displays an error message instead of the Meeting Page.

16. Delete Meeting

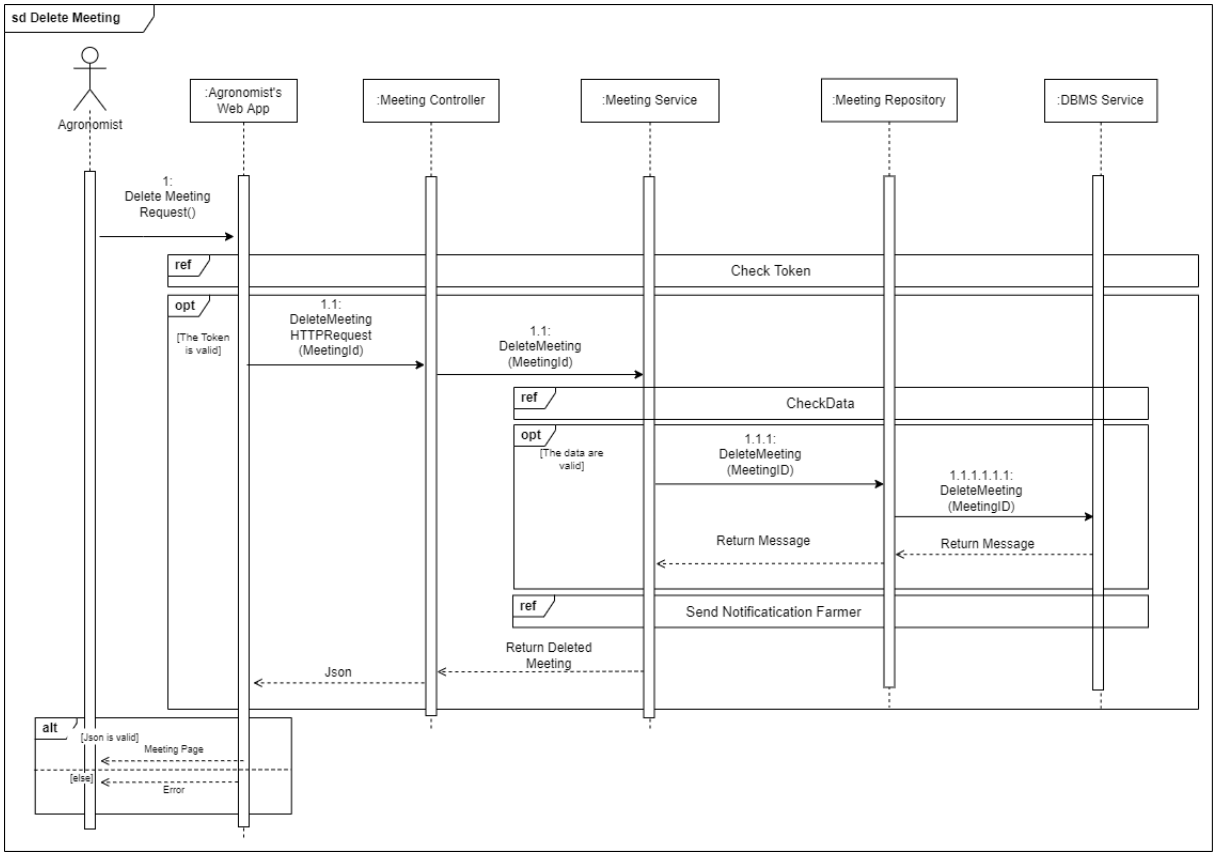


Figure 20: Runtime View - Delete Meeting

The sequence diagram just shown represents the Delete Meeting sequence that is only allowed to Agronomists. The Agronomist uses the Agronomist's Web Application to Delete an existing Meeting. First the token is checked, if valid the Agronomist's Web Application sends a DELETE HTTP Request to the Meeting Controller. The Delete Meeting Request is propagated until the Meeting Service. The latter calls a delete request of the Meeting Repository and then a delete query is sent and executed in the DBMSService. The Meeting Service also send a notification to the farmer to worn him about the deletion of the meeting. The information about the deleted Meeting are sent back to the Meeting Controller and it is serialized in a JSON file. Lastly, the JSON file reaches the Agronomist's Web Application where it is checked. The Agronomist's Web Application then shows the Meeting Page with the updated Meetings. If a token is not valid, the Web App does not forward the request. Then, an error message is serialized in the JSON file by the check token, sent to the Agronomist's Web Application and eventually the application displays an error message instead of the Meeting Page.

17. Accept or Reject Meeting

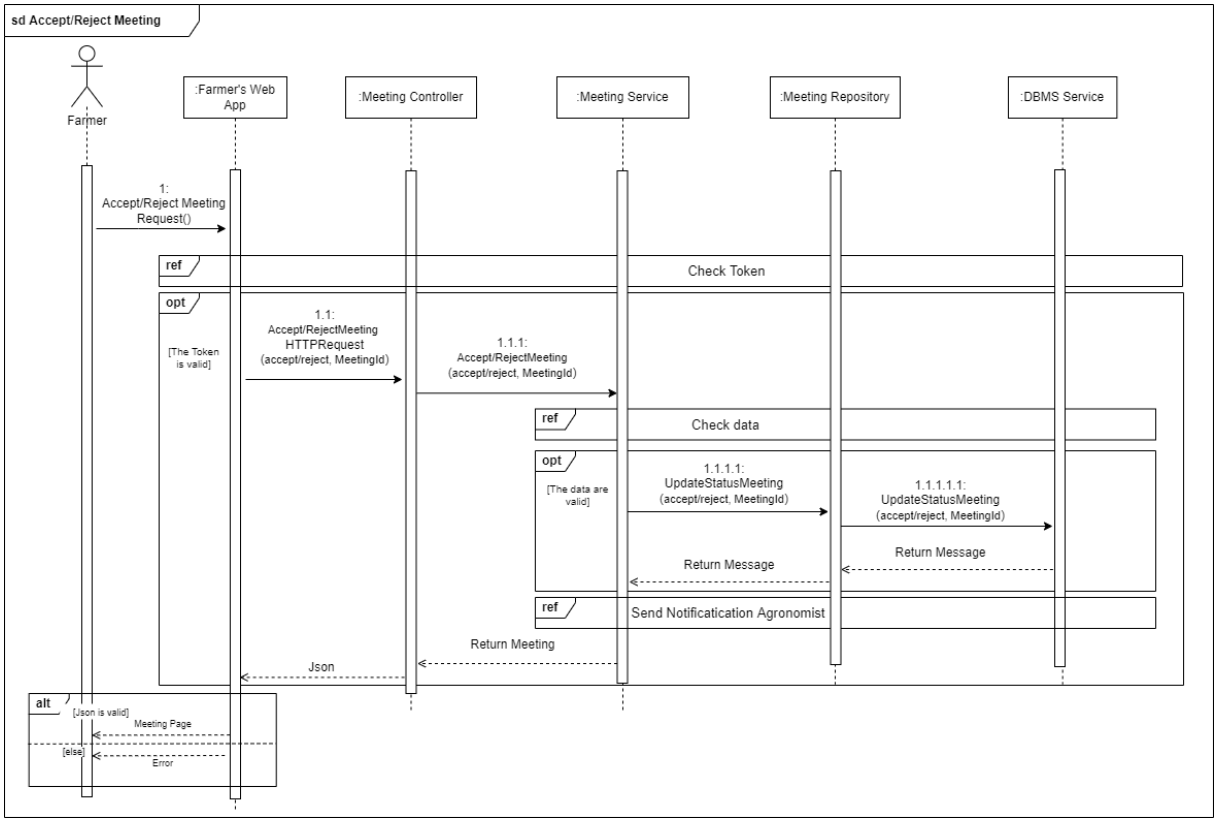


Figure 21: Runtime View - Accept or Reject Meeting

The sequence diagram just shown represents the Acceptation or the Rejection of a Meeting sequence that is only allowed to Farmers.
The Farmer uses the Farmer's Web Application to Accept or Reject a Meeting. First, the token is checked and if it's valid the Farmer's Web Application sends a PUT HTTP Request to the Meeting Controller. Then the Accept/Reject Meeting Request is propagated until the Meeting Service. The Meeting Service, first check if the farmer can actually accept or reject the meeting or it's too late with the check data sequence. If the data are valid the Meeting Service calls an update request of the Meeting Repository that will send an update query to the DBMS Service where is executed. In that way the status of the meeting is changed in Confirmed or Rejected status. The Meeting Service also send a notification to the agronomist to worn him that the meeting has been confirmed or rejected. The information about the Meeting are sent back to the Meeting Controller and it is serialized in a JSON file. Lastly, the JSON file reaches the Farmer's Web Application where it is checked. The Farmer's Web Application then shows the Meeting Page with the updated Meetings. If a token is not valid, the web app does not forward the request. Then, an error message is serialized in the JSON file by the check token, sent to the Farmer's Web Application and eventually the application displays an error message instead of the Meeting Page.

18. Show Topics

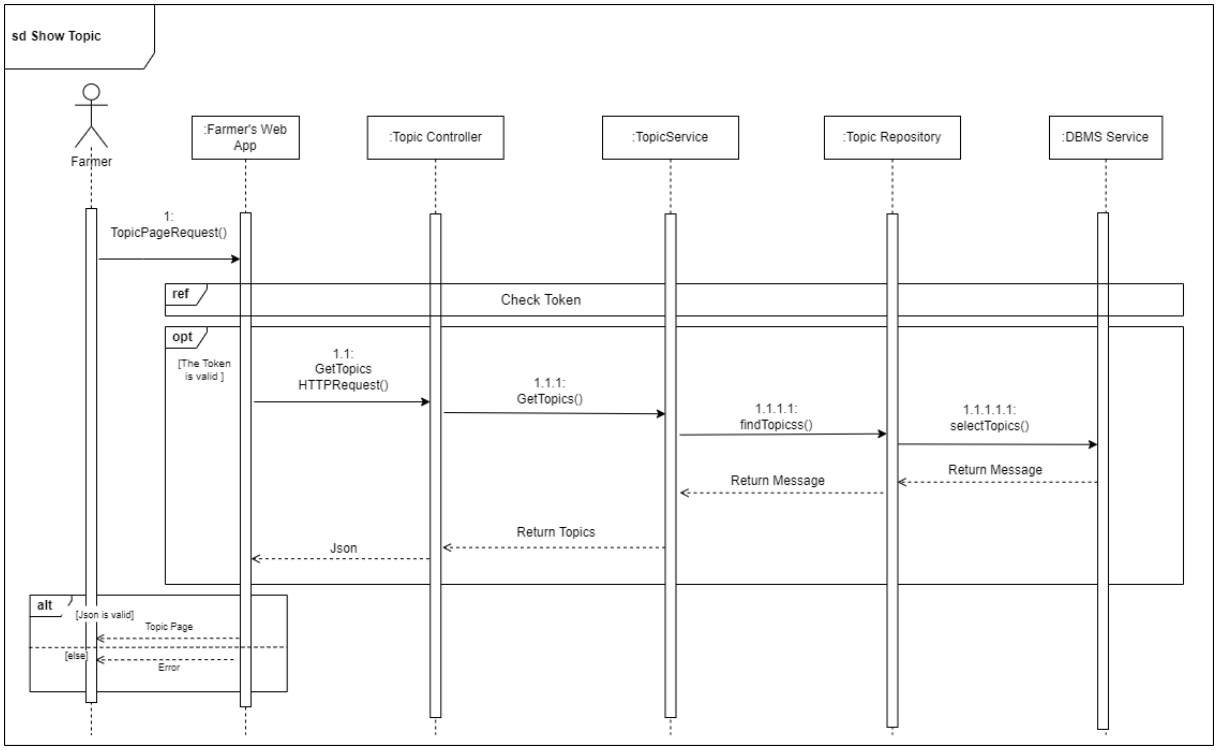


Figure 22: Runtime View - Show Topics

The sequence diagram just shown represents the process that allows to visualize the Farmer's Topic Page and the information contained in it.
The Farmer selects the Topics Page's section in the Farmer's Web Application. As previously specified, the Topic Page contains all the topics that are concerning the Farmer.
First, the authentication token is checked. If the token is valid, the GET HTTP Request is sent by the Web App to the Topic Controller, that call the get topic of the Topic Service. The latter sent a findTopic request to the Topic repository that forward the select query to the DBMS Service wehere eventually is performed on the database. The query result then is brought back to the Topic Controller. Finally it is serialized in a JSON file, brought back to the Farmer's Web Application.
If the token is not valid, an error message is serialized and the request is not propagated. The Farmer's Web Application will show an error message as well.

19. Create Topic

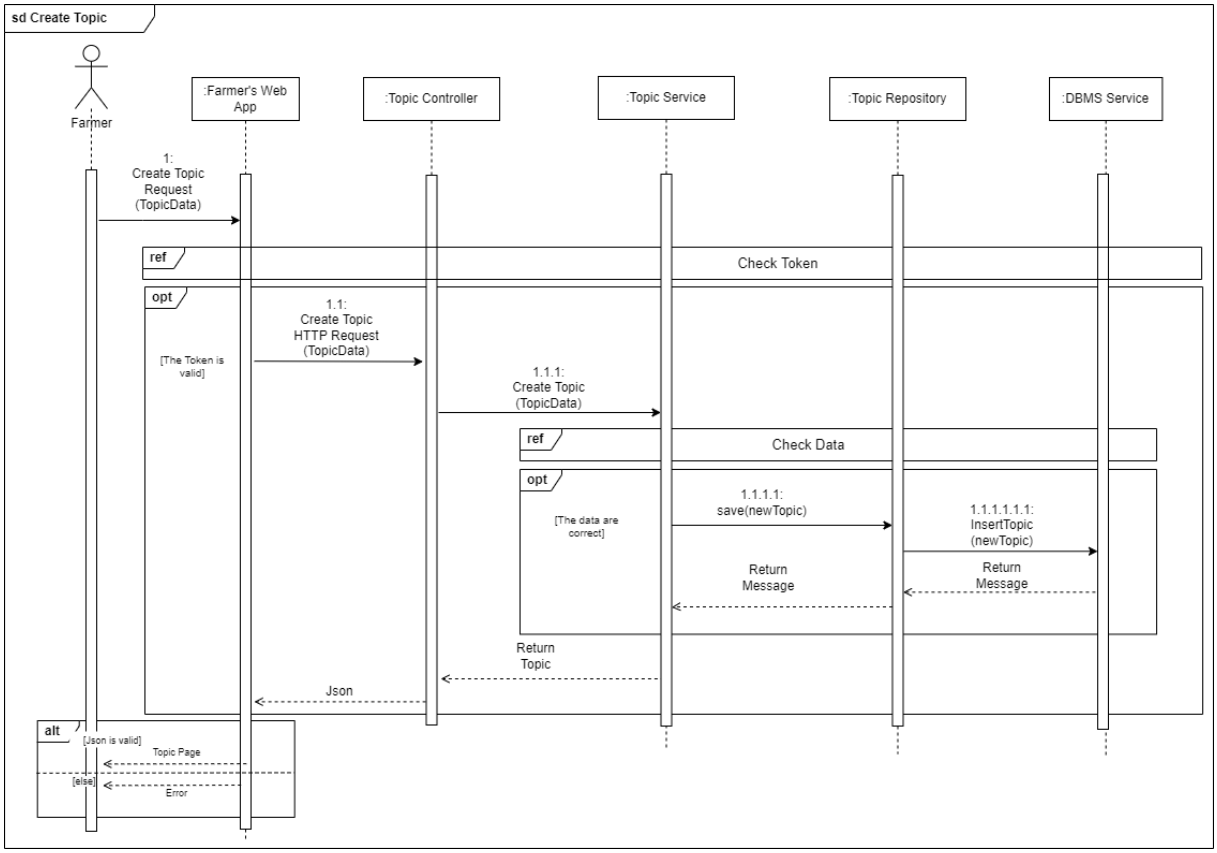


Figure 23: Runtime View - Create Topic

The sequence diagram just shown represents the Create Topic sequence that is only allowed to Farmers. The Farmers uses the Farmer's Web Application to create a new Topic. Before forwarding the HTTP Request to the controller, the token is checked and if valid, the Farmer's Web Application sends a POST HTTP Request to the Topic Controller and then forwarded to the Topic Service. The Topic Service before actually saving the new topic checks the data inserted by the farmer to be sure that there no duplicate and then calls a save method of the Repository and then a query is executed in the DBMSService. The information about the Topic are sent back to the Topic Controller and it is serialized in a JSON file. The JSON file then reaches the Farmer's Web Application where it is checked. The Farmer's Web Application then shows the Topic Page with the updated Topics. If a token is not valid, the Web App does not forward the request. Then, an error message is serialized in the JSON file, sent to the Farmer's Web Application and eventually the application displays an error message instead of the Topic Page.

20. Comment Topic

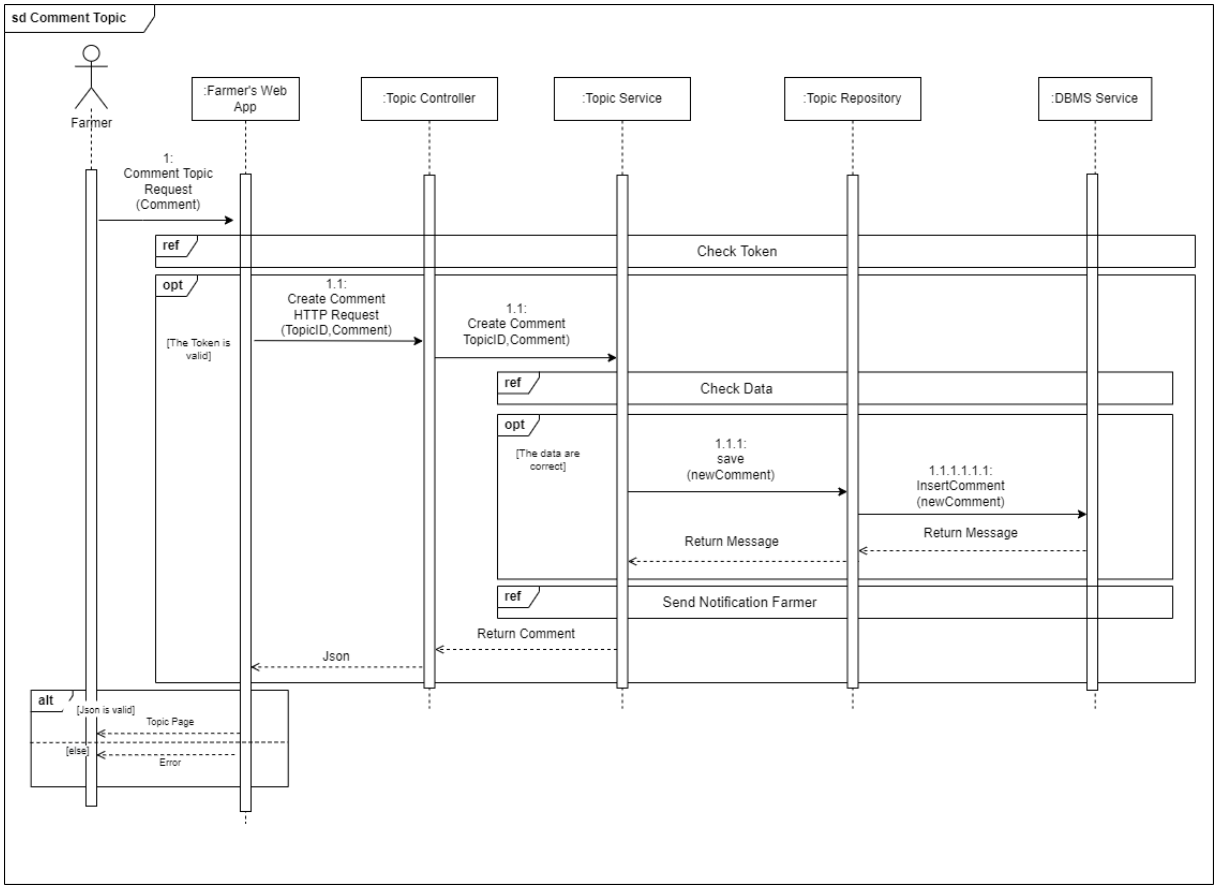


Figure 24: Runtime View - Comment Topic

The sequence diagram just shown represents the Comment Topic sequence that is only allowed to Farmers.
The Farmers uses the Farmer's Web Application to create a new Comment. Before forwarding the HTTP Request to the controller, the token is checked and if valid, the Farmer's Web Application sends a POST HTTP Request to the Comment Controller and then forwarded to the Comment Service. The Comment Service before actually saving the new comment checks the data inserted by the farmer to be sure that there no duplicate and then calls a save method of the Repository and then a query is executed in the DBMSService. To inform the creator of the topic about the new comment a notification is sent to the creator to let him aware of the new comments. The information about the Comment are sent back to the Comment Controller and it is serialized in a JSON file. The JSON file then reaches the Farmer's Web Application where it is checked. The Farmer's Web Application then shows the Topic Page with the updated comment section. If a token is not valid, the Web App does not forward the request. Then, an error message is serialized in the JSON file, sent to the Farmer's Web Application and eventually the application displays an error message instead of the Topic Page.

21. Send Requests for Help

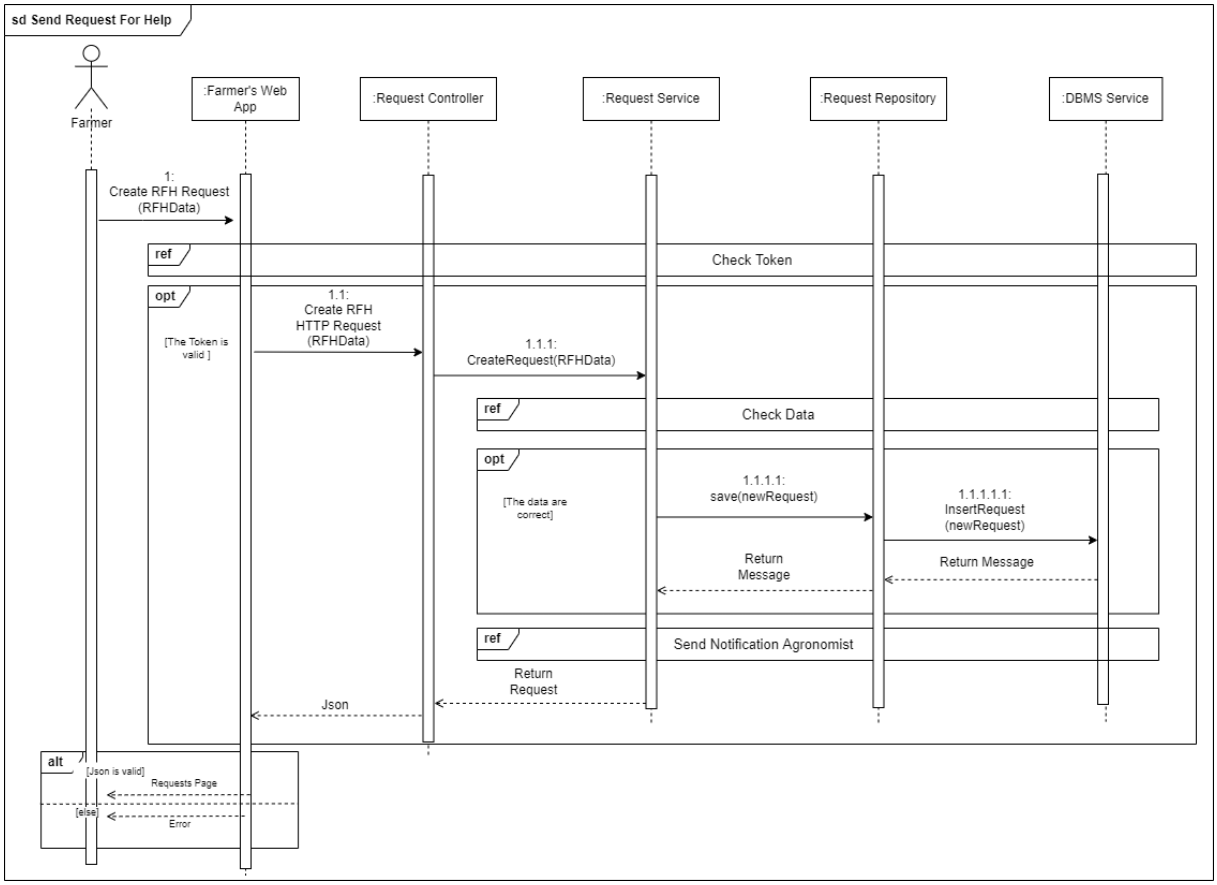


Figure 25: Runtime View - Send Request for Help

The sequence diagram just shown represents the Send a Request For Help sequence that is only allowed to Farmers.
The Farmers uses the Farmer's Web Application to Send a Request For Help. First, the authentication token si checked, if valid the Farmer's Web Application sends the POST HTTP Request to the Request Controller. Then, the RFH Request is propagated until the Request Service. The Request Service, first check the request data with the check data and select the agronomist that will recive the request. Then, the service create the request and calls the save method of the Request Repository and then an insert query is executed in the DBMSService. The Request Service also send a notification to the Local Agronomist about the new request. The information about the Request for Help are sent back to the Request Controller and it is serialized in a JSON file. The JSON file then reaches the Farmer's Web Application where it is checked. The Farmer's Web Application then shows the Requests Page with the updated Requests. If a token is not valid, the Web App does not forward the request. Then, an error message is serialized in the JSON file, sent to the Farmer's Web Application and eventually the application displays an error message instead of the Requests Page.

22. Answer Requests for Help

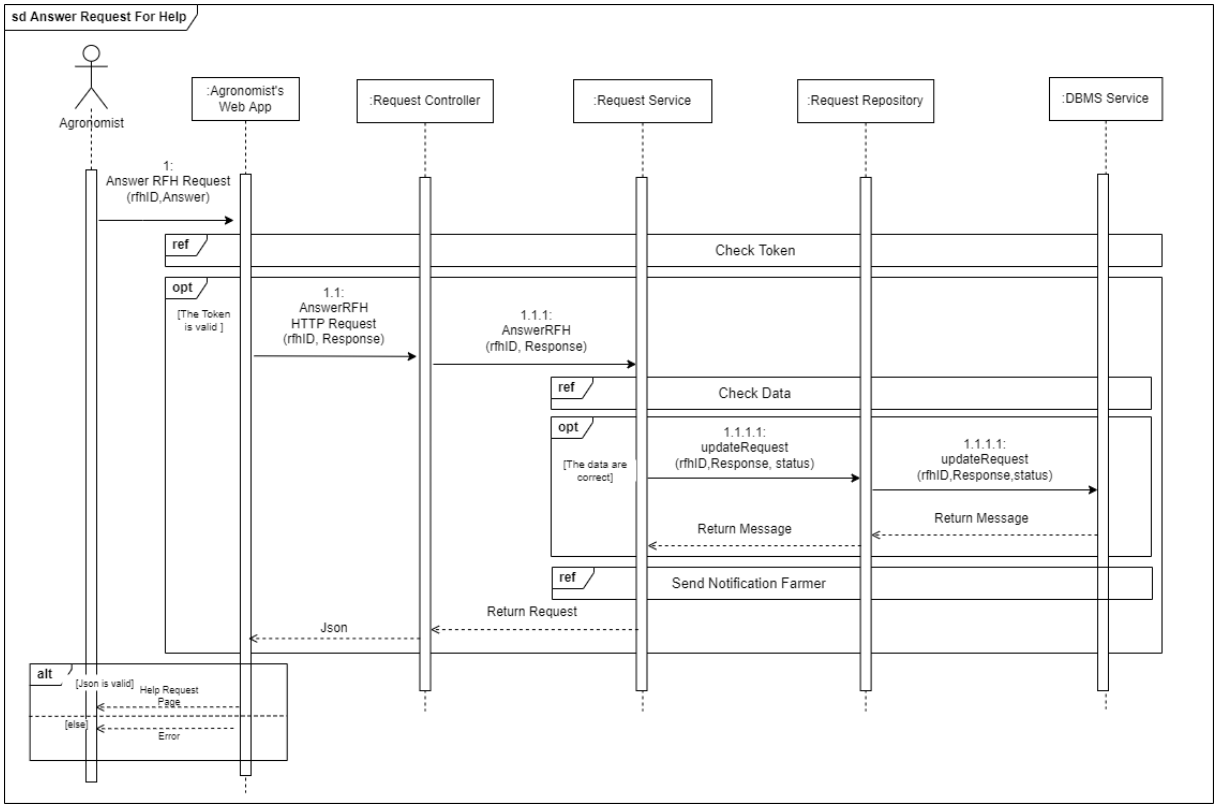


Figure 26: Runtime View - Answer Requests for Help

The sequence diagram just shown represents the Answer a Request For Help sequence that is only allowed to Agronomist.
The Agronomist uses the Agronomist's Web Application to Answer a Request For Help. First, the authentication token si checked, if valid the Farmer's Web Application sends the PUT HTTP Request to the Request Controller. Then, the RFH Request is propagated until the Request Service. The Request Service, first check the Response data with the check data. Then, the service calls the update method of the Request Repository and then an update query is executed in the DBMSService, to update the response of the request. The Request Service also send a notification to the Farmer about the response. The information about the Request for Help are sent back to the Request Controller and it is serialized in a JSON file. The JSON file then reaches the Farmer's Web Application where it is checked. The Agronomist's Web Application then shows the Requests Page with the updated Requests. If a token is not valid, the Web App does not forward the request. Then, an error message is serialized in the JSON file, sent to the Agronomist's Web Application and eventually the application displays an error message instead of the Requests Page.

23. Show Farm Info

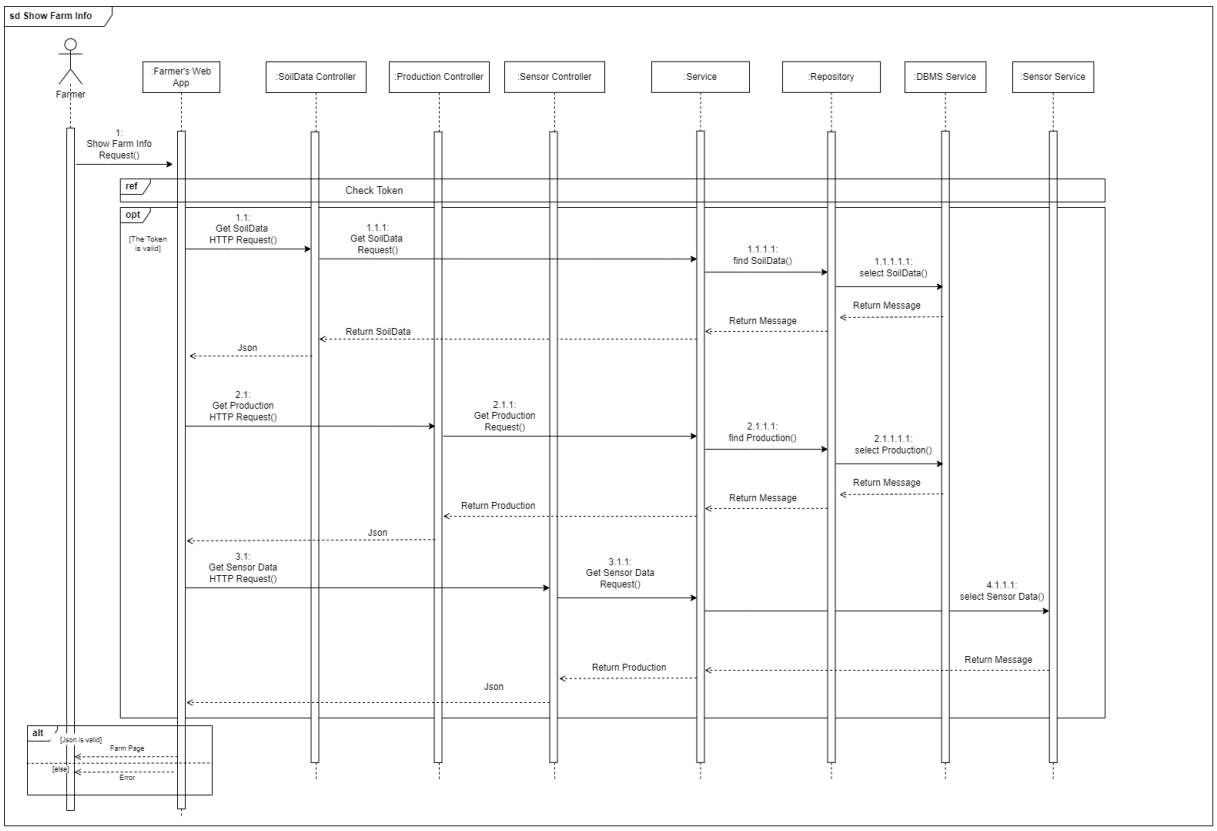


Figure 27: Runtime View - Show Farm Info

The sequence diagram just shown represents the Show Farm Info sequence that is only allowed to Farmer. The Farmer uses the Farmer's Web Application to look at the farm information. The Token is first checked by the check token sequence, if valid the Farmer's Web Application sends a GET HTTP Request to the SoilData Controller and propagated until the Soil Data Service. The SoilData Service calls a find method of the Repository and then a query is executed in the DBMSService. The information about the Soil are sent back to the SoilData Controller. The same process is executed for the production. Lastly, the Sensor Controller calls a method of the Sensor Service to get instant farm data. The information about the farm are sent back to the Controllers and all these data are serialized in a JSON file. The JSON file then reaches the Farmer's Web Application where it is checked. The Farmer's Web Application then shows the Farm Page with the updated Farm Info. If the token is not valid, the Web App does not forward the request. Then, an error message is serialized in the JSON file, sent to the Farmer's Web Application and eventually the application displays an error message instead of the Production Page.

24. Insert Farm

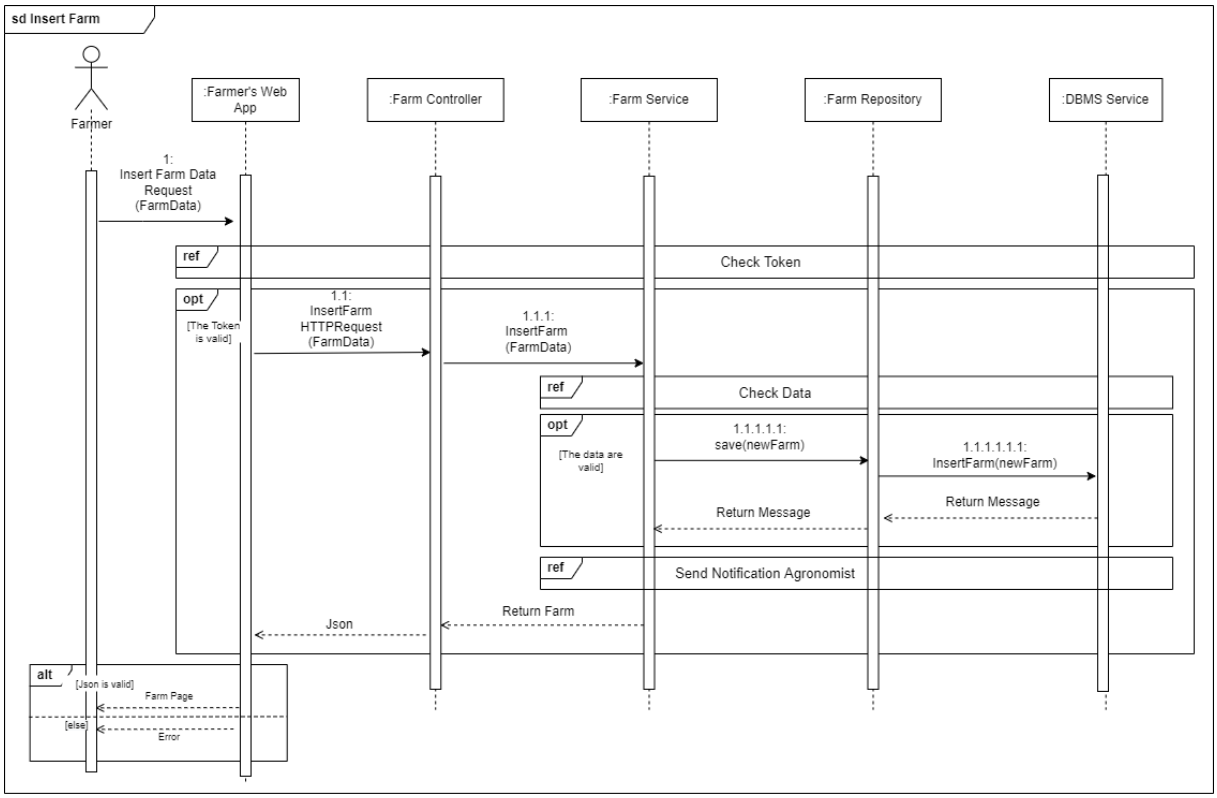


Figure 28: Runtime View - Insert Farm

The sequence diagram just shown represents the Insert the Farm sequence that is only allowed to Farmer. The Farmer uses the Farmer's Web Application to Insert Farm Data, like the location, adress and km square of the area managed by the farmer. Before sending the HTTP Request the token is checked and only if valid the Farmer's Web Application sends a POST HTTP Request through the Farm Controller and towards the Farm Service. The Farm Service checks the data inserted like the existence of the location with the check data if the data are valid, the service calls the save method of the Repository and then an insert query is executed in the DBMSService. To warn the local Agronomist about the new farmer in his area a notification is sent by the Send Notification Agronomist. It's important to highlight the fact that all the operation that concern the Agronomist, Farm and Production are blocked untill the Farmer insert the farm. The information about the Farm are sent back to the Farm Controller and it is serialized in a JSON file. The JSON file then reaches the Farmer's Web Application where it is checked. The Farmer's Web Application then shows the Farm Page and the other pages are unlocked. If the token is not valid, the Web App does not forward the request. Then, an error message is serialized in the JSON file, sent to the Farmer's Web Application and eventually the application displays an error message instead of the Farm Page.

25. Insert Soil Data

The sequence diagram just shown represents the Insert Soil Data sequence that is only allowed to Agronomist.

The Agronomist uses the Agronomist's Web Application to Insert Soil Data. First the authentication token is checked and if valid, the Agronomist's Web Application sends a POST HTTP Request through the SoilData Controller and towards the SoilData Service. The SoilData Service then check the data inserted by the Agronomist like the fact that he can actually put the soil data of that farmer, using the

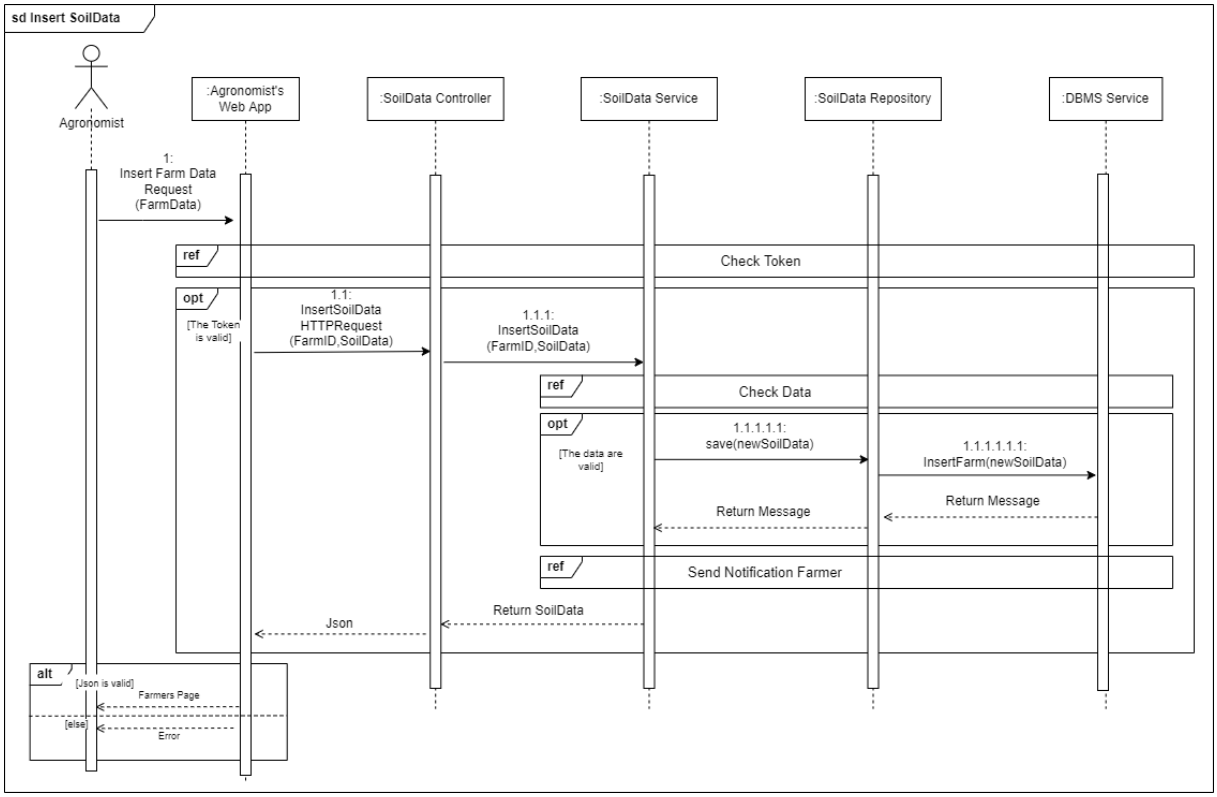


Figure 29: Runtime View - Insert Soil Data

check data. If the data are valid, the soilData Service calls the save method of the SoilData Repository and then a insert query is executed in the DBMSService. To inform the farmer about the new Soil Data a notification is sent to the farmer by the Send Notification Farmer. The information about the Soil Data are sent back to the Soil Data Controller and it is serialized in a JSON file. The JSON file then reaches the Agronomist's Web Application where it is checked. The Agronomist's Web Application then shows the Farmers Page with the updated Soil Data. If the token is not valid, the Web App does not forward the request. Then, an error message is serialized in the JSON file, sent to the Agronomist's Web Application and eventually the application displays an error message instead of the Farmers Page.

26. Insert Product

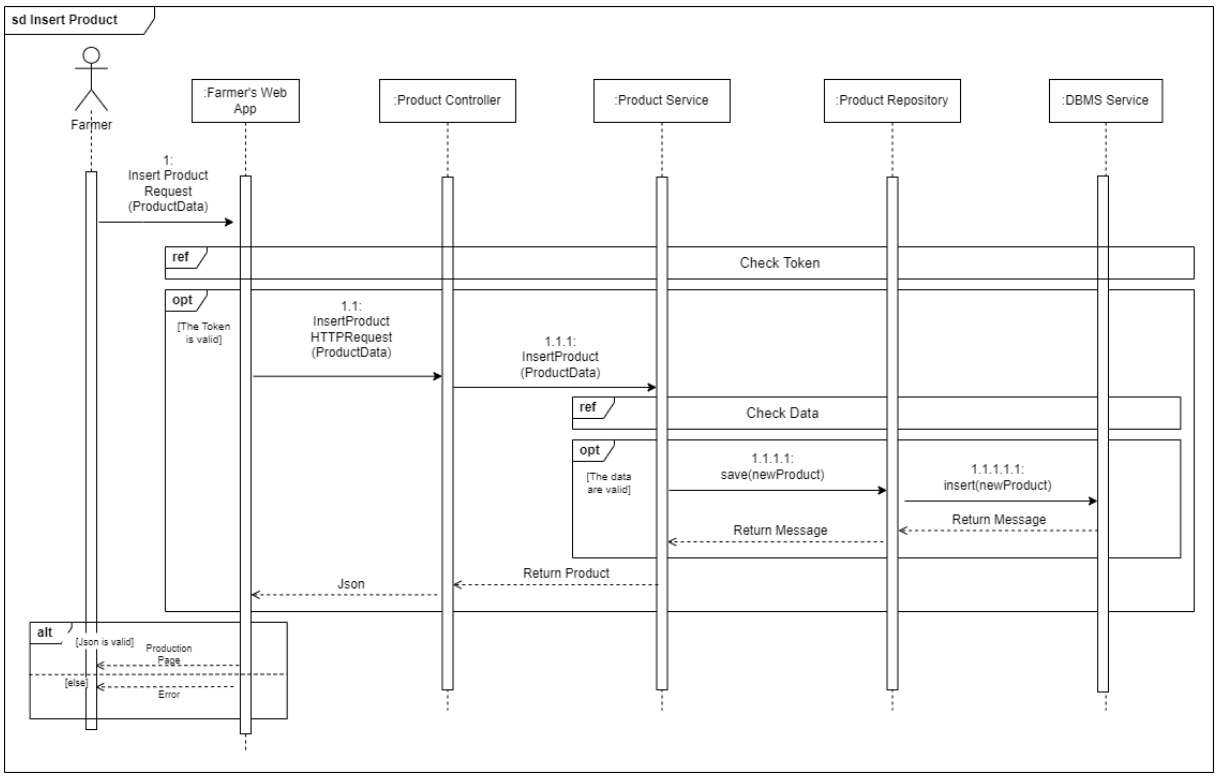


Figure 30: Runtime View - Insert Product

The sequence diagram just shown represents the Insert Product sequence that is only allowed to Farmer. The Farmer uses the Farmer's Web Application to Insert a new Product. Firt the authentication token is checked and if valid the Farmer's Web Application sends a POST HTTP Request through the Product Controller and towards the Product Service. The Product Service, first, check the data inserted by the farmer and if they're valid it calls the save method of the Product Repository and then a query is executed in the DBMSService. The information about the Product are sent back to the Product Controller and it is serialized in a JSON file. The JSON file then reaches the Farmer's Web Application where it is checked. The Farmer's Web Application then shows the Farm Page with the updated Product. If the token is not valid, the Web App does not forward the request. Then, an error message is serialized in the JSON file, sent to the Farmer's Web Application and eventually the application displays an error message instead of the Farm Page.

27. Insert Production

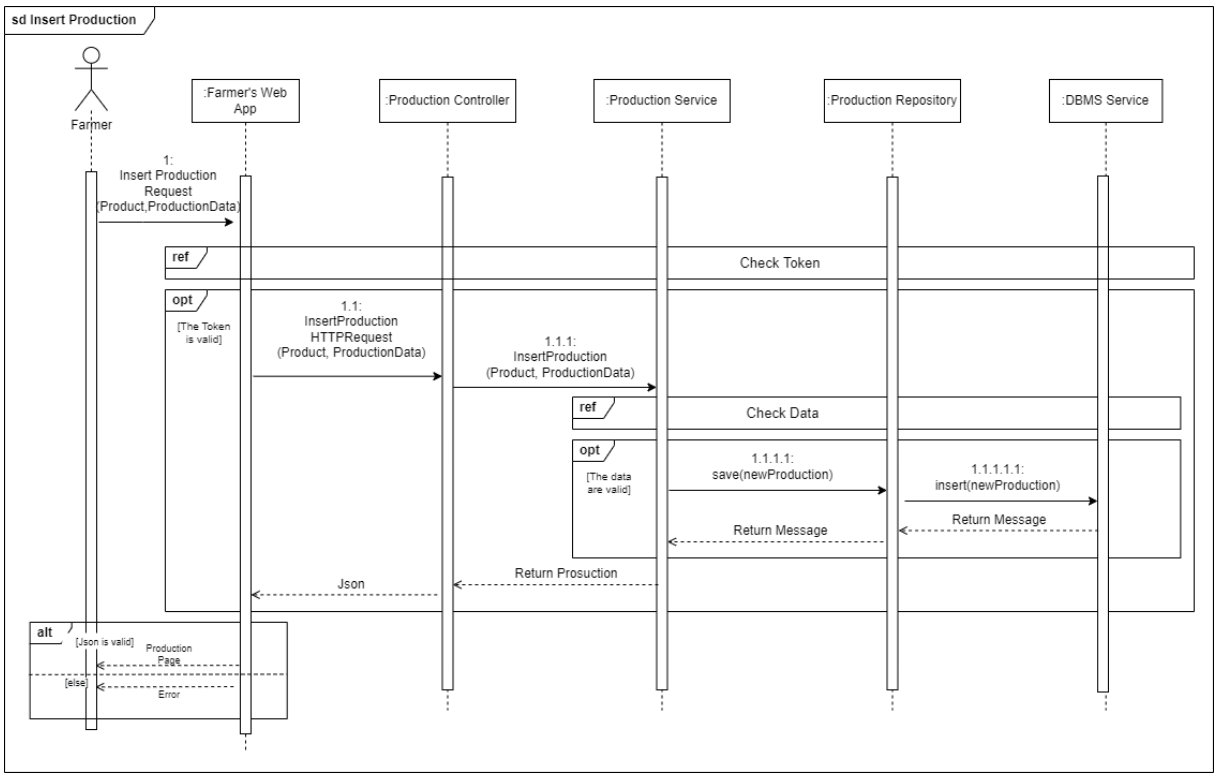


Figure 31: Runtime View - Insert Production

The sequence diagram just shown represents the Insert Production sequence that is only allowed to Farmer.
The Farmer uses the Farmer's Web Application to Insert a new Production. Firt the authentication token is checked and if valid the Farmer's Web Application sends a POST HTTP Request through the Production Controller and towards the Production Service. The Production Service, first, check the data inserted by the farmer and if they're valid it calls the save method of the Production Repository and then a query is executed in the DBMSService. The information about the Production are sent back to the Production Controller and it is serialized in a JSON file. The JSON file then reaches the Farmer's Web Application where it is checked. The Farmer's Web Application then shows the Farm Page with the updated Production. If the token is not valid, the Web App does not forward the request. Then, an error message is serialized in the JSON file, sent to the Farmer's Web Application and eventually the application displays an error message instead of the Farm Page.

28. Grade Farmer

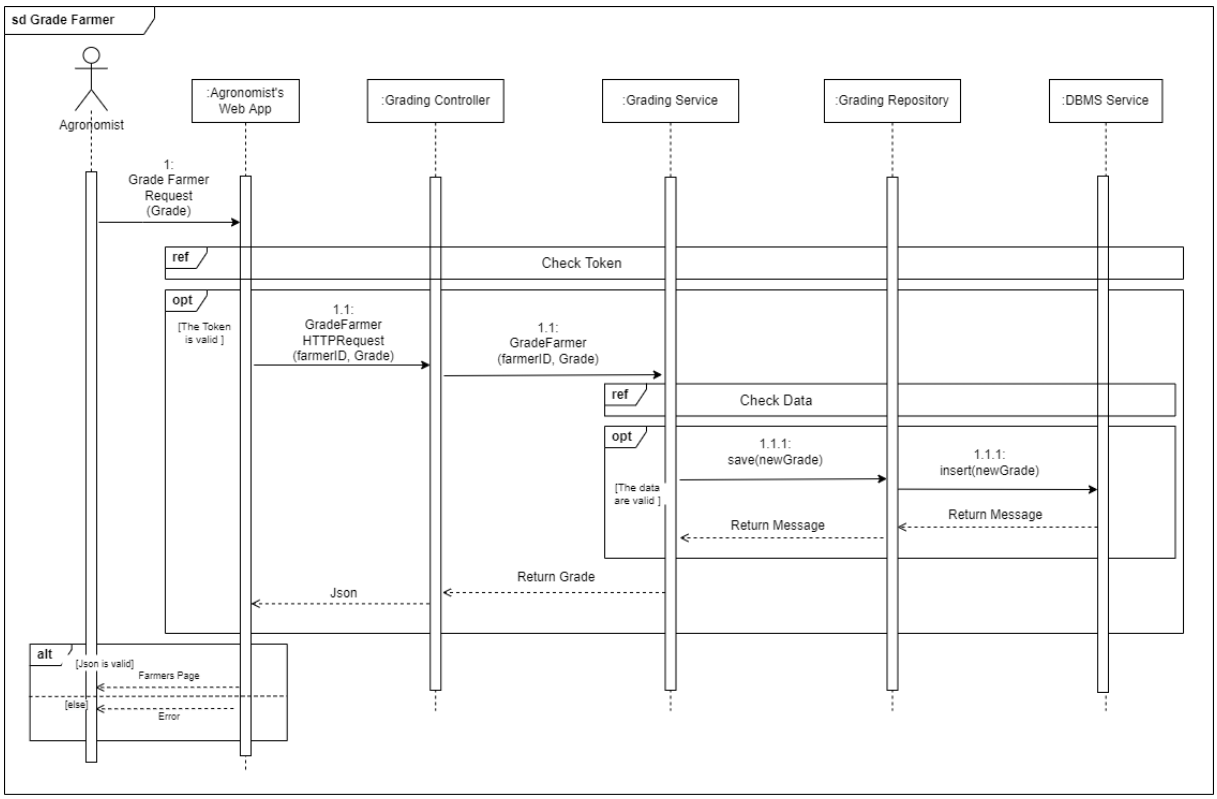


Figure 32: Runtime View - Grade Farmer

The sequence diagram just shown represents the Grade Farmer sequence that is only allowed to Agronomist. The Agronomist uses the Agronomist's Web Application to Grade the Farmer. Before sending the HTTP Request, the authentication token is checked and if valid the Agronomist's Web Application sends a POST HTTP Request through the Grade Controller and towards the Grade Service. The Grading Service check the data inserted by the agronomist and if valid calls a save method of the Grading Repository and then a query is executed in the DBMSService. The information about the Grade are sent back to the Grading Controller and it is serialized in a JSON file. The JSON file then reaches the Agronomist's Web Application where it is checked. The Agronomist's Web Application then shows the Farmers Page with the updated grade. If the token is not valid, the Web APP does not forward the request. Then, an error message is serialized in the JSON file, sent to the Agronomist's Web Application and eventually the application displays an error message instead of the Farmers Page.

2.5 Component interfaces

This section will show the main interfaces of the components present in the system.
The names of the interfaces and variables used are however provisional and therefore may vary in the final version.

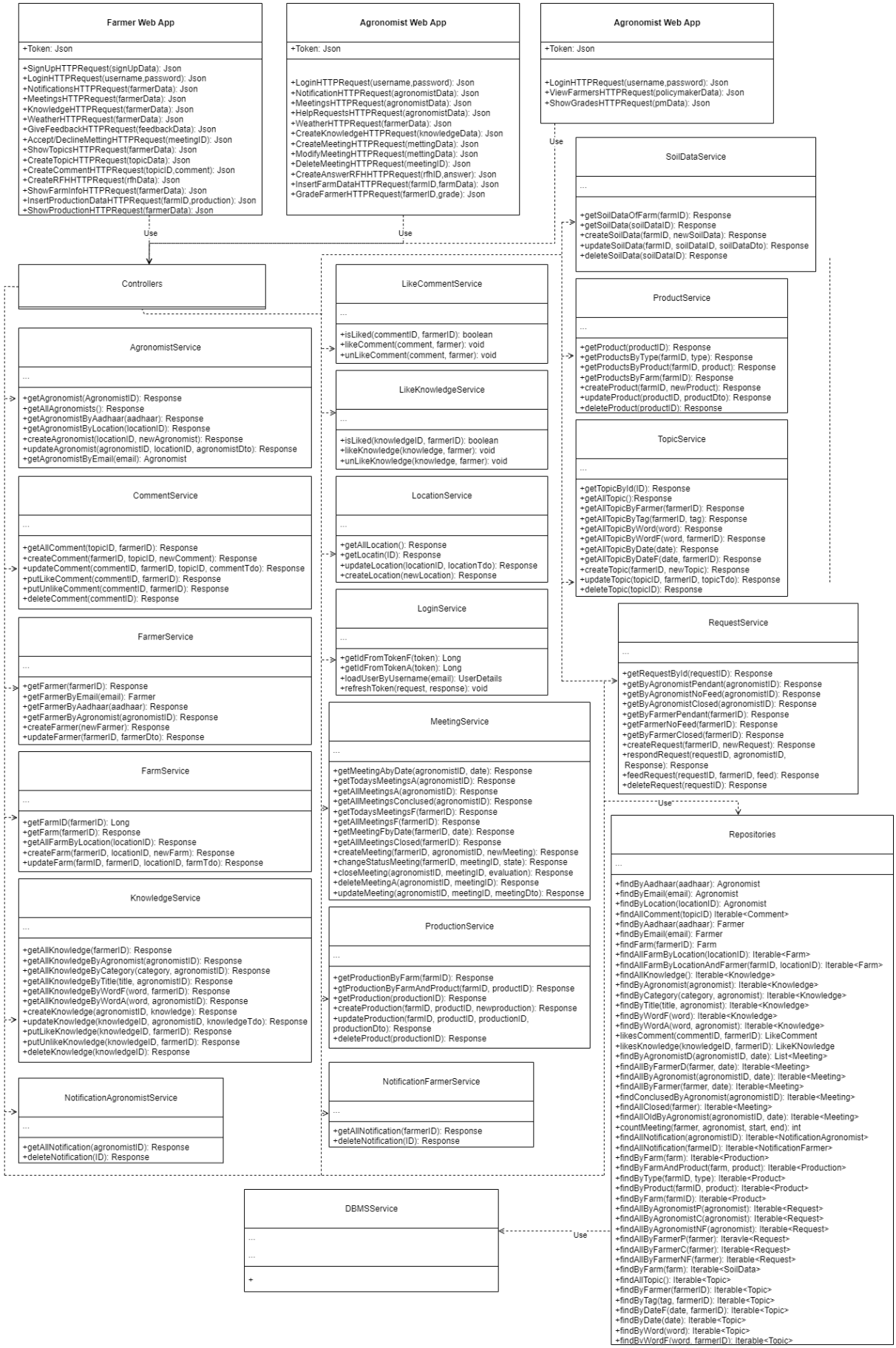


Figure 33: Component Interfaces

2.6 Selected architectural styles and patterns

The system is composed of four main parts following the SpringBoot Architecture, respectively: presentation layer, business logic layer, persistence layer and database layer. More in details, Spring Boot is a module of the Spring Framework and it is used to create stand-alone, production-grade Spring Based Applications with minimum efforts. It is developed on top of the core Spring Framework.
Spring Boot follows a layered architecture in which each layer communicates with the layer directly below or above in a hierarchical structure:

- The **presentation layer** which is represented by the Farmer's Web Application, the Agronomist's Web Application, and the Policy Maker's Web Application and the Controllers;
- The **business logic layer** which is made of the Services;
- The **persistence layer** which is made of the Repositories.
- The **database layer** which is made of the Database.

As previously explained in **Paragraph 2.1**, the reason for choosing this kind of architecture is to provide high scalability, high reliability, high security, and, finally, to reduce the development time.
For what concerns the security measures, we chose to encrypt all sensitive information, such as the passwords, stored in the database. Other then that, we chose to develop a system that allows to perform determined requests only to users with pre-defined roles. We also chose to develop a system of access and refresh tokens to guarantee security. This system to secure the back-end through Authentication and Authorization will be developed using SpringBoot, SpringSecurity and JWT (JSON Web Tokens). It is also important to highlight how the connection between client and server are provided through HTTP requests.
Finally, as it is clear from the runtimeview diagrams, each controller can serialize files in JSON format in order to bring back information to the Farmer's Web Application, the Agronomist's Web Application and the Policy Maker's Web Application concerning, for instance, the authentication process, data about shared Knowledges or topics, booked meetings or successful signup or related error messages.

Another choice that has been made was to use the **MVC or Model View Controller** architecture pattern. The MVC architecture is structured based on three separate components as follows:

- **Model:** The backend which contain all the data logic, used by the Controller;
- **View:** the frontend or graphical user interface (GUI) which displays readable data to the user;
- **Controller:** it can be thought as the "brain" of the application which connects View and Model and is responsible for controlling how data are displayed, modified and retrieved.

The reason behind this choice is that MVC allows the so called **separation of concerns** helping to breakup frontend and backend code into separate components making it much easier to manage, especially since the development process is performed by more than one person.

2.7 Other design decisions

The three web applications will be built with ReactJS.
React is a free and open-source front-end JavaScript library used for building web interfaces. The ReactJS's approach allows the development of single-page applications that dynamically rewrite the current web page, instead of loading entire new pages. The goal is to have faster transactions, more similar to a native app.

React applications can also be expanded with the use of numerous open-source libraries, for instance for routing, and the reuse of supported components, such as weather and calendar components.
Therefore, this approach allows to develop the UI in a faster and more maintainable way. The main purpose of React is to be fast, scalable, and simple. This corresponds to the View in the MVC template.

2.8 Algorithm

2.8.1 Pseudocode

The meeting between a farmer and an agronomist is associated with a status. This status can assume three different values:

- **Pendant:** the meeting has been created by the agronomist but is not accepted or rejected by the farmer;
- **Confirmed:** the meeting has been accepted by the farmer. Nevertheless, the farmer can still rejected in another time or the agronomist can change it or delete it, these option are available only 24h before the meeting;
- **Rejected:** the meeting has been rejected by the farmer, so the agronomist can propose a different day by changing the meeting;
- **Concluded:** it's an automatic operation that change the status of the meeting after the start of it, only if the meeting is in this status the agronomist can start writing the evaluation of the farmer;
- **Closed:** when the meeting is concluded and the evaluation of the farmer has been inserted by the agronomist.

When a meeting is created by an agronomist first is checked the overlapping of the meeting with other meetings, then if there's no overlapping the status is changed to pendant and the meeting is saved.

```
1
2  public CreateMeeting(newMeeting) {
3      if(is newMeeting overlapping with other existing meeting?){
4          return error;
5      }
6      else{
7          newMeeting.setStatus("pendant");
8          save(newMeeting);
9          return success;
10     }
11 }
12
```

Figure 34: Create Meeting

A farmer can confirm or reject a meeting only until the day before the meeting, in that case the status of the meeting is changed from the status he was before to the confirmed/ rejected status.

```
1
2  public confirmMeeting(meeting) {
3      if(Today is day before meetingDay?){
4          return message("to late to confirm meeting")
5      }
6      else{
7          meeting.setStatus("confirmed");
8          save(meeting);
9          return success;
10     }
11 }
12
13 public rejectMeeting(meeting) {
14      if(Today is day before meetingDay?){
15         return message("to late to reject meeting")
16     }
17     else{
18         meeting.setStatus("rejected");
19         save(meeting);
20         return success;
21     }
22 }
23
```

Figure 35: Confirm Reject Meeting

The cancellation of a meeting can be done only until the day before the meeting and in case is made by the farmer the status is changed in rejected, in that way the agronomist can simply modify the meeting without the need to create a new one. In case the cancellation is made by the agronomist the meeting is actually deleted from the db.

```
1
2  public cancelMeetingfarmer(meeting) {
3      if(Today is day before meetingDay?){
4          return message("to late to cancel meeting")
5      }
6      else{
7          meeting.setStatus("rejected");
8          save(meeting);
9          return success;
10     }
11 }
12
13 public cancelMeetingAgronomist(meeting) {
14      if(Today is day before meetingDay?){
15         return message("to late to cancel meeting")
16     }
17     else{
18         delete(meeting);
19         return success;
20     }
21 }
22
```

Figure 36: Cancel Meeting

The update meeting can be done only until the day before the meeting, the status of the meeting is set in the pendant status cause the changes has to be accepted by the farmer.

```
1
2  public updateMeeting(oldMeeting, newMeeting) {
3      if(is newMeeting overlapping with existing meeting? ){
4          return error;
5      }
6      else{
7          oldMeeting.update(newMeeting);
8          oldMeeting.setStatus("pendant");
9          save(oldMeeting);
10         return success;
11     }
12 }
13
```

Figure 37: Update Meeting

Every time the agronomist open the tab of meetings concluded page and the function getMeetingConclused the system first get all old meeting in the database. For each old meeting if the status is confirmed, we suppose that the meeting should have been done, so the status is set as conclused. Otherwise if the meeting is in pendant or rejected status it get deleted to minimize the meetings in the database.

```
1
2  public getMeetingConclused() {
3      get oldMeetings;
4      Meeting conclused;
5     for(m in oldMeeting){
6         if(m.getStatus()=="confirmed"){
7             m.setStatus("conclused");
8             conclused.add(m);
9         }
10        else if(m.getStatus()=="rejected" || m.getStatus()=="pendant")
11        {
12            delete(m);
13        }
14    }
15    return conclused
16 }
17
```

Figure 38: Conclude Meeting

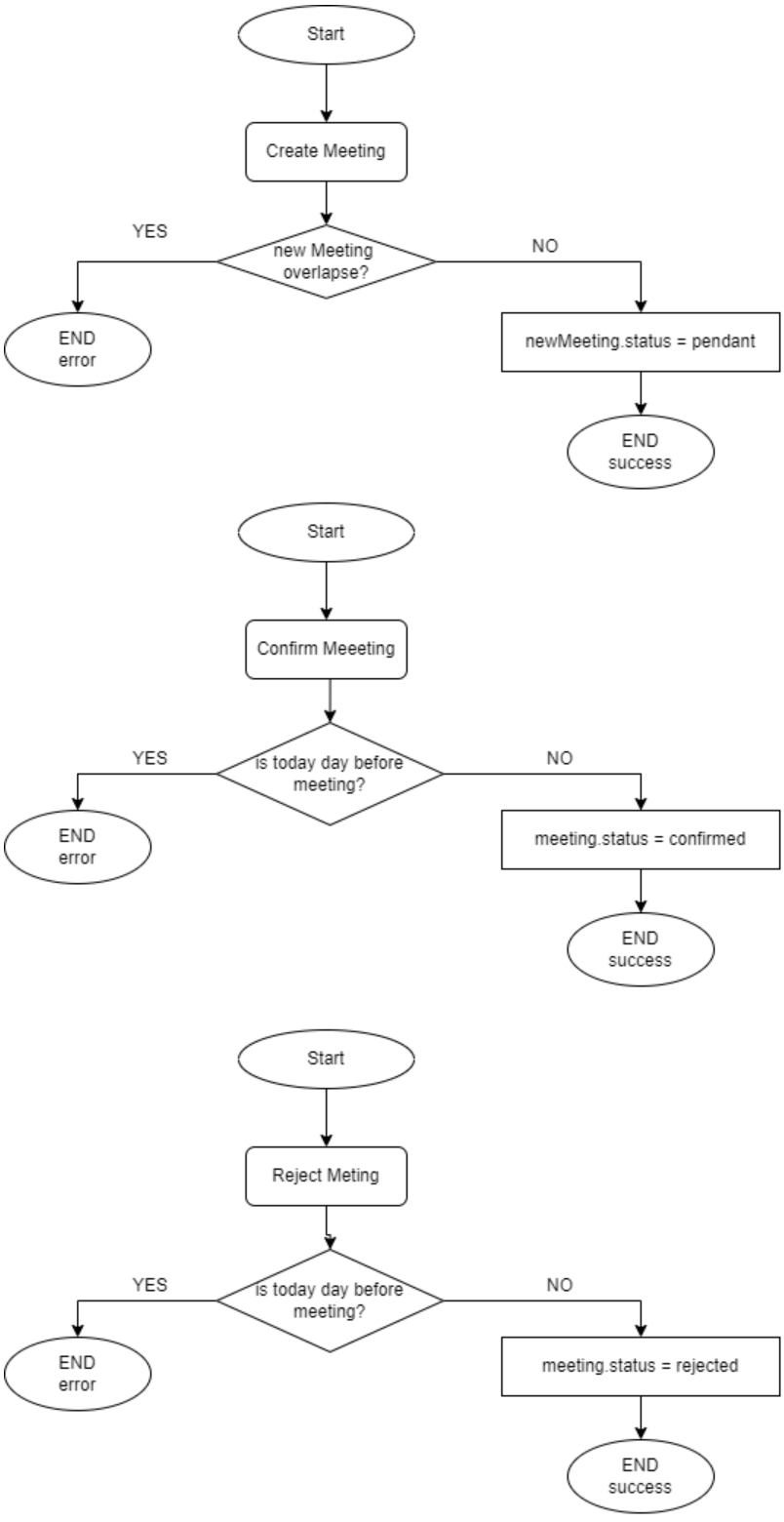
When the agronomist want to evaluate the meeting with a farmer first is checked that the meeting is actually happened, we assume that if the meeting is set on a date and hour the agronomist will respect that timing.

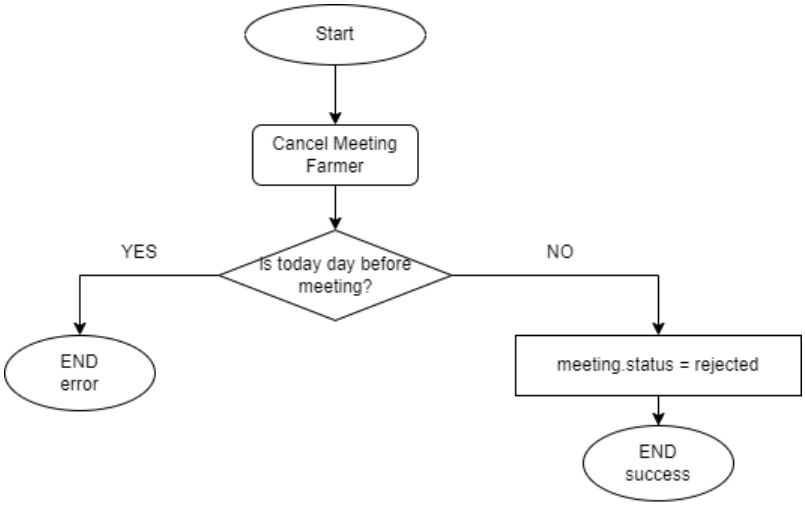
```
1
2  public closeMeeting(meeting, evaluation) {
3     if(is meeting conclused?)
4     {
5         meeting.setEvaluation(evaliation);
6         update(meeting);
7         return success;
8     }
9     else{
10        return error;
11    }
12 }
13
```

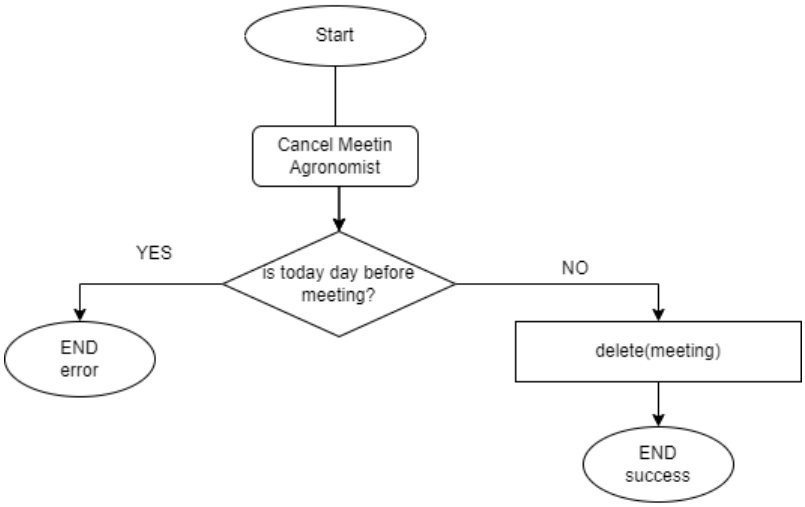
Figure 39: Close Meeting

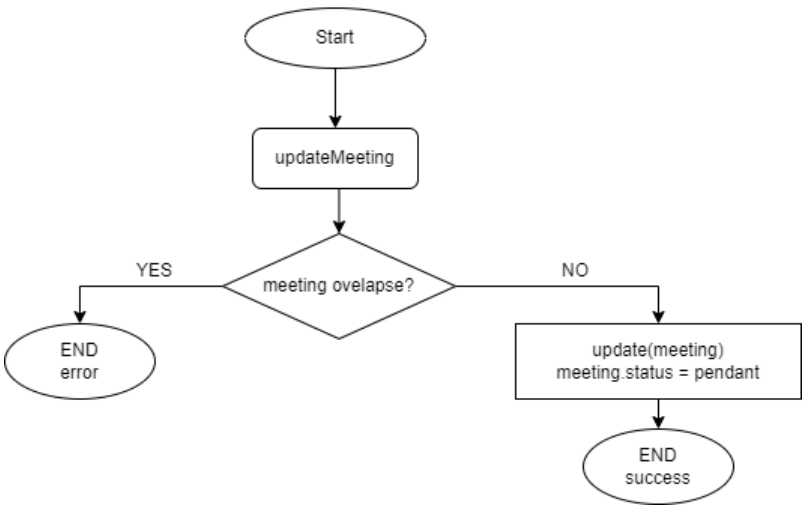
2.8.2 Flowcharts

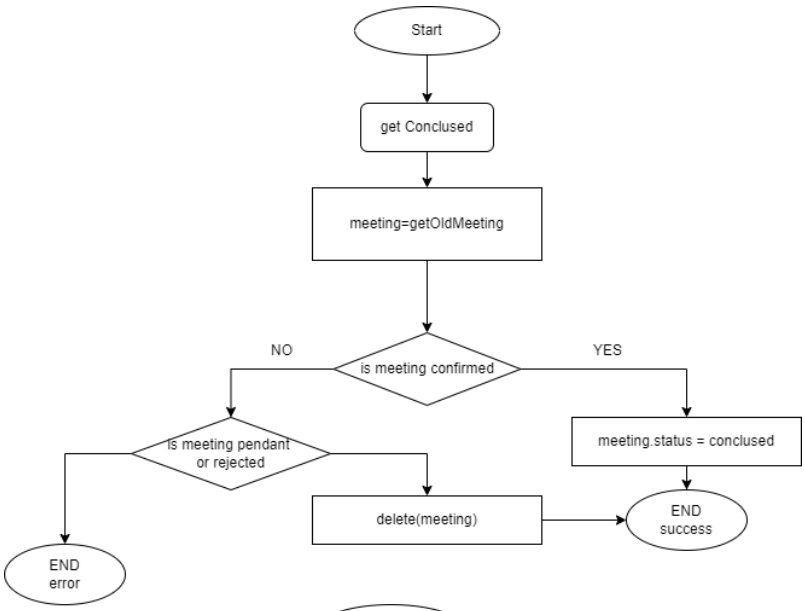
Lastly, a series of flowcharts corresponding with the above pseudocode examples.

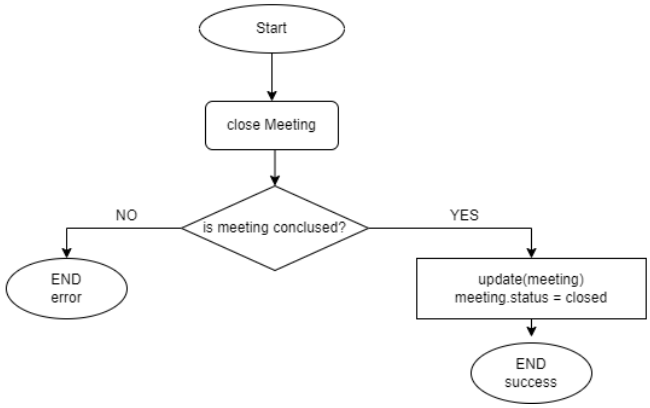












3 User Interface Design

3.1 Overview

In this section the User Interface mockups will be presented. All three web applications follow the "Bootstrap Italia" guidelines, the Bootstrap 4 theme for the development of web applications for the Italian public administration. The choice was made to make the system easy and intuitive, with well-contrasted texts and large buttons. Furthermore, a single page approach was followed, dividing navigation and tasks on different tabs.
Our discussion will start by describing the common sections of the three web applications and subsequently those specific to the three actors.

3.2 Mock-up common sections

Figure 40 shows the page where the three actors land for the first time before registration or login. This page will show an overview of the system and some useful information. There is also a button at the top right to reach the Login page.

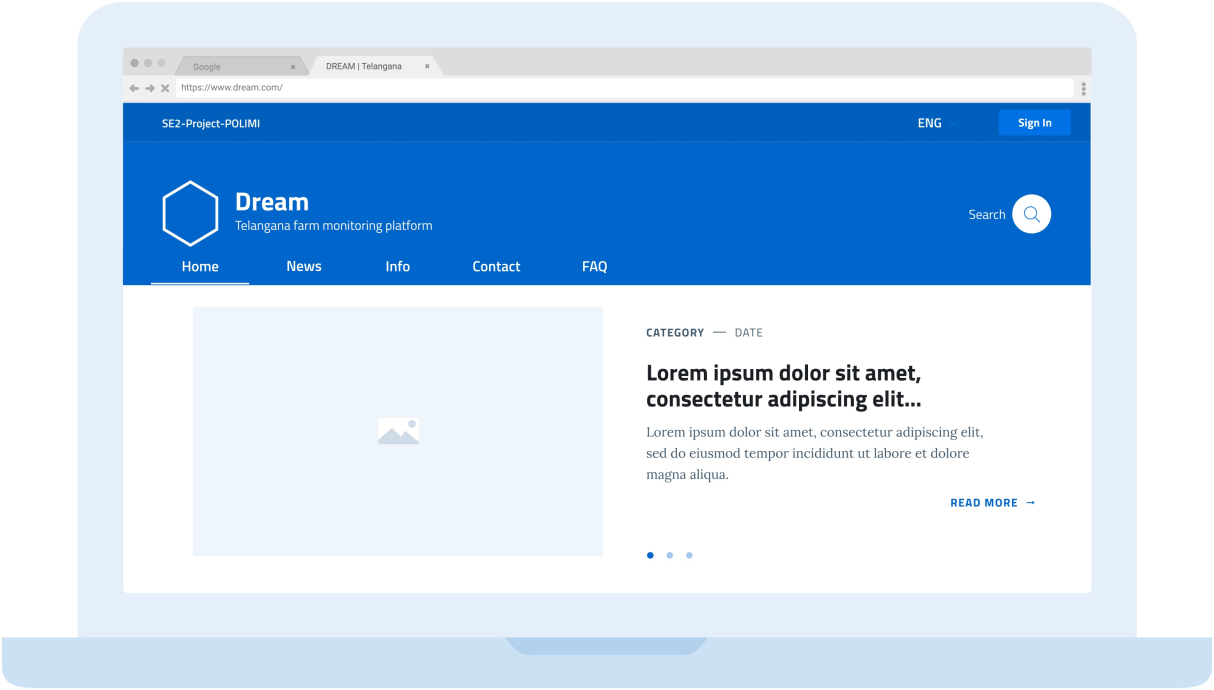


Figure 40: Mock-up - Landing page

As shown in figure 41, on this page it is possible to log in with the user's login credentials, e-mail and password in our case. It is also possible to check the appropriate field to remember the credentials and be already logged in in a next session. In the event that the user is not already registered (only for the farmer), it is possible to click the button that will redirect to the registration page, figure 42, where it is possible to carry out the procedure for the first insertion of the credentials in the system .

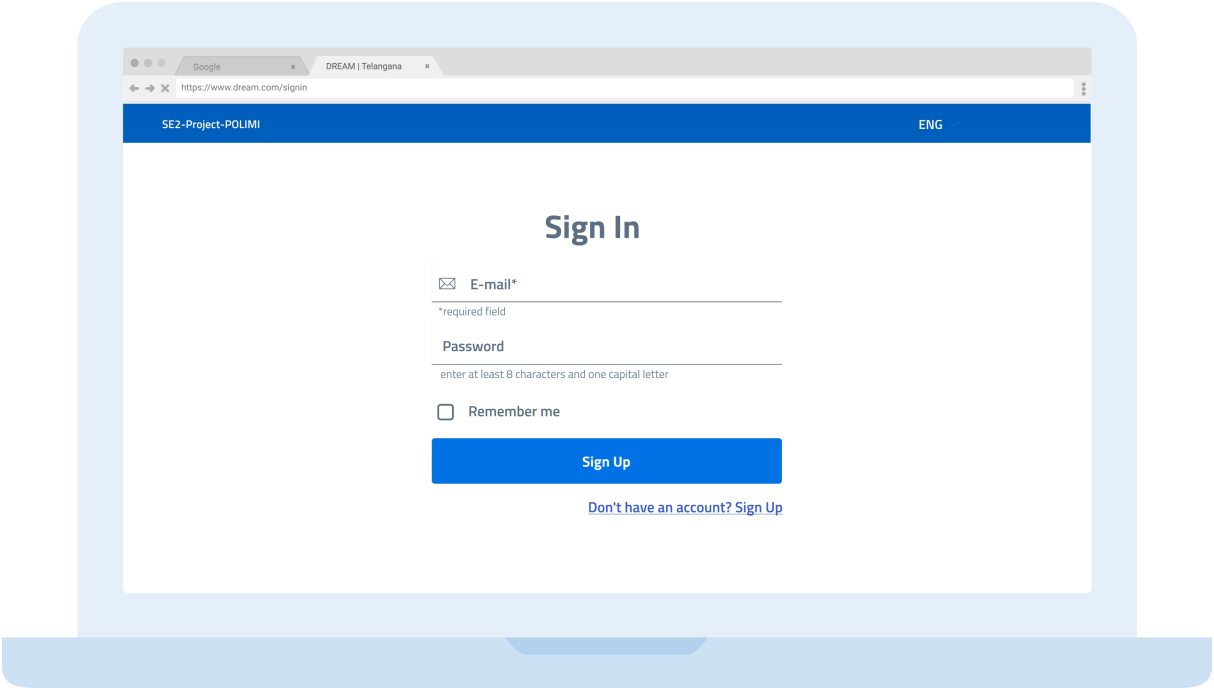


Figure 41: Mock-up - Login In page

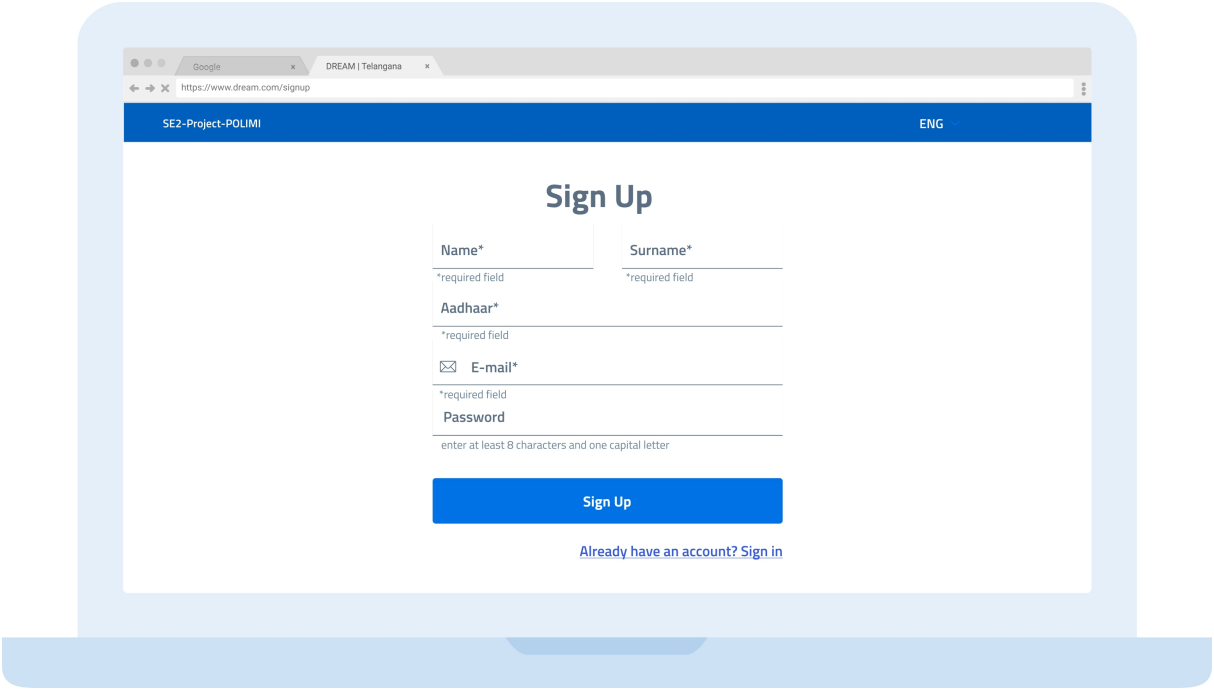


Figure 42: Mock-up - Registration page

3.3 Mock-up Farmer's Web Application

After logging in, the farmer will find himself on his home page as shown in figure 43. It shows an overview of the key functions of the web application. This page contains several subsections, such as the weather, some Knowledge, a summary of upcoming meetings and some notifications. Furthermore, like every screen, this too is composed of a header where the accessible tabs are present, allowing to change screens based on the functionality of the page.

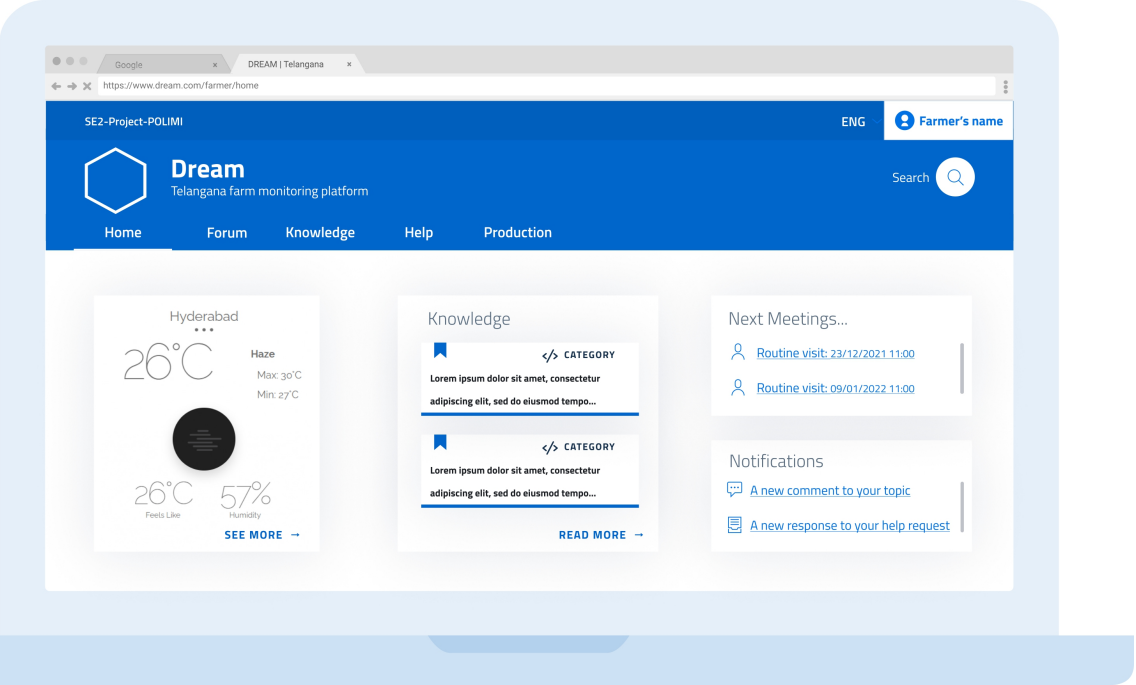


Figure 43: Mock-up - Farmer Home page

Figure 44 shows the Forum section. In this tab the farmer can explore the latest topics written by other farmers, see comments, reply and write a new topic.
Figure 45 shows the farmer Knowledge section. In this tab it is possible to explore the Knowledge written by agronomists and rate them.
Figure 46 shows the Help section. In this tab the farmer can send a request for help to the agronomist in your area by specifying a title, the category and a more detailed description of the problem.
Figure 47 shows the Production section. In this tab the farmer can view the overview of his production, the farm soil data and the instantaneous data obtained from the installed sensors.

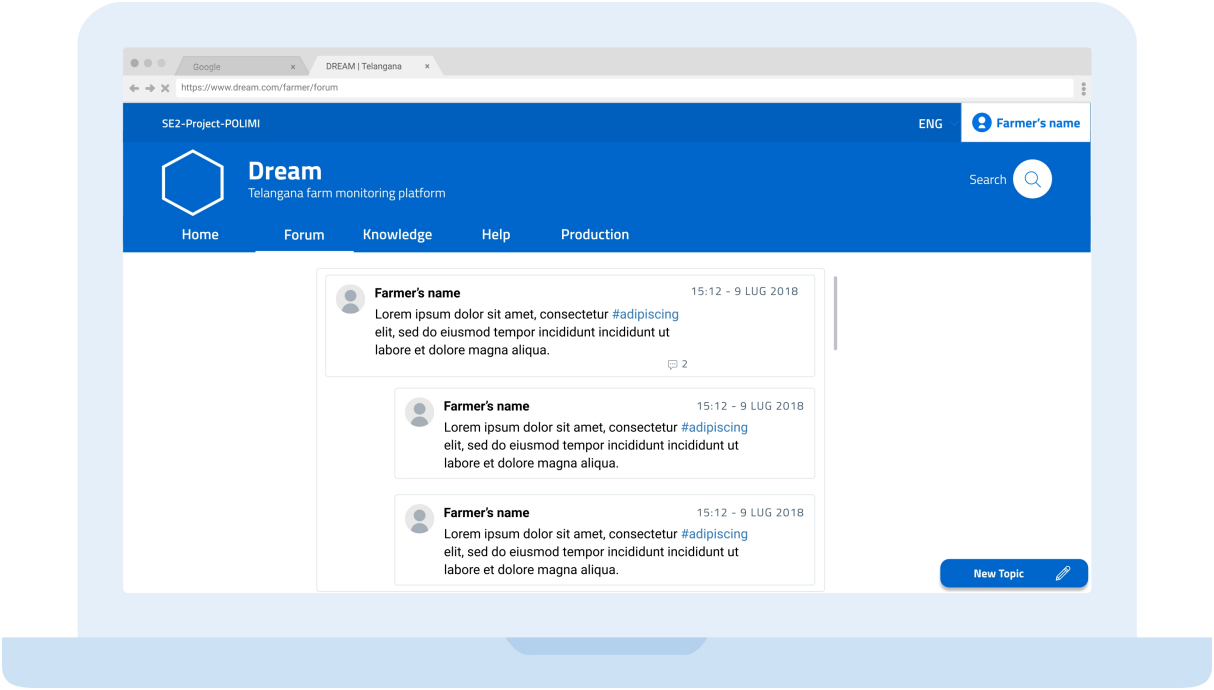


Figure 44: Mock-up - Forum page

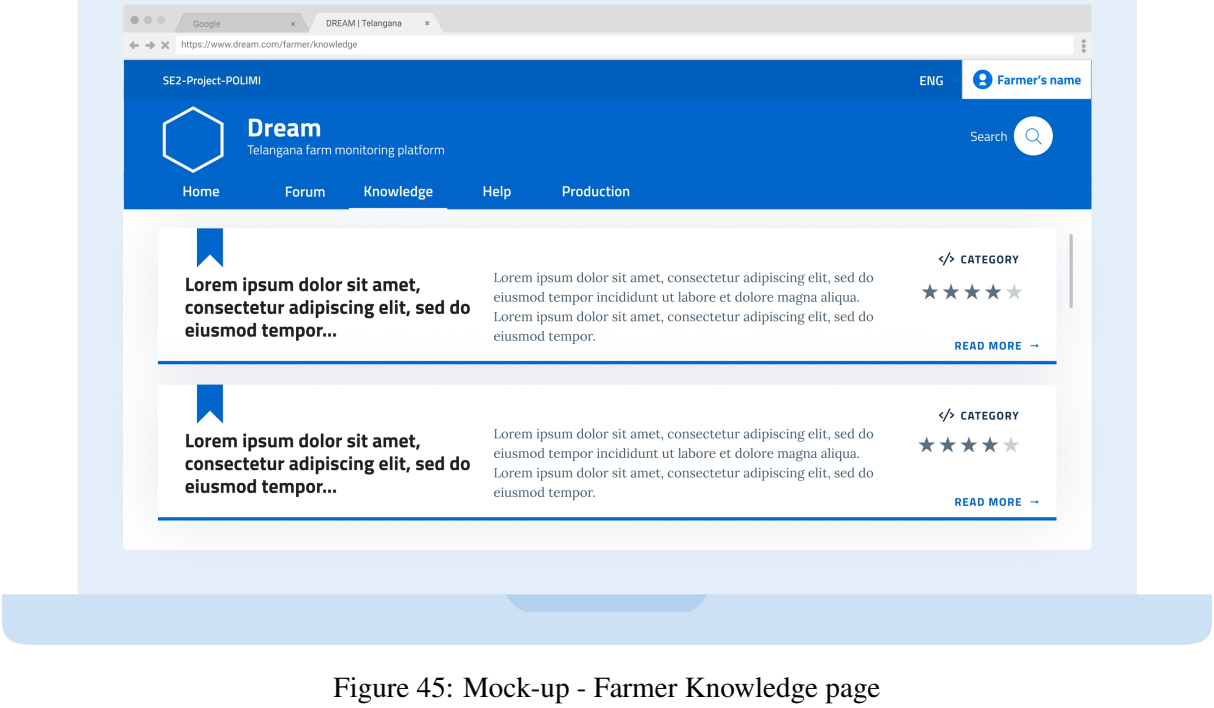


Figure 45: Mock-up - Farmer Knowledge page

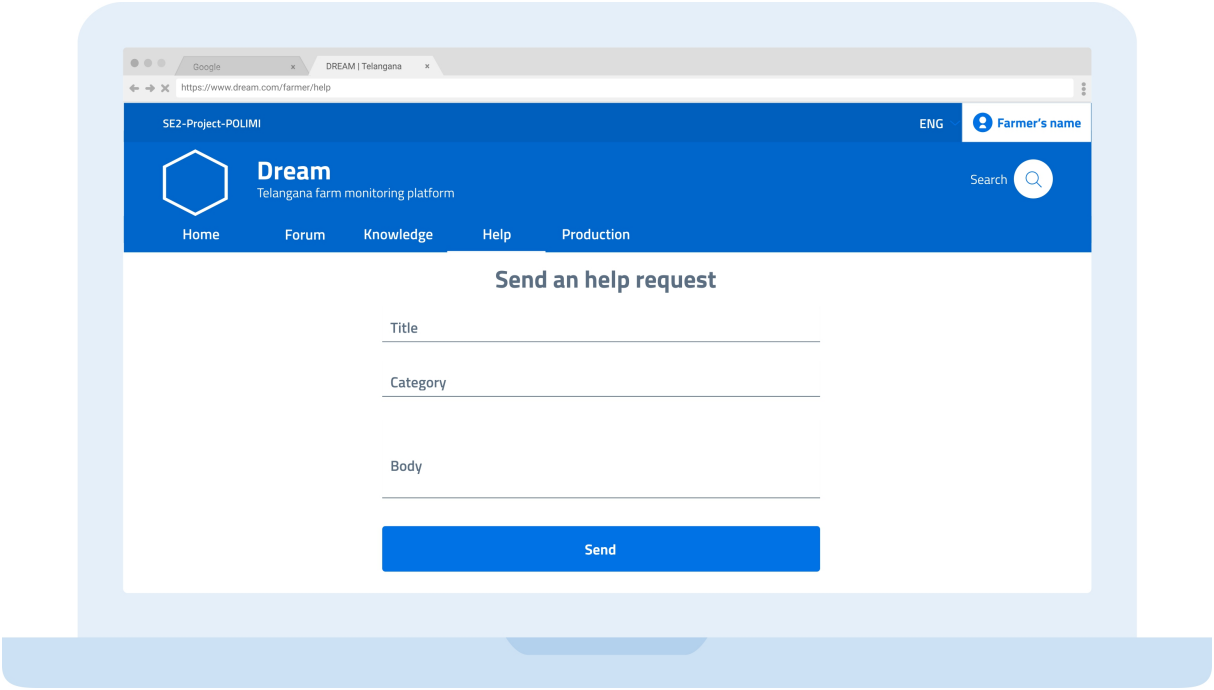


Figure 46: Mock-up - Help page

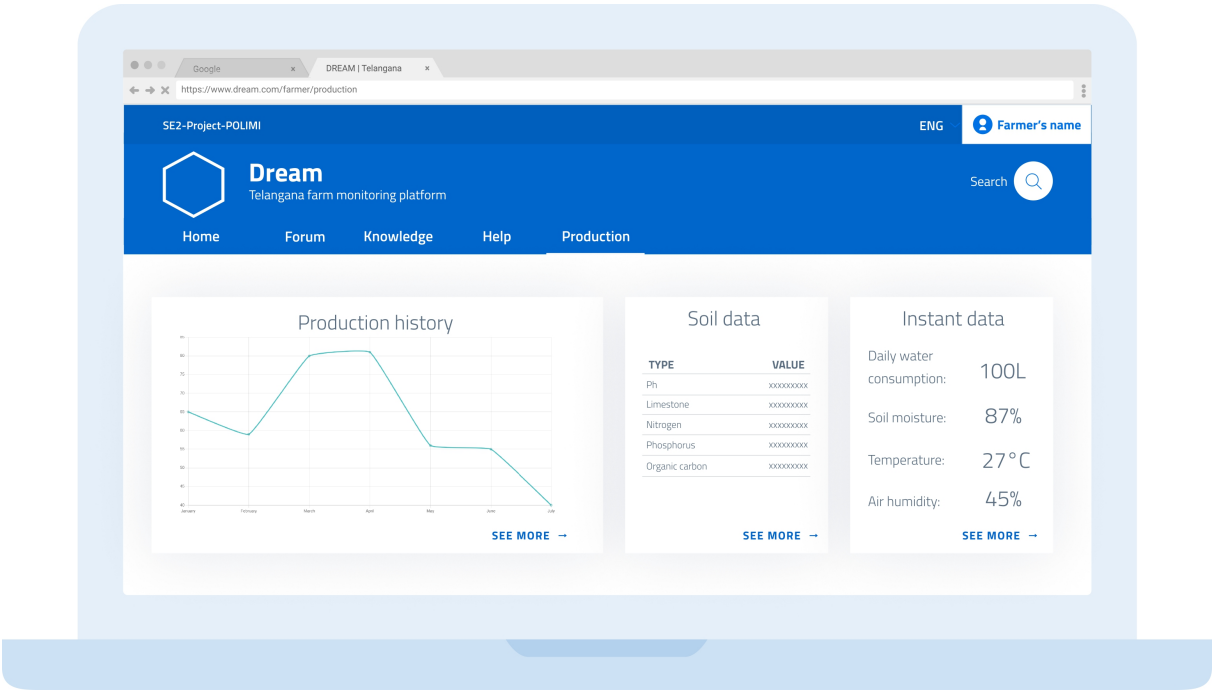


Figure 47: Mock-up - Production page

3.4 Mock-up Agronomist's Web Application

After logging in, the agronomist will find himself on his home page as shown in figure 48. It shows an overview of the key functions of the web application. This page contains several subsections, such as the weather, the latest help requests, a summary of upcoming meetings and some notifications. Furthermore, like every screen, this too is composed of a header where the accessible tabs are present, allowing to change screens based on the functionality of the page.

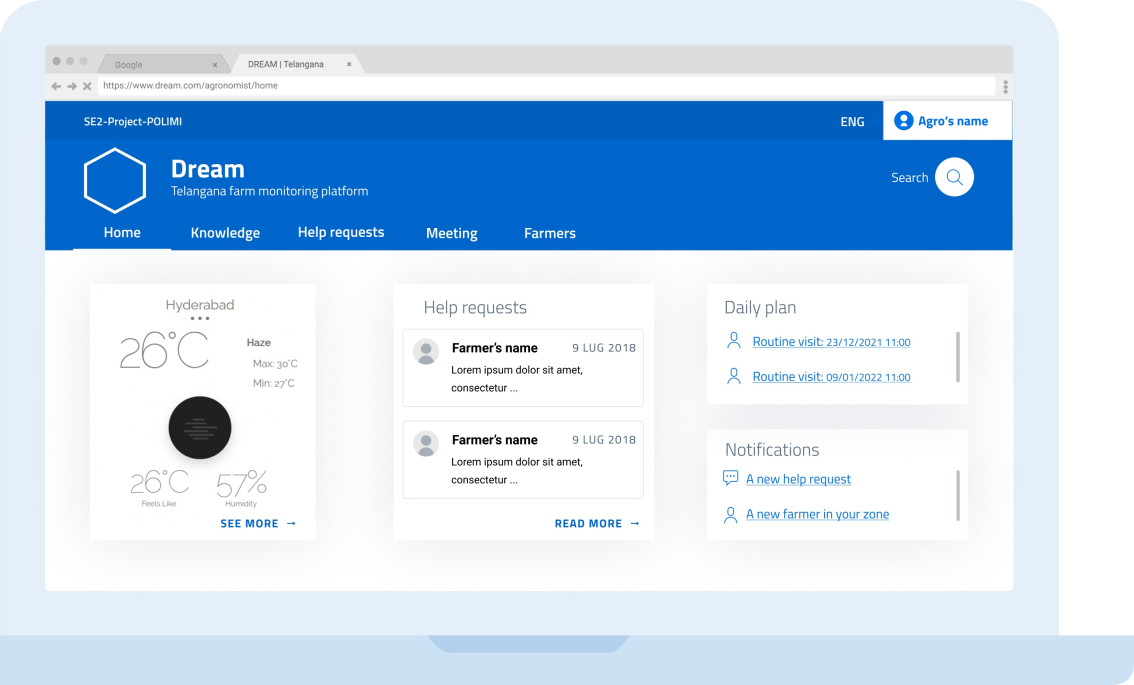


Figure 48: Mock-up - Agronomist Home page

Figure 49 shows the agronomist Knowledge section. In this tab the agronomist can view his own Knowledge, modify or delete them and write new ones.
Figure 50 shows the agronomist Help requests section. In this tab the agronomist can view the requests for help sent by the farmers in his area and respond to them.
Figure 51 shows the agronomist Meeting section. In this tab the agronomist can view the next meetings with the farmers, modify or delete them and create new ones.
Figure 52 shows the agronomist Farmer details section. In this tab the agronomist can view the details of the farmers in his area in tabular form showing the surname, first name, address of the farm and the resilience score. It is also possible to view alerts for the farmer where some important information is indicated such as that a visit is needed or that he has sent a request for help. All these data can also be organized alphabetically or numerically, ascending or descending. There is also a button at the bottom right where you can confirm the insertion of a new farmer in the system after the first visit.

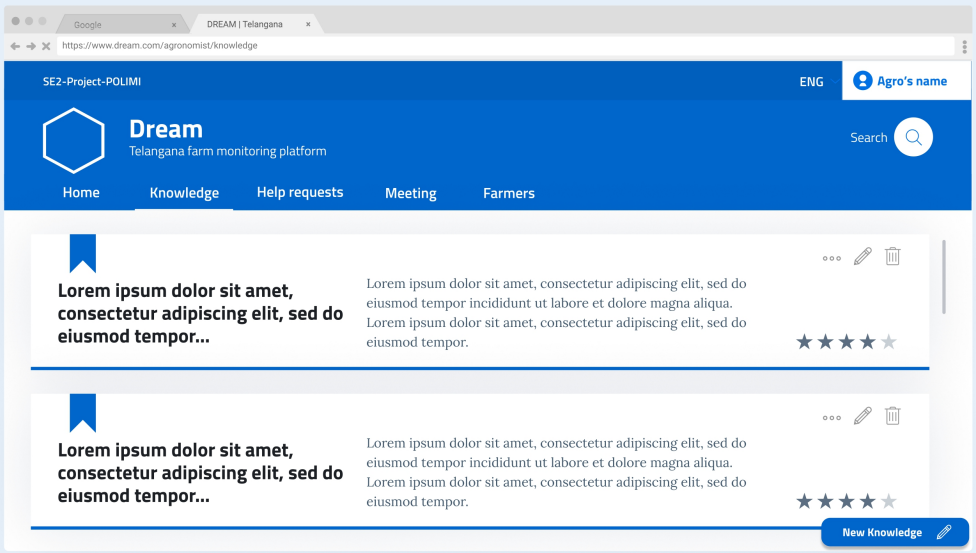


Figure 49: Mock-up - Agronomist Knowledge page

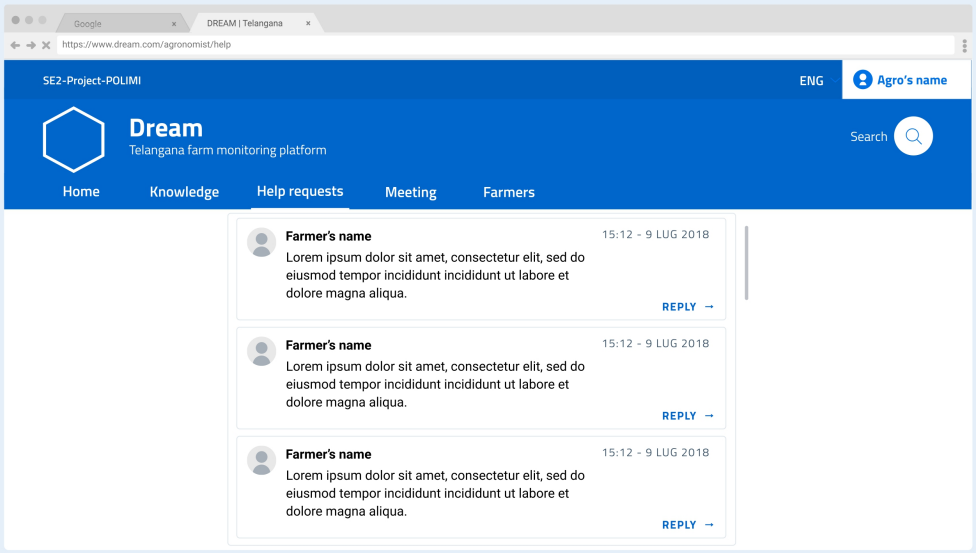


Figure 50: Mock-up - Help requests page

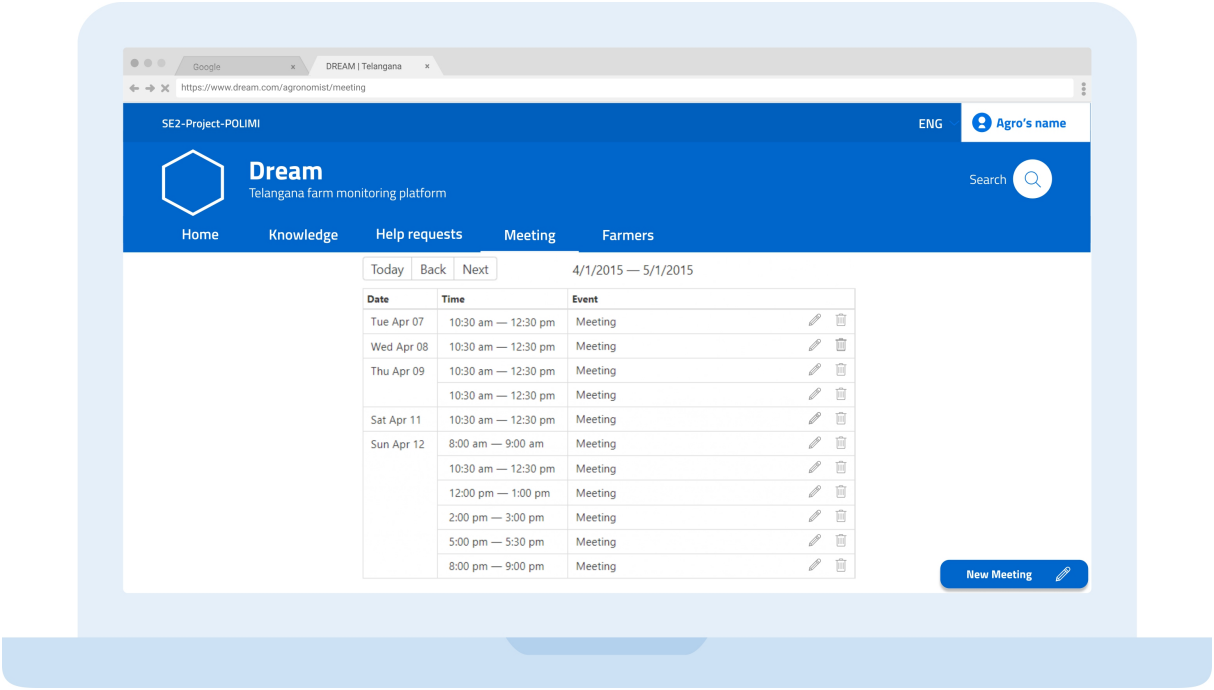


Figure 51: Mock-up - Meeting page

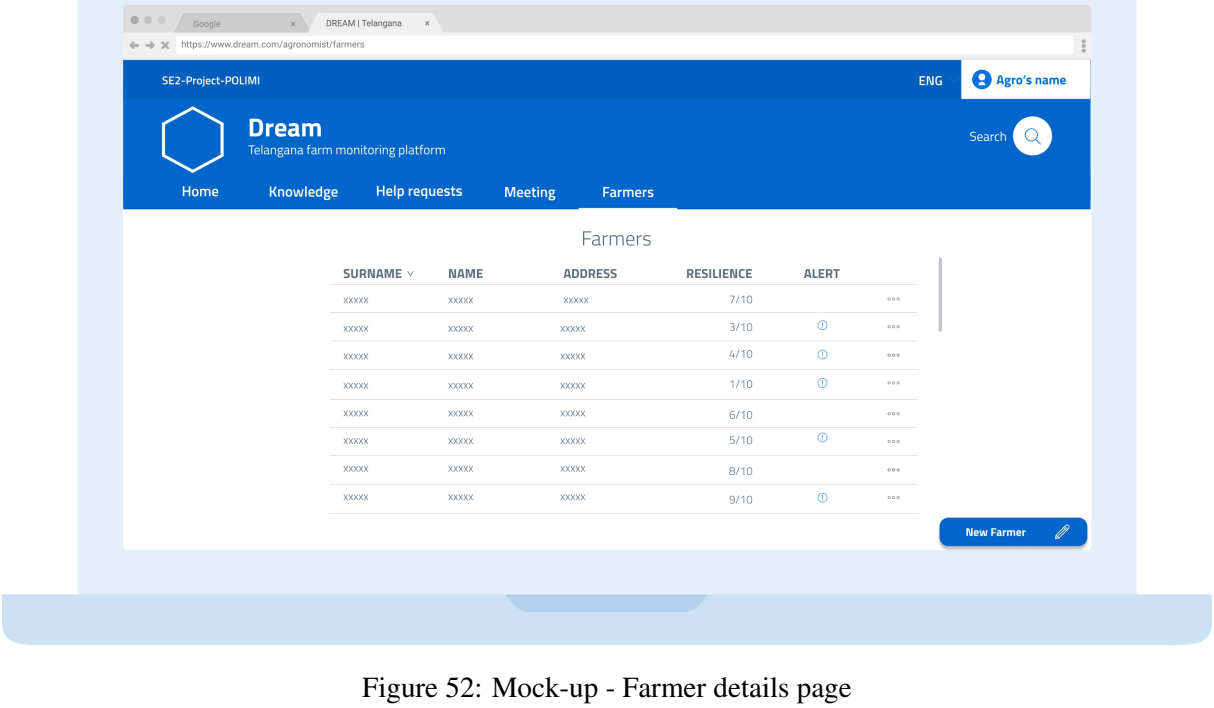


Figure 52: Mock-up - Farmer details page

3.5 Mock-up Policy Maker's Web Application

After logging in, the policy maker will find himself on his home page as shown in figure 53. In this tab the PM can view the details of the farmers in the country in tabular form showing the surname, first name, address of the farm and the resilience score. It is also possible to view alerts for the farmer where some important information is indicated such as, for example, that he is performing worse than last period. All these data can also be organized alphabetically or numerically, ascending or descending. There are also filters on the right where you can search farmers by first or last name or filter by area.

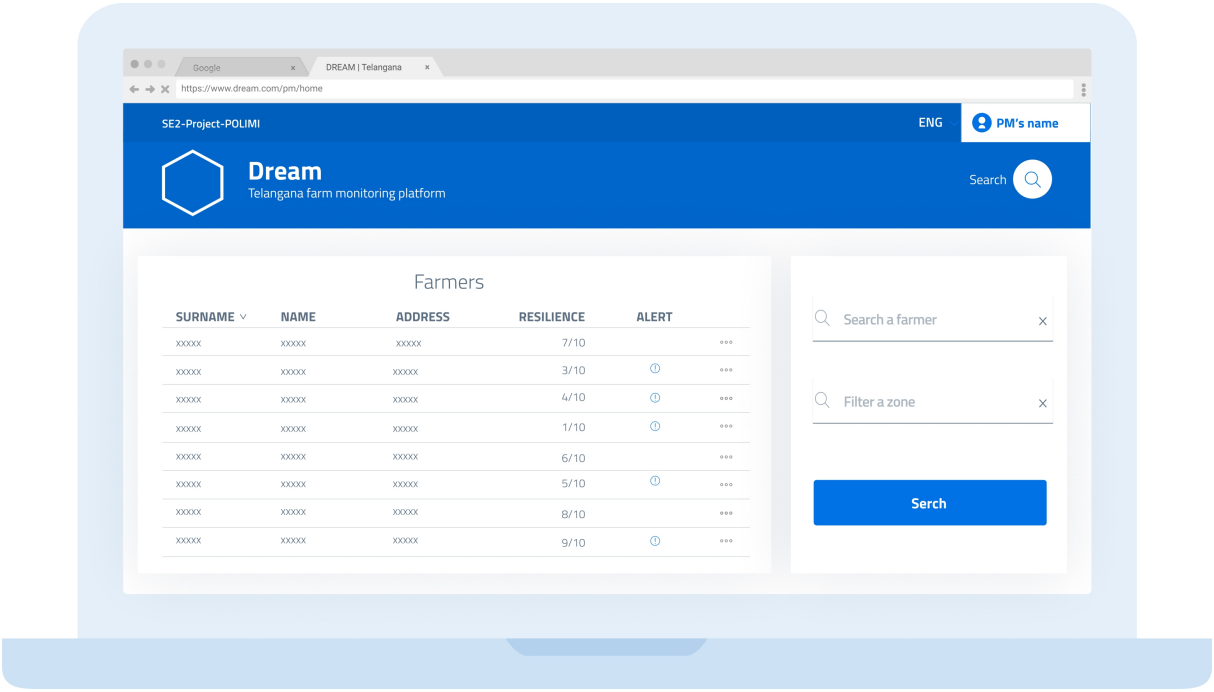


Figure 53: Mock-up - PM Home page

4 Requirements Traceability

The mapping between the requirements given in the RASD paper and the components mentioned above are provided in this section. First, a high-level overview is provided, allowing you to check that the stated components meet all of the requirements. Then a more in-depth examination is taken. Note that to make the schema more clear we put together Controller, Service and Repository since they all work together and they would be always coupled up.

Requirement	Description
C.1	Farmer WebApp
C.2	Agronomist WebApp
C.3	PolicyMaker WebApp
C.4	AgronomistController/Service/Repository
C.5	CommentController/Service/Repository
C.6	FarmController/Service/Repository
C.7	LocationController/Service/Repository
C.8	LoginController/Service
C.9	ProductController/Service/Repository
C.10	ProductionController/Service/Repository
C.11	SoilDataController/Service/Repository
C.12	TopicController/Service/Repository
C.13	KnowledgeController/Service/Repository
C.14	RequestController/Service/Repository
C.15	MeetingController/Service/Repository
C.16	FarmerController/Service/Repository
C.17	NotificationController
C.18	GradingController
C.19	NotificationAgronomistService/Repository
C.20	NotificationFarmertService/Repository
C.21	LikeCommentService/Repository
C.22	LikeKnowledgeService/Repository
C.23	DBMS
C.24	WeatherService
C.25	SensorService

	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**
C1	X						X			X
C2		X	X	X	X	X		X	X	
C3										
C4		X	X	X	X	X		X	X	
C5										
C6	X									
C7										
C8	X	X	X	X	X	X	X	X	X	X
C9	X									
C10	X	X								
C11										
C12										
C13										
C14										
C15			X	X	X	X			X	X
C16	X						X			X
C17							X	X		
C18										
C19								X		
C20						X				
C21										
C22										
C23	X	X	X	X	X	X	X	X	X	X
C24										
C25										

	R11	R12	R13	R14	R15	R16	R17	R18	R19	R20
C1	X			X	X	X	X	X		X
C2		X	X						X	
C3										
C4										
C5										
C6										
C7										
C8	X	X	X	X	X	X	X	X	X	X
C9										
C10										
C11										
C12					X	X	X			
C13									X	X
C14	X	X	X	X						
C15										
C16										
C17										
C18										
C19										
C20										
C21										
C22										
C23	X	X	X	X	X	X	X	X	X	X
C24								X		
C25										

	R21	R22	R23	R24	R25	R26	R27
C1							X
C2	X	X			X	X	
C3			X				
C4							
C5							
C6							
C7							X
C8	X	X	X	X	X	X	X
C9							
C10							
C11							
C12							
C13							
C14							
C15		X					
C16					X		
C17							
C18		X	X				
C19						X	
C20							
C21							
C22							
C23	X	X	X	X	X	X	X
C24	X						
C25							

G1	Allowing farmer to share their knowledge and problems with other farmers and help them with their farm by creating and commenting a topic
R15	The system allows the farmer to publish a topic on the forum
R16	The system allows all farmers to view the forum
R17	The system allows all farmers to publish an answer to a topic on the forum
R24	The system requires a sign up and a login to access to the data
R27	The system allows the farmer to insert their position when they first register
C1	Farmer WebApp
C8	LoginController/Service
C12	TopicController/Service/Repository
C23	DBMS
C7	LocationController/Service/Repository

G2	Allowing farmer to keep track of their production
R1	The system allows the farmer to add data concerning his production
R24	The system requires a sign up and a login to access to the data
R27	The system allows the farmer to insert their position when they first register
C1	Farmer WebApp
C6	FarmController/Service/Repository
C8	LoginController/Service
C9	ProductController/Service/Repository
C10	ProductionController/Service/Repository
C16	FarmerController/Service/Repository
C23	DBMS
C7	LocationController/Service/Repository

G3	Showing the farmer personalized advice regarding specific subjects of their need, shared by the agronomist
R2	The system allows the agronomist to view the data concerning their farmers' production
R19	The system allows the agronomists to share knowledge with the farmer
R20	The system allows the farmers to visualize the knowledges shared by their agronomist
R24	The system requires a sign up and a login to access to the data
R27	The system allows the farmer to insert their position when they first register
C2	Agronomist WebApp
C4	AgronomistController/Service/Repository
C8	LoginController/Service
C10	ProductionController/Service/Repository
C23	DBMS
C13	KnowledgeController/Service/Repository
C7	LocationController/Service/Repository
C1	Farmer WebApp

G4	Allowing farmers to make a request for help to agronomists
R11	The system allows the farmer to send a help request to the agronomist
R14	The system allows the farmer to view all the help requests sent by him
R24	The system requires a sign up and a login to access to the data
R27	The system allows the farmer to insert their position when they first register
C1	Farmer WebApp
C8	LoginController/Service
C14	RequestController/Service/Repository
C23	DBMS
C7	LocationController/Service/Repository
yellow!25**C19**	DBMS Service

G5	Allowing agronomists to help farmers in their area by answering to their request
R12	The system allows the agronomist to answer to the help requests that are sent to him
R13	The system allows the agronomist to view all the help requests sent to him
R24	The system requires a sign up and a login to access to the data
R25	The system shows to each agronomist the list of all farmers they are responsible for
R26	The system notifies the agronomists when a new farmer has registered under their area
R27	The system allows the farmer to insert their position when they first register
C2	Agronomist WebApp
C8	LoginController/Service
C14	RequestController/Service/Repository
C23	DBMS
C16	FarmerController/Service/Repository
C7	LocationController/Service/Repository

G6	Allowing agronomists to help farmers in their area by sharing their knowledge and giving them advice
R2	The system allows the agronomist to view the data concerning their farmers' production
R19	The system allows the agronomists to share knowledge with the farmer
R20	The system allows the farmers to visualize the knowledges shared by their agronomist
R21	The system allows all agronomists to check the weather forecasts for their area
R24	The system requires a sign up and a login to access to the data
R25	The system shows to each agronomist the list of all farmers they are responsible for
R26	The system notifies the agronomists when a new farmer has registered under their area
R27	The system allows the farmer to insert their position when they first register
C2	Agronomist WebApp
C4	AgronomistController/Service/Repository
C8	LoginController/Service
C10	ProductionController/Service/Repository
C23	DBMS
C13	KnowledgeController/Service/Repository
C1	Farmer WebApp
C24	WeatherService
C16	FarmerController/Service/Repository
C19	NotificationAgronomistService/Repository
C7	LocationController/Service/Repository

G7	Allowing agronomists to plan visits with farmers to check their progress and if needed, help them
R3	The system allows the agronomist to book an appointment with the farmer by showing all available slots
R4	The system allows the agronomist to modify a booked appointment with the farmer
R5	The system allows the agronomist to cancel a booked appointment with a farmer
R6	The system check that the agronomist booked at least two appointments per year with each farmer
R7	The system alerts the farmer before the appointment
R8	The system alerts the agronomist before the appointment
R9	The system allows the agronomist to view the calendar
R10	The system allows the farmer to view the calendar
R24	The system requires a sign up and a login to access to the data
R25	The system shows to each agronomist the list of all farmers they are responsible for
R26	The system notifies the agronomists when a new farmer has registered under their area
R27	The system allows the farmer to insert their position when they first register
C2	Agronomist WebApp
C4	AgronomistController/Service/Repository
C8	LoginController/Service
C15	MeetingController/Service/Repository
C23	DBMS

C20	NotificationFarmerService/Repository
C1	Farmer WebApp
C16	FarmerCntroller/Service/Repository
C17	NotificationController
C19	NotificationAgronomistService/Repository
C7	LocationController/Service/Repository

G8	Allowing policy makers to monitor the performance of the farmers to help them or to give them special incentives to encourage them to share their experience
R22	The system allows the agronomists to publish a grade for each farmer after each visit
R23	The system allows the policy makers to view the grades of all farmers of Telangana
R24	The system requires a sign up and a login to access to the data
C3	PolicyMaker WebApp
C8	LoginController/Service
C15	MeetingController/Service/Repository
C18	GradingController
C23	DBMS
C3	PolicyMaker WebApp

G9	Showing customers weather forecasts allowing them to have a better approach with the climate changes
R18	The system allows the farmers to check the weather forecasts for their area
R24	The system requires a sign up and a login to access to the data
R27	The system allows the farmer to insert their position when they first register
C1	Farmer WebApp
C7	LocationController/Service/Repository
C8	LoginController/Service
C23	DBMS
C24	WeatherService

5 Implementation, Integration, and Test Plan

This section aims to give a view of what the plan for the implementation part of the system will be. Also, the integration and testing phases are explained in detail.

5.1 Implementation

The implementation of the system will follow a strategy of thread type. This kind of implementation is thought to be composed of many portions (or threads) of different modules which, working together, will provide the desired functionalities. This means that a thread at a time will be integrated, until the end. Please note that an explanation of the components can be found in the Component view Paragraph (2.2) and to keep the document more slim we decided to just leave the name of the components here. Also, note that to keep the diagrams more clear the folder "Repositories" is meant to comprehend all the repositories previously considered and that they will be developed along with their correspondent Controller and Service. The order of implementation will be the following:

1. DBMS
2. Farmer WebApp
3. Agronomist WebApp
4. PolicyMaker WebApp
5. SoilDataRepository
6. SoilDataController
7. SoilDataService
8. AgronomistRepository
9. AgronomistController
10. AgronomistService
11. FarmRepository
12. FarmCotroller
13. FarmService
14. LocationRepository
15. LocationController
16. LocationService
17. ProductRepository
18. ProductController
19. ProductService
20. ProductionRepository
21. ProductionController
22. ProductionService

23. FarmerRepository
24. FarmerController
25. FarmerService
26. TopicRepository
27. TopicController
28. TopicService
29. MeetingRepository
30. MeetingController
31. MeetingService
32. NotificationAgronomistRepository
33. NotificationFarmerRepository
34. NotificationController
35. NotificationAgronomistService
36. NotificationFarmerService
37. RequestRepository
38. RequestController
39. RequestService
40. CommentRepository
41. CommentController
42. CommentService
43. LikeCommentRepository
44. LikeCommentService
45. LikeKnowledgeRepository
46. LikeKnowledgeService
47. KnowledgeRepository
48. KnowledgeController
49. KnowledgeService
50. LoginController
51. LoginService
52. ExternalComponents

5.2 Integration

In the following paragraph, the process of integration of the components previously described will be discussed and more details will be given.
Obviously, the infomation shuld be coherent with the thread approach and the implementation order just described.

- **First Phase:** The basis of several components will be developed, including the Database, the Repositories, the Farmer WebApp, the Agronomist WebApp, and the PolicyMaker WebApp.
- **Second Phase:** Here, the majority of controllers and the corresponding services will be developed. In parallel the development of the Web Apps, of the repositories and the integration in the DBMS will continue. In detail, SoilDataController and Service, The AgronomistController and Service, the FarmController and Service, the LocationController and Service, the ProductController and Service, the ProductionController and Service and the FarmerController and Service.
- **Third Phase:** Here, new Controllers and Services will be added and the development of the WebApp and repositories will continue. The TopicController and Service, the MeetingController and Service, the NotificationController, NotificationAgronomistService and NotificationFarmerService, the RequestController and Service, the KnowledgeController and Service, the CommentController and Service, the LikeCommentService and the LikeKnowledgeService will be added.
- **Fourth Phase:** Eventually, the External Components the LoginController and the LoginService will be developed while completing the development of the Web Apps, repositories and DBMS.

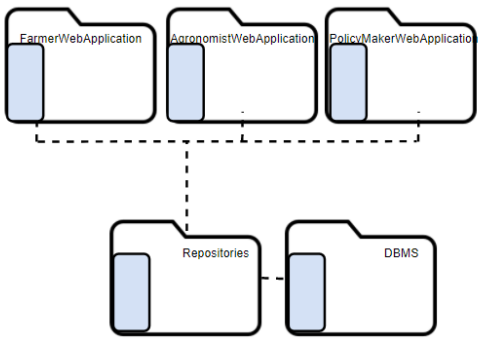


Figure 54: First Phase

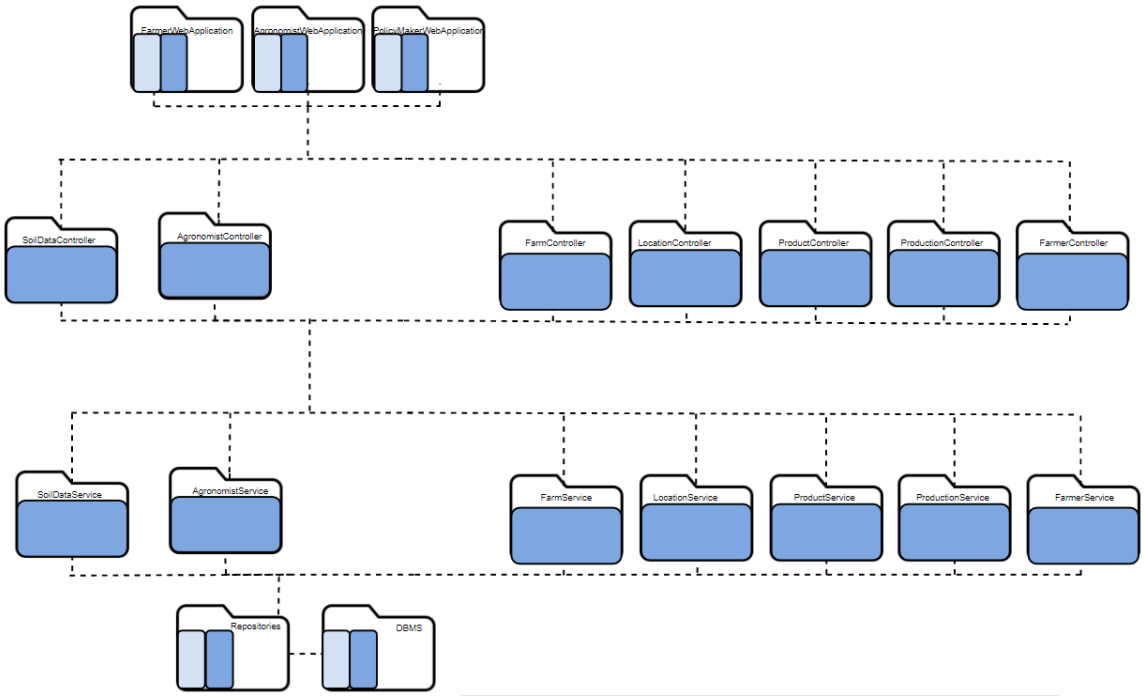


Figure 55: Second Phase

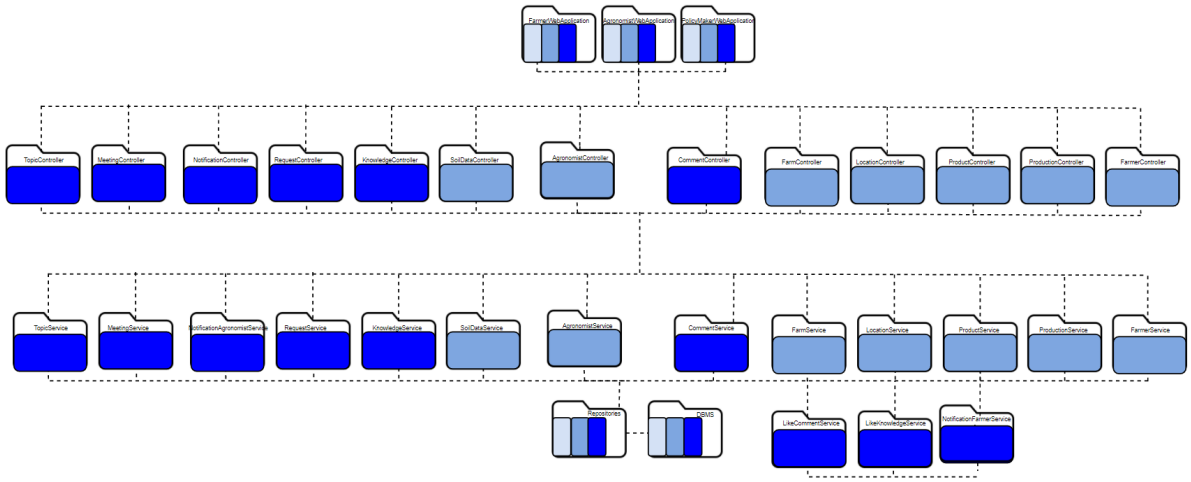


Figure 56: Third Phase

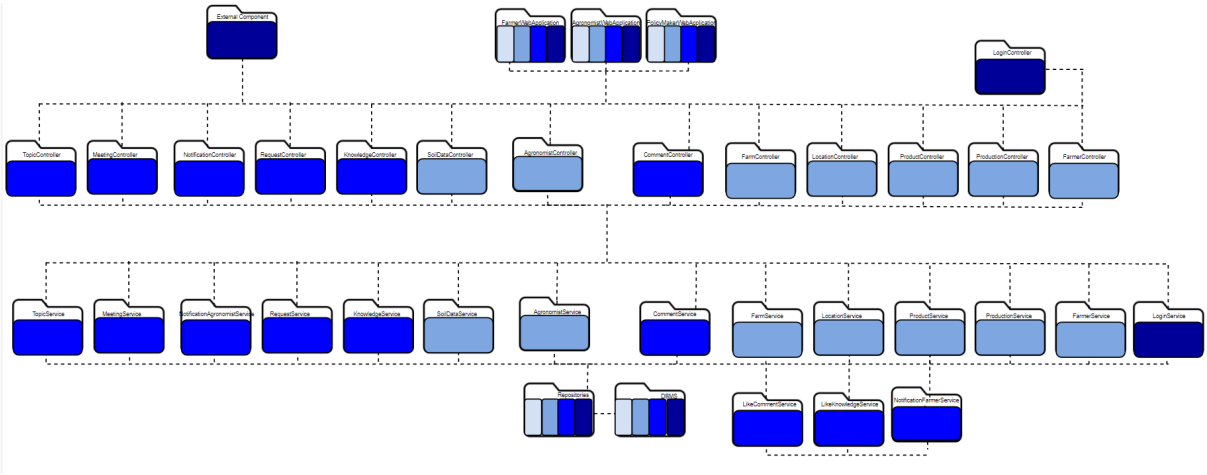


Figure 57: Fourth Phase

5.3 Testing

In the previous section the phases of implementation and integration have been discussed and the testing process will follow the same pattern by performing it while the various components will be developed and released.
The goal is to perform the testing of each component once it gets integrated in the system with all the others already integrated. During this time, the next component is supposed to be developed and, in case the test of the previous component is passed, the newest component will be integrated and tested as well. Eventually, the whole system will be tested in order to verify the functioning of the whole system all together.

6 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 (or contributors, if more than one) for each topic but should not be interpreted as a lack of participation by other team members for that topic.

Topic	**Member**	**Hours**
General initial brainstorming and creation of the document	Buttiglione, Caffagnini, Faouzi	5
Introduction integration	Caffagnini	1
Arhitectural Design: Overview	Caffagnini	3
Component definition and sketch of Component Diagram	Buttiglione, Caffagnini, Faouzi	4
Component Diagram improvements	Faouzi	3.5
Deployment View	Faouzi	3
Runtime view diagram	Buttiglione, Caffagnini, Faouzi	25
Component Interfaces	Buttiglione	4
Research on architectures and patterns	Caffagnini	3
Other Design Decision	Buttiglione	1
Mock-Ups and User Interface Design	Buttiglione	5
Requirement traceability	Faouzi	4
Implementation, Integration and Test plan	Buttiglione, Caffagnini, Faouzi	5
Implementation, Intergation and Testing Integration	Caffagnini	3
Final Improvement	Buttiglione, Caffagnini, Faouzi	6
Second version's improvements	Buttiglione, Caffagnini, Faouzi	40

7 References

- The diagrams have been made with: https://www.visual-paradigm.com/ and https://lucid.co/it
- The mockups have been made with: https://www.figma.com/
- UI Design: https://designers.italia.it/kit/progettazione-interfaccia/
- Sequence Diagram Reference: https://www.uml-diagrams.org/sequence-diagrams-reference.html
- IBM – Three-Tier Architecture - https://www.ibm.com/cloud/learn/three-tier-architecture

