



POLITECNICO DI MILANO

Computer Science and Engineering

# Design Document

DREAM

Software Engineering 2 Project

Academic year 2021 - 2022

08/01/2022

Version 1.0

*Authors:*

Stefano Staffolani,  
Stefano Taborelli,  
Matteo Viafora

*Professor:*

Damian Andrew Tamburri

## Revision History

---

<b>Deliverable:</b>	DD
<b>Title:</b>	Design Document
<b>Authors:</b>	Stefano Staffolani, Stefani Taborelli e Matteo Viafora
<b>Version:</b>	1.0 - First Release
<b>Date:</b>	08-January-2022
<b>Download page:</b>	<a href="#">Github</a>
<b>Copyright:</b>	Copyright © 2022, Staffolani, Taborelli, Viafora – All rights reserved

---

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Purpose	4
1.2	Scope	4
1.3	Glossary	5
1.3.1	Definitions	5
1.3.2	Acronyms	5
1.3.3	Abbreviations	5
1.4	Reference Documents	6
1.5	Document Structure	6
<b>2</b>	<b>Architectural Design</b>	<b>7</b>
2.1	Overview	7
2.2	Component View	8
2.2.1	High Level	8
2.2.2	DREAM Server	9
2.2.3	Sensor Server	10
2.3	Deployment view	11
2.3.1	Presentation Tier	11
2.3.2	Business Logic Tier	11
2.3.3	Data Management Tier	12
2.3.4	Load Balancer	12
2.4	Run-time View	12
2.4.1	Login	12
2.4.2	Sensor Interaction	13
2.4.3	Report a Problem (Farmer)	14
2.4.4	Report a problem (TPM/Agronomist)	15
2.4.5	WikiFarm	16
2.4.6	Weather	17
2.4.7	Report Production (Farmer)	18
2.4.8	Report Production (Agronomist/TPM)	19
2.4.9	Help (Ask)	20
2.4.10	Forum	20
2.4.11	Agenda	21
2.4.12	Visualize Initiatives	22
2.4.13	Ranking	23
2.4.14	Chat	24
2.5	Component Interface	25
2.5.1	DREAM Server	25
2.5.2	Sensor Server	29
2.6	Selected architectural styles and patterns	30
2.7	Other Design Decision	31
2.7.1	Database Structure	31
<b>3</b>	<b>User Interface Design</b>	<b>32</b>
3.1	Farmer's WebApp	32
3.2	Agronomist's WebApp	36
3.3	TPM's WebApp	39
3.4	Demo	41
<b>4</b>	<b>Requirement Traceability</b>	<b>42</b>

4.1	Mapping Interfaces on Requirements . . . . .	42
<b>5</b>	<b>Implementation, Integration and Test Plan . . . . .</b>	<b>45</b>
5.1	Plan Definition . . . . .	45
5.1.1	Introducing the model and the Entity Manager . . . . .	45
5.1.2	Authentication Implementation . . . . .	45
5.1.3	Adding Functionalities . . . . .	46
5.1.4	Integrating the Forecast Service . . . . .	47
5.1.5	Testing the Sensor . . . . .	48
5.1.6	Web App Integration . . . . .	49
5.1.7	Final Testing . . . . .	50
5.2	Technologies Used . . . . .	51
5.2.1	Database . . . . .	51
5.2.2	WebApp . . . . .	51
5.2.3	DREAM Server e Sensor Server . . . . .	51
5.2.4	Testing Tool . . . . .	51
<b>6</b>	<b>Effort Spent . . . . .</b>	<b>52</b>
6.1	Team Work . . . . .	52
6.2	Stefano Staffolani . . . . .	52
6.3	Stefano Taborelli . . . . .	52
6.4	Matteo Viafora . . . . .	52

# 1 Introduction

## 1.1 Purpose

Agriculture plays a pivotal role in India's economy as over 58% of rural households depend on it as the principal means of livelihood, 80% of whom are smallholder farmers with less than 5 acres of farmland. More than a fifth of the smallholder farm households are below poverty. The COVID-19 pandemic has greatly highlighted the massive disruption caused in food supply chains exposing the vulnerabilities of marginalized communities, small holder farmers and the importance of building resilient food systems. It has become even more important now that we develop and adopt innovative methodologies and technologies that can help bolster countries against food supply shocks and challenges.

The purpose of Data-dRiven prEdictive fArMing (DREAM) is to help Telangana's government in designing, developing, and demonstrating anticipatory models for food systems using digital public goods and the community of agricultural worker. To achieve this DREAM allows the communication between the users, gives the possibility to retrieve data from the sensors in Telangana, to track the evolutions of the work of farmers, and to access the data from farmers or from any other sources.

This document contains an explanation of the design decisions made for the entire system, from the general architecture to the specific components and interfaces.

## 1.2 Scope

Data-dRiven prEdictive fArMing (DREAM) is an easy-to-use and intuitive application which aims to settle for various problems faced by the member of the agricultural community of Telangana.

It allows Telangana's Policy Makers (TPM) to access data and analyze the performance of all farmers in the country, identify which farmers are coping well with adversity and which ones need help, understanding which initiatives are performing best, in order to replicate them in the whole Nation, and they can contact the resilient farmers to give them special incentives.

To improve their cultivations, farmers can get personalized suggestions, report any problem they encountered during their work, request more support by an agronomist or another farmer, discuss in the about any topic in the forum, but they can also provide their data of production.

Agronomists can use this system to increase the effectiveness of their work as they can manage their daily work and keep track of the visits they have planned throughout the year and evaluating the performance of farmers they are responsible of. Furthermore, they have access to the weather forecast and the data collected by the system to better take decisions with farmers or receive requests for help from farmers.

## 1.3 Glossary

This section explains the terms used throughout the document.

### 1.3.1 Definitions

Term	Definition
<b>Resilient Farmers</b>	Farmers who performed well, despite they faced an adverse weather event
<b>Bad Farmers</b>	Farmers that are not able to sustain their production
<b>Problem faced by farmers</b>	Any difficulty that a farmer faced during his/her work, i.e., a bug infestation of the cultures
<b>Type of Production</b>	How the cultivation is grown, i.e., Shifting Cultivation, intensive, and many more
<b>Data of Production</b>	Information concerning a single cultivation of a farmer. It includes tons produced per acre of land used, fertilizer used, humidity of the soil, type of production, water usage, seed planted and the zone of the cultivation
<b>Credentials</b>	Username and Password given by Telangana to a user to access DREAM
<b>Zone</b>	It refers to a District and a Mandala

### 1.3.2 Acronyms

Acronym	Term
<b>DREAM</b>	Data-dRiven PrEdictive FArMing in Telangana
<b>TPM</b>	Telangana Policy Makers
<b>API</b>	Application Programming Interface
<b>GUI</b>	Graphical User Interface

### 1.3.3 Abbreviations

Abbreviation	Term
<b>e.g.</b>	Exempli Gratia
<b>i.e.</b>	Id est
<b>R</b>	Requirement

## 1.4 Reference Documents

Below is presented the list of all the documents used to create this document.

- Project assignment specification document
- DREAM, Requirements Analysis and Specification Document.
- ISO/IEC/IEEE 29148 - Systems and software engineering
- Course slides on WeBeeP
- UNDP India's [Github](#)
- Telangana's Forecast [WebPage](#)
- Slide of the course: "Sistemi Informativi"
- Slide of the course: "Distributed System"
- [WebPage](#) to support the choice of load balancers.
- Slide of the course: "Ingegneria del Software"

## 1.5 Document Structure

This document is presented as it follows:

1. **Introduction** contains an overall description of the main functionalities of the system and the main focus of this document.
2. **Architecture Design** contains a description of the architecture used to implement the system. It goes from a general, high-level description to the composition of the database and how the individual components work.
3. **User Design Interface** contains a representation of the system's WebApp GUI.
4. **Requirement Traceability** contains a cross-reference that traces components to the requirements contained in RASD document.
5. **Implementation, Integration and Test Plan** presents a description of how the individual components and the functioning of the whole system will be implemented, integrated and tested.
6. **Effort Spent** keeps track of the time spent to complete this document. The first table defines the hours spent as a team for taking the most important decisions or reviewing contents, the seconds contain the individual hours spent working on this project

## 2 Architectural Design

### 2.1 Overview

The system will be developed from scratch and the architectural choices that have been considered during the design of DREAM reflect the distributed nature of the system. Indeed, the functionalities offered by the system shall be used at the same time by various users located across the Telangana state. In order to maintain the scalability and ease to use experience of the application a three-tier approach has been implemented.

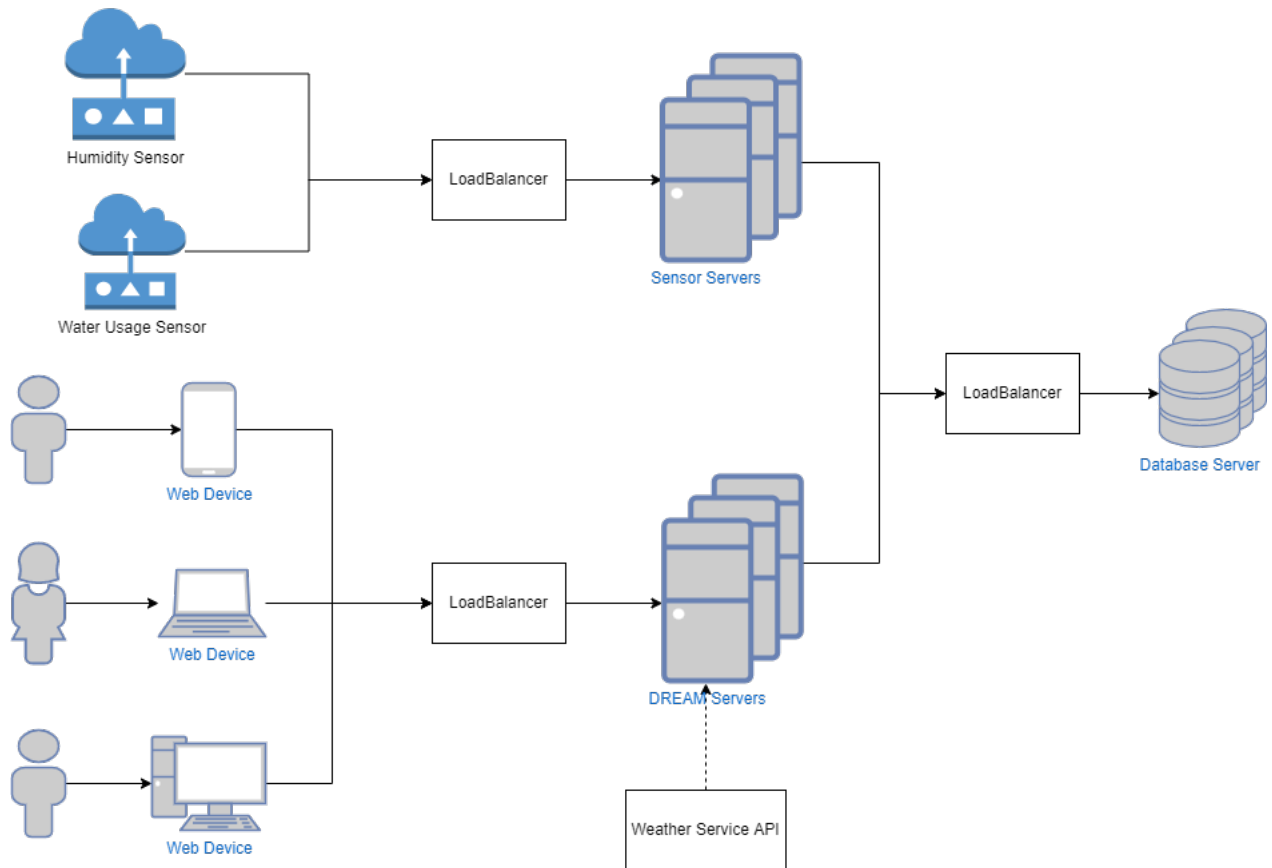


Figure 1: *Overview Description.*

The high-level system consists of:

- **Web Device:** Device enabled for internet connection and with a web browser installed to access the system. I.e., it can be a PC or a smartphone.
- **Water Usage Sensor:** Sensor used to detect the water consumed by a farmer.
- **Humidity Sensor:** Sensor used to detect moisture in the soil.
- **Weather Service API:** API used to access the weather information requested by the user.
- **Sensor Server:** Server used to manage communication between the sensors and the database.
- **DREAM Server:** Server where all the logic is located. It communicates with other servers and is the central point of the system.
- **Database Server:** Server where all the data are stored.



- **LoadBalancer:** Server with the only task of redirecting the load in an equal manner between the components.

## 2.2 Component View

### 2.2.1 High Level

The following component diagram highlights all the components of the system and describes both its internal and external interactions. A general overview of the whole system is shown in Figure.

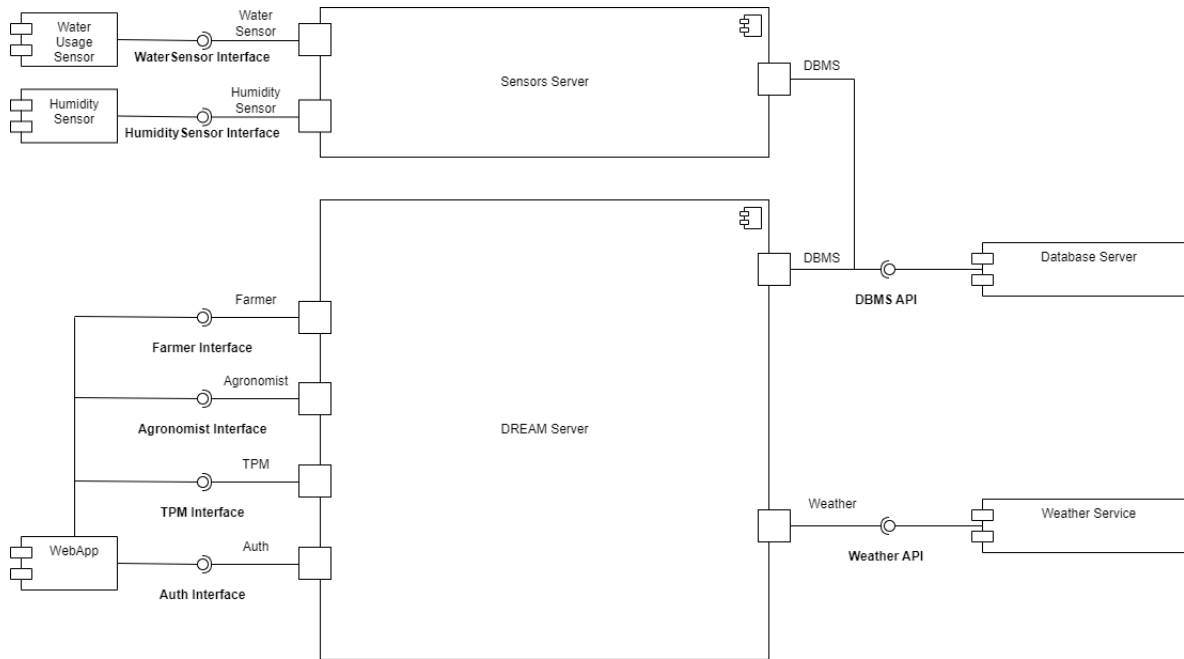


Figure 2: *High Level Overview.*

The components shown in Figure are:

- **DREAM Server:** DREAM Server: it contains the business logic of the entire system. This component allows each user to use the services offered by DREAM, and also retrieves the data contained in the database and accesses the Telangana meteorological service.
- **WebApp** represents the interface with which users can access DREAM, via their preferred web browser. After the user has logged in via **Auth Interface**, it guarantees the correct access to the platform according to the functionalities that the user can use. I.e., **Farmer Interface** allows access to a farmer to "Report Production".
- **WaterUsageSensor** is a sensor that detects the flow of water flowing in a pipe. The sensor will be installed at the water supply point for each farmer in order to calculate his/ her exact consumption. The device is equipped with a 2G SIM to connect to DREAM and exchange information and to avoid excessive energy consumption.
- **HumiditySensor** is a sensor that detects the amount of moisture present in the soil. Each plot will be equipped with one of these sensors, and to make the user experience easier, sensors will be used that do not require a continuous source of electricity but can be self-sufficient through the use of mini solar panels. In addition, the device is equipped with a 2G SIM to connect to DREAM and exchange information, and to avoid excessive energy consumption.

- **Sensor Server** is the server responsible for communicating the data collected by the sensors distributed throughout Telangana and the database.
- **Database Server** provides the Sensor Server and DREAM Server with access to the data in the system.
- **Water Service** provides the interface to DREAM to retrieve the weather forecasts requested by a farmer or agronomist.

## 2.2.2 DREAM Server

The following component diagrams describe the internal structure of the application server, which contains the business logic of the system.

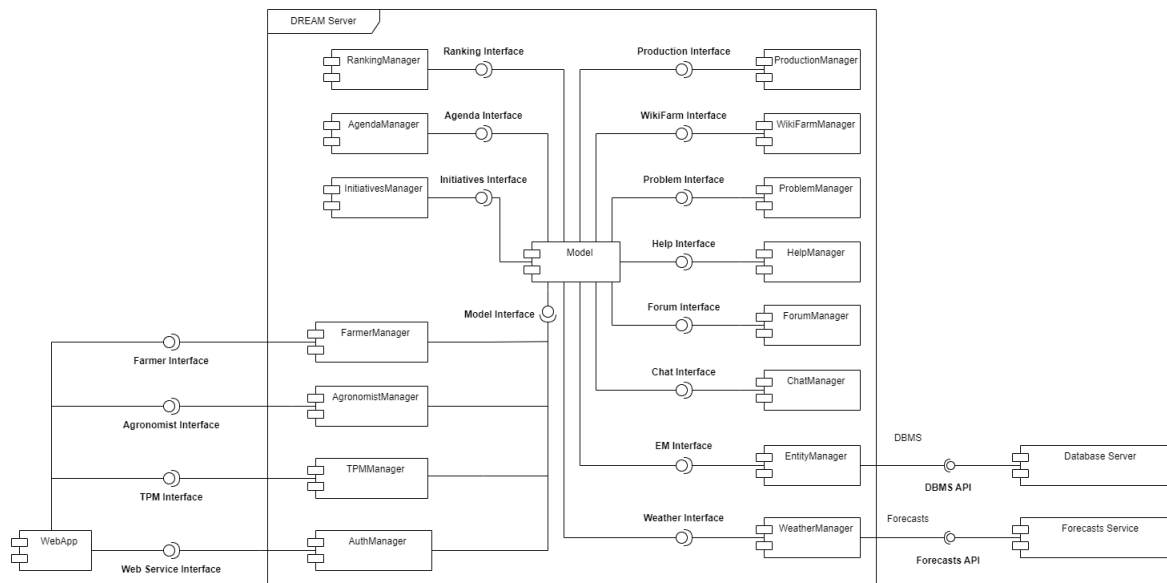


Figure 3: *DREAM Server Schema.*

The component used in the DREAM Server are:

- **FarmerManager:** Component that manages the requests from a farmer's WebApp to functionality that it can access.
- **AgronomistManager:** Component that manages the requests from an agronomist WebApp to functionality that it can access.
- **TPMManager:** Component that manages the requests from a TPM WebApp to functionality that it can access.
- **AuthManager:** Component that verifies that the credentials entered by the user are correct.
- **Model:** This is the central component of DREAM Server. It generates the authentication token for each user and is responsible for managing all interactions between the server modules and, therefore, for querying the correct component according to the function desired by the user. In addition, it is the component that is responsible for querying the **EntityManager** and the **Telangana Weather Service API**.
- **RankingManager:** Component that computes with the ranking of the farmers present in an area selected by a user.

- **AgendaManager:** Component that allows manage the personal agenda of an agronomist. It allows to insert a new event, view the events scheduled for a selected day, confirm or modify an event saved in the diary and upload the report of a visit to a farmer that has just been completed.
- **InitiativesManager:** Component that allows a TPM to retrieve all the reports uploaded by agronomists to help a farmer.
- **Production Manager:** Component that allows a farmer to upload production data for his farm.
- **WikiFarm:** Component that suggests the most suitable fertilizers and seeds in the selected area.
- **ProblemManager:** Component that allows a farmer to upload details about a problem he/she has encountered. It also allows TPM and agronomists to view all the problems encountered by farmers in an area.
- **HelpManager:** Component that allows a farmer to enter a help request and allows agronomists and TPMs to view problems that have occurred in an area they have requested.
- **ForumManager:** Component that allows a farmer to create a new discussion or that allows him/her to reply to an already started discussion.
- **ChatManager:** Component that allows two users to communicate with each other.
- **EntityManager:** Component that allows two users to communicate with each other.
- **WeatherManager:** Component responsible for finding the weather forecast for a specific time slot requested by a user, by connecting to the Telangana weather portal.

### 2.2.3 Sensor Server

The following component diagrams describe the internal structure of the Sensor Server.

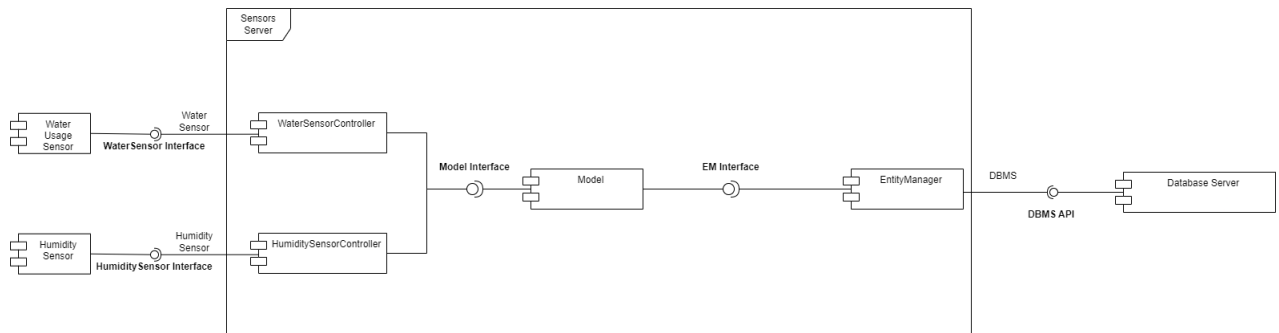


Figure 4: *Sensor Server Schema.*

The component used in the Sensor Server are:

- **WaterSensorController:** Component responsible for communication between the server and the water usage sensor
- **HumiditySensor:** Component responsible for communication between the server and the humidity sensor in a terrain.
- **Model:** It is the central component of the Sensor Server, it is responsible for managing the flow of information between the sensors to the EntityManager.
- **EntityManager:** Component responsible for managing the interactions between the database and the Sensor Server.

## 2.3 Deployment view

In the following section, the deployment of DREAM is discussed with the visual help of a deployment diagram. This architecture offers three tiers, which are separated by two layers of load balancers. This choice will be motivated in the following section.

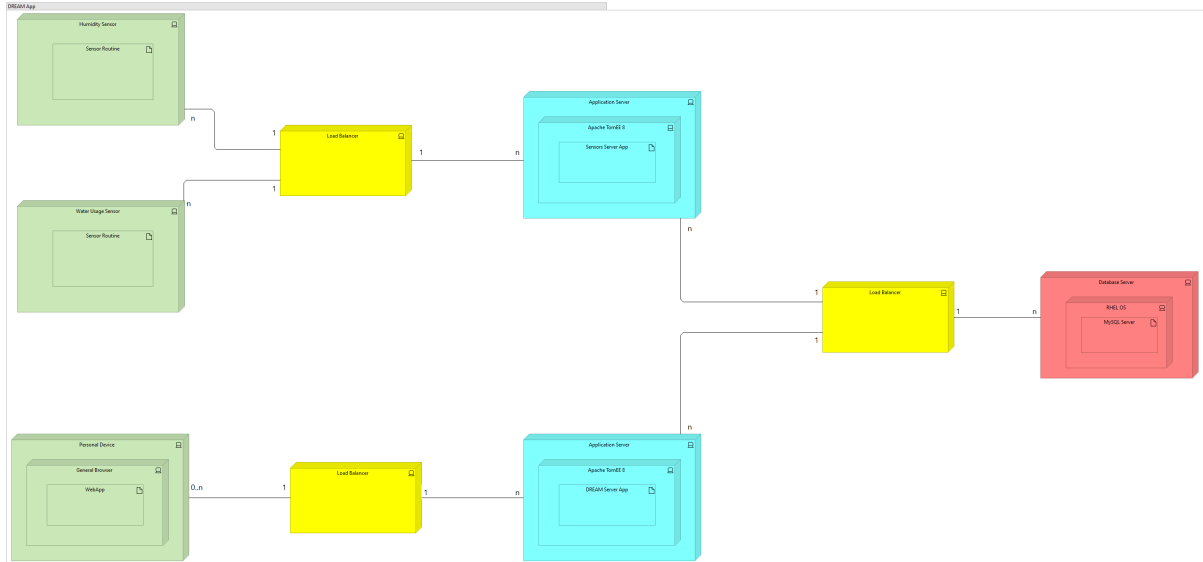


Figure 5: *Deployment View of the System.*

### 2.3.1 Presentation Tier

The components of the presentation tier are presented below:

- **Humidity Sensor and Water Usage Sensor:** These components only interact with the Sensors Server to send data acquired from their routine.
- **Personal Device:** DREAM's system can be accessed from any device that utilizes any web browser. This will be the front-end layer, which consists of the user interface (this will be different for each role). The web-based approach has been chosen in order to guarantee access from any kind of device, with no restriction on the OS and without minimum requirements.

### 2.3.2 Business Logic Tier

The business logic tier contains the logic that drives the application's core capabilities. The components are:

- **Sensor Server:** The role of this component is to receive the data routinely sent from the sensors and send them in an organized manner to the Third Tier, the database. The artifact for this component should be light, and it's based on Apache TomEE 8.
- **DREAM Server:** This is the main logic component. This server has to manage each request from the First Tier, and it has to interact with the database to retrieve and update data. It also has the job to generate a different token for each role (i.e. Agronomists), which will then be used to show the correct user interface, and to guarantee access only to the allowed requests. The DREAM Server App is installed on a dedicated server with a running instance of Apache TomEE 8.

### 2.3.3 Data Management Tier

The data management tier comprises the database/data storage system and data access layer. Data is stored in a persistent way into a SQLServer database and accessed by the business logic tier via API calls.

### 2.3.4 Load Balancer

The role of the load balancers that divide the First and the Second Tier is to guarantee a correct load distribution between the multiple instances of the servers. Each server component can be duplicated in order to guarantee a more efficient managing of the requests. Both the Sensors Server and the DREAM Server could have as many instances as required, but in order to keep the system as light (and economic) as possible only two instances per server are required. This also provides an improved availability, since if only one of the two instances requires maintenance, the system can continue to work.

The load balancers that divide the Second and the Third Tier have a slightly different role. In fact, since many components require access to the database, either to retrieve data or to update it, three instances of the database are implemented, with one that act as a primary database and periodically sends “Sync Requests” to the replicas. This means that the load balancers will redirect every “write” operation to the primary database, while the “read” operations can be divided between the primary and the replicas.

## 2.4 Run-time View

### 2.4.1 Login

WebApp calls AuthManager, sending the credentials as parameters. AuthManager will then hash the password, and send the credentials to Model, which will create a query to verify the correctness of the data. If the inserted credentials are correct, both Model and WebApp will receive a token, which will then be used for every further interaction.

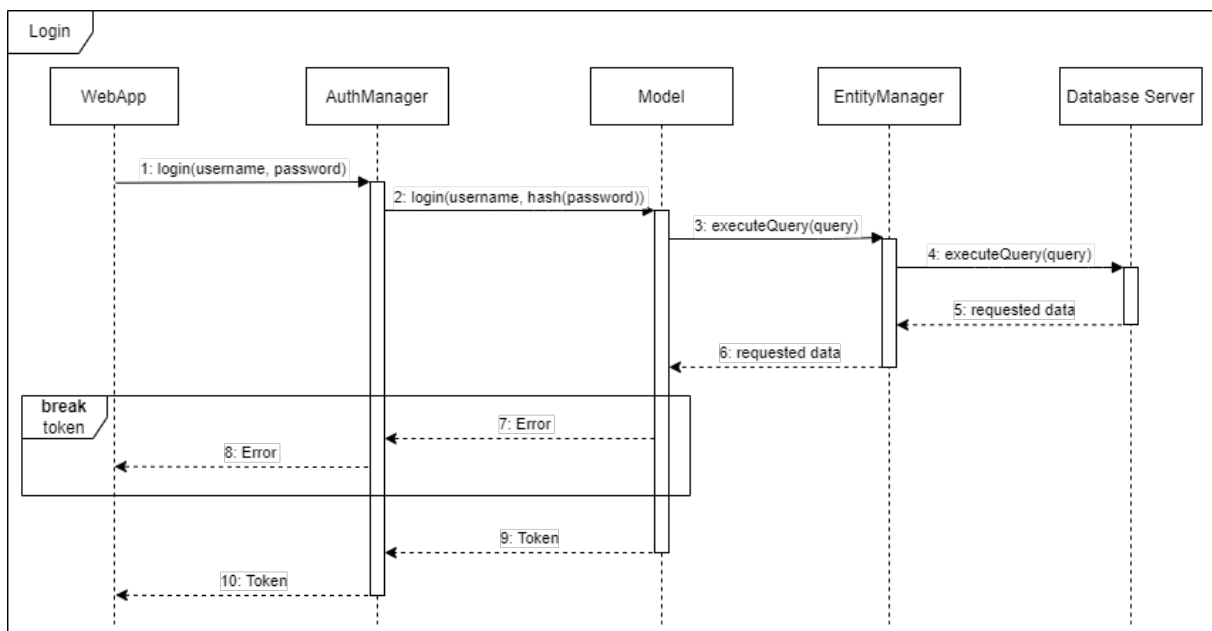


Figure 6: *LogIn* Sequence Diagram.

### 2.4.2 Sensor Interaction

Sensors will routinely send the acquired data to the SensorController, which will send this information to the Model along with the type of sensor that sent the data. The model will then create the appropriate query to update the database, which will be executed by the EntityManager.

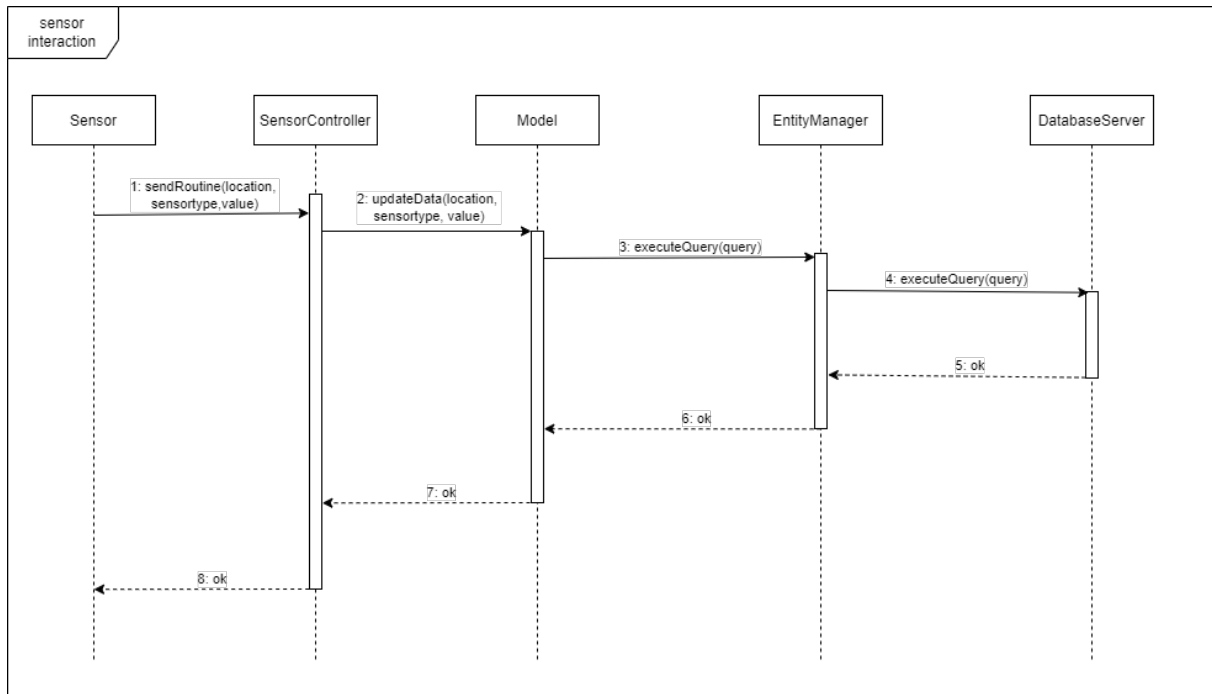


Figure 7: *Sensor* Sequence Diagram.

### 2.4.3 Report a Problem (Farmer)

WebApp will send a request to FarmerManager, which will contain the title and the description of the problem, along with the token. FarmerManager sends the token to the Model, which verifies it and sends the result back to FarmerManager. This interaction happens every time a request is made by any user, and in order to improve readability it will not be reported in further diagrams. FarmerManager will then send to the Model the title and description of the problem, which will call ProblemManager in order to generate the query to insert it in the database.

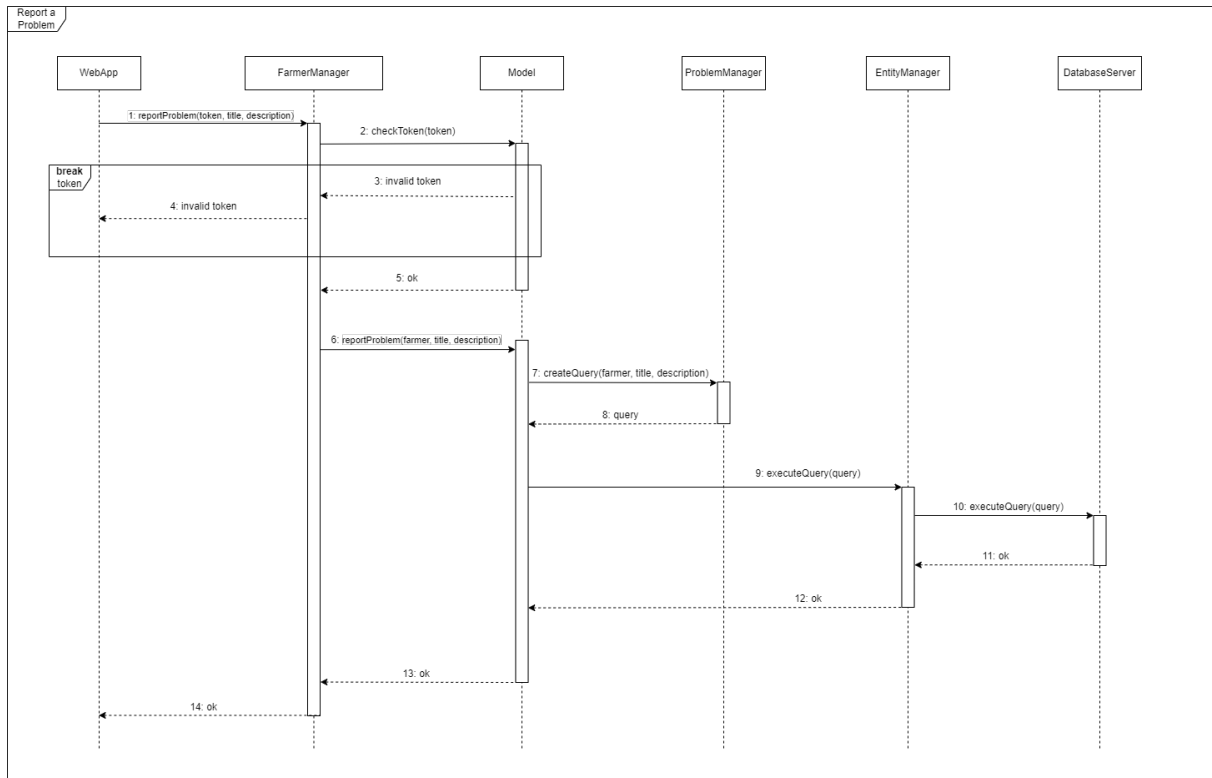


Figure 8: *Report a Problem(Farmer)* Sequence Diagram.

## 2.4.4 Report a problem (TPM/Agronomist)

After the token is verified, Model calls ProblemManager in order to create the query to retrieve the list of problems reported in the selected zone. The user will select the problem to open, and, after the token gets verified again, ProblemManager creates the query to retrieve all the data concerning the problem.

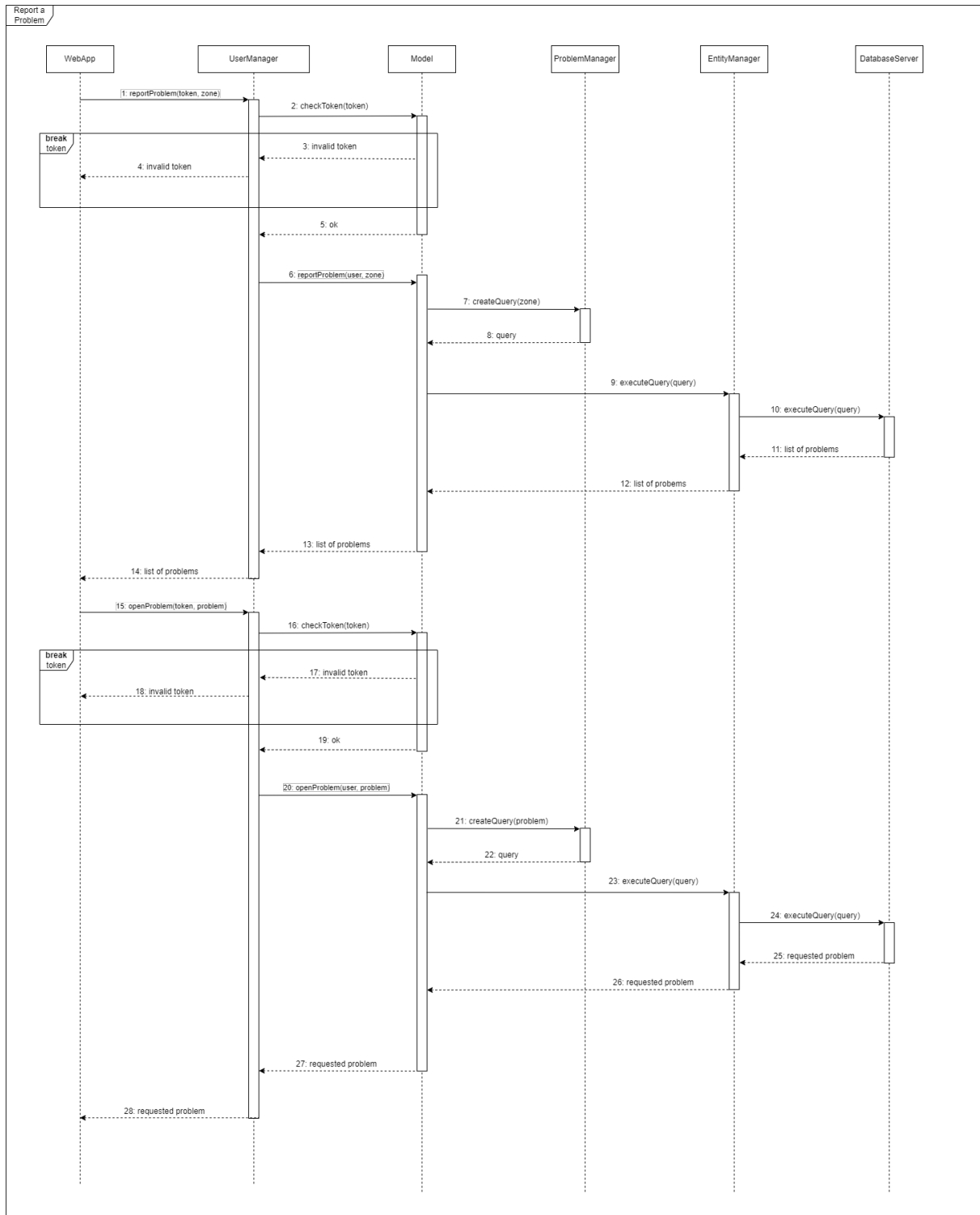


Figure 9: *Report a Problem*(Agronomist/TPM) Sequence Diagram.



### 2.4.5 WikiFarm

After the token is verified, Model calls WikiFarm to create a query that selects the best performing farmers in the selected zone which have a cultivation with the selected type of production. The query will then retrieve the fertilizers and the seeds employed by these farmers.

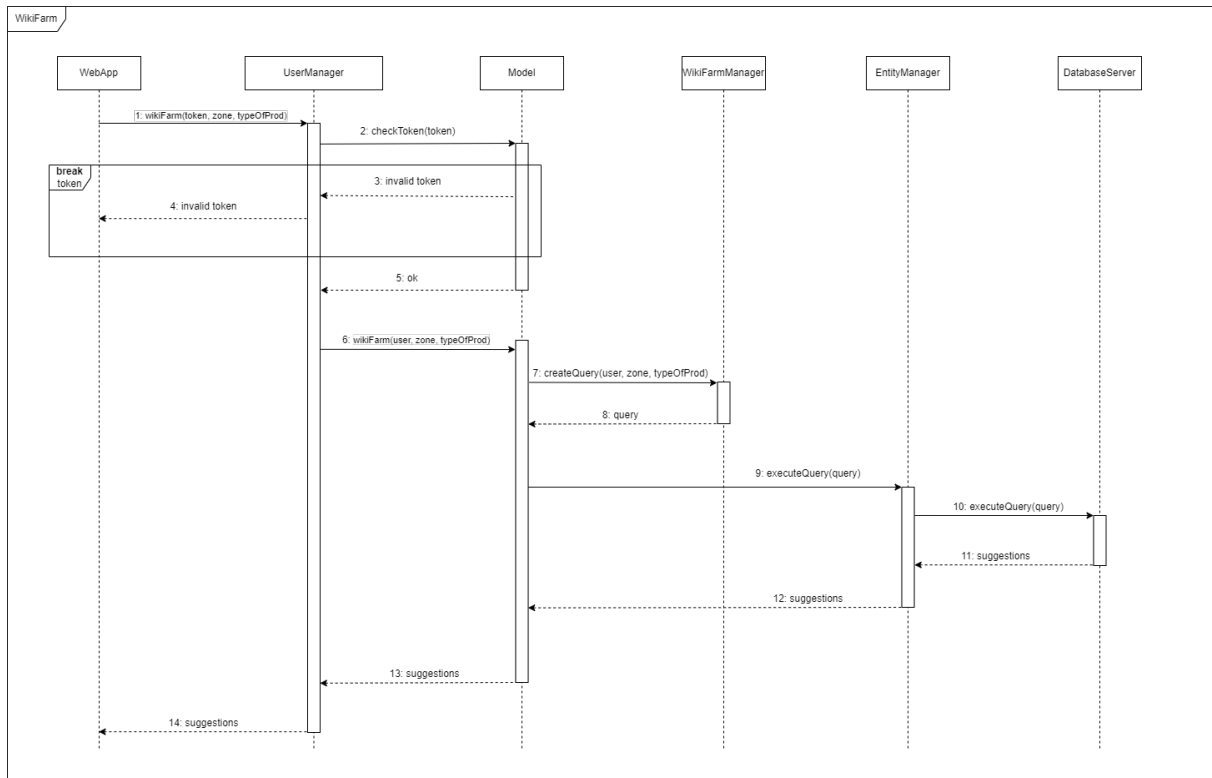


Figure 10: *WikiFarm* Sequence Diagram.

### 2.4.6 Weather

After the token is verified, the Model calls WeatherManager, which interacts with WeatherService in order to retrieve the requested forecast.

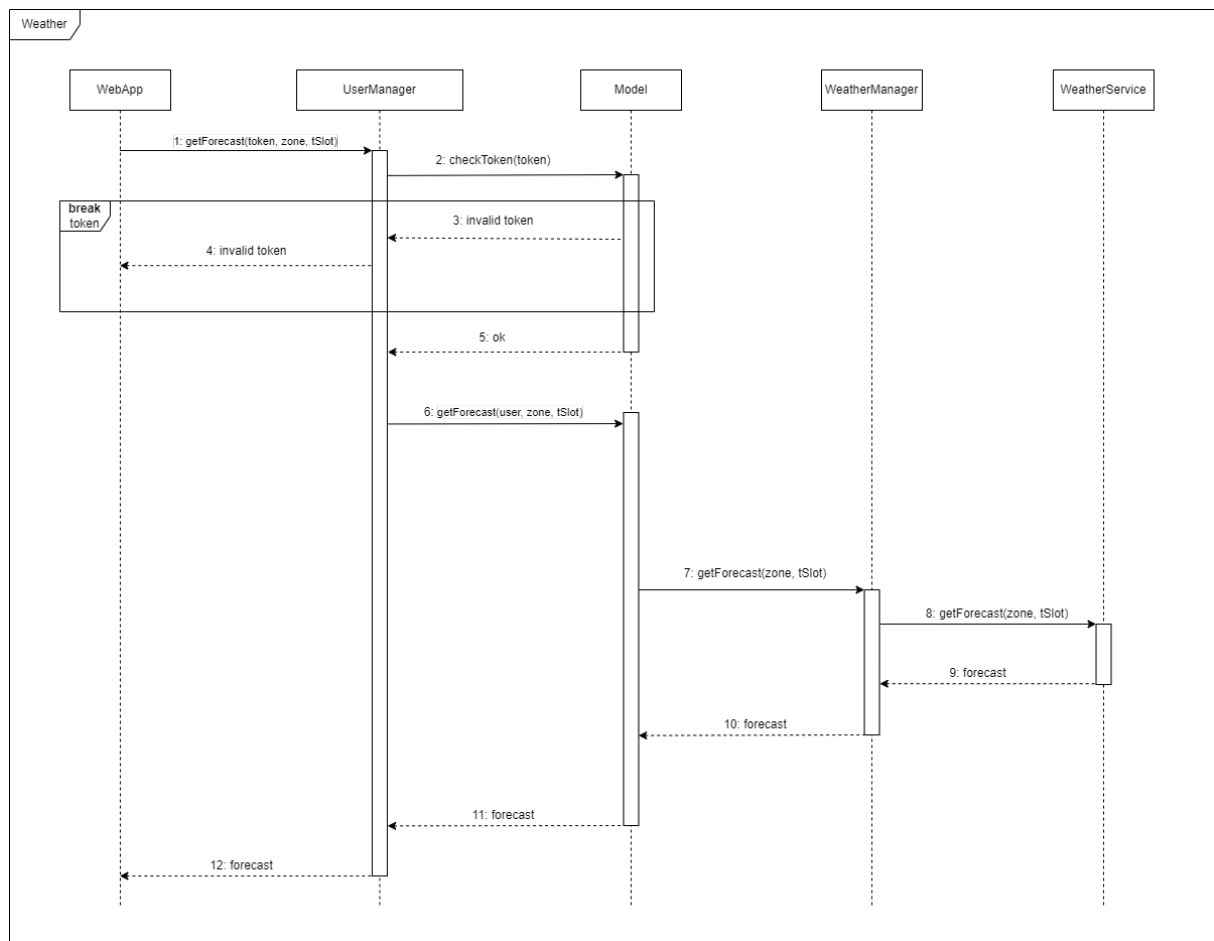


Figure 11: *Weather* Sequence Diagram.

### 2.4.7 Report Production (Farmer)

After the token is verified, the Model calls ProductionManager, which creates a query used to insert the reported production in the database.

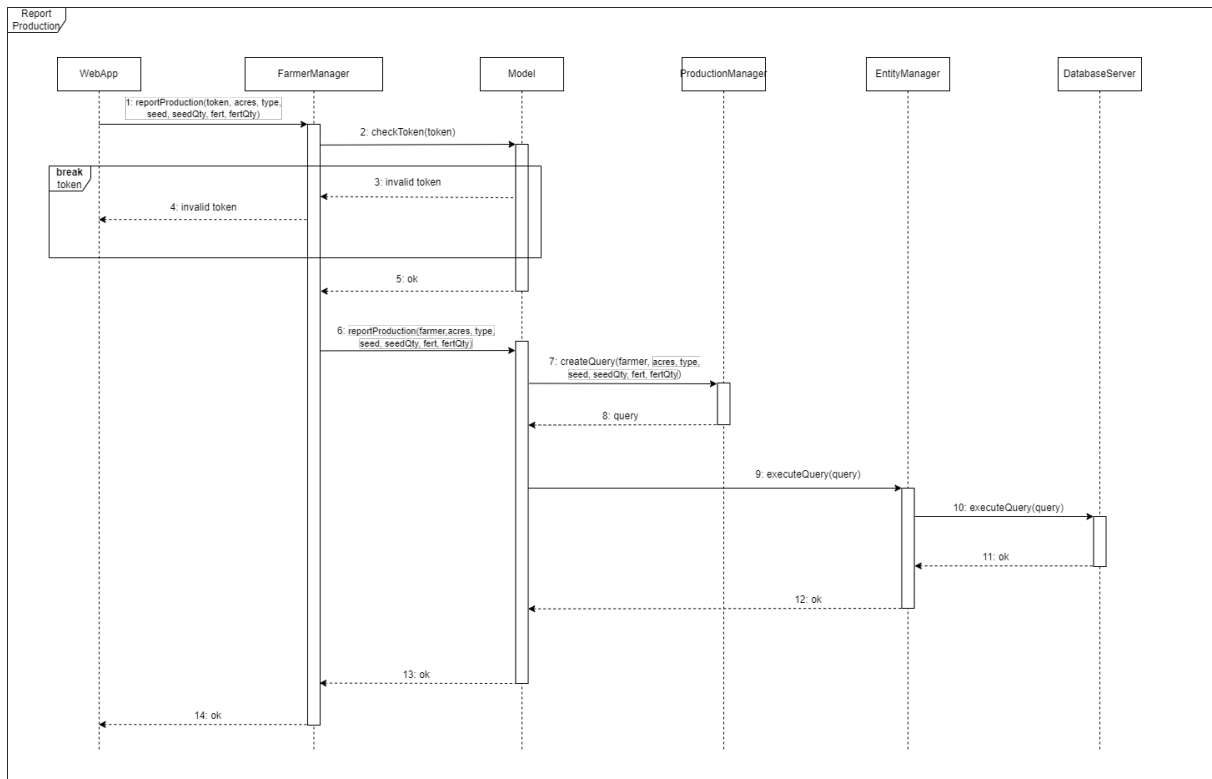


Figure 12: *Report Production (Farmer)* Sequence Diagram.

### 2.4.8 Report Production (Agronomist/TPM)

After the token is verified, the Model calls ProductionManager, which takes the requested farmer as a parameter in order to generate a query to retrieve the reported productions. The token will then be verified again, and Model will call ProductionManager one more time, passing the selected report as a parameter. ProductionManager will then generate a query to retrieve all the data concerning the selected report.

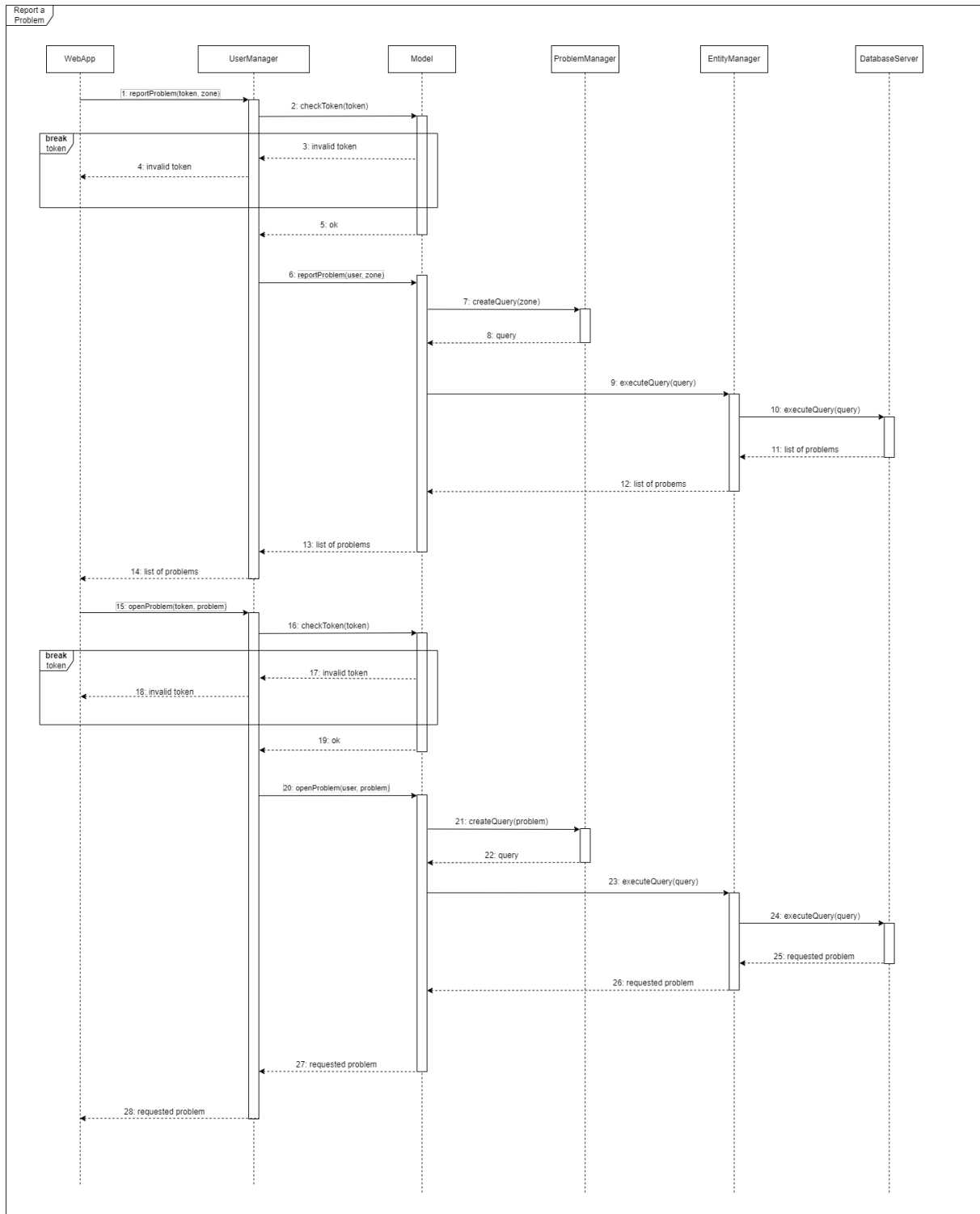


Figure 13: Visualize Production Sequence Diagram.

### 2.4.9 Help (Ask)

After the token is verified, the Model calls HelpManager in order to generate a query to insert the request in the database.

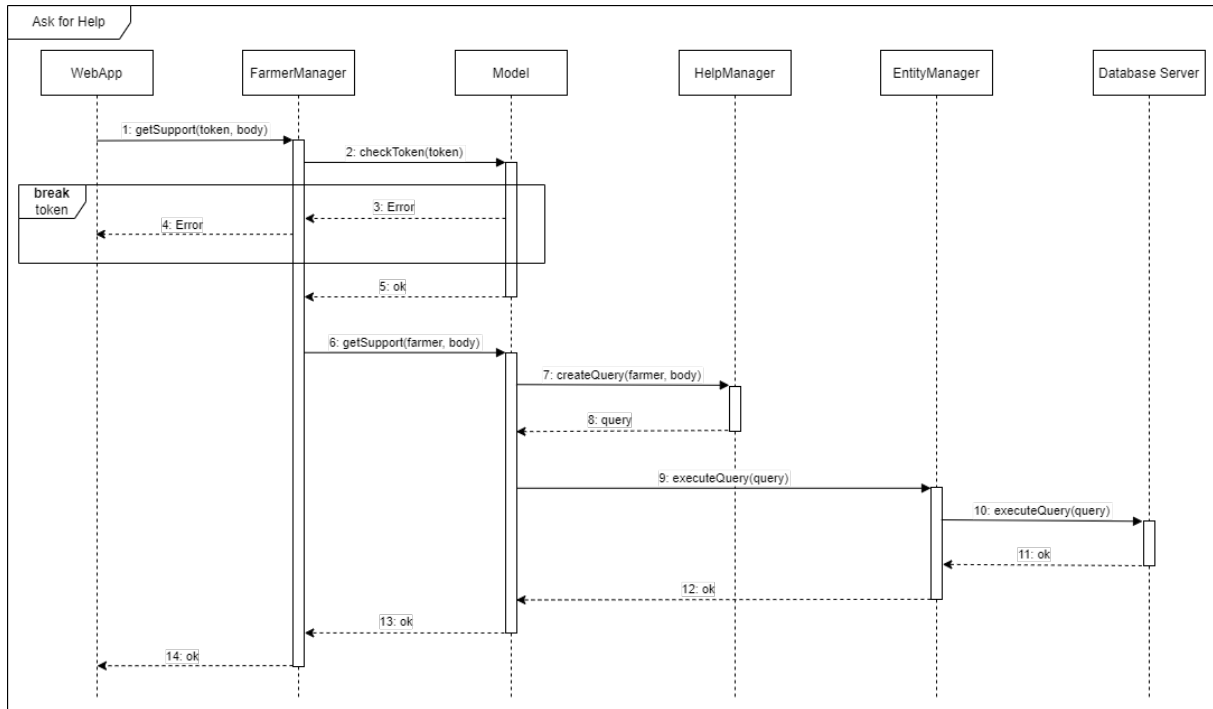


Figure 14: *Ask For Help* Sequence Diagram.

### 2.4.10 Forum

After the token is verified, the Model calls ForumManager in order to generate a query to insert the newly created discussion in the database.

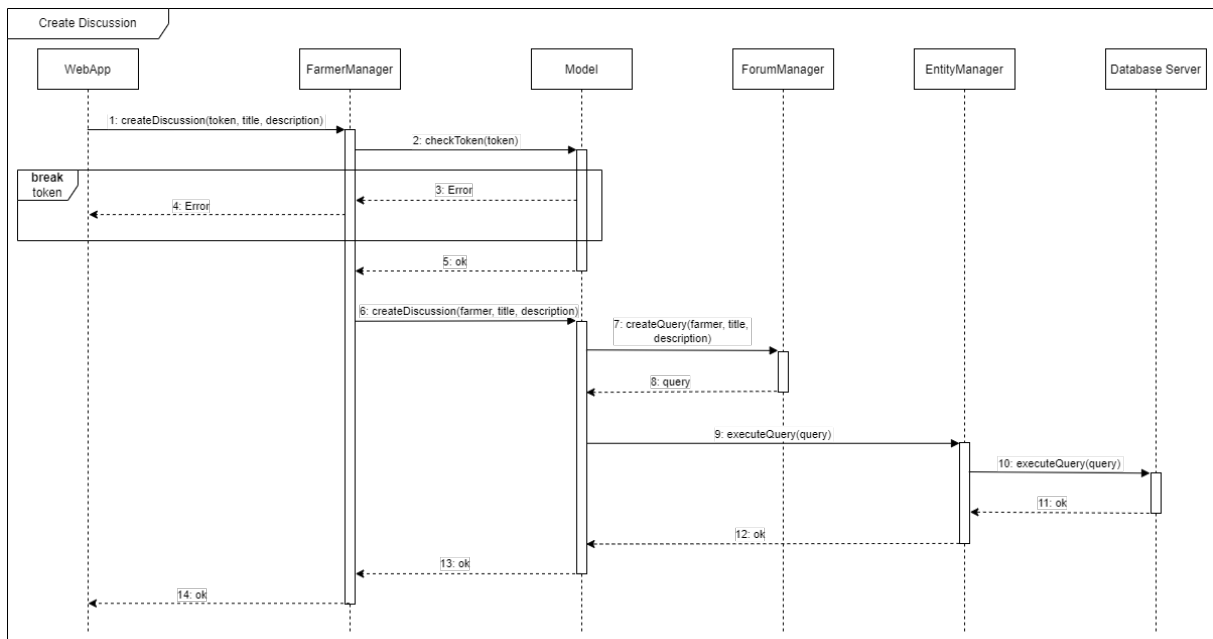


Figure 15: Sequence Diagram.

## 2.4.11 Agenda

After the token is verified, the Model calls AgendaManager in order to create a query to insert the new event in the database.

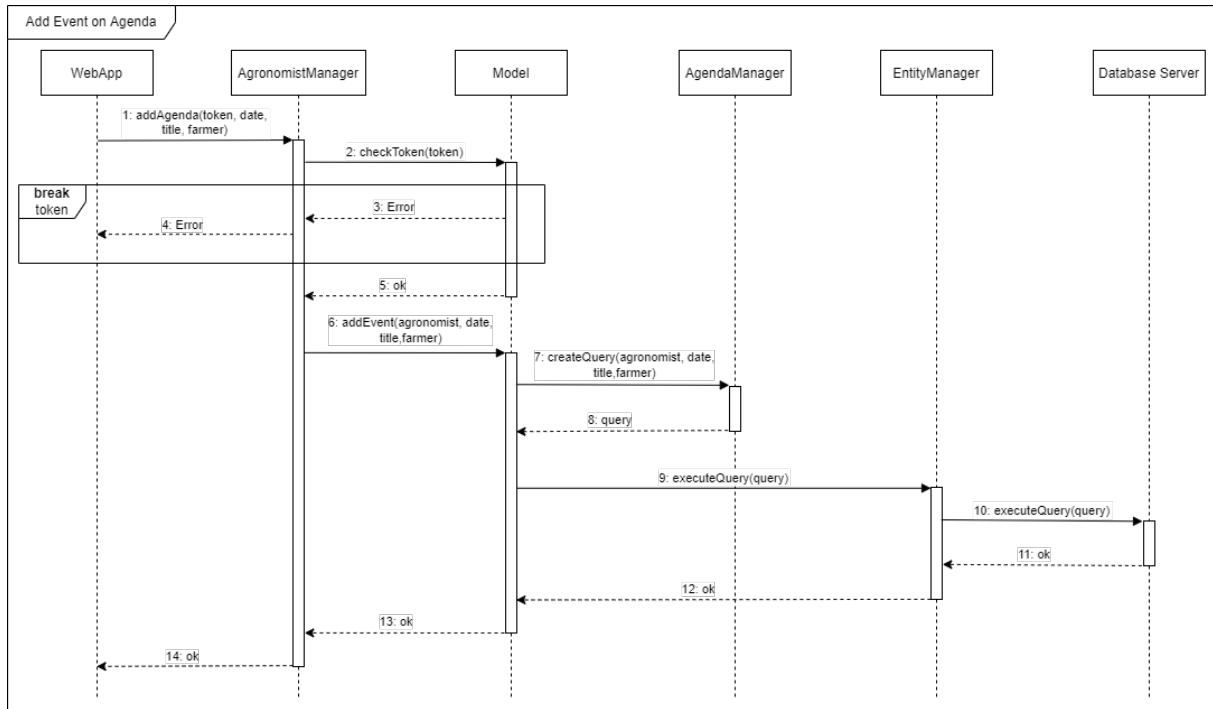


Figure 16: *Creating a new Event Sequence Diagram.*

After the token is verified, the Model calls AgendaManager in order to create a query to retrieve all the planned events associated to the agronomists in the selected date

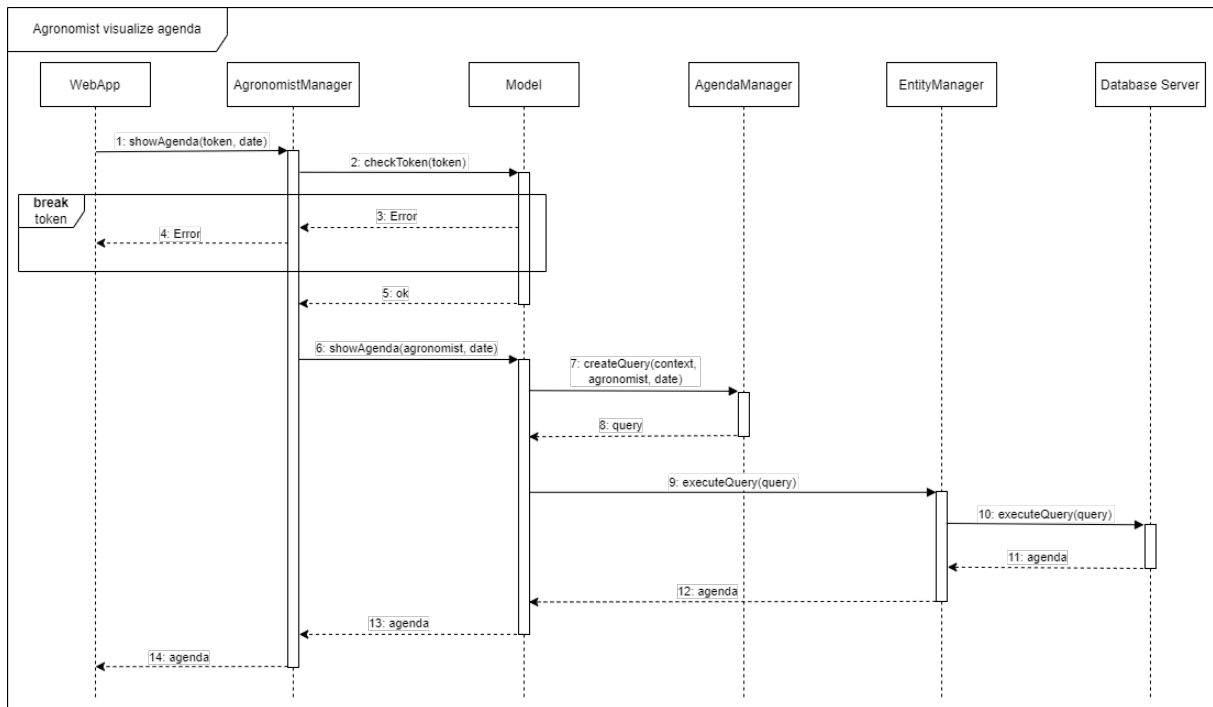


Figure 17: *Visualize Events Sequence Diagram.*

## 2.4.12 Visualize Initiatives

After the token is verified, the Model calls InitiativesManager in order to create a query to retrieve from the database all the reports associated to the requested farmer.

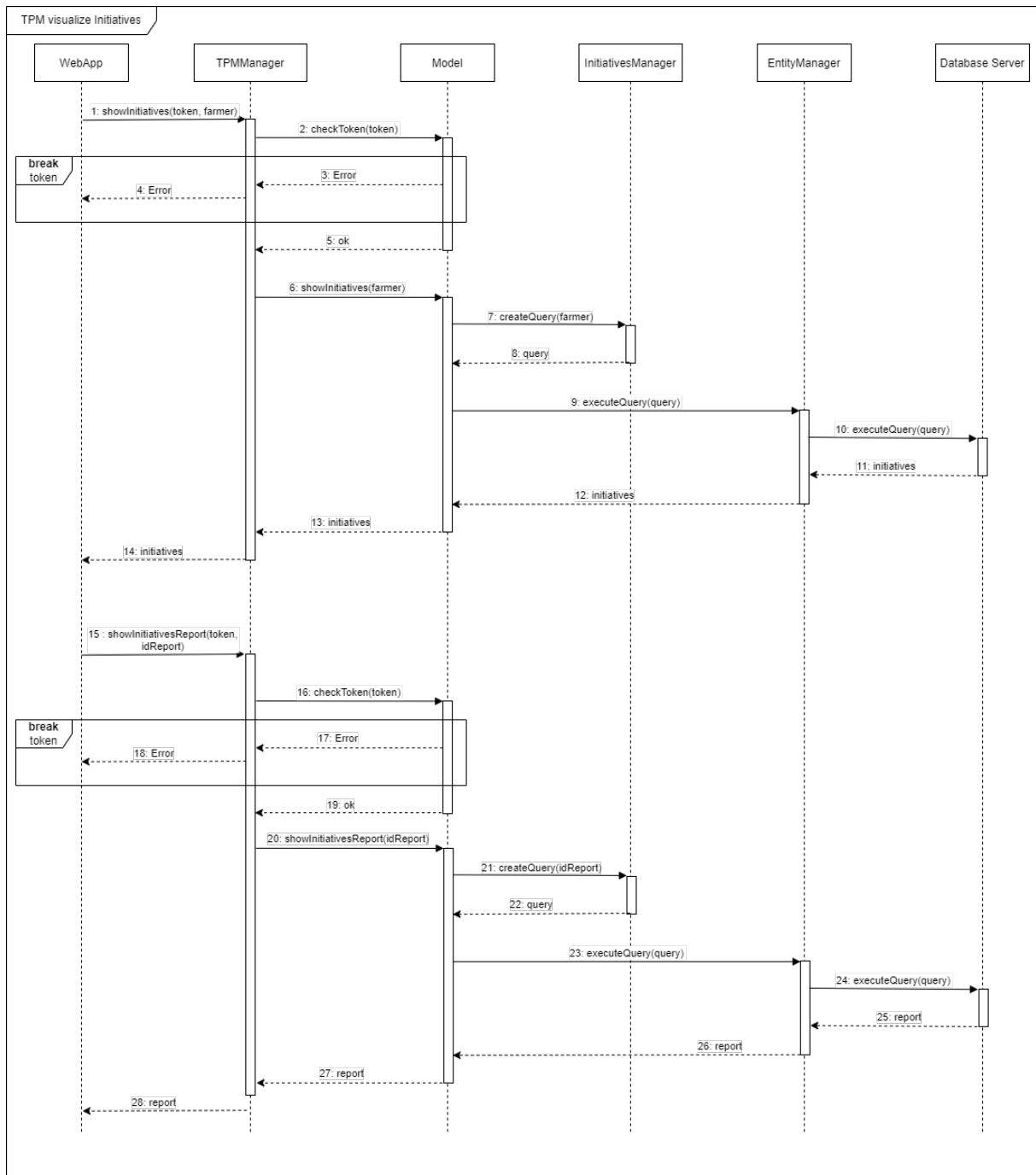


Figure 18: *Visualize Initiatives* Sequence Diagram.

### 2.4.13 Ranking

After the token is verified, the Model call RankingManager in order to create a query to retrieve the ranking of all the farmers associated to the selected zone. If no zone is selected, the query will request the global ranking.

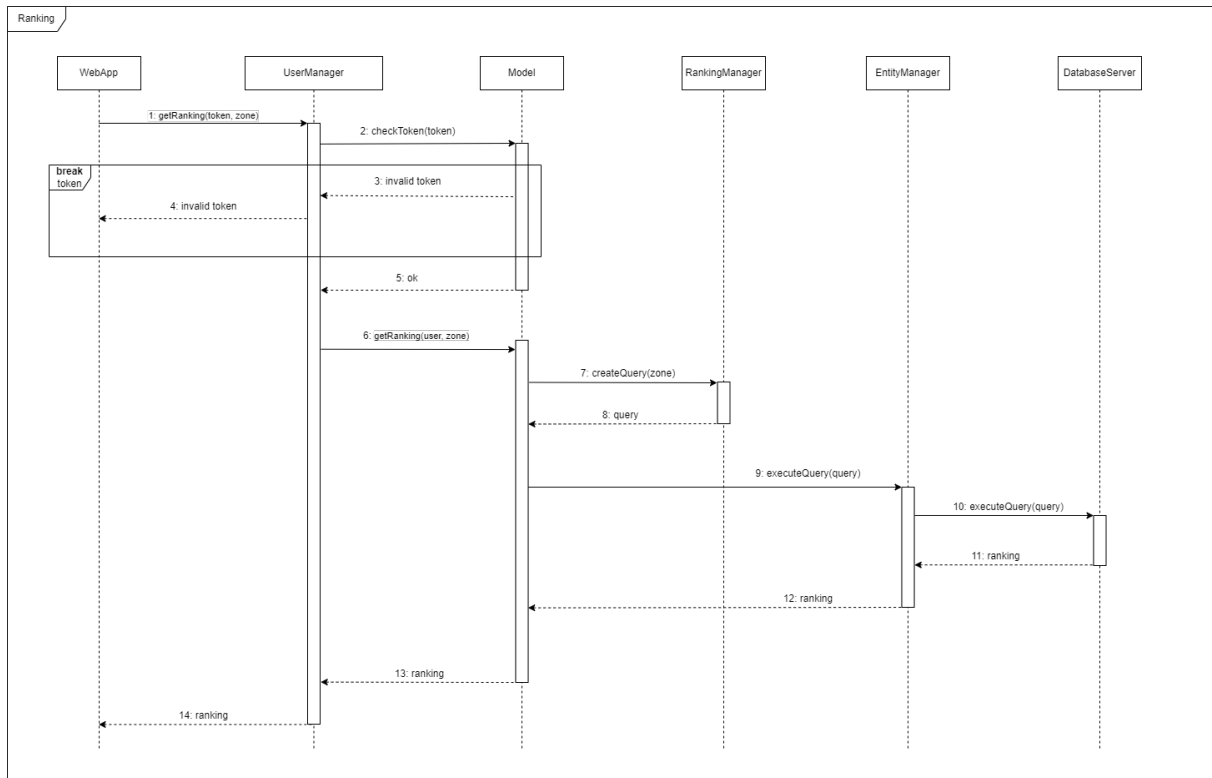


Figure 19: *Ranking* Sequence Diagram.



## 2.4.14 Chat

This diagram represents WebApp1, which is a user that sends a message, and WebApp2, which is a user that visualize a message. To send a message, the token has to be verified, then the Model will call ChatManager in order to generate a query that saves in the database the message, along with the receiver and the sender. Instead, to visualize a message, the token has to be verified, then Model will call ChatManager in order to generate a query that retrieves all the messages associated to the user.

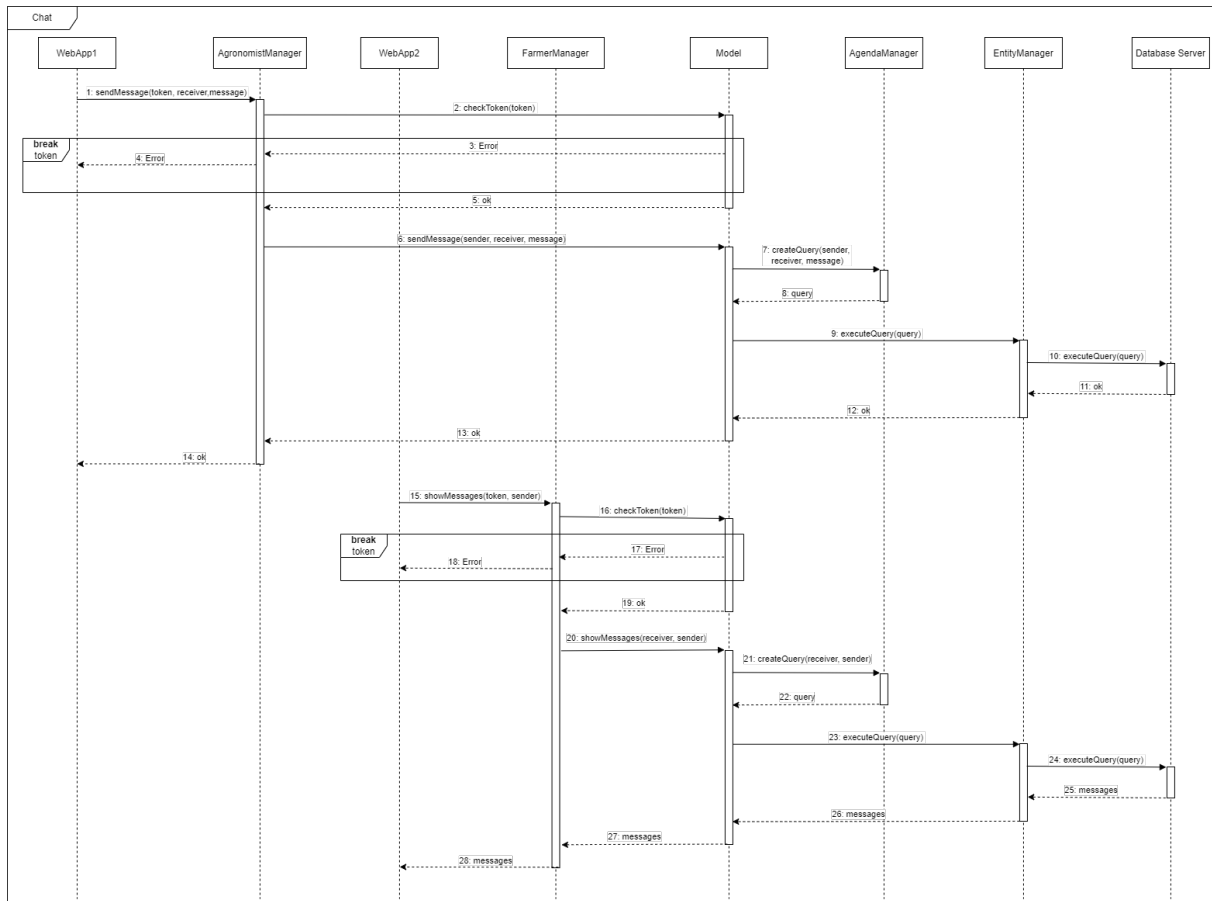


Figure 20: Chat Sequence Diagram.

## 2.5 Component Interface

This section lists all the methods that each component interface provides to the other components.

### 2.5.1 DREAM Server

#### **AuthManager**

- login(username, password)

#### **FarmerManager**

- getHomePage(token)
- getReportProblem(token)
- reportProblem(token, title, description)
- getHelpHomePage(token)
- getSupport (token)
- getSupport(token, body)
- openHelp(token)
- openHelp(token, idRequest)
- answerHelp(token, idRequest, answer)
- getForum(token)
- createDiscussion(token)
- createDiscussion(token, title, description)
- answerDiscussion(token, idDiscussion)
- answerDiscussion(token, idDiscussion, answer)
- getForecast(token)
- getForecast(token, zone, tSlot)
- wikiFarm(token)
- wikiFarm(token, zone, typeOfProd)
- reportProduction(token)
- reportProduction(token, acres, type, seed, seedQty, fert, fertQty)
- showChats(token)
- newChat(token)
- newChat(token, username)
- sendMessage(token, receiver, message)
- showMessages(token, sender)

## **AgronomistManager**

- getHomePage(token)
- getForecast(token)
- getForecast(token, zone, tSlot)
- showAgenda (token)
- showAgenda(token, date)
- addAgenda (token, date)
- addAgenda (token, date, title, farmers)
- showDetails(token, idVisit)
- editReport(token, idReport)
- editReport(token, idReport, report)
- setVisisted(token, idVisit)
- confirmDailyPlan(token, date)
- confirmDailyPlan(token, date)
- getProductionOthers(token, farmer)
- getProductionOthers(token, idReport)
- reportProblem(token)
- reportProblem(token, zone)
- openProblem(token, problem)
- wikiFarm(token)
- wikiFarm(token, zone, typeOfProd)
- showChats(token)
- newChat(token)
- newChat(token, username)
- sendMessage(token, receiver, message)
- showMessages(token, sender)
- getRanking(token, zone)
- openHelp(token)
- openHelp(token, idRequest)
- answerHelp(token, idRequest, answer)

## **answerHelp(token, idRequest, answer)**

- getHomePage(token)

- getRanking(token, zone)
- reportProblem(token)
- reportProblem(token, zone)
- openProblem(token, problem)
- showChats(token)
- newChat(token)
- newChat(token, username)
- sendMessage(token, receiver, message)
- showMessages(token, sender)
- getProductionOthers(token)
- getProductionOthers(token, farmer)
- getProductionOthers(token, idReport)
- showInitiatives(token)
- showInitiatives(token, farmer)
- showInitiativesReport(token, idReport)

## **Model**

- login(username, hashPassword)
- checkToken(token)
- reportProblem(farmer, title, description)
- reportProblem(user, zone)
- openProblem(user, problem)
- getSupport(farmer, body)
- openHelp()
- openHelp(idRequest)
- answerHelp(user, idRequest, answer)
- getForum()
- createDiscussion(farmer, title, description)
- answerDiscussion(idDiscussion)
- answerDiscussion(farmer, idDiscussion, answer)
- getForecast(user, zone, tSlot)
- wikiFarm(user, zone, typeOfProd)
- reportProduction(farmer, acres, type, seed, seedQty, fert, fertQty)

- getProductionOthers(user, farmer)
- getProductionOthers(user, idReport)
- showChats(farmer)
- sendMessage(sender, receiver, message)
- showMessages(receiver, sender)
- showAgenda(agronomist, date)
- addEvent(agronomist, date, title, farmers)
- showDetails(idVisit)
- editReport(idReport)
- editReport(idReport, report)
- setVisisted(idVisit)
- confirmDailyPlan(agronomist, date)
- getRanking(user, zone)
- showInitiatives(farmer)
- showInitiativesReport(idReport)

### **RankingManager**

- createQuery(zone)

### **AgendaManager**

- createQuery(context, agronomist, date)
- createQuery(agronomist, date, title, farmers)
- createQuery(context, idVisit)
- createQuery(idReport)
- createQuery(idReport, report)

### **InitiativesManager**

- createQuery(farmer)
- createQuery(idReport)

### **ProductionManager**

- createQuery(farmer, acres, type, seed, seedQty, fert, fertQty)
- createQuery(farmer)
- createQuery(idReport)

### **WikiFarmManager**

- createQuery(user, zone, typeOfProd)

### **ProblemManager**

- createQuery(farmer, title, description)
- createQuery(zone)
- createQuery(problem)

### **HelpManager**

- createQuery(farmer, body)
- createQuery()
- createQuery(idRequest)
- createQuery(user, idRequest, answer)

### **ForumManager**

- createQuery()
- createQuery(farmer, title, description)
- createQuery(idDiscussion)
- createQuery(farmer, idDiscussion, answer)

### **ChatManager**

- createQuery(farmer)
- createQuery(sender, receiver, message)
- createQuery(receiver, sender)

### **EntityManager**

- executeQuery(query)

### **WeatherManager**

- getForecast(zone, tSlot)

## **2.5.2 Sensor Server**

### **SensorController**

- sendRoutine(location, sensortype, value)

### **Model**

- updateData(location, sensortype, value)

### **EntityManager**

- executeQuery(query)

## 2.6 Selected architectural styles and patterns

Below are presented the decisions made during the design architecture.

- **Three-tiers architecture:** As described in section 2.3, the system adopts this architecture to distinguish between client, server, and database.
- **Client-Server architecture:** This architecture has been used in conjunction with a thin client approach. A client makes a request for a service and receives a reply to that request; a server receives and processes a request and sends back the required response.
- **Model-View-Controller:** software design pattern that divides the program logic into three interconnected elements.
  - **Model:** central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic, and rules of the application.
  - **View:** any visual representation of such as a chart, diagram, or table.
  - **Controller:** responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it and then passes the input to the model.
- **Adapter:** It's a pattern used to match interfaces of different classes. The adapter is used as a middle layer between the UserManagers and the Model, as there is a unique interface that connects those components. E.g. FarmerManager will send a farmer object, while AgronomistManager will send an agronomist object, but they both use the same interface.
- **Facade:** This pattern is used to create a component that offers an interface used to simplify and adapt the usage of another component. It's used in the WeatherManager component to act as a connector with the external WeatherService.
- **Observer:** This is a way to notify a change to other classes. It's used in the Chat function to implement the exchange of messages between two users.

## 2.7 Other Design Decision

### 2.7.1 Database Structure

All the data will be stored in a database implemented using SQLServer, since it is an open-source relational database management system. The following diagram explains how the database is structured.

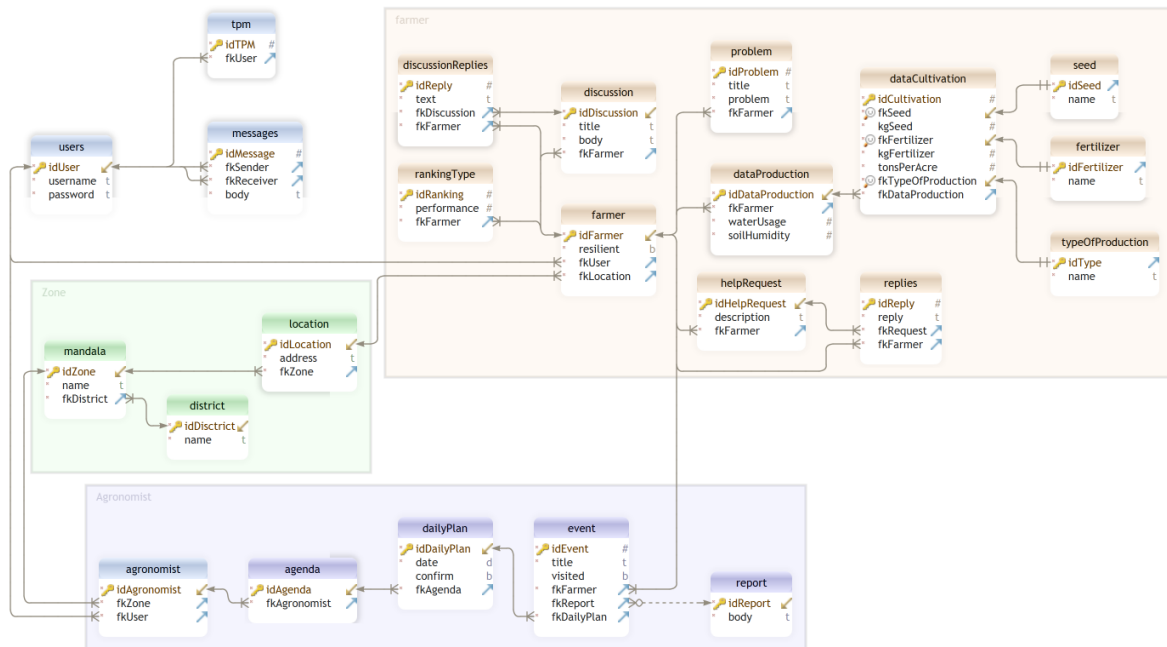


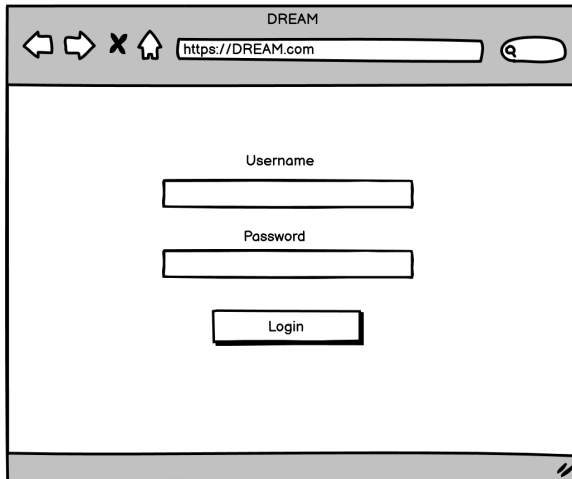
Figure 21: Database Schema.



### 3 User Interface Design

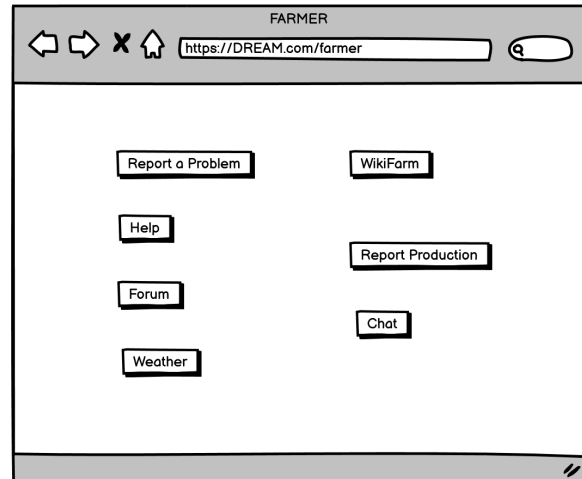
In this section, mockups are divided into 3 sections in order to represent the DREAM's GUI.

#### 3.1 Farmer's WebApp



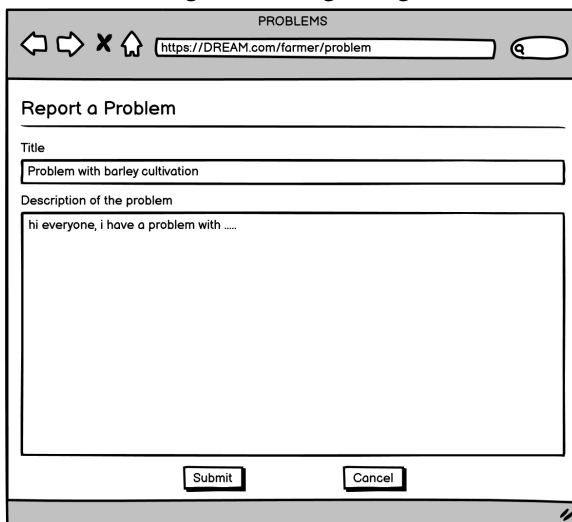
A web browser window titled "DREAM" with the address bar showing "https://DREAM.com". The page contains a login form with two input fields labeled "Username" and "Password", and a "Login" button below them.

Figure 22: LogIn Page.



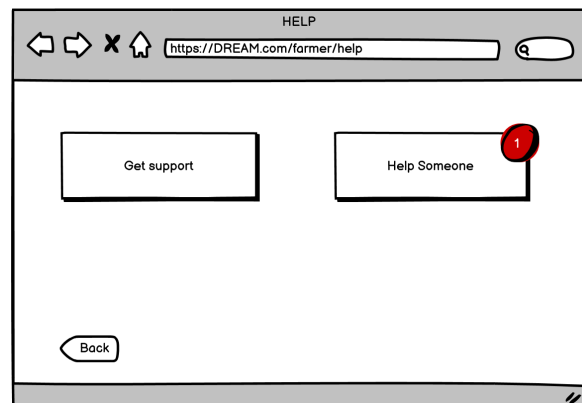
A web browser window titled "FARMER" with the address bar showing "https://DREAM.com/farmer". The page displays a grid of buttons: "Report a Problem", "WikiFarm", "Help", "Report Production", "Forum", "Chat", and "Weather".

Figure 23: Home Page.



A web browser window titled "PROBLEMS" with the address bar showing "https://DREAM.com/farmer/problem". The page is titled "Report a Problem" and contains a "Title" input field with the text "Problem with barley cultivation", a "Description of the problem" text area with the text "hi everyone, i have a problem with ....", and "Submit" and "Cancel" buttons at the bottom.

Figure 24: Report a Problem Section.



A web browser window titled "HELP" with the address bar showing "https://DREAM.com/farmer/help". The page contains two buttons: "Get support" and "Help Someone" (which has a red notification bubble with the number "1" next to it). A "Back" button is located at the bottom left.

Figure 25: Help Section.

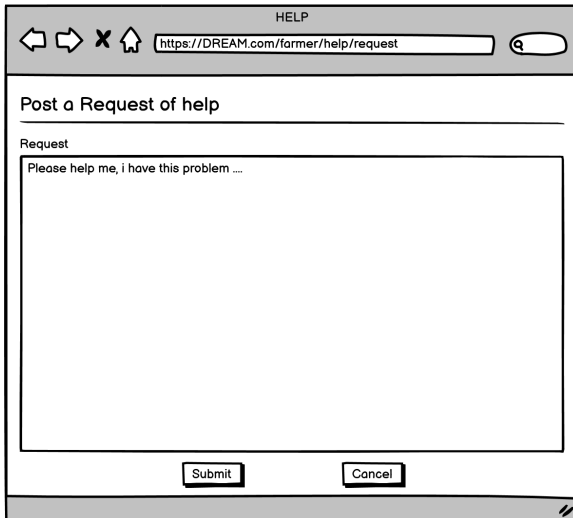


Figure 26: Description of a problem.

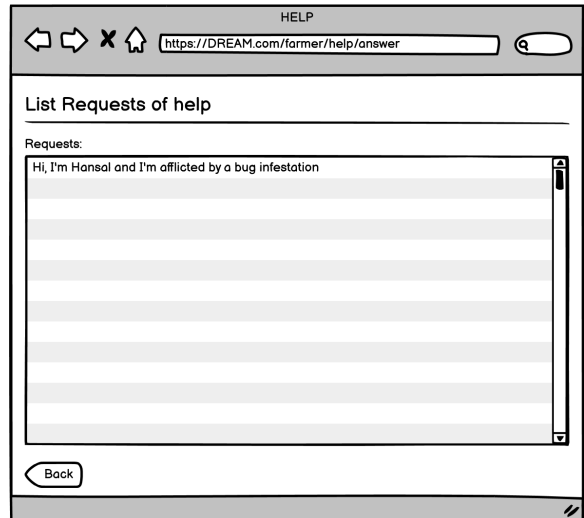


Figure 27: List of helps.

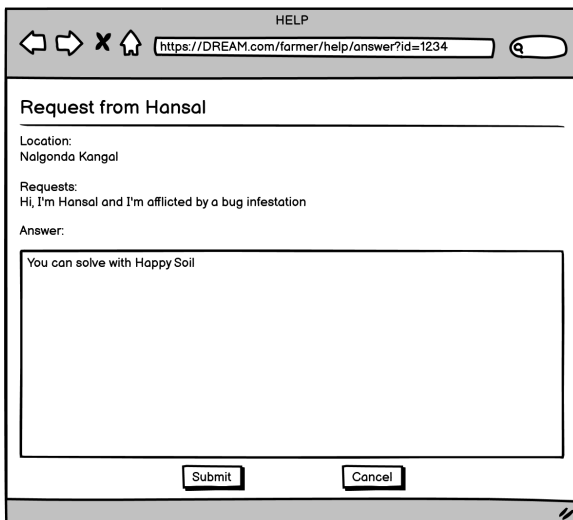


Figure 28: Reply to a request.

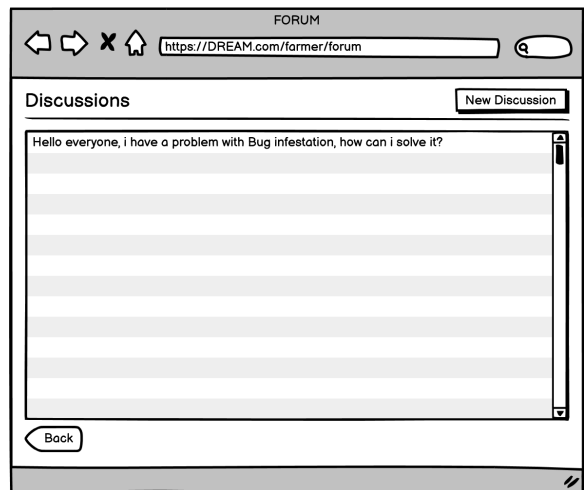


Figure 29: Forum Section.

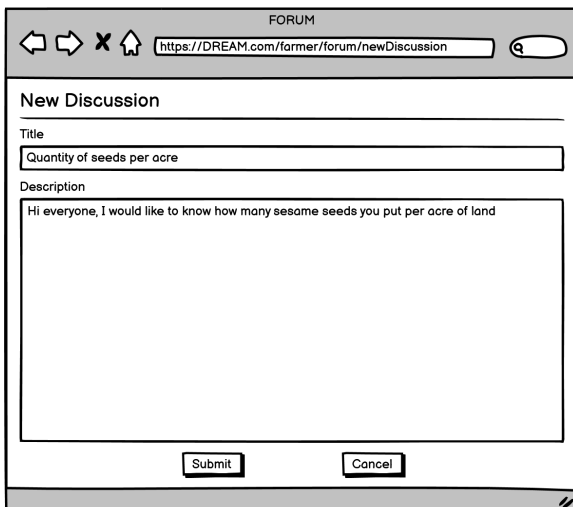


Figure 30: Creating a new Discussion.

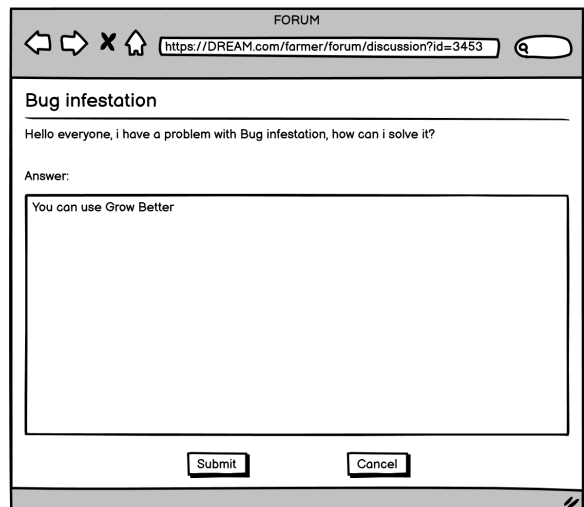


Figure 31: Reply to discussion.

WEATHER

https://DREAM.com/farmer/weather

Weather

District: Mandya

Mandala: Maddur

Start period: 09/01/2022

End period: 12/01/2022

Search

Back

Figure 32: Weather Section.

WEATHER

https://DREAM.com/farmer/weather?search=...

Weather

Back

Figure 33: Weather Result.

WIKIFARM

https://DREAM.com/farmer/wikifarm

WikiFarm

District: Mandya

Mandala: Maddur

Type of Production: Intensive farming

Search

Back

Figure 34: WikiFarm Section.

WIKIFARM

https://DREAM.com/farmer/wikifarm?search=suggestion

WikiFarm

Crops	Fertilizer
Cotton	Grow Better
Sesame	Happy Soil

Back

Figure 35: WikiFarm Result.

PRODUCTION

https://DREAM.com/farmer/production

Cultivation

Acres of land: 24

Type of production: Intensive farming

Seed

Seed type: Cotton

Quantity (tons): 4

Fertilize

Fertilize type: Grow Better

Quantity (tons): 0.5

Next Finish Cancel

Figure 36: Report Production.

PRODUCTION

https://DREAM.com/farmer/production

Cultivation

Acres of land:

Type of production:

Seed

Seed type:

Quantity (tons):

Fertilize

Fertilize type:

Quantity (tons):

Next Finish Cancel

Figure 37: Reporting another cultivation.

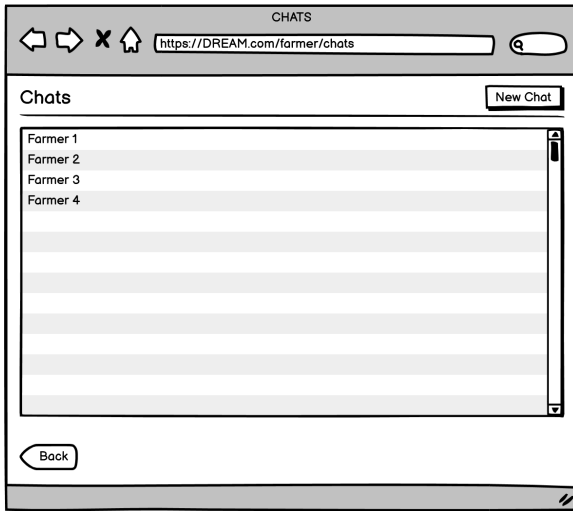


Figure 38: Chat Section.

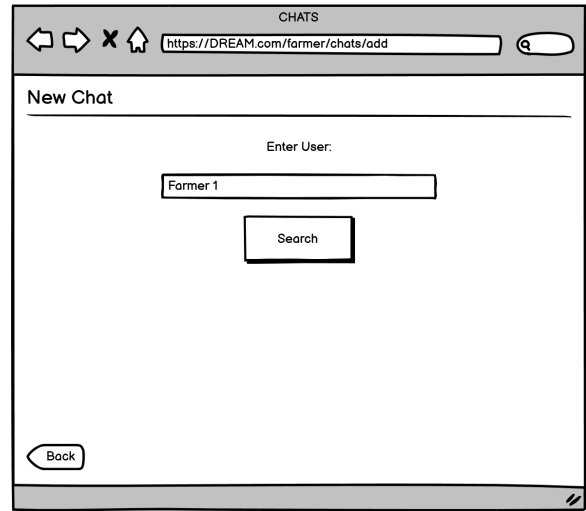


Figure 39: New Chat.

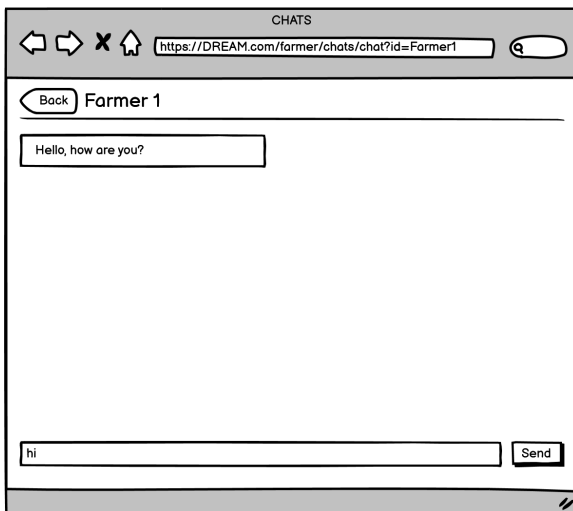


Figure 40: Chat Reply.

### 3.2 Agronomist's WebApp

Figure 41 shows a login page within a browser window titled 'DREAM'. The address bar shows 'https://DREAM.com'. The page has a simple layout with a 'Username' input field, a 'Password' input field, and a 'Login' button centered below them.

Figure 41: LogIn Page.

Figure 42 shows the agronomist's home page in a browser window titled 'AGRONOMIST'. The address bar shows 'https://DREAM.com/agronomist'. The page features a grid of eight buttons: 'Weather', 'WikiFarm', 'Agenda', 'Chat', 'Report Production', 'Ranking', 'Report a Problem', and 'Help'.

Figure 42: Agronomist's Home Page.

Figure 43 shows the weather section in a browser window titled 'WEATHER'. The address bar shows 'https://DREAM.com/agronomist/weather'. The page has a form with dropdown menus for 'District' (Mandya) and 'Mandala' (Maddur), date pickers for 'Start period' (09/01/2022) and 'End period' (12/01/2022), and a 'Search' button.

Figure 43: Weather Section.

Figure 44 shows the weather result page in a browser window titled 'WEATHER'. The address bar shows 'https://DREAM.com/agronomist/weather?search=...'. The page displays a large placeholder box with a diagonal cross, suggesting the weather data or image failed to load.

Figure 44: Weather Result.

Figure 45 shows the agenda home page in a browser window titled 'AGENDA'. The address bar shows 'https://DREAM.com/agronomist/agenda?date=09/01/2022'. The page features a table of visits for the selected date (09/01/2022). The table has columns for Title, Location, Farmer, and Details. The data shown is:

Title	Location	Farmer	Details
First visit to Menroosh	Street 23	Menroosh	<a href="#">Details</a>
Extra visit to Hanu	Street 7	Hanu	<a href="#">Details</a>

Below the table are 'Back' and 'Confirm DailyPlan' buttons.

Figure 45: Agenda Home.

Figure 46 shows the 'Add a visit' page in a browser window titled 'AGENDA'. The address bar shows 'https://DREAM.com/agronomist/agenda?date=09/01/2022'. The page has a form with a 'Title' field (First visit to Menroosh), an 'Insert Farmer' field (Menroosh), and an 'Add' button.

Figure 46: Add a visit.

AGENDA

https://DREAM.com/agronomist/agenda?pospose=15987

Details:

Title: First visit to Menroosh

Location: Street 23

Farmer: Menroosh

Date: 11/02/2022

Edit the report:

Click to make the visit ended:

Figure 47: Detail of a Event.

AGENDA

https://DREAM.com/agronomist/agenda?id=15987

Report Menroosh

We decided to plant cotton ...

Figure 48: Add a report.

PRODUCTION

https://DREAM.com/agronomist/production

Production

Insert Farmer:

List of Report:

Report 1

Report 2

Figure 49: List of reports.

PRODUCTION

https://DREAM.com/agronomist/production?id=Hanu

Production Report 1

Soil Humidity (%): 25 Water used (L): 10

Acres of land	Type of production	Seed Type	Quantity of seed	Fertilize Type	Quantity of fertilize
24	Intensive farming	Cotton	4	Grow Better	0.5

Figure 50: Report Detail.

PROBLEM

https://DREAM.com/agronomist/problem

Problems faced by Farmers

District:  Mandala:

Problem1

Problem2

Figure 51: Search a Problem.

PROBLEM

https://DREAM.com/agronomist/problem?id=22334

Problems1 by Hanu

I have this problem ...

Figure 52: Detail of a Problem.

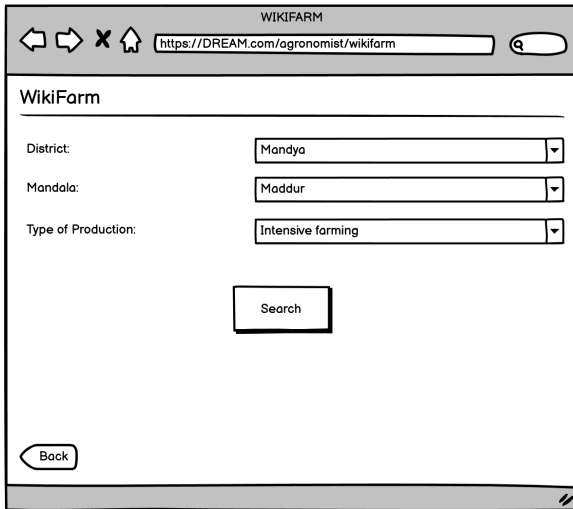


Figure 53: WikiFarm Home.

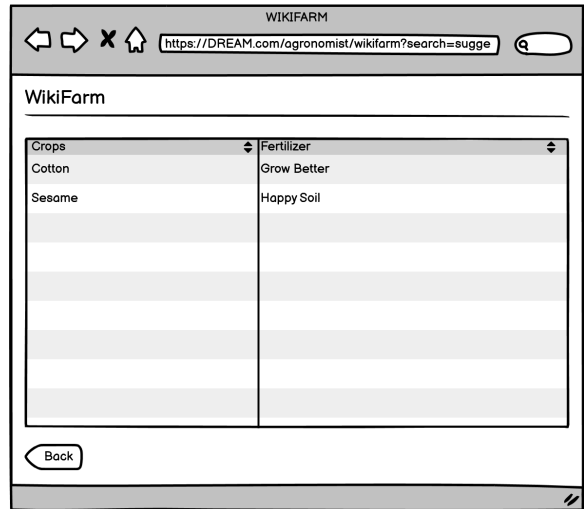


Figure 54: WikiFarm's Result.

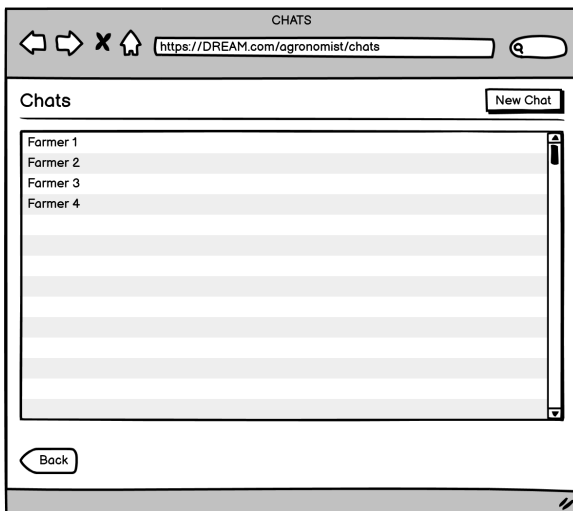


Figure 55: List of Chats.

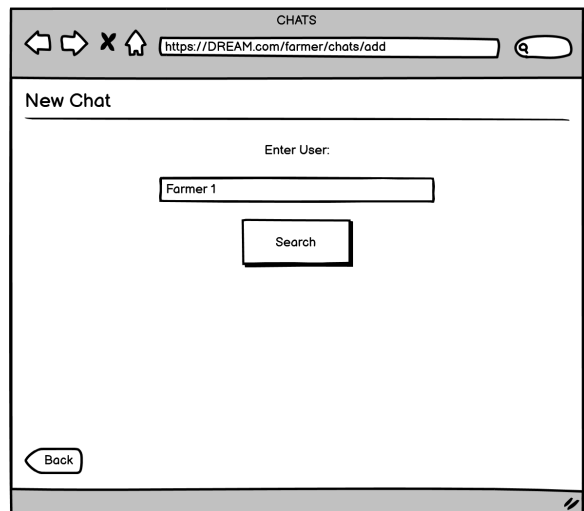


Figure 56: New Chat.

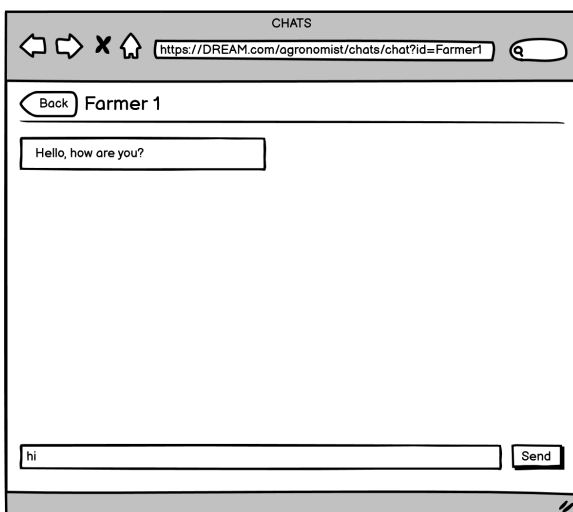


Figure 57: Chat Detail.

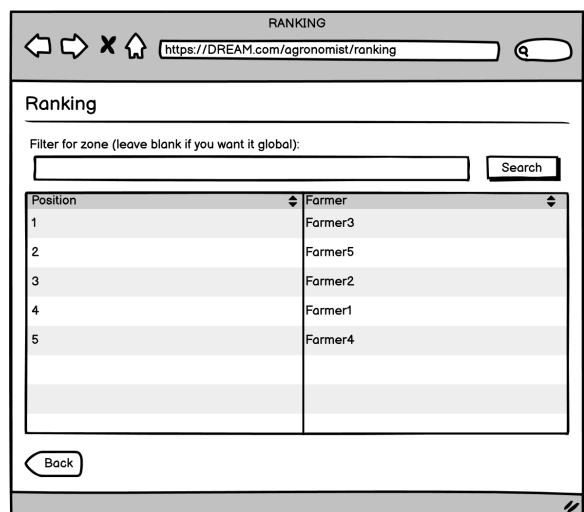


Figure 58: Ranking of the Farmers.

### 3.3 TPM's WebApp

Figure 59 shows a login page within a browser window titled 'DREAM'. The address bar shows 'https://DREAM.com'. The page layout is centered and includes a 'Username' input field, a 'Password' input field, and a 'Login' button.

Figure 59: LogIn Page.

Figure 60 shows the TPM's Home Page within a browser window titled 'TPM'. The address bar shows 'https://DREAM.com/tpm'. The page features a vertical list of buttons: 'Ranking', 'Report a Problem', 'Chat', 'Report Productions', and 'Visualize Initiatives'.

Figure 60: TPM's HomePage.

Figure 61 shows the Ranking Section within a browser window titled 'RANKING'. The address bar shows 'https://DREAM.com/tpm/ranking'. The page includes a 'Ranking' header, a filter for zone, and a table with the following data:

Position	Farmer
1	Farmer3
2	Farmer5
3	Farmer2
4	Farmer1
5	Farmer4

Figure 61: Ranking Section.

Figure 62 shows the Search for Problems page within a browser window titled 'PROBLEM'. The address bar shows 'https://DREAM.com/tpm/problem'. The page includes a 'Problems faced by Farmers' section with dropdown menus for 'District' (Setúbal) and 'Mandala' (Palmela), and a 'Search' button.

Figure 62: Search for Problems.

Figure 63 shows the Detail of a Problem page within a browser window titled 'PROBLEM'. The address bar shows 'https://DREAM.com/tpm/problem?id=22334'. The page includes a 'Problems1 by Hanu' section with a text area for the problem description.

Figure 63: Detail of a Problem.

Figure 64 shows the List of Chats page within a browser window titled 'CHATS'. The address bar shows 'https://DREAM.com/tpm/chats'. The page includes a 'Chats' section with a list of farmers (Farmer 1, Farmer 2, Farmer 3, Farmer 4) and a 'New Chat' button.

Figure 64: List of Chats.



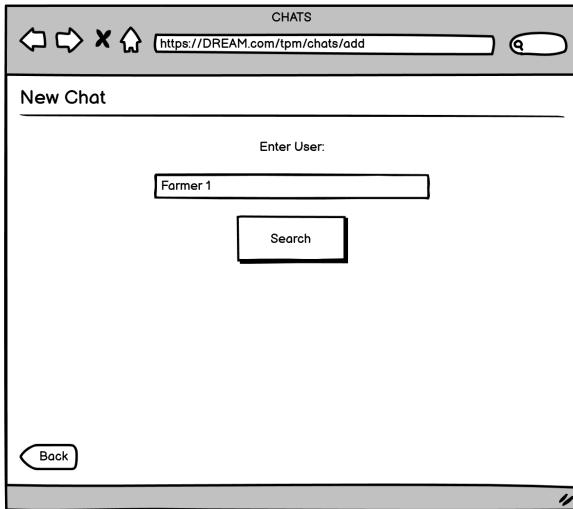


Figure 65: New Chat.

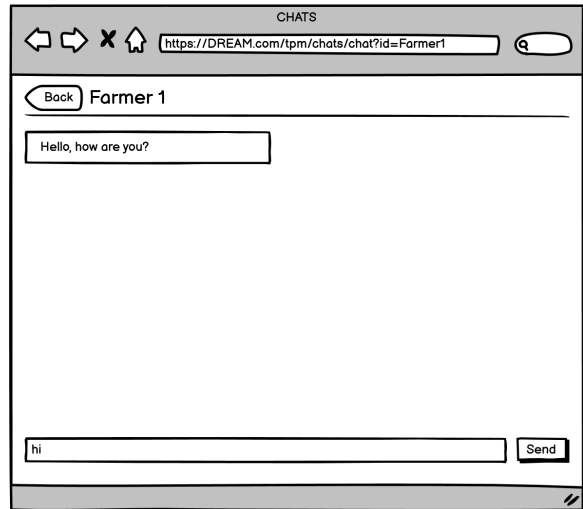


Figure 66: Reply to a Chat.

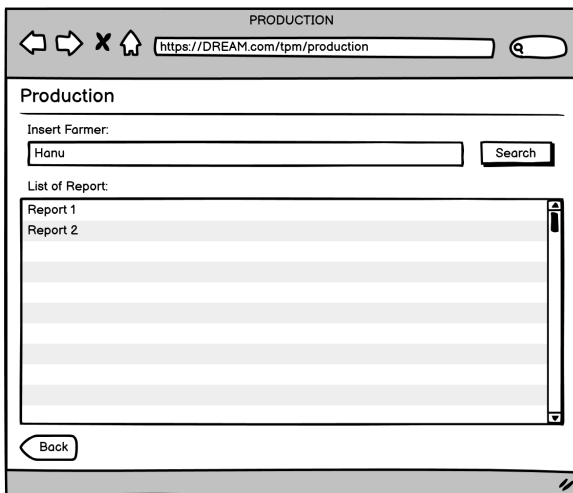


Figure 67: Search for a Production of a farmer.

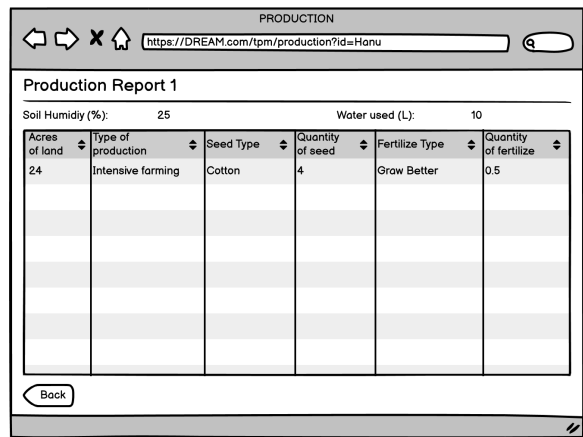


Figure 68: Details of a Production.

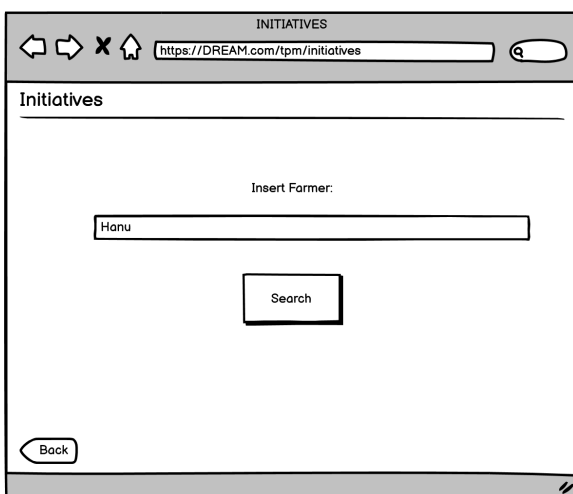


Figure 69: Initiatives Section.

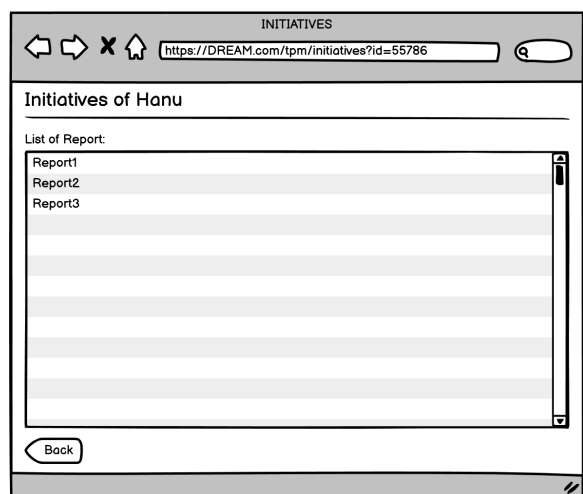


Figure 70: List of Initiatives taken.

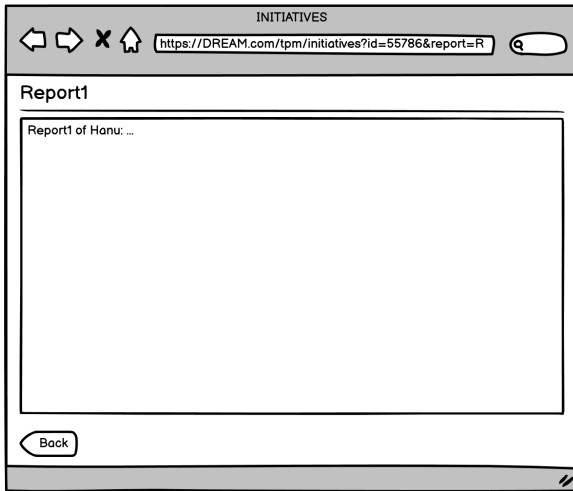


Figure 71: Detail of a Report.

### 3.4 Demo

In the [Github](#) page of this document there are 3 videos showing a demo version of each version of the WebApp. The following qr-codes link directly to the corresponding demo.



Figure 72: Agronomist Demo.



Figure 73: Farmer Demo.



Figure 74: TPM Demo.

## 4 Requirement Traceability

In this section is presented the mapping between the Requirement described in our Requirements Analysis and Specifications Document.

### 4.1 Mapping Interfaces on Requirements

Every requirement is connected to Model (except R47), which is not reported for easier readability

- R.1** Data in the system are never deleted  
**EntityManager**
- R.2** System shall retrieve the data from a sensor of humidity in a land  
**HumiditySensorController, EntityManager**
- R.3** System shall update data of humidity in land every hour  
**HumiditySensorController, EntityManager**
- R.4** System shall retrieve the amount of water supplied to a farmer  
**WaterSensorController, EntityManager**
- R.5** System shall weekly update the amount of water consumed by a farmers  
**WaterSensorController, EntityManager**
- R.6** System shall compute a ranking of the farmers' performances  
**RankingManager, EntityManager**
- R.7** System shall allow any user to communicate with any other user using the chat function  
**ChatManager, EntityManager**
- R.8** System shall allow a farmer to insert his/her type of production  
**WebApp, FarmerManager, ProductionManager, WikiFarmManager**
- R.9** System shall store the location of each farmer  
**EntityManager**
- R.10** System shall store all the types of production of each farmer  
**EntityManager**
- R.11** System shall allow a farmer to filter the type of data he/she wants to visualize  
**WebApp, FarmerManager**
- R.12** System shall receive information about local weather forecast  
**WeatherManager**
- R.13** System shall allow a farmer to visualize the weather forecast  
**WebApp, FarmerManager, WeatherManager**
- R.14** System shall store all the fertilizers available in Telangana  
**EntityManager**
- R.15** System shall suggest fertilizers based on the retrieved data  
**WikiFarmManager, EntityManager**
- R.16** System shall store all crops that can be planted in Telangana  
**EntityManager**

- R.17** System shall suggest crops based on the retrieved data  
**WikiFarmManager, EntityManager**
- R.18** System shall allow a farmer to insert his/her data of production  
**WebApp, FarmerManager, ProductionManager**
- R.19** System shall store all data of production of farmers  
**EntityManager**
- R.20** System shall allow a farmer to insert data about a problem that he/she faced  
**WebApp, FarmerManager, ProblemManager**
- R.21** System shall store all data of problems faced by farmers  
**EntityManager**
- R.22** System shall allow a resilient farmer to share best practices with other farmers  
**WebApp, FarmerManager, ForumManager, EntityManager**
- R.23** System shall allow a farmer to ask for help  
**WebApp, FarmerManager, HelpManager, EntityManager**
- R.24** System shall allow a farmer to reply to a request of help  
**WebApp, FarmerManager, HelpManager, EntityManager**
- R.25** System shall allow a farmer to create a new discussion in the Forum  
**WebApp, FarmerManager, ForumManager, EntityManager**
- R.26** System shall allow a farmer to reply to a discussion in the Forum  
**WebApp, FarmerManager, ForumManager, EntityManager**
- R.27** System shall allow an agronomist to select the area he/she is responsible of  
**WebApp, AgronomistManager, EntityManager**
- R.28** System shall allow an agronomist to visualize a request of help in the area he/she is responsible of  
**WebApp, AgronomistManager, HelpManager, EntityManager**
- R.29** System shall allow an agronomist to reply to a request of help in the area he/she is responsible of  
**WebApp, AgronomistManager, HelpManager, EntityManager**
- R.30** System shall allow an agronomist to visualize the weather forecast for the area he/she is responsible of  
**WebApp, AgronomistManager, WeatherManager**
- R.31** System shall allow an agronomist to visualize the performances of farmers in the area he/she is responsible of  
**WebApp, AgronomistManager, RankingManager, EntityManager**
- R.32** System shall allow an agronomist to visualize data of production inserted by farmers  
**WebApp, AgronomistManager, ProductionManager, EntityManager**
- R.33** System shall allow an agronomist to visualize problems reported by farmers in the area he/she is responsible of  
**WebApp, AgronomistManager, ProblemManager, EntityManager**
- R.34** System shall store the daily plan of all agronomists  
**EntityManager**

- R.35** System shall allow an agronomist to visualize his/her daily plan  
**WebApp, AgronomistManager, AgendaManager, EntityManager**
- R.36** System shall allow an agronomist to update his/her daily plan  
**WebApp, AgronomistManager, AgendaManager, EntityManager**
- R.37** System shall allow an agronomist to confirm his/her daily plan  
**WebApp, AgronomistManager, AgendaManager, EntityManager**
- R.38** System shall allow an agronomist to insert the deviations from his/her original daily plan  
**WebApp, AgronomistManager, AgendaManager, EntityManager**
- R.39** System shall store all the initiatives took by a farmer and an agronomist  
**EntityManager**
- R.40** System shall allow an agronomist to insert an initiative he/she proposed to a farmer  
**WebApp, AgronomistManager, AgendaManager, EntityManager**
- R.41** System shall allow TPM to visualize the ranking of farmers' performances  
**WebApp, TPMManager, RankingManager, EntityManager**
- R.42** System shall allow TPM to visualize data about the production of a farmer  
**WebApp, TPMManager, ProductionManager, EntityManager**
- R.43** System shall allow TPM to visualize the problems reported by farmers  
**WebApp, TPMManager, ProblemManager, EntityManager**
- R.44** System shall allow TPM to visualize initiatives taken by agronomist during his/her visits  
**WebApp, TPMManager, InitiativesManager, EntityManager**
- R.45** System shall allow TPM to ask to a resilient farmer to share best practices  
**WebApp, TPMManager, ChatManager, EntityManager**
- R.46** System shall allow TPM to signal bad farmers to Agronomists  
**WebApp, TPMManager, ChatManager, EntityManager**
- R.47** System shall allow any user to access through a browser  
**WebApp**
- R.48** System shall allow any user to log in using his/her credentials  
**WebApp, AuthManager, EntityManager**
- R.49** System shall give access to any user who has the right permission. For example, a farmer could not access to "Agenda"  
**WebApp, AuthManager, FarmerManager, AgronomistManager, TPMManager**

## 5 Implementation, Integration and Test Plan

This section describes the process and the reasons for the choices that were made during the implementation, integration, and test plan phases. All development follows a bottom-up approach in order to facilitate the search for bugs and to create a system that is easy to update. External services do not need to be unit tested since it is assumed that they are reliable, only the correct integration with the system is checked.

### 5.1 Plan Definition

The implementation focuses on the integration of the central server because an MVC architecture with thin clients was used where the model, which resides in the DREAM Server, is responsible for all interactions. The other components outside the central server, such as the WebApp or the Sensor Server, are tested locally first and then tested as they interact with the other components.

**Note for the reader:** Throughout the implementation and testing process, drivers are used to manage the components that have not yet been fully implemented, so that the parts that are to be tested can be controlled.

#### 5.1.1 Introducing the model and the Entity Manager

For the first step, the model and the Entity Manager of the DREAM Server are implemented, as they are the components responsible for interaction with the Database and are required by most other components.

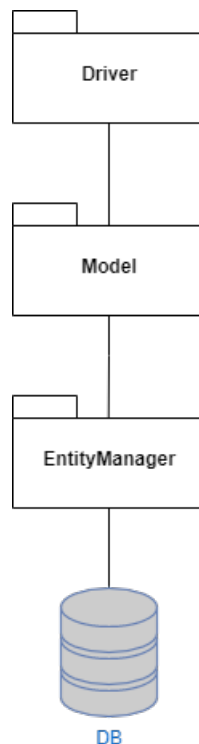


Figure 75: *Model and Entity Manager* implementation.

#### 5.1.2 Authentication Implementation

In this step, the components responsible for authentication are implemented and tested, because access to DREAM data and functionalities is a crucial aspect of the system.

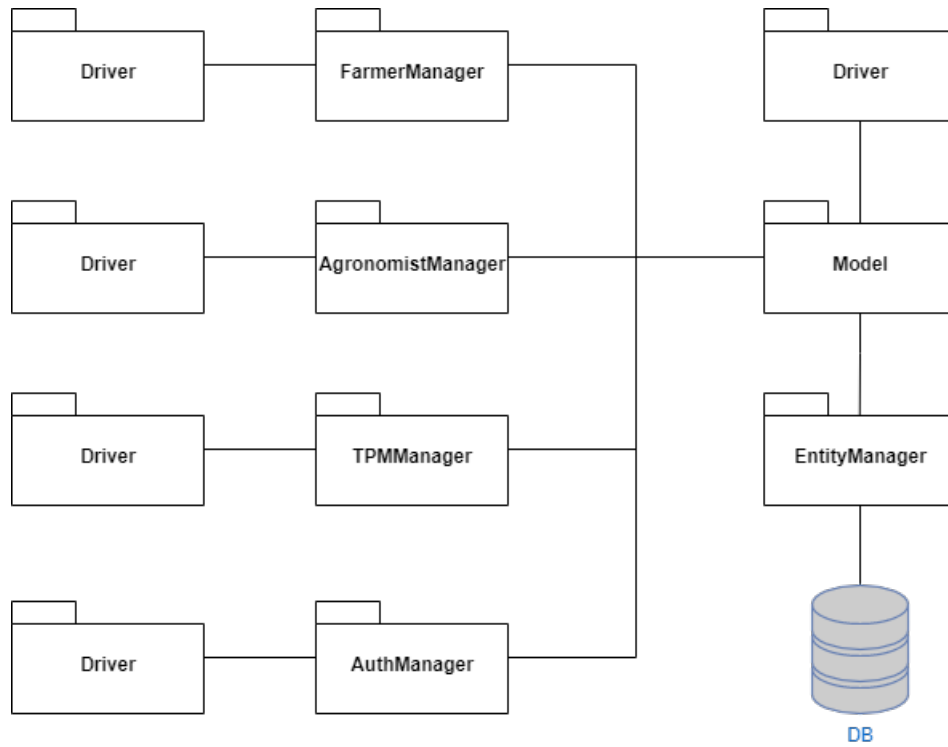


Figure 76: *Authentication Manager* implementation.

### 5.1.3 Adding Functionalities

In this step, the functionalities available in DREAM are implemented one by one. The only functionality that is not yet working is *WeatherManager*, because it relies on the functioning of the external Forecast Service. While the functionalities dedicated to the production of a farmer are implemented using test data, inserted specifically for the test, since the Sensor Server has not yet been implemented.

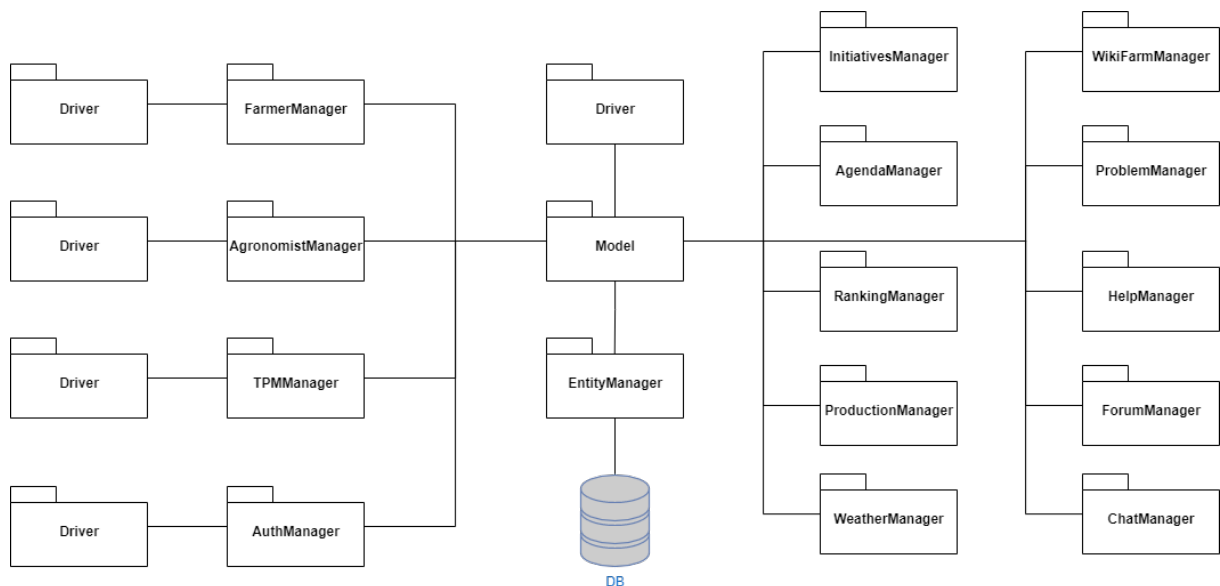


Figure 77: *Functionalities* implementation.

### 5.1.4 Integrating the Forecast Service

In this step, the WeatherManager functionality is implemented and tested by integrating the functionality with the model and implementing the API provided by the Telangana Weather Service.

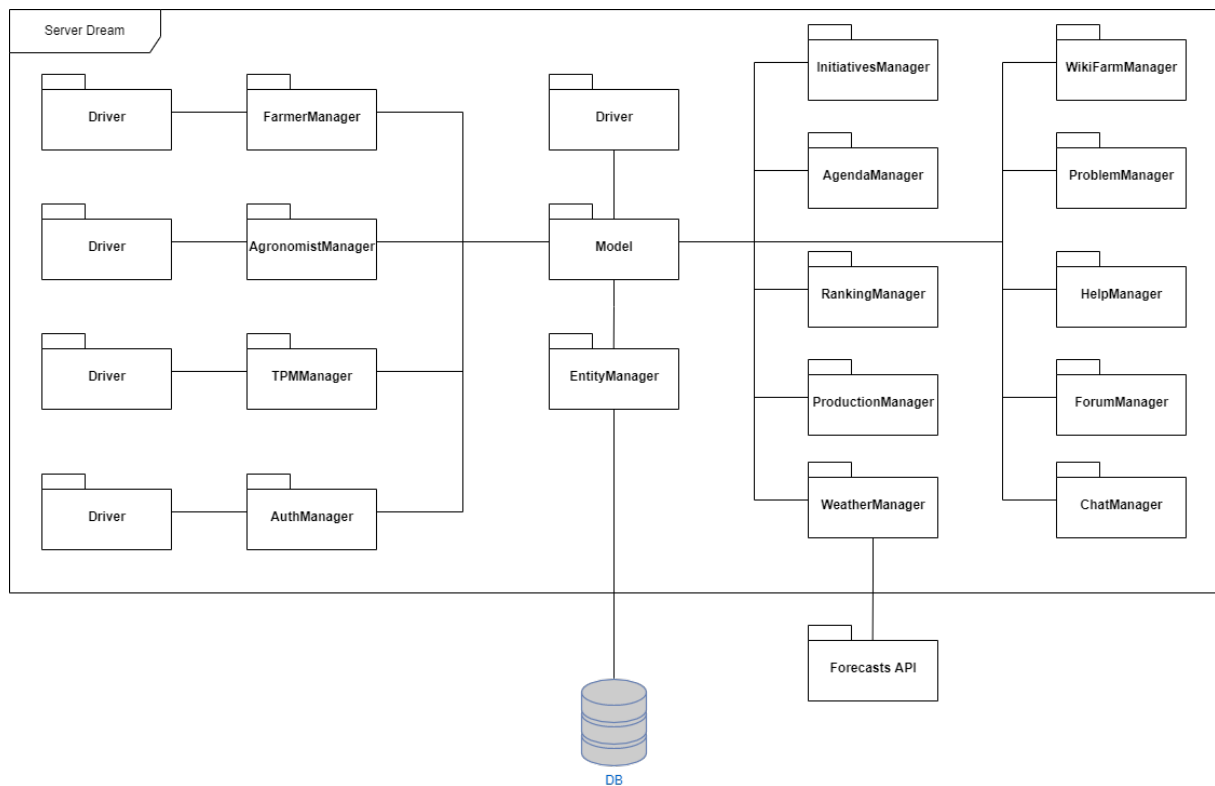


Figure 78: *Weather Service* implementation.



### 5.1.5 Testing the Sensor

The next step is to integrate the server dedicated to managing the Water Usage Sensor and the Humidity Sensor. Before connecting it with the DREAM Server, the operation of the Sensor Server model with sensors is tested.

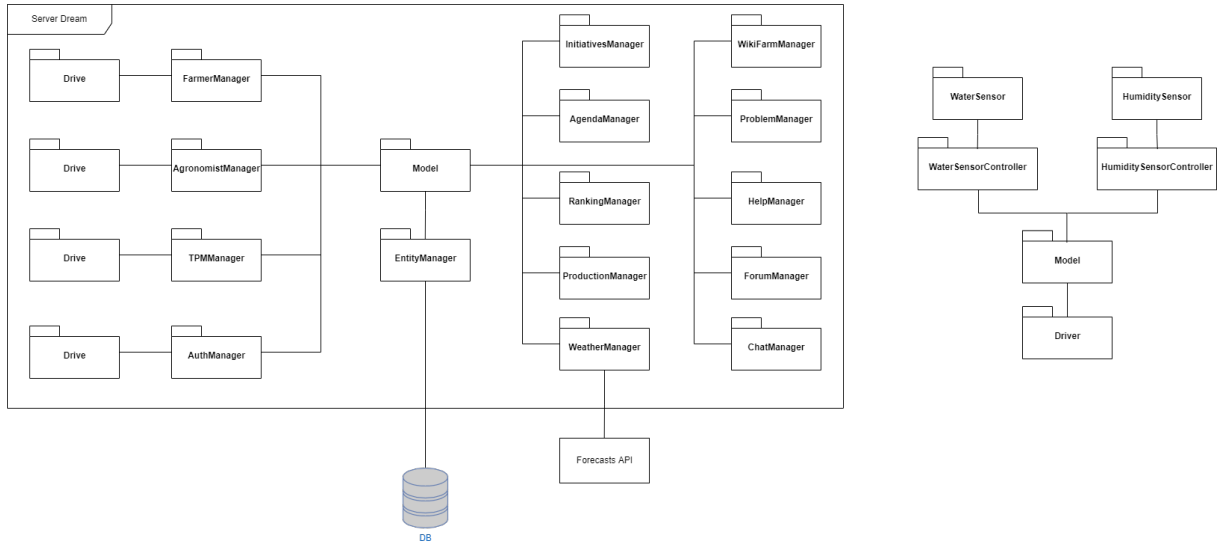


Figure 79: *Sensor Server* testing.

Once the correct functioning of the Sensor Server has been verified, we proceed to connect it to the Database using the same type of Entity Manager used in the DREAM Server.

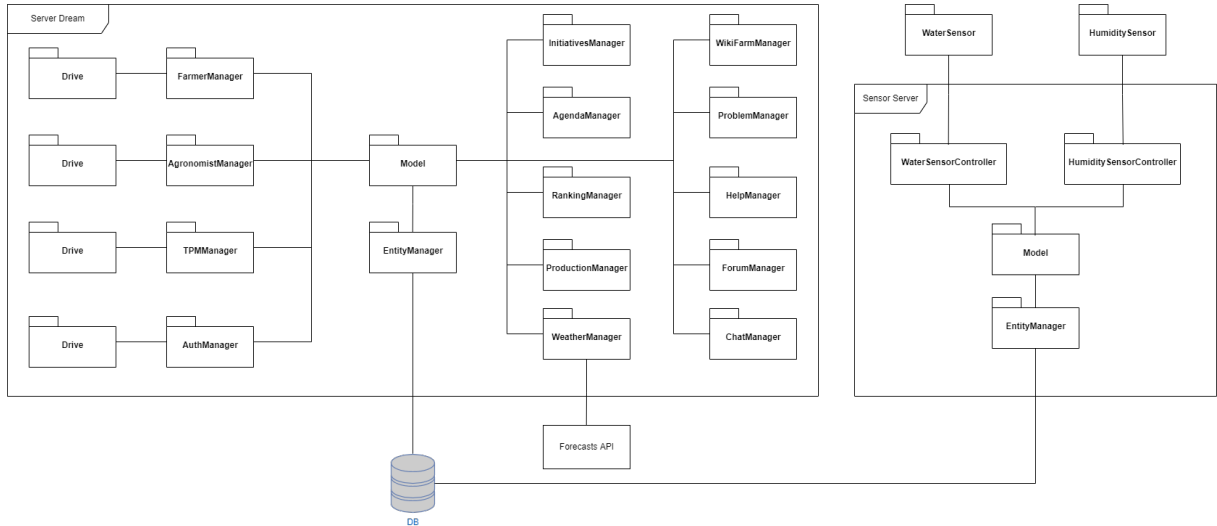


Figure 80: *Sensor Server* implementation.

### 5.1.6 Web App Integration

In this step, the thin client graphics are implemented, and it is checked that all DREAM functions are working properly.

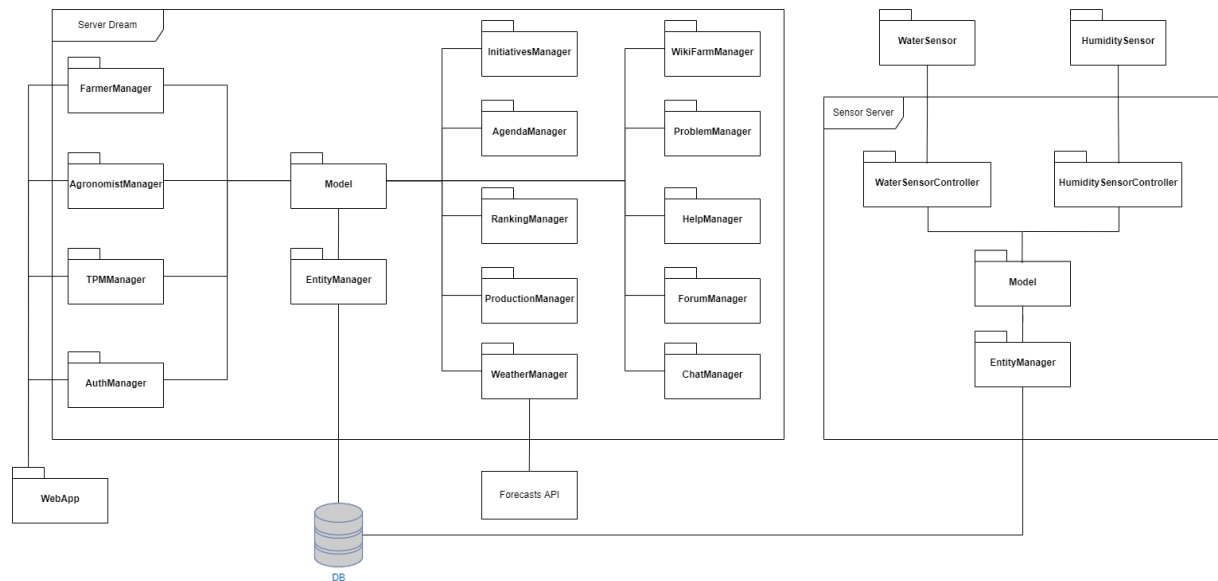


Figure 81: *Final* implementation.

### 5.1.7 Final Testing

Already in the previous step the system was working, but in this step the load balancers are tested and implemented, so as to verify and test the correct functioning of the system at full capacity and with a realistic workload.

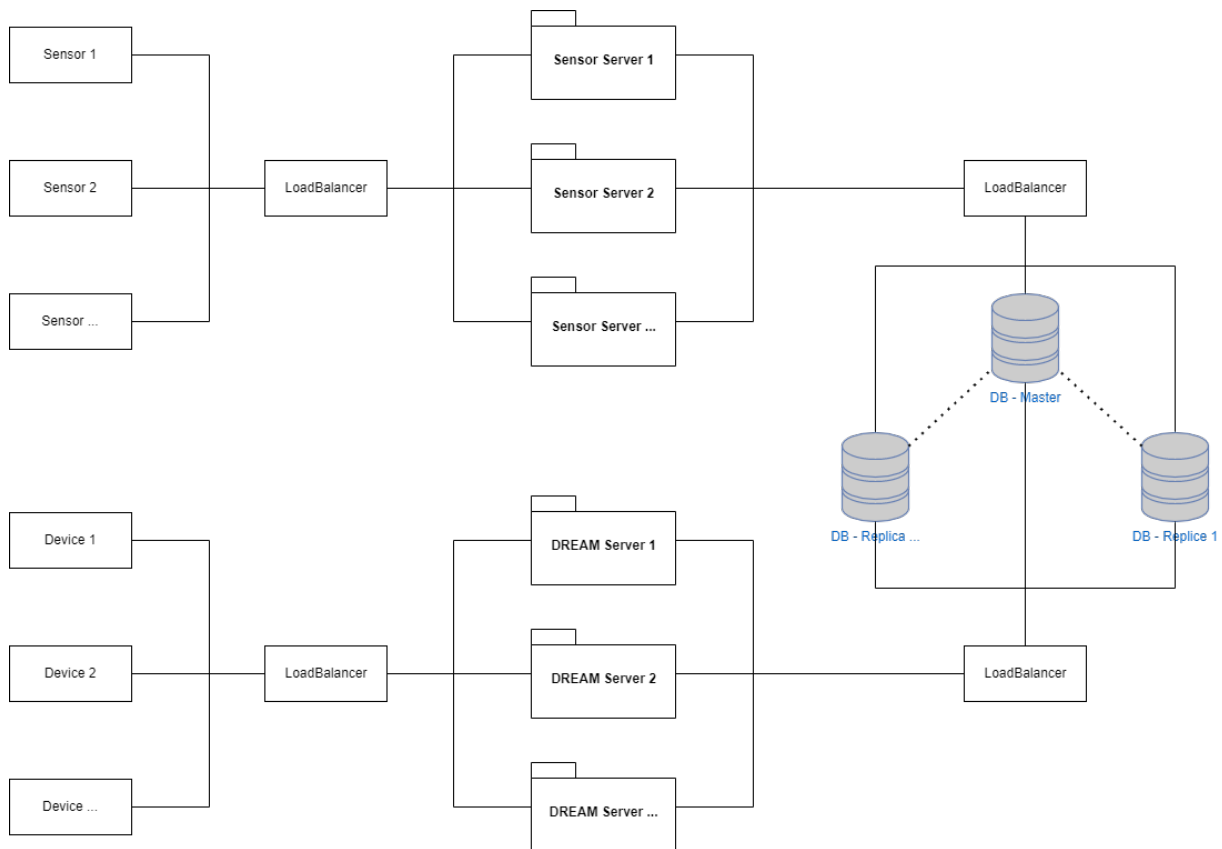


Figure 82: *Workload Testing.*

## **5.2 Technologies Used**

This section will analyze the possible technologies available to develop each component of the system.

### **5.2.1 Database**

SQLServer was chosen to implement the database because it is one of the most popular and widely used database software in the world, is open source, reliable and easy to keep up to date.

### **5.2.2 WebApp**

The WebApp developed must be compatible with all devices and browsers used by the user, which is why it will be implemented using a dynamic page that adapts automatically. For this reason the application will be developed in Java, using Thymeleaf for the template.

### **5.2.3 DREAM Server e Sensor Server**

The Sensor Server and DREAM Server will be implemented in Java EE, using TomEE application Web Server. We chose Java EE because it is the most widely used language in server development, is highly supported and offers numerous specially developed APIs. For example, the Java Persistence API (JPA) allows rapid interaction with the database.

### **5.2.4 Testing Tool**

Automated tests help ensure that the applications perform correctly before publishing them, while retaining features and bug fixes velocity. Junit is used for the unit testing framework for the Java programming language. It allows to define the test for each of the parts developed in a Java application. It is very well documented and allows to be integrated with other testing tools.

## 6 Effort Spent

In this section is provided an overview of the effort spent to write this document.

### 6.1 Team Work

Task	Hours
Initial briefing	2
Chapter 2	8
Mock-ups	2
Document Revision	6

### 6.2 Stefano Staffolani

Task	Hours
Mock-ups	10,5
Revision	1
Sequence Diagram	6,5
Database Schema	1

### 6.3 Stefano Taborelli

Task	Hours
Section 2	3
Revision	1
Section 5	2
Section 1	1
L <sup>A</sup> T <sub>E</sub> X	7

### 6.4 Matteo Viafora

Task	Hours
Mapping	4
Archiemate	2
Sequence Diagram	6,5