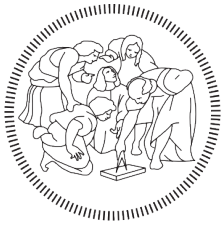


POLITECNICO
MILANO 1863

Computer Science and Engineering

Design Document

DREAM - Data-Driven Predictive Farming in Telangana

Software Engineering 2 Project
Academic year 2021 - 2022

January 2022
Version 1.0

Authors:
Samuele Mariani [10622653]
Alessandro Molteni [10623928]
Matteo Monti [10622780]

Professor:
Matteo Giovanni ROSSI

Contents

Chapter 1

Introduction

1.1 Purpose

The purpose of this document is to present a complete and detailed *Design description* of the DREAM system, by illustrating each major design decision made for the system.

The document contains a complete overview of the system, from the general architecture layout to the design of each specific component and their interfaces. The document will also contain information concerning the physical deployment of the components and a presentation of some of the *user graphical interfaces*. A discussion on the Implementation and Testing plan will also be addressed.
This *Design Document* will also provide a detailed description showing how the requirements and use cases discussed in the *Requirement Analysis and Specification Document*, are implemented by the system.

This document is aimed for the software developers and testers in order to provide them with a roadmap.

1.2 Scope

The DREAM system allows the user to retrieve useful information based on the user role: *farmer*, *agronomist* or *policy maker*.

Each farmer has access to a *Personal page* from which they can navigate to all the different sections of the application.
The section named *My farm* contains useful information about the farm that can be edited and updated. The farm production can be updated in the *Manage Crops* section which also provides a list of the current crops.
Additionally, farmers have access to a *Forums* section where they can create a specific forum tackling a particular topic or join conversations on already existing forums in order to exchange ideas and tips with other farmers.
The personal page also grant access to a *Weather Forecasts* section that will display up-to-date weather conditions for the incoming days as well as data regarding temperature and humidity.
Finally farmers have the possibility to ask for help on specific problems in the *Help!* by submitting an help request that will be answered by other competent farmers and by the agronomist that oversees the area.

Agronomists have access to an *Agronomist page* from which they can navigate to a *Region overview* section that provides an overview of the area of their competence complete with a

ranked list of the farmers of the area. Agronomists can visualize the personal page of each farmer in the list. They have access to the *Help!* section described above where they can answer to questions made by farmers of their area.
Finally agronomists can access their *Daily plan* section which displays an up-to-date daily schedule about farms that are yet to be visited. The schedule can be modified by the agronomist at every time.

Policy makers have access to a *Region overview* section in which a list of the best farmers in the whole region is displayed.

1.3 Glossary

1.3.1 Definitions

Term	Definition
Farmers	Identifies Telangana's farmers logged into the application.
Agronomists	Identifies the agronomists who have access to the system.
Policy makers	Identifies the policy makers who are involved in the project.
System	The set of hardware devices and software applications involved in the provided operations.
Forum	A web portal in which farmers can talk to each others and exchange advice.

1.3.2 Acronyms

Acronyms	Term
DREAM	Data-dRiven PrEdictive FArMing in Telangana
UI	User Interface

1.3.3 Abbreviations

Abbreviations	Term
R	Requirement

1.4 Reference documents

- Project assignment specification document.

1.5 Document Structure

This document is presented as it follows:

1. **Introduction**: contains a general description of the main functionalities of the system and a description of the whole document.

2. **Architectural design**: gives an high level overview of the system and subsystems architecture it also describes how the different subsystems interact with each other in order to provide all the functionalities.

3. **User Interface Design**: it describes the system functionalities from a user perspective by presenting all the different graphical interfaces. This section gives an idea on how the user will interact with the application.

4. **Requirements Traceability**: this section provides a cross-reference analysis to show how the requirements included in the RASD document are mapped on the different design elements defined in the Architectural design section.

5. **Implementation, Integration and Test Plan**: contains a detailed description of the implementation, integration and testing phases of the system's subcomponents.

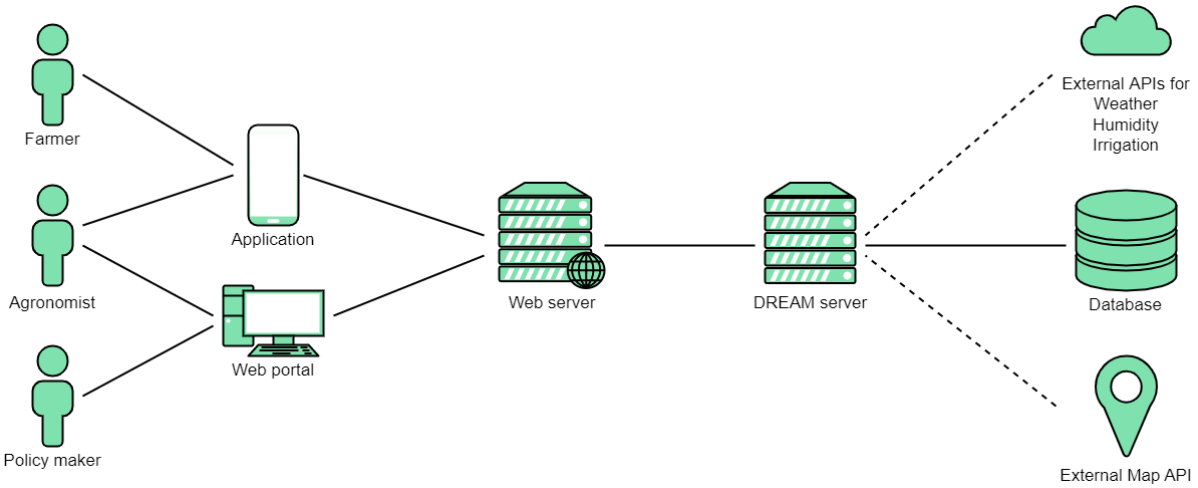
6. **Effort Spent**: keeps track of the time spent to complete this document.

7. **References**: contains a list of the different documents, materials and sources used to complete this document.

Chapter 2

Architectural Design

2.1 Overview



The system is composed by two main parts:

- **Client part**
  - *Mobile application*: it will be installed on the user's device. Farmers and agronomists will be able to download and login into the app with their credentials. This is the only way for farmers to interact with the system, and it is also recommended its use by agronomists for operations such as updating the daily schedule or answering help requests.
  - *Web portal*: agronomists and policy makers can access to the system via web browser. This portal is recommended for operations that require a wide screen, for example to visualize weather forecasts related to an entire region, or to visualize a farmers' ranking with the relative region map.
- **Server part**
  - *DREAM server*: this server will handle the communication through the web server and the DB. It will also retrieve the necessary data from the Map API. This will be the main server, where all the logic is located, and will take charge of all the complex operations.
  - *Web server*: this server will communicate with the clients through the internet.
  - *Database*: it will handle data storage.

– *External Map API*: will be used to convert farm addresses into GPS coordinates, in order to visualize the farms on the map. This API will also be useful to compute the distance between two farms, to create a better daily schedule for the agronomists and to optimize their visit plan.
– *External APIs for weather, humidity, irrigation*: will be used to provide weather forecasts to the stakeholders and to evaluate farmers' work. Irrigation and humidity APIs will be provided by Telangana state.

2.2 Component view

2.2.1 DREAM server

The following diagram describes the internal structure of the DREAM server, which is the core of the application. Indeed, it guarantees the communication between the web server, the Map API and the DB, and it also contains most of the business logic.

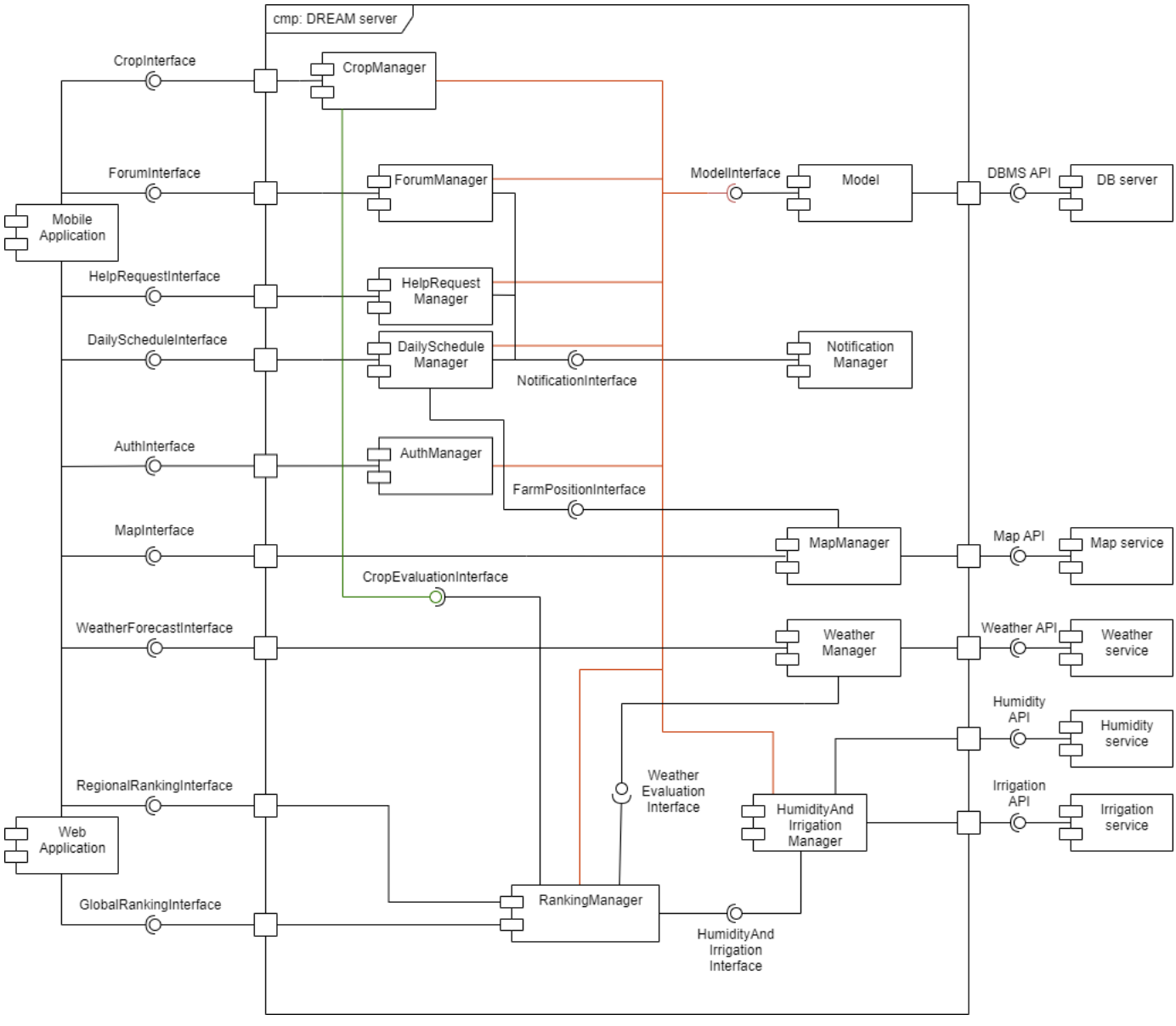


Figure 2.1: Server components

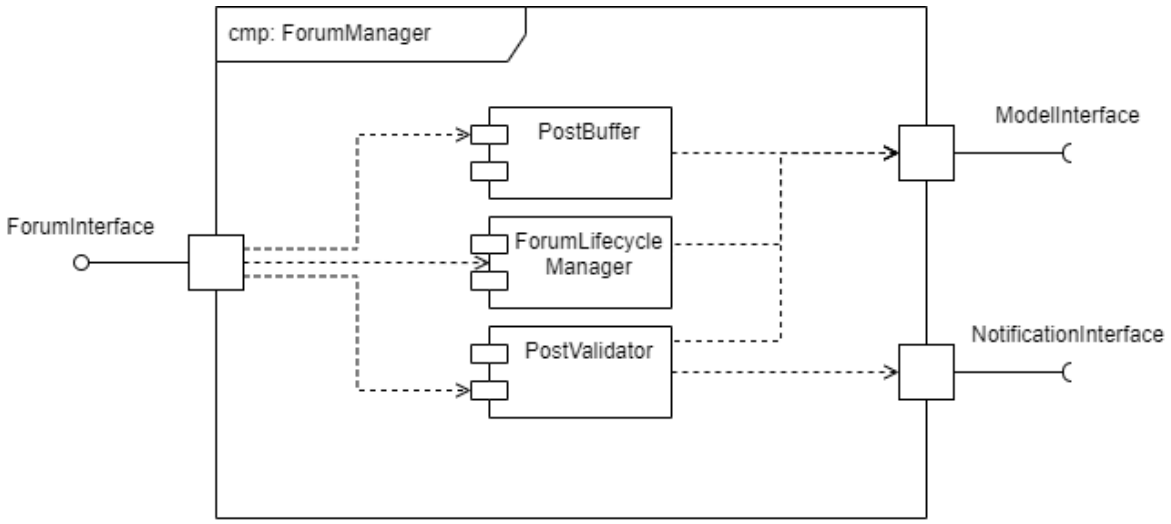
The components that make up the DREAM server are described in the following paragraphs, while the other components in the figure above will be discussed below.

- **Mobile application**: it is the application that will be installed on farmers' and agronomists' mobile devices. Upon authentication via **AuthInterface**, users will be allowed to perform actions based on their specific role. Most of the interfaces are common with the **Web application**, indeed agronomists can use either web or mobile app, performing the same actions.
- **Web application**: it represents the web portal that will be accessible from browser by agronomists and policy makers. Similar to **Mobile application**, the authentication is performed via **AuthInterface** and grants different permissions to the two possible categories of users.
- **DB server**: it contains all the system data, and provides the **DBMS API** interface to allow the DREAM server to write, read and delete them.
- **Map service**: it represents the external service used to deal with maps. It offers the **Map API** to visualize the farms on the map and to calculate the distance between two farms.
- **Weather service**: it represents the external service used to get weather forecasts and the past precipitation data. It offers the **Weather API** to retrieve these data.
- **Humidity service**: it represents the external service used to get data concerning the humidity of the soil, and will be provided by Telangana state. It offers the **Humidity API** to retrieve these data.
- **Irrigation service**: it represents the external service used to get data concerning the irrigation and water consumption for every farm, and will be provided by Telangana state. It offers the **Irrigation API** to retrieve these data.

2.2.2 Crop manager

Handles the operations regarding crop management. In particular, through the **CropInterface**, it allows farmers to insert a new **Planted** crop and to set a crop as **Harvested**. It also offers the **CropEvaluationInterface**, which sends relevant crop data to the RankingManager in order to evaluate farmers' crops. Farmers can also visualize a complete list of all crops **Harvested** crops they inserted during the years.

2.2.3 Forum manager



Handles the app features about forum management. It provides the mobile application with the **ForumInterface**, which allows farmers to create and close their own forums, and to create and validate new posts. The sub-components in the figure above are:

- **PostBuffer**: loads posts as them are read, indeed a forum may contain thousands of posts and if they are loaded all in once, the application may slow down significantly.
- **ForumLifecycleManager**: allows a farmer to create a forum, upon checks discussed in the RASD document, and also enables the forum admin to close it at any time.
- **PostValidator**: it performs a two-steps check on post creation
  - An automatic check for inappropriate language
  - A manual check by forum admin

  When a post is approved, a notification will be sent to its author

All the components described above interface with the Model, and the PostValidator component also interfaces with the NotificationManager.

2.2.4 HelpRequestManager

Handles the operation connected to help requests: creation and answering, both by farmers and agronomists. Furthermore, agronomists can also insert personal suggestions to farmers who requested for help. It provides the **HelpRequestInterface** to enable farmers and agronomists to perform the actions described above. Furthermore, this interface will be available for both mobile and web application.
This component interfaces with the Model and also with the NotificationManager: when an help request is created, a notification will be sent to the designed agronomist, and when the agronomist (or another farmer) answers to the request, the system will notify to the farmer that created the request.

2.2.5 DailyScheduleManager

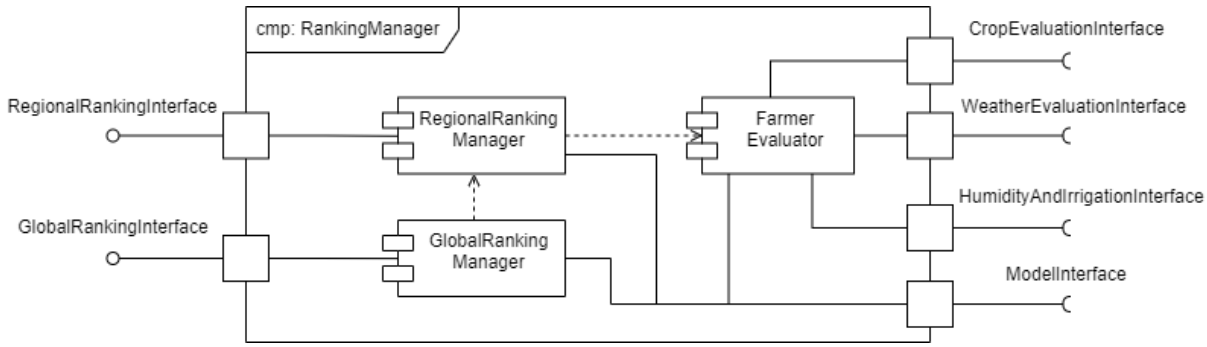
Allows agronomists to manage their own daily schedule through **DailyScheduleInterface** they can consult their scheduled visit, set a visit as **Completed** and move a visit to another day. The schedule is computed through the **FarmPositionInterface** to minimize the agronomist's displacements, also holding into account the minimum number of visits per year discussed in the RASD document.

2.2.6 AuthManager

This component will handle the login of each type of user. It will perform a two-steps authentication:

- **Authentication**: prompts the user for a username and password to verify their identity
- **Authorization**: if the submitted credentials are valid, the system grants to the user permissions based on its role

2.2.7 RankingManager



Handles the creation and visualization of farmers rankings. It provides the **RegionalRankingInterface** to both mobile and web application, and the **GlobalRankingInterface** to the web application. The sub-components that make up the RankingManager are:

- **FarmerEvaluator**: interfaces with the Model to require the list of **Harvested** crops for a certain farmer, and then evaluates every crop through the **CropEvalutaionInterface**, **WeatherEvaluationInterface**, **HumidityAndIrrigationInterface**, in order to assign a precise score to each farmer.
- **RegionaRankingManager**: requires to the Model the list of farmers in a certain region and then evaluates them one by one through the FarmerEvaluator. Finally, sorts the farmers by ranking
- **GlobalRankingManager**: requires to the Model the list of regions, and then for every region creates the farmers' list through the RegionalRankingManager

2.2.8 Model

Is the component that interfaces with the DB Server, so almost every other component in the DREAM Server needs to interface with it through the **ModelInterface**, that allows to retrieve, modify and delete data. Before data modification, checks will be performed to verify if the operation is allowed, and in negative case an exception will be sent to the caller.

2.2.9 NotificationManager

Handles the sending of notifications, providing the **NotificationInterface** to ForumManager, HelpRequestManager and DailyScheduleManager.

2.2.10 MapManager

This component interfaces with the external map API. It provides the **MapInterface** to both mobile and web application to visualize maps and farm locations, and the **FarmPositionInterface** to the DailyScheduleManager to evaluate farms' relative position.

2.2.11 WeatherManager

This component interfaces with the external weather API. It provides the **WeatherForecastInterface** to allow app users to see weather forecasts, and the **WeatherEvaluationInterface** to evaluate in a more precise way the farmers' work.

2.2.12 HumidityAndIrrigationManager

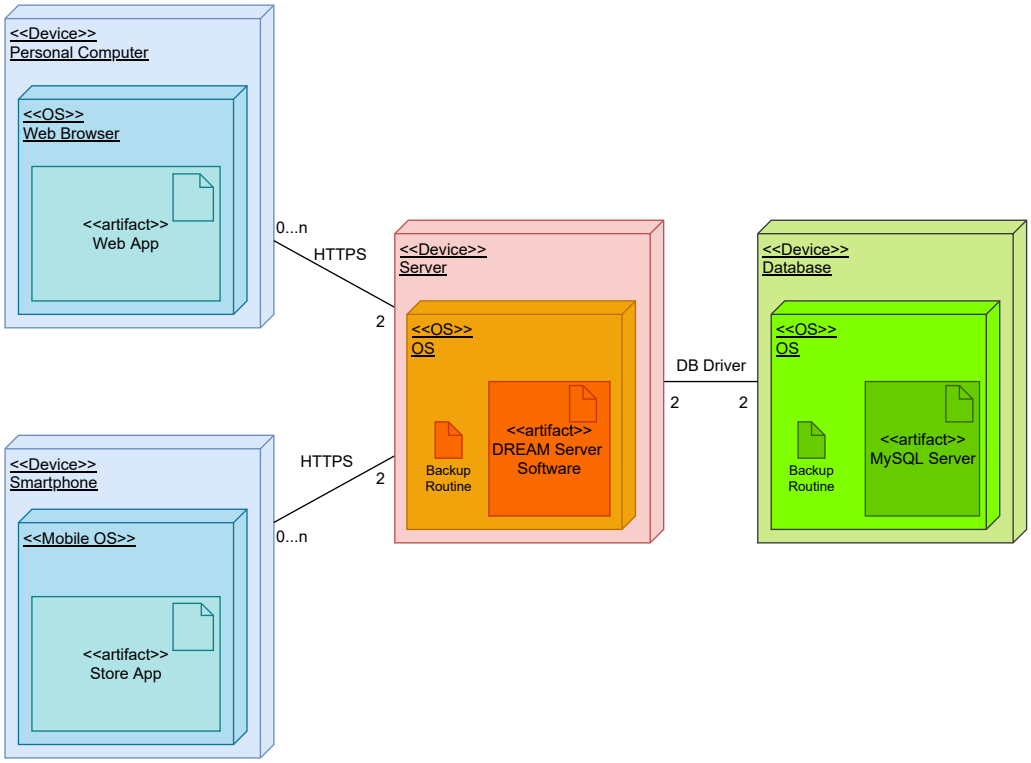
This component interfaces with the external humidity and irrigation APIs. It provides the **HumidityAndIrrigationInterface** to better rank the farmers, and is connected to the model through the **ModelInterface** to store the data coming from the APIs.

2.3 Deployment view

The system is developed using a three tier architecture, so each layer is implemented on a different hardware and can be scaled, developed or improved without necessarily impacting the others.

- **Presentation Layer**
  This is the front end and contains the user interfaces. It can communicate with other layers using API queries with the HTTPS protocol. The application can both be used on a smartphone (either IOS or Android) or via web, with the web app.
  - Personal Computer: This device is any computer using any browser acting as a client. This means the server requires an HTTPS access point that should support many devices.
  - Smartphone: Any mobile device should be able to install the application, either with Android or IOS. Developing two different dedicated application, one for Android and one for IOS, is a prefered solution since it is possible to exploit benefits of the two operating systems, but even a common application solution is possible.
- **Business logic Layer**
  It allows multiple components to access this layer and it contains the main functionalities of the application. This tier needs to be installed on a dedicated device.
- **Data Management Layer**
  It is the MySQL database that contains all informations about farmers, agronomists, policy maker and others related to them are stored. There is the data access layer which can be accessed via API calls containing queries.

In order to prevent some failures of the server or the database, a Backup routine takes place regularly which copies the content of both on another device. So, the server and the database are duplicated, but in order to maintain a unique state of the data, just a single database is update by the server, while the other is a copy of the main one update by the Backup routine. Differently, both servers are active at the same time and this not only increase the fault tolerance, but it also increases the performance since the client requests are evenly divided among the two servers.



2.4 Runtime View

This section describes the interactions happening between the different components when performing one of the core operations allowed by the system. Each operation shown below is described using an appropriate sequence diagram.

2.4.1 Mobile App user login

Each user will be authenticated by the system after having submitted their credential during the login procedure. The procedure is analogous for the Web application.

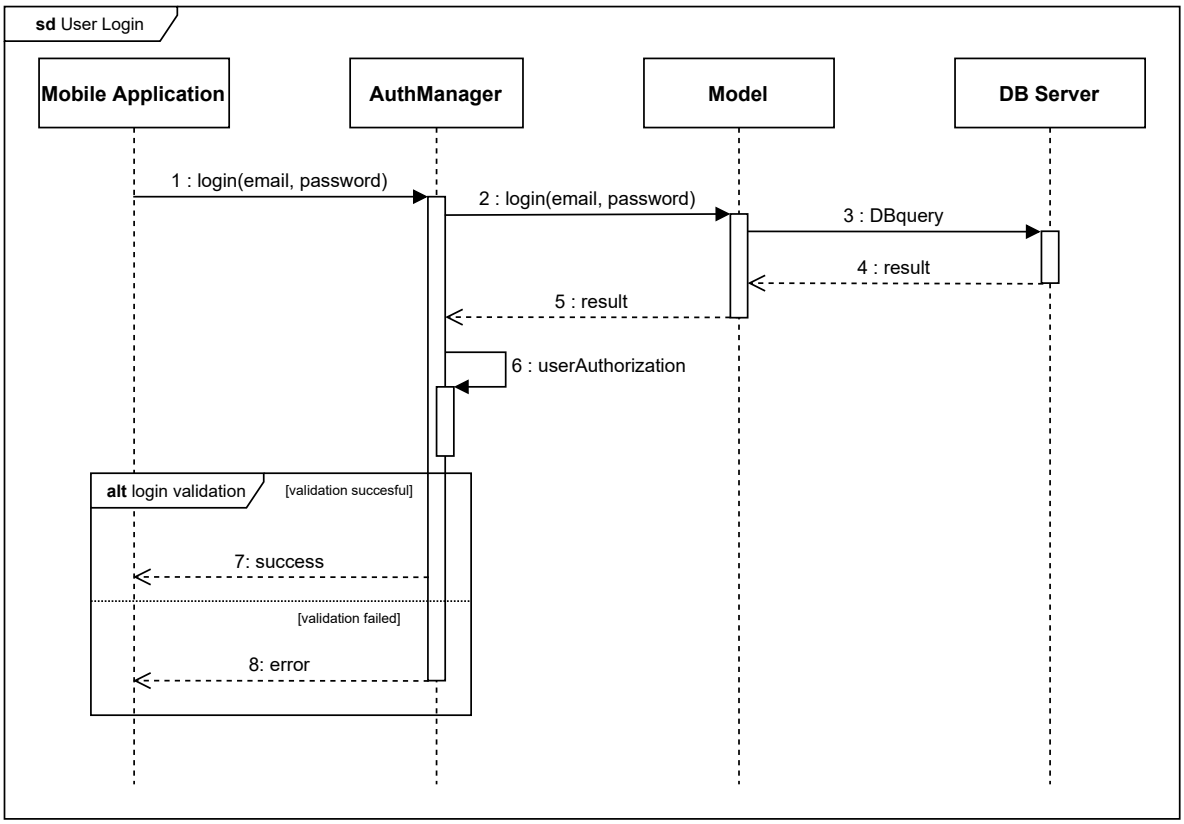


Figure 2.2: mobile app login sequence diagram

2.4.2 Mobile App user registration

When a new user tries to register, the system checks if an already registered user with the same email and name as the new one exists, if not the DB is updated with the new user and the procedure terminates. If such user exists an error message is displayed and the procedure has to be repeated.

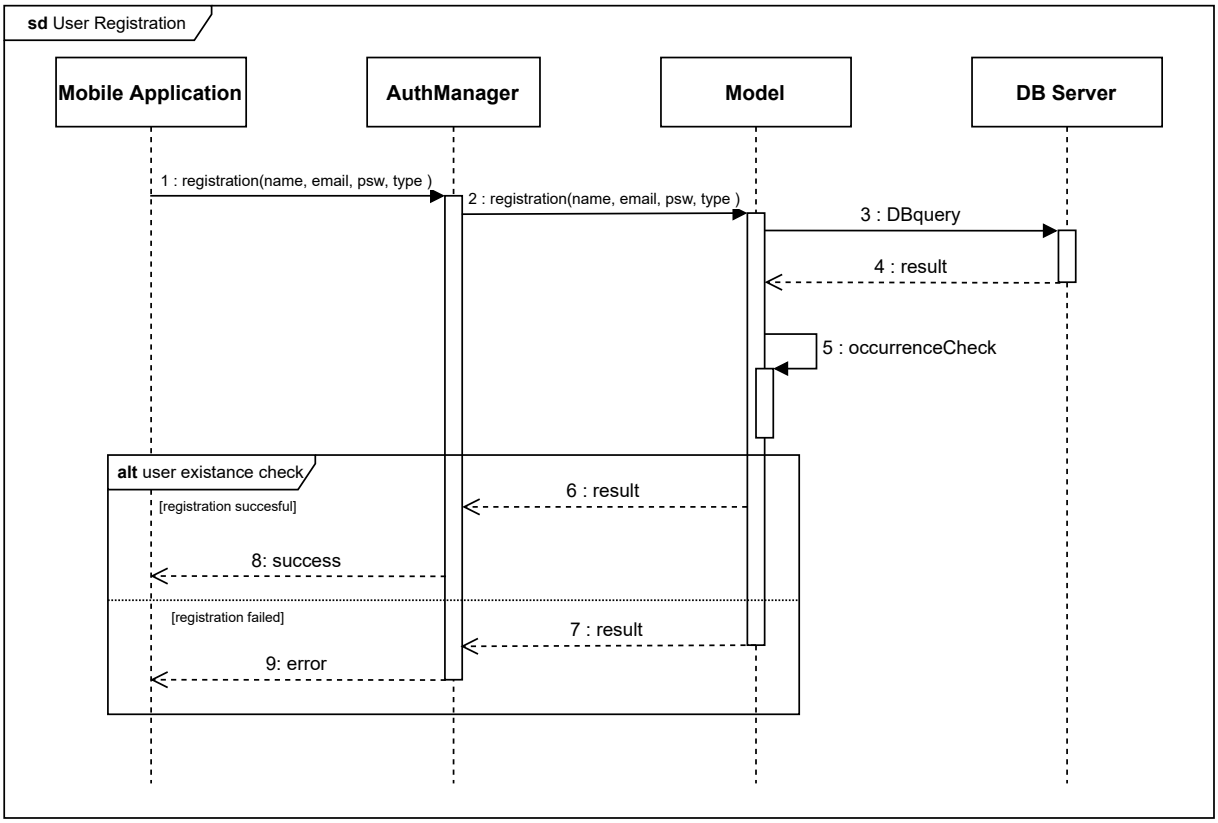


Figure 2.3: mobile app registration sequence diagram

2.4.3 The farmer's production is updated

An authenticated user of type **farmer** has the option to modify his production by adding newly planted crops or by harvesting already present ones. The diagram below shows the system's components interactions during these two operations.

When a new crop is added the system's DB is updated by the model and the procedure terminates.

When an existing crop is set as **harvested** the DB is updated with the new crop's status, the farmer new ranking is then calculated by the *RankingManager* component using the analytics retrieved from the *Humidity and Irrigation Manager* and the *Weather Manager* components, finally the DB is updated with the new ranking.

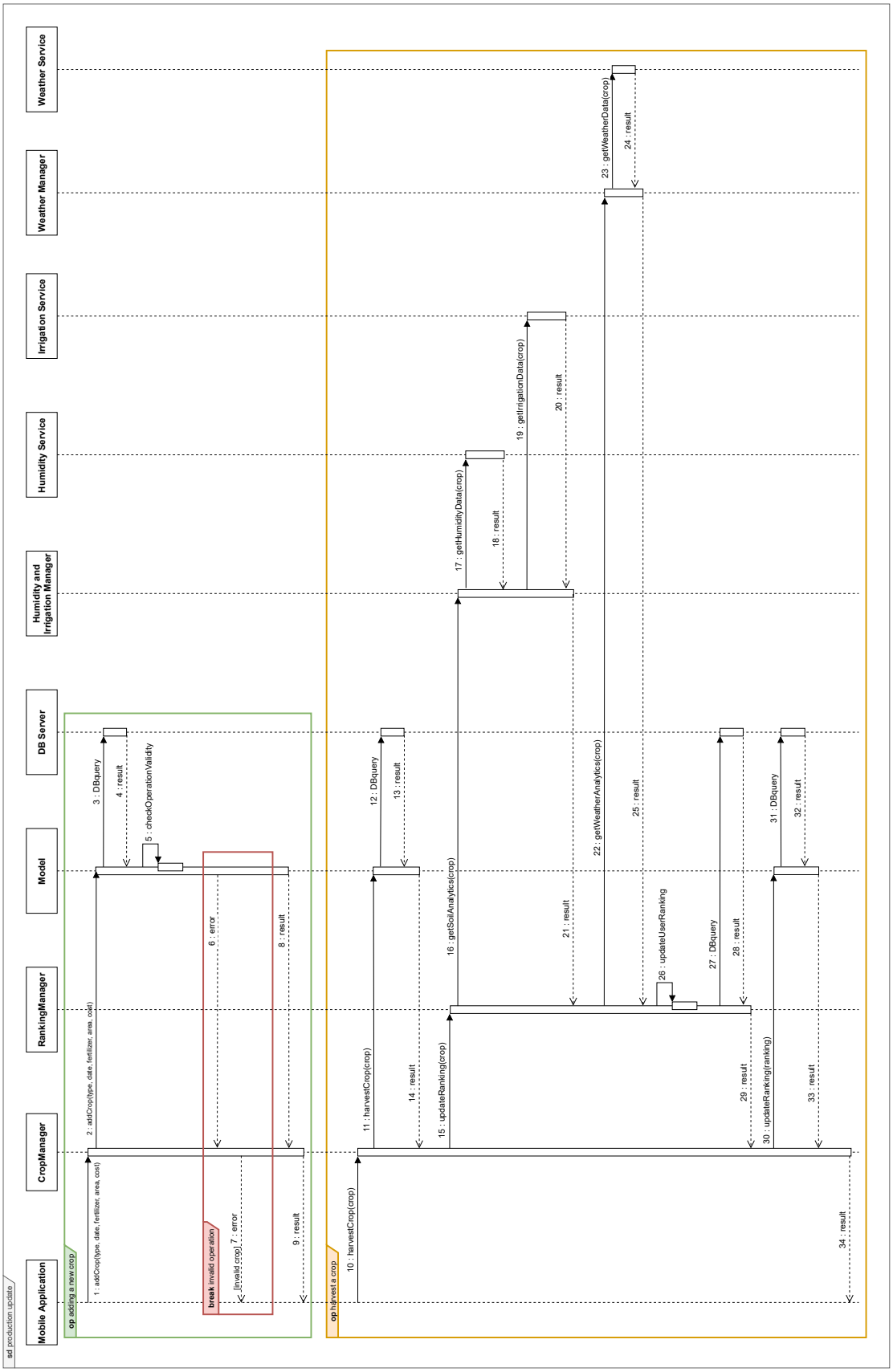


Figure 2.4: mobile app production update

2.4.4 Forum creation

When an authenticated user of type **farmer** creates a new forum, the models checks in the system's DB for an already existing forum with same name and topic, if such forum doesn't already exists the forum is created and the DB is updated, otherwise an error message is displayed.

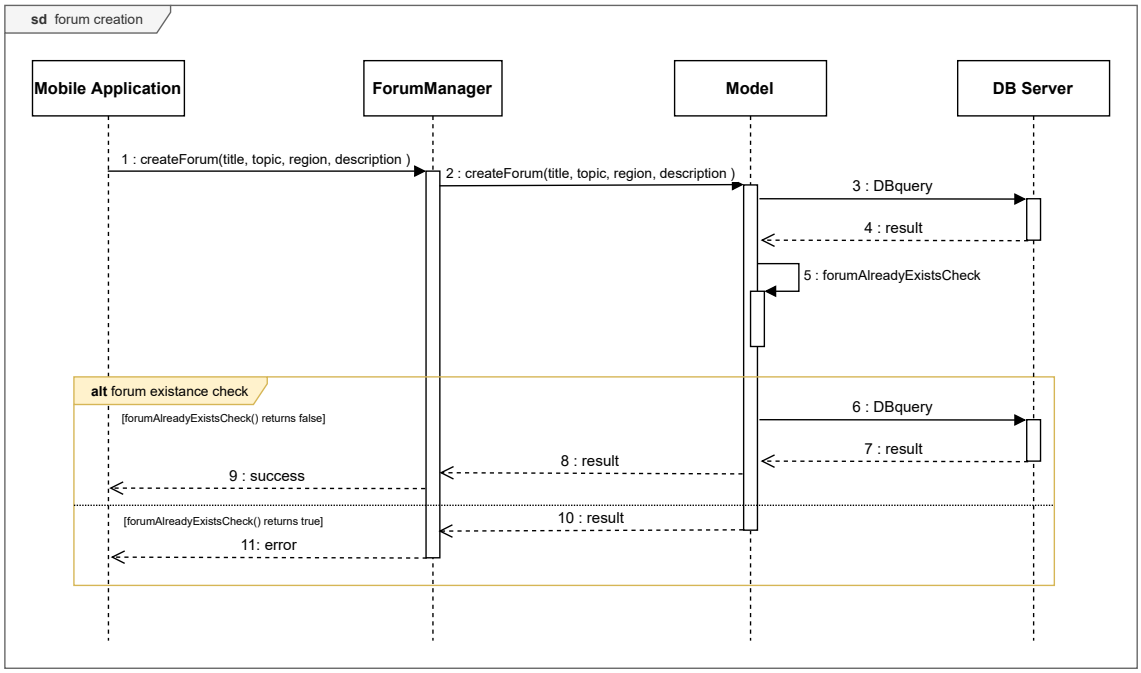


Figure 2.5: mobile app forum creation

2.4.5 Forum post submission

When an authenticated user of type **farmer** submits a new post the forum manager component performs the *post validation* routine to check for inappropriate language, if found the post is rejected, otherwise the DB is updated and a notification is sent to the user.

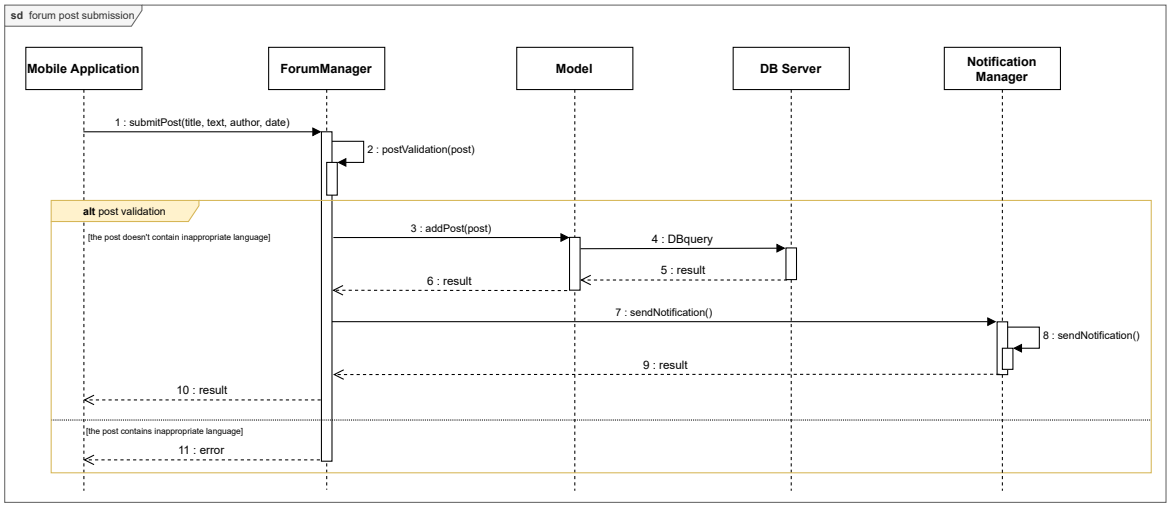


Figure 2.6: mobile app forum post submission

2.4.6 Help request submission

When an help request is submitted by an authenticated farmer the system DB is updated with the new request and a notification is sent to the agronomist of the farmer's region.

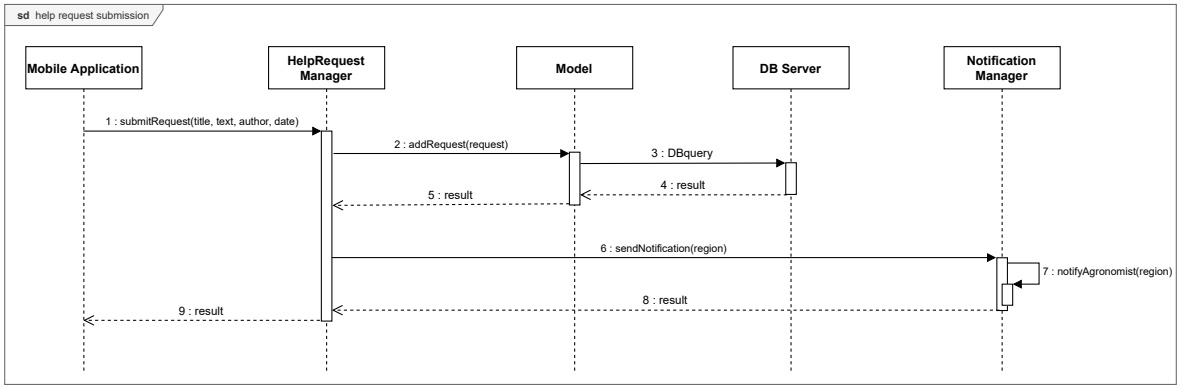


Figure 2.7: mobile app help request submission

2.4.7 Daily schedule update

An authenticated agronomist can modify his personal daily planner by setting completed visits as *completed* and by scheduling new visits.
When a visit is set as *completed* the Agronomist's schedule is recalculated in order to make sure that it's always optimized and then the DB is updated by the model component with the new schedule.
When a new visit is added in the schedule, the agronomist inserts the farmer's location and his new schedule is calculated by the *DailySchedule Manager* component using the data obtained from the *Map Manager*.

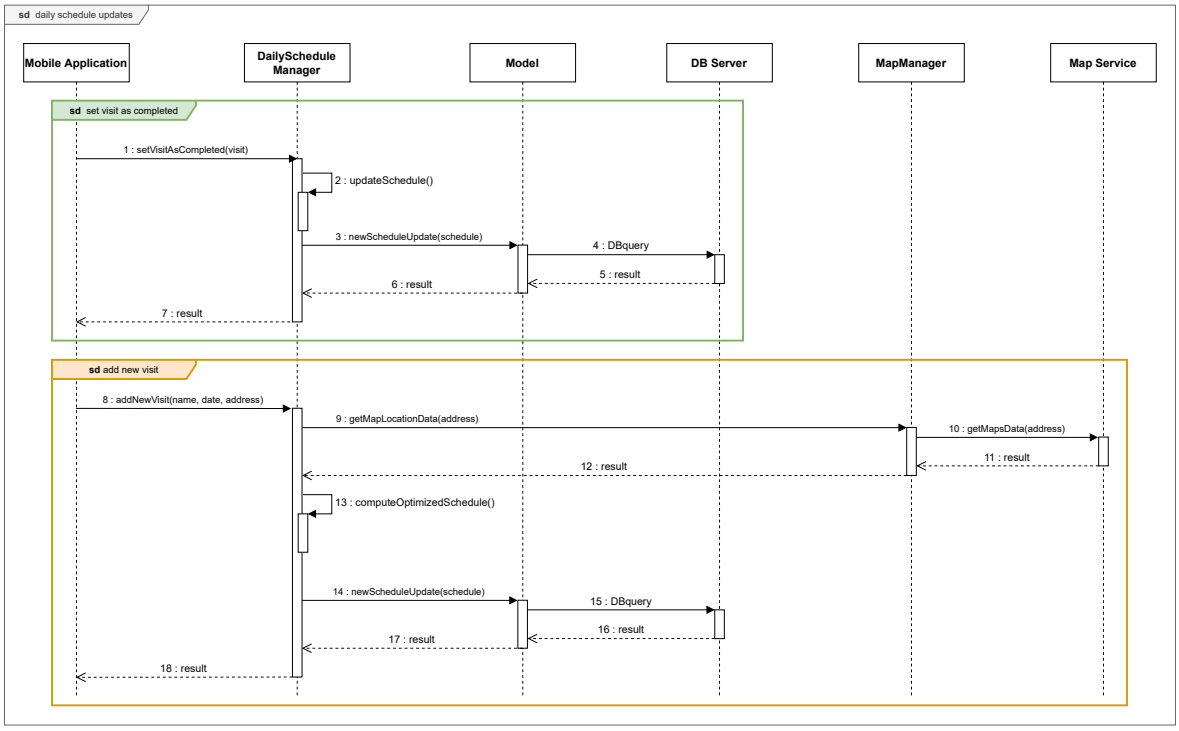


Figure 2.8: Agronomist's **daily schedule** updates

2.4.8 Help request reply submission

When an authenticated agronomist replies to a farmer help request, the request is set as *Answered* and a notification is sent to the farmer that submitted the help request.

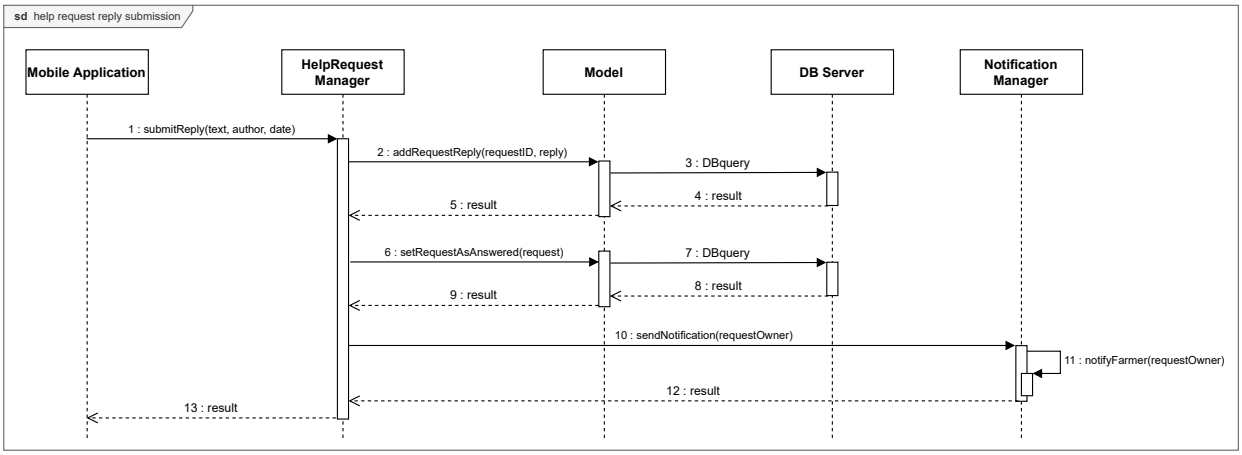


Figure 2.9: An agronomist replies to a farmer help request

2.4.9 Regional ranking check

An authenticated Policy Maker can visualize regional ranking for each region of the state as well as a global ranking including all farmers of the state. The ranking is retrieved by the Ranking manager component.

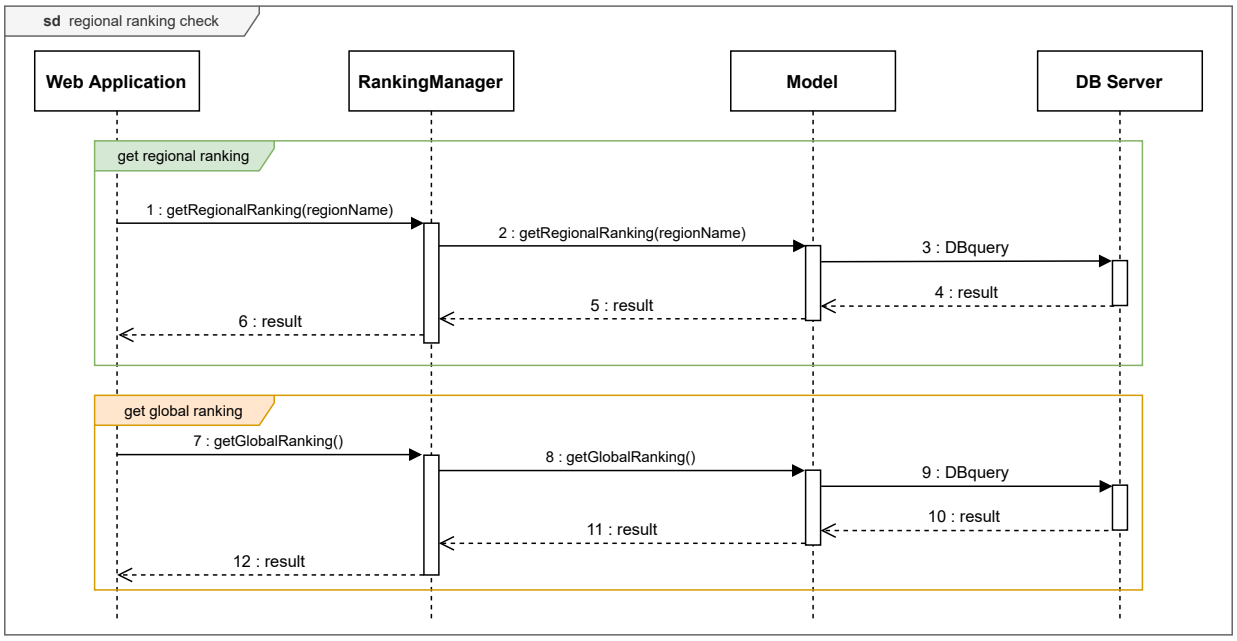


Figure 2.10: A policy maker checks regional and global ranking

2.5 Component Interfaces

This section contains the interfaces various component offer with their respective dependencies (shown as arrows). In particular, all method's signature are displayed.

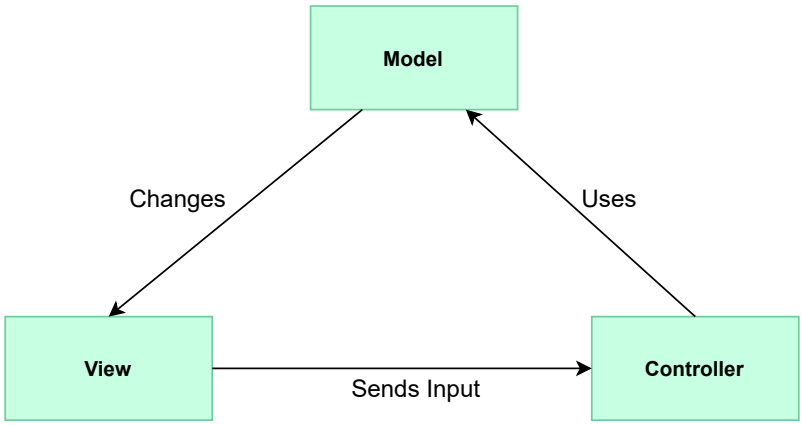
- CropInterface
  - addCrop(type, date, fertilizer, area, cost)
  - harvestCrop(crop)
  - updateRanking(crop)
  - updateRanking(ranking)
- ForumInterface
  - createForum(title, topic, region, description)
  - addPost(post)
  - postValidation(post)
  - sendNotification()
- ModelInterface
  - checkOperationValidity()
  - DBQuery()
- FarmPositionInterfaceInterface
  - getFarmPosition(farm)
- CropEvaluationInterface
  - evaluateCrop (crop)
- weatherEvaluationInterface()
  - evaluateWeather()
  - assignEvaluationToFarmer(farmer. date)
- HelpRequestInterface
  - addRequestReply(requestID, reply)
  - setRequestAsAnswered(request)
  - sendNotification(requestOwner)
  - addRequest(request)
  - sendNotification(region)
- DailyScheduleManager
  - updateSchedule()
  - getMapLocationDate(address)
  - computeOptimizedSchedule()
  - newScheduleUpdate(schedule)

- NotificationInterface
  - sendNotification()
  - notifyFarmer(requestOwner)
  - notifyAgronomist(region)
- AuthInterface
  - login(email, password)
  - registration (name, email, password, type)
- MapInterface
  - getMapData(address)
- WeatherInterface
  - getWeatherData(crop)
- HumidityAndIrrigationInterface
  - getHumidityData(crop)
  - getIrrigationData(crop)
- RegionalRankingInterface
  - getRegionalRanking(regionName)
  - getSoilAnalytics(crop)
  - getWeatherAnalytics(crop)
  - updateUserRanking()
- GlobalRankingInterface
  - getGlobalRanking()
  - getSoilAnalytics(crop)
  - getWeatherAnalytics(crop)
  - updateUserRanking()
- WebApplication
  - login(email, password)
  - registration(name, email, password, type)
  - getRegionalRanking(regionName)
  - getGlobalRanking()
- MobileApplication
  - login(email, password)
  - registration(name, email, password, type)
  - createForum(title, topic, region, description)
  - submitPost (title, text, author, date)
  - submitReply(text, author, date)
  - setVisitAsCompleted(visit)

- addNewVisit(name, date, adress)
- submitRequest(title, text, auther, date)
- addCrop(type, date, fertilizer, area, cost)
- harvestCrop(crop)

2.6 Selected architectural and styles and patterns

- **Model View Controller**: It's a software design pattern that divides the program logic into three different components. This architecture is used to have a better organised software with each part having a specific job.
  - Model: It is independent from the user interface and it's the central part which manages the data from the database. This part is competence of the server.
  - View: It is the part directly related to the user. Each device display a different view which can be used to interact with the application.This part is competence of the client.
  - Controller: It is the input part that register the user's actions, check their validity, and afterwards sends them to the model which will actually perform those actions on the database. This part is competence of the client, but it can be divide among client and server.



- **Three tier architecture**: as described in chapter 2.3, the DREAM application uses the three tier architecture that divides the presentation, server and database layer in three different hardware components. Since the DREAM application is a distributed system, this architecture is used to better handle different types of clients.
- **Client Server Architecture**: in this Architecture clients and server are logically separated and acts as different entities. They communicate over a network where the client sends requests to the server and the server responds. This architecture is used to handle connections and exchange of informations between every client and the database.
- **Rest API**: It is a series of commands used for data gathering. This data can be collected with HTTPS and stored in any format, for example in JSON. This architecture is used for the Weather API.

Chapter 3

User Interfaces Design

3.1 UI mockups

In the following section the DREAM mobile and web application core **User Interfaces** are presented, the DREAM application is available to users in both mobile and web form depending on their user type. Farmers will download and use the app on their mobiles, policy makers are required to use the web version while agronomist may use both versions.

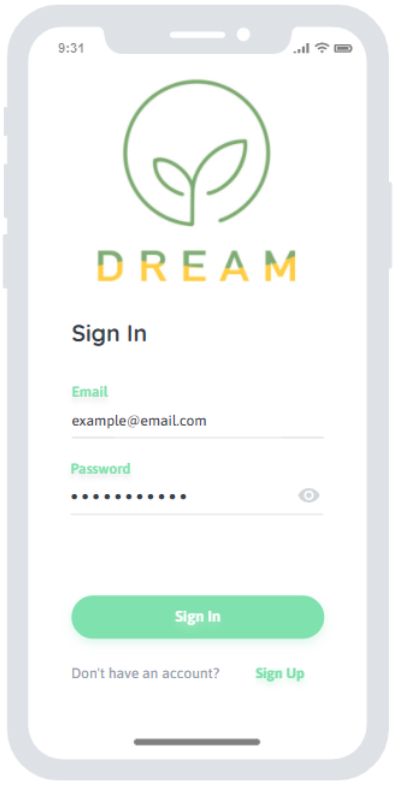


Figure 3.1: mobile app *Sign In* interface

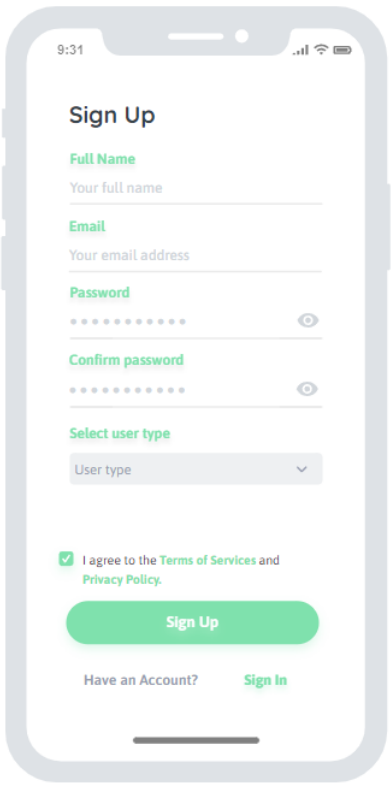


Figure 3.2: mobile app *Sign Up* interface

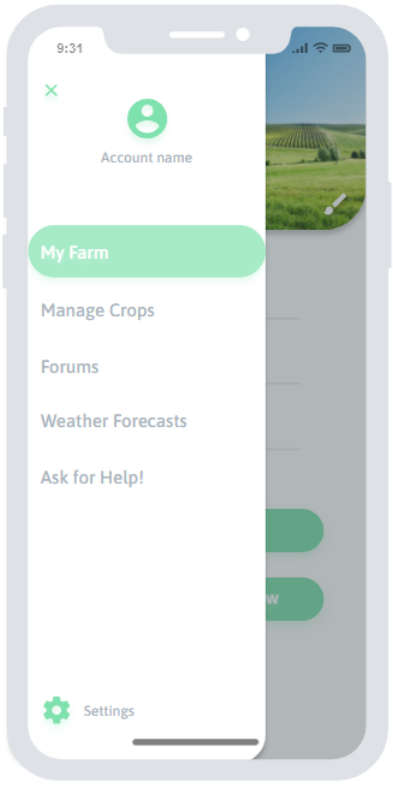


Figure 3.3: Farmer's *Navigation Menu* interface

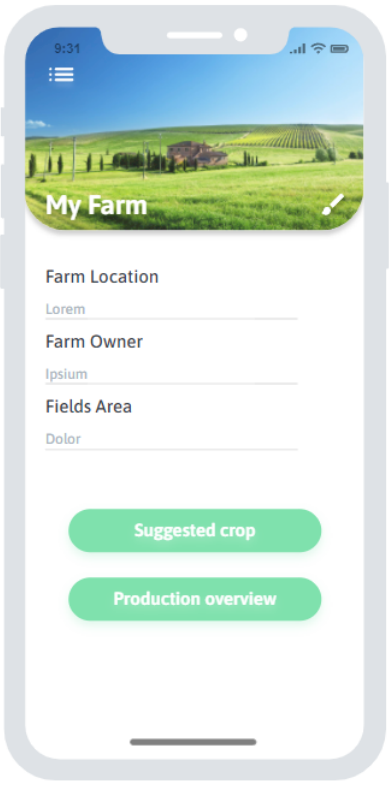


Figure 3.4: Farmer's *My Farm* interface

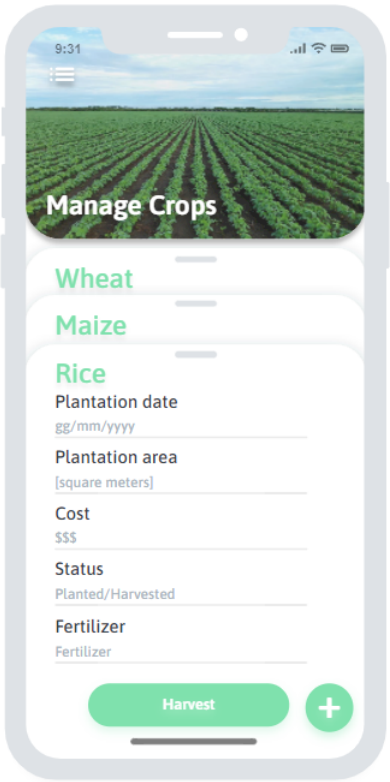


Figure 3.5: Farmer's *Manage Crops* interface

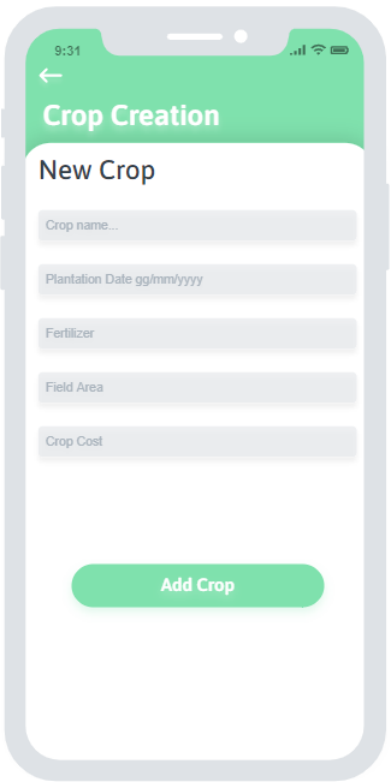


Figure 3.6: Farmer's *Crop creation* interface

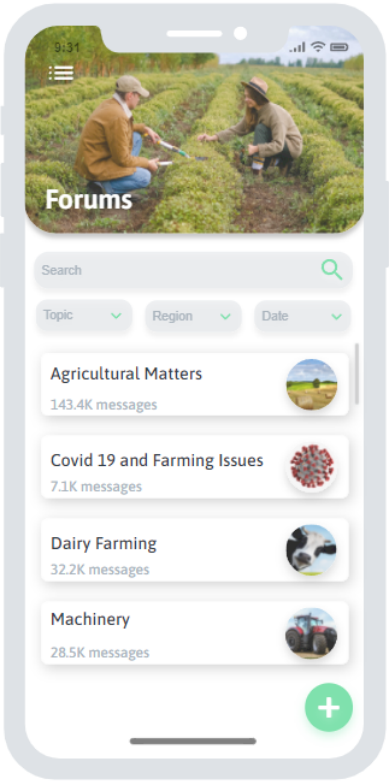


Figure 3.7: Farmer's *Forum* interface

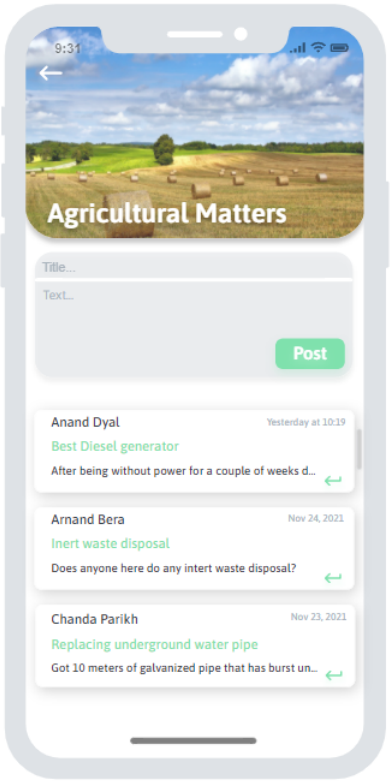


Figure 3.8: Farmer's *Example Forum* interface

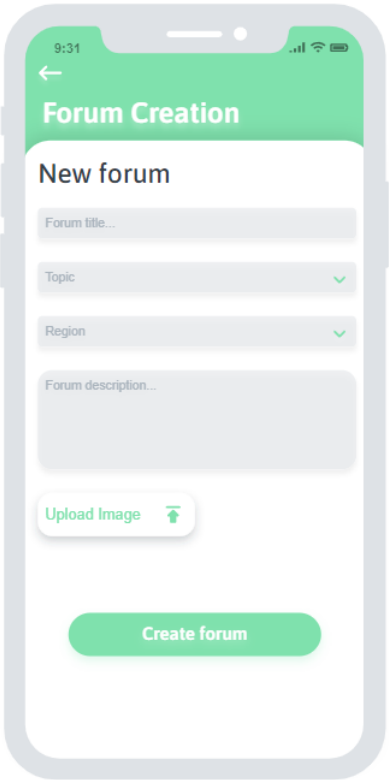


Figure 3.9: Farmer's *Forum creation* interface

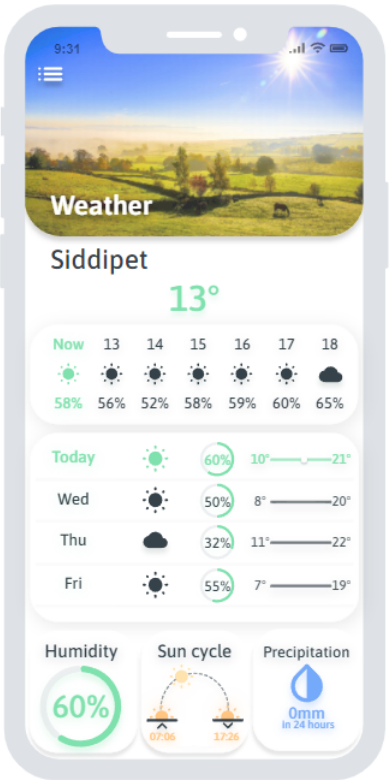


Figure 3.10: Farmer's *Weather* interface

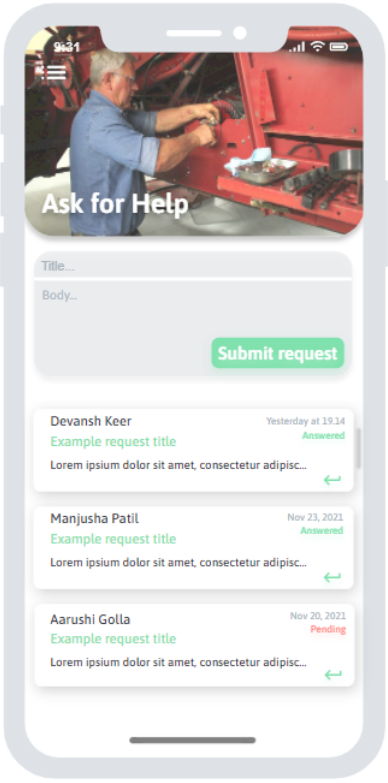


Figure 3.11: Farmer's *Ask for Help!* interface

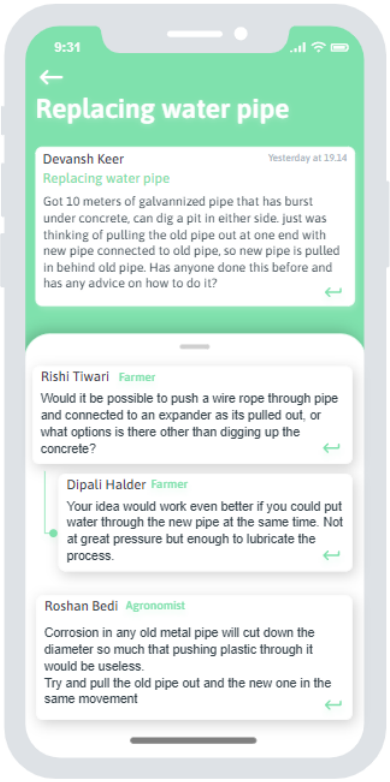


Figure 3.12: Farmer's *example Help request* interface

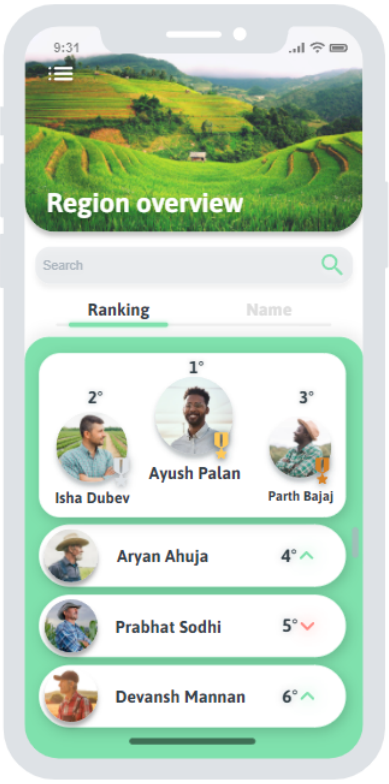


Figure 3.13: Agronomist's *Region overview* interface

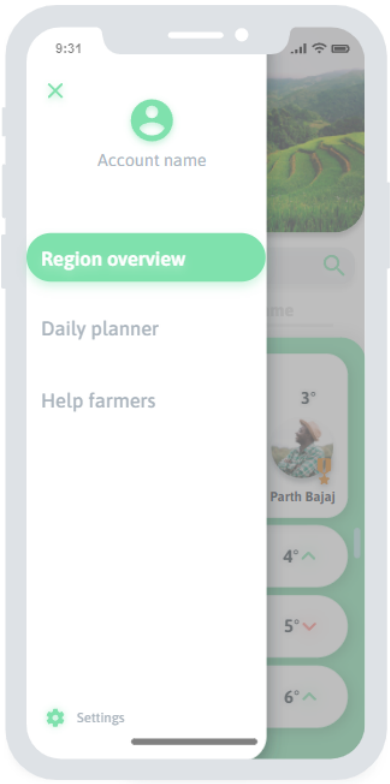


Figure 3.14: Agronomist's *Navigation menu* interface

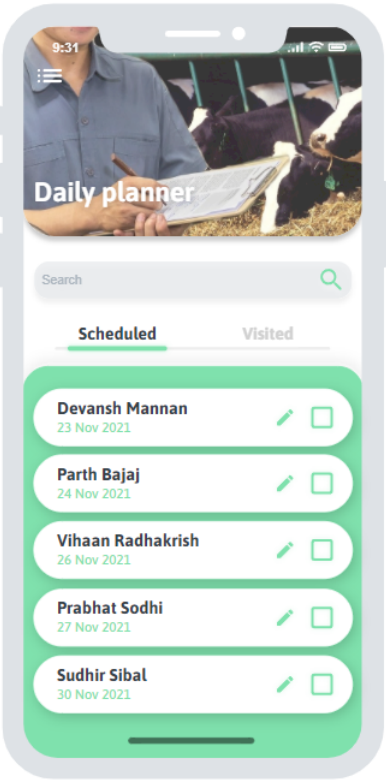


Figure 3.15: Agronomist's *Daily planner* interface

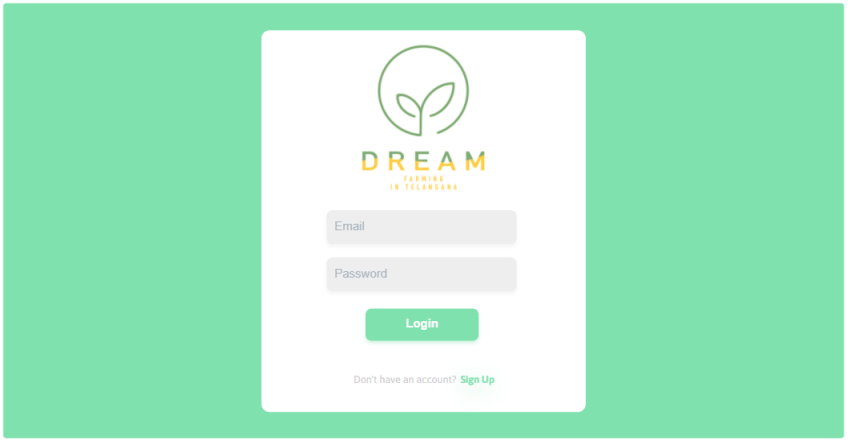


Figure 3.16: Website *Sign In* interface

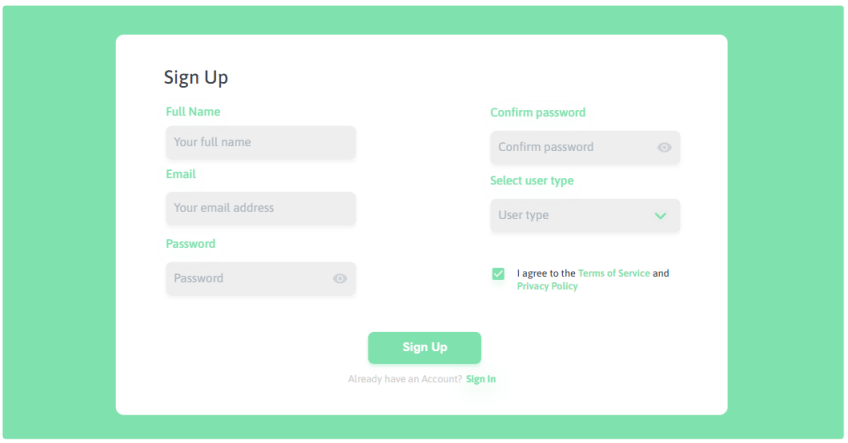


Figure 3.17: Website *Sign Up* interface

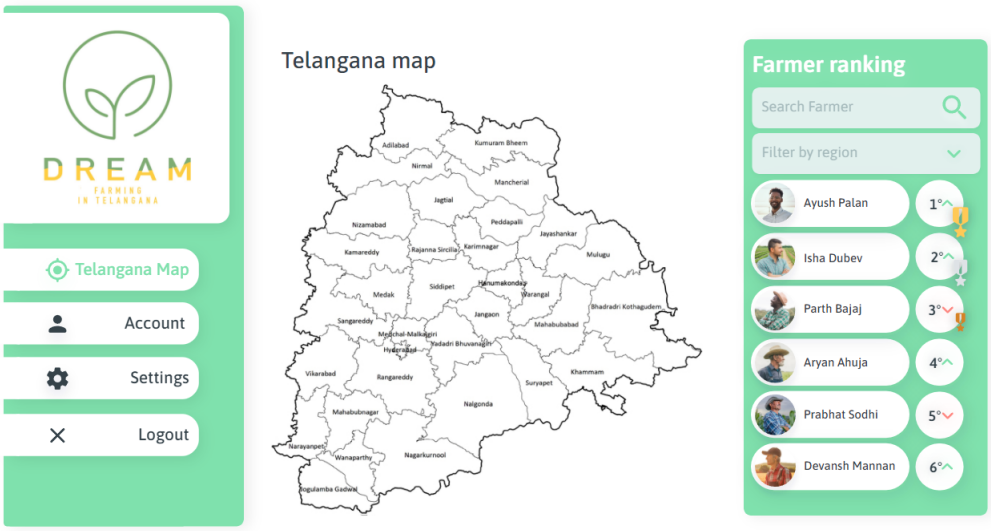


Figure 3.18: Policy maker's dashboard interface

3.2 Mobile application flow diagram

The following diagrams represent the user (of type *Farmer* and *Agronomist*) interface flow through the mobile application.

3.2.1 User login and registration flow

Once the user has logged in the application, his main interface is displayed. In the diagram below a user of the type *Farmer* is shown his *My Farm* interface.

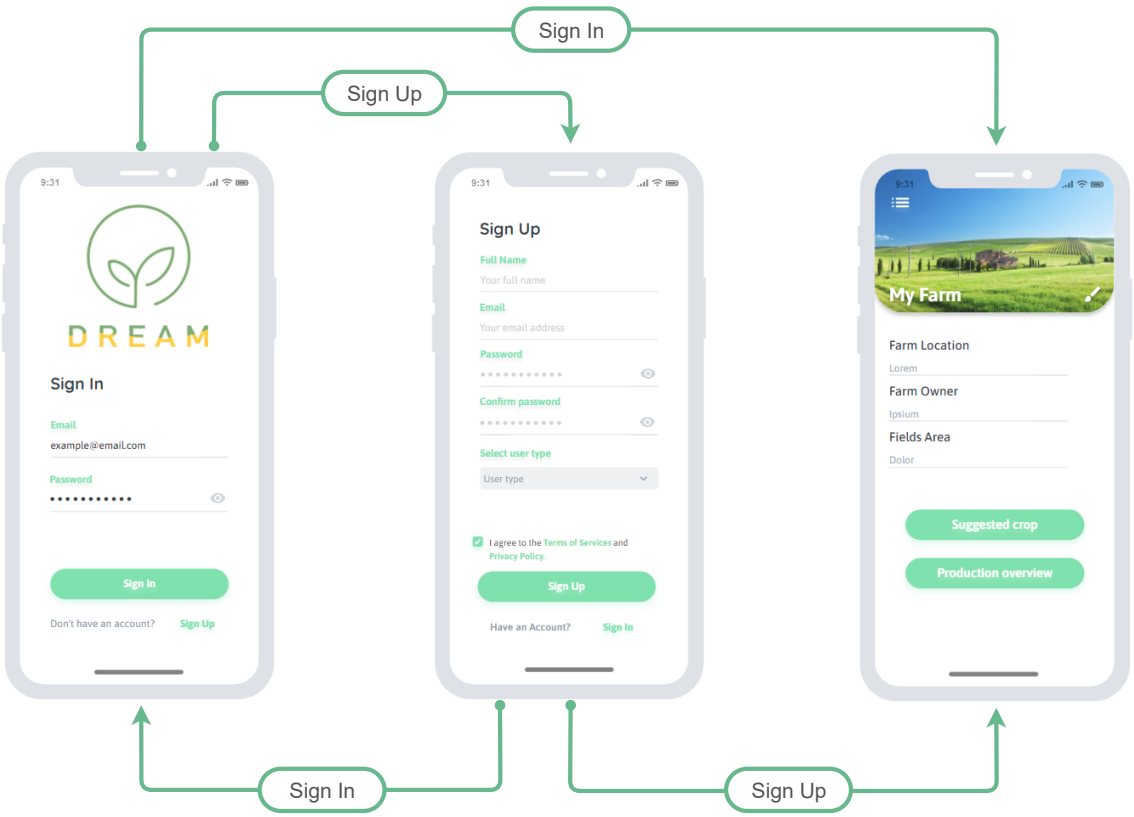


Figure 3.19: Login and Registration user interfaces flow

3.2.2 Farmer's UI flow

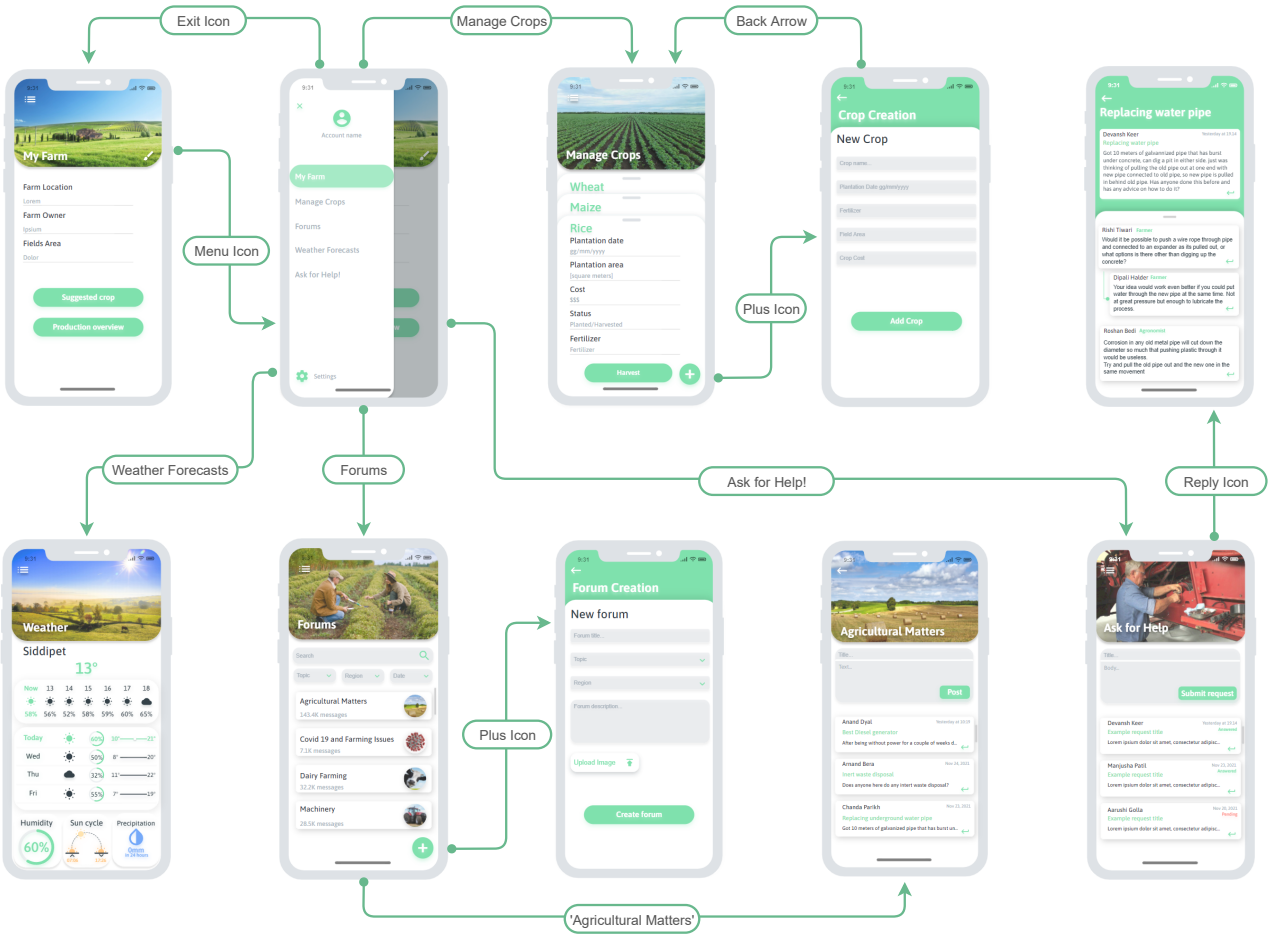


Figure 3.20: Farmer User Interfaces flow

3.2.3 Agronomist's UI flow

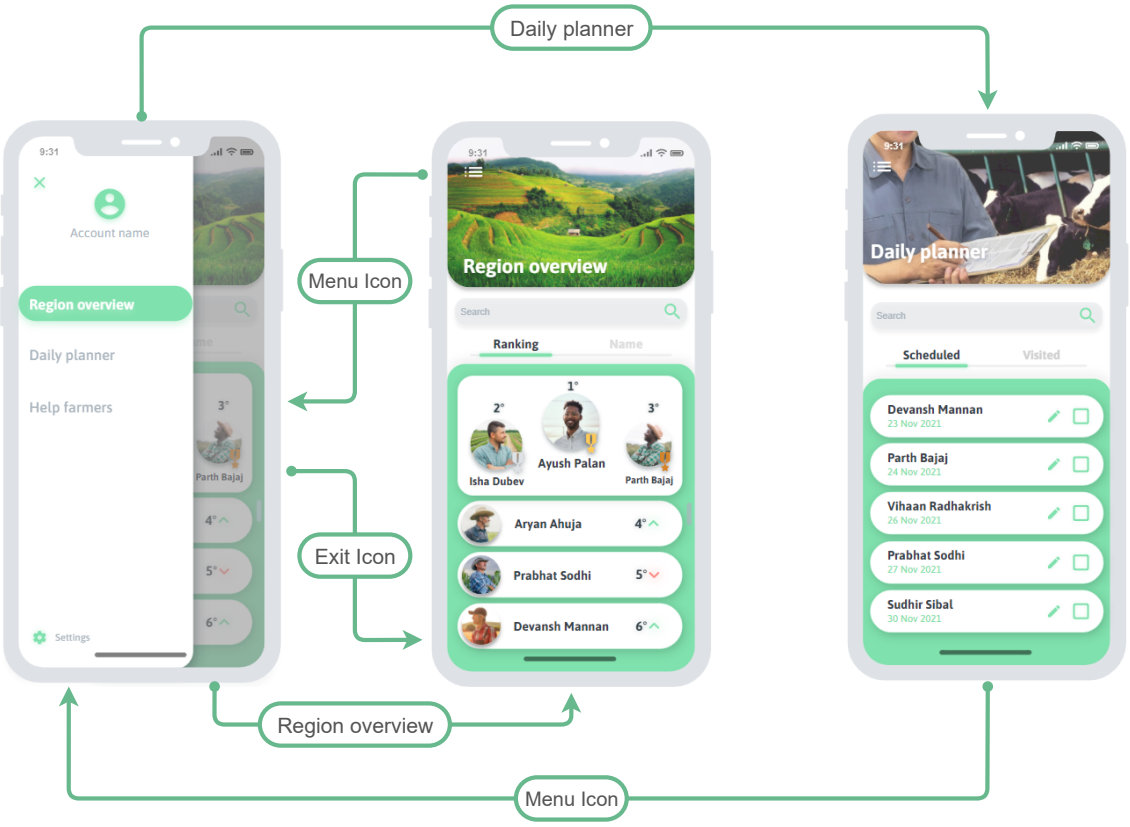


Figure 3.21: Agronomist User Interfaces flow

Chapter 4

Requirements Traceability

The **Model** component is needed to satisfy every requirement, so it will not be listed not to be repetitive.

4.1 Farmer

R.1 The system shall allow an unregistered user to register
Mobile Application, **AuthManager**

R.2 The system shall allow a registered user to unregister
Mobile Application, **AuthManager**

R.3 The system shall display weather forecasts in the farmer's personal page
Mobile Application, **WeatherManager**

R.6 The system shall display the old production of a farmer in the personal page
Mobile Application, **CropManager**

R.7 A farmer should be able to update his production
Mobile Application, **CropManager**

R.8 The system shall allow a farmer to store data about multiple productions
Mobile Application, **CropManager**

R.10 A farmer should be able to create a new forum
Mobile Application, **ForumManager**

R.11 A farmer should be able to post something on an existing forum
Mobile Application, **ForumManager**, **NotificationManager**

R.12 A farmer should be able to close a forum
Mobile Application, **ForumManager**

R.13 The farmer who created a forum, should be able to change its topic
Mobile Application, **ForumManager**

R.14 A farmer should be able to send a request to an agronomist
Mobile Application, **HelpRequestManager**, **NotificationManager**

R.15 A farmer should be able to view his and other farmer's help requests and responses
Mobile Application, **HelpRequestManager**

R.16 A farmer should be able to answer to other farmer's help requests
Mobile Application, **HelpRequestManager**, **NotificationManager**

4.2 Agronomist

R.1 The system shall allow an unregistered user to register
Mobile Application, **Web Application**, **AuthManager**

R.2 The system shall allow a registered user to unregister
Mobile Application, **Web Application**, **AuthManager**

R.17 An agronomist should be able to view his daily schedule
Mobile Application, **Web Application**, **DailyScheduleManager**

R.18 An agronomist should be able to postpone a meeting with a farmer
Mobile Application, **Web Application**, **DailyScheduleManager**

R.19 The system should send notifications to agronomists when a meetin is close
Mobile Application, **Web Application**, **DailyScheduleManager**, **NotificationManager**

R.20 The system shall delete a completed meeting from the calendar
DailyScheduleManager

R.21 An agronomist should be able to view all help requests and related responses from farmers of his area
Mobile Application, **Web Application**, **HelpRequestManager**, **NotificationManager**

R.22 An agronomist should be able to respond to all farmer's help requests from his area
Mobile Application, **Web Application**, **HelpRequestManager**, **NotificationManager**

R.23 An agronomist should be able to see the ranking of best farmers of his area
Mobile Application, **Web Application**, **RankingManager**
MapManager, **WeatherManager**, **HumidityAndIrrigationManager**

4.3 Policy Maker

R.1 The system shall allow an unregistered user to register
Web Application, **AuthManager**

R.2 The system shall allow a registered user to unregister
Web Application, **AuthManager**

R.4 The system shall store information about the irrigation system
HumidityAndIrrigationManager

R.5 The system shall store information about soil humidity
HumidityAndIrrigationManager

R.9 A policy maker should be able to see the worst farmers of a specific region
Web Application, **RankingManager**, **MapManager**, **WeatherManager**, **HumidityAndIrrigationManager**

R.24 A policy maker should be able to see the best farmers of the whole Telangana state
Web Application, **RankingManager**, **MapManager**, **WeatherManager**, **HumidityAndIrrigationManager**

R.25 A policy maker should be able to see the worst farmers of the whole Telangana state
Web Application, **RankingManager**, **MapManager**, **WeatherManager**, **HumidityAndIrrigationManager**

R.26 A policy maker should be able to see the best farmers of a specific region
Web Application, **RankingManager**, **MapManager**, **WeatherManager**, **HumidityAndIrrigationManager**

Chapter 5

Implementation, Integration and Test Plan

5.1 General Implementation

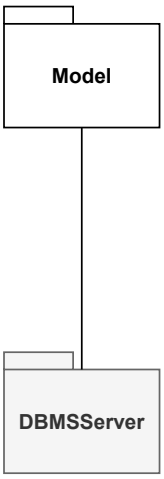
This section provides an implementation overview which will follow a buttom-up approach in order to make the testing phase easier and closer to the implementation one. The system can be divided into some components whose integration will be explained in the next chapter.

5.2 Integration

The integration phase is described in different phases in order to progressively develop the components. The relation lines shown in each diagram represent the dependencies and the suggested order in which to implement the whole part, but it is not mandatory since all parts can be implemented in parallel and linked afterwards.

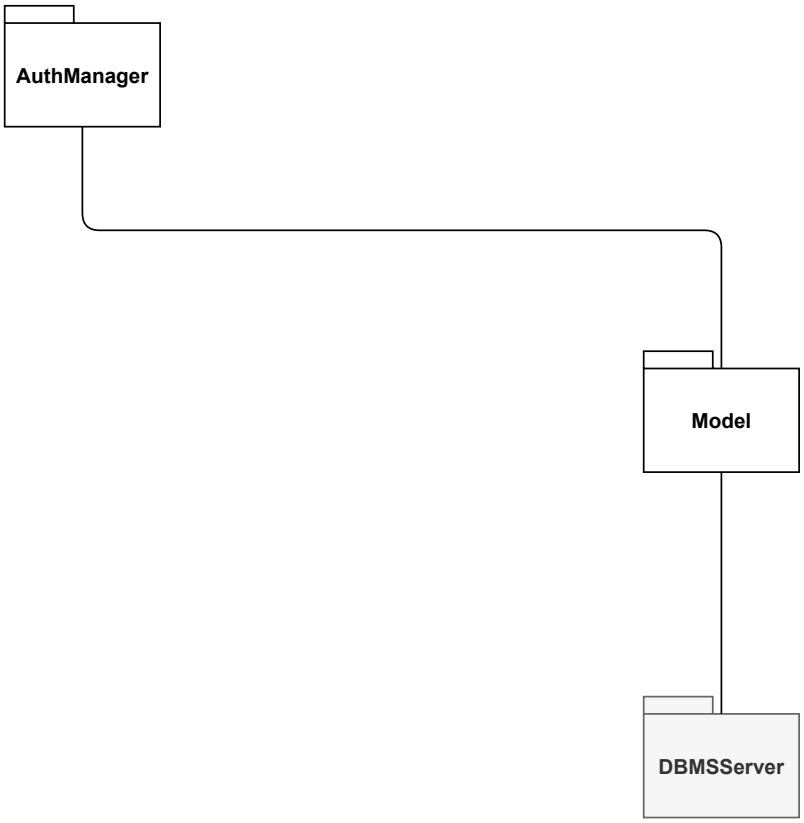
5.2.1 Integration with the database

This is highly advised to be the first phase since everything rotates around storing, updating and deleting informations. So the first step is to create the link between the model and the database.



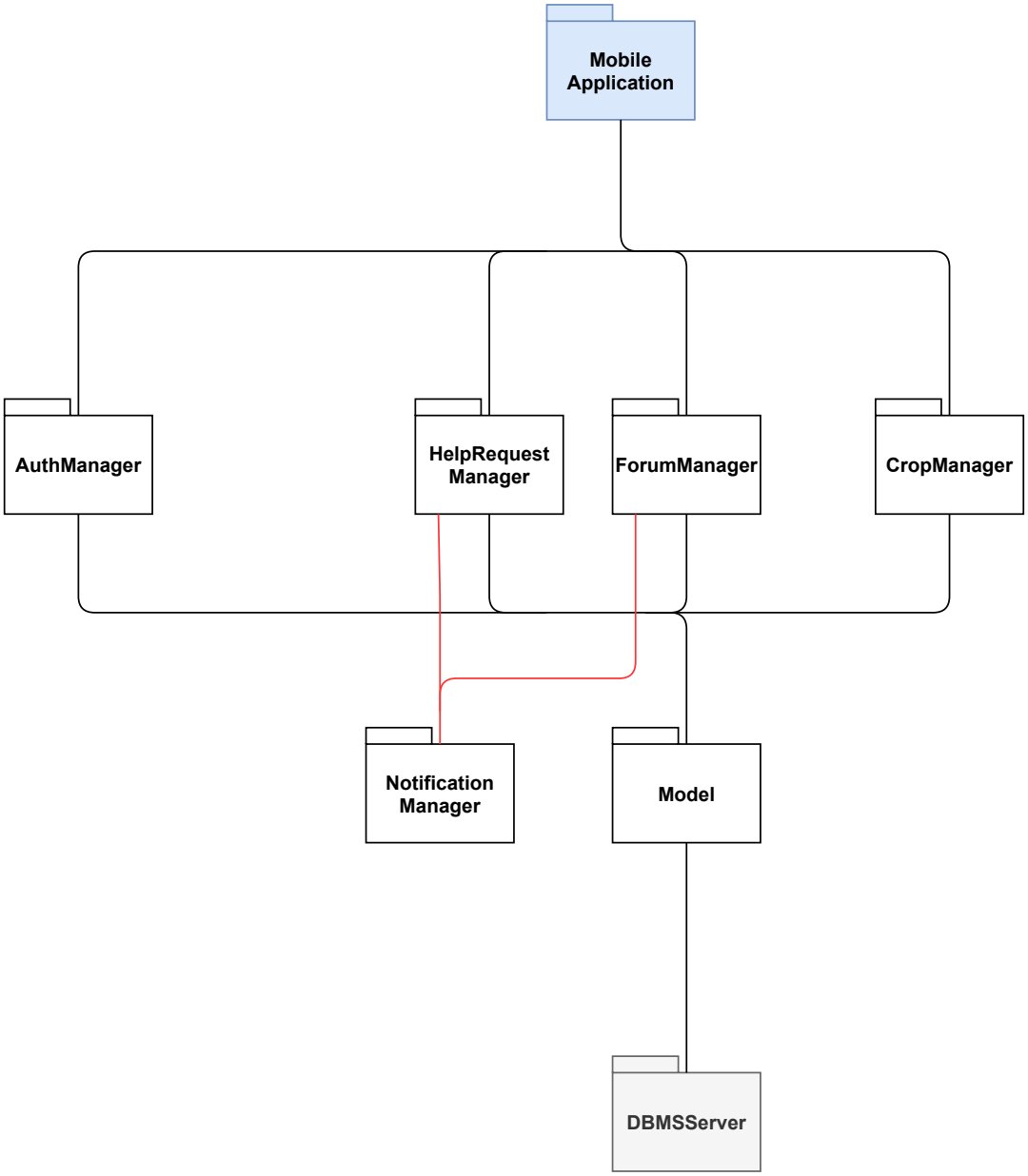
5.2.2 Authentication

In the second phase the authentication components are implemented in order to prioritise the login mechanics. This part is still competence of the server and the connection with the clients, which is still not present, can be simulated.



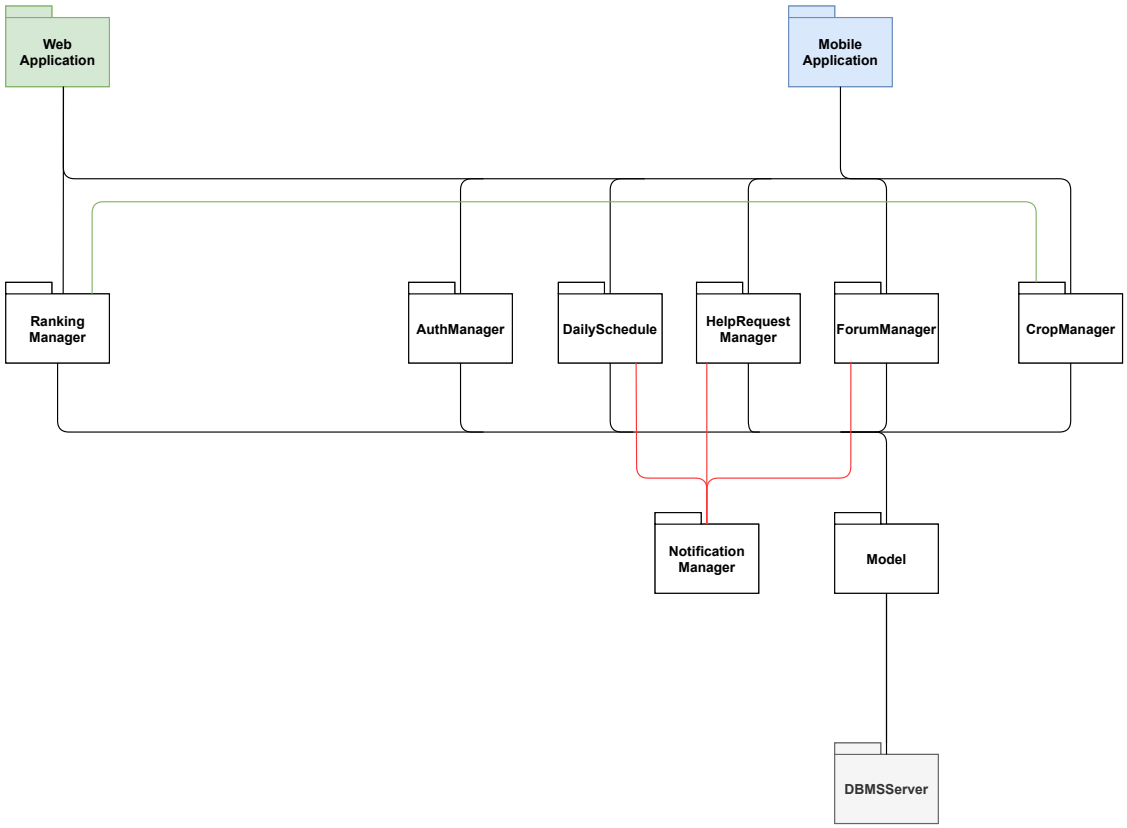
5.2.3 Farmer client software

The first client should be the farmer client software since it has a more central role in the whole project. In this step the connection between the client and server (via app) needs to be done.



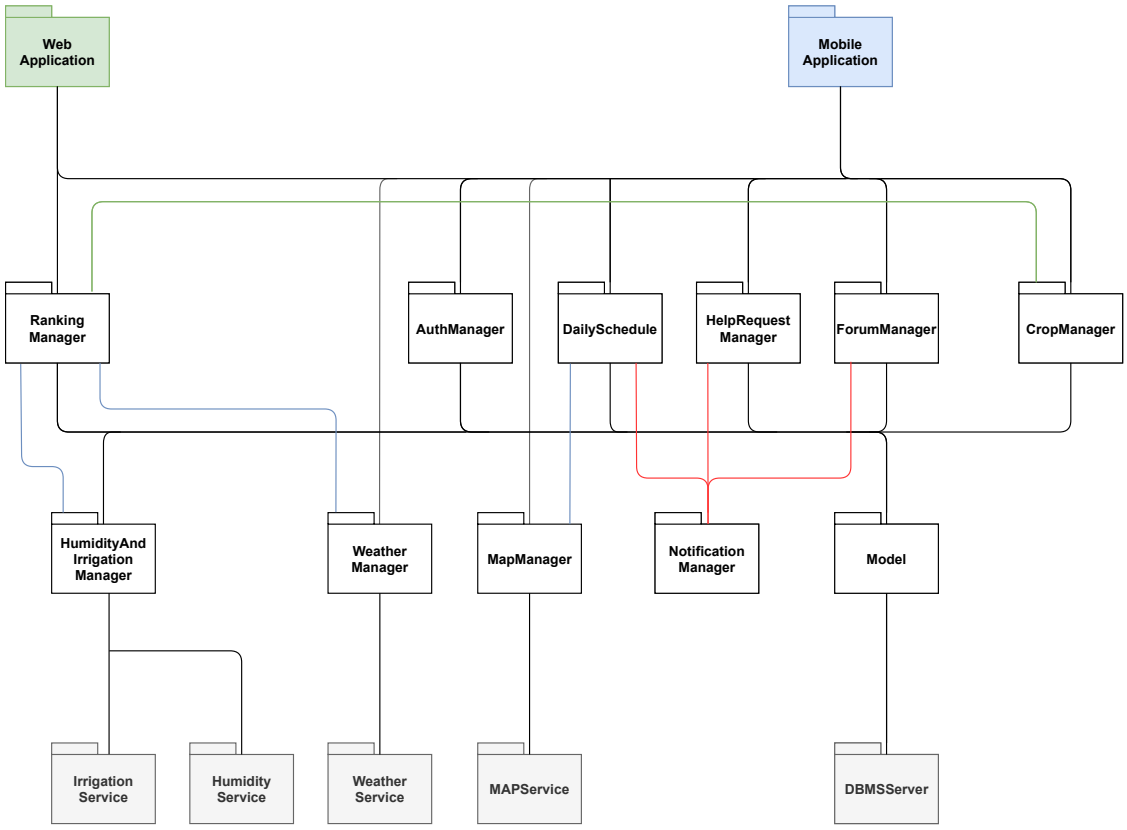
5.2.4 Agronomist client (web) software

The next step is to implement the Agronomist part, since they need to have an overview of the best farmers and they have to plan their daily schedule. This is why it is advisable to implement them after the farmers. Since the connection with the mobile was already implemented, the application part of the agronomist is suggested to be done first and only after the web part.



5.2.5 External Services and PolicyMaker features

Now it is advised to develop the external services like the Map Manager, WeatherManager and the HumidityAndIrrigationManager. The final step is to complete the rankingManager in order to allow the policy maker to view the global ranking and every regional ranking.



5.3 Testing

The testing should follow step by step the implementation of the program. Each method should be tested before implementing a new one and, moreover, there should be tests of every dependency in order to guarantee a correct integration of every component (this means there should be a series of testing for every link of the diagrams above). Another key part is the testing of components working together, for example, all farmer's components should be tested all together to guarantee a correct working .

Chapter 6

Effort Spent

6.1 Monti Matteo

Task	Hours
Initial briefing	2
Introduction	5
Runtime view	7
User Interface Design	10
Implementation Integration and Test Plan	2
Document Revision	4
Total	30

6.2 Mariani Samuele

Task	Hours
Initial briefing	2
System overview	3
Component view	15
Requirements traceability	6
Revision	4
Total	30

6.3 Molteni Alessandro

Task	Hours
Initial briefing	2
Deployment View	4
Component Interfaces	8
Selected architectural styles and pattern	4
Implementation, Integration and Test Plan	8
Revision	4
Total	30

Chapter 7

References

- Course slides on WeBeeP.
- Course lectures notes.
- ISO/IEC/IEEE 29148 - Systems and software engineering.
- All *class*, *activity* and *sequence* diagrams were made using the **draw.io** online tool.