AY 2021/2022

Politecnico di Milano

DD: Design Document

Ottavia Belotti Alessio Braccini Riccardo Izzo

Professor
Elisabetta Di Nitto

Version 1.0
January 20, 2022

Contents

1 Introduction **1**
- 1.1 Purpose . . . 1
- 1.2 Scope . . . 1
- 1.3 Definitions, acronyms, abbreviations . . . 1
- 1.4 Revision history . . . 2
- 1.5 Reference documents . . . 3
- 1.6 Document structure . . . 3

2 Architectural Design **4**
- 2.1 Overview . . . 4
- 2.2 Component view . . . 7
- 2.3 Deployment view . . . 11
- 2.4 Runtime view . . . 13
- 2.5 Component interfaces . . . 27
- 2.6 Selected architectural styles and patterns . . . 28
- 2.7 Other design decisions . . . 29
  - 2.7.1 Servers availability and response time . . . 29
  - 2.7.2 Farmers ranking . . . 29
  - 2.7.3 Farmers location . . . 31
  - 2.7.4 Agronomist daily plan . . . 31

3 User Interface Design **32**

4 Requirements Traceability **36**

5 Implementation, Integration and Test Plan **41**
- 5.1 Implementation Plan . . . 41
- 5.2 Integration Strategy . . . 42
  - 5.2.1 Integration and Testing . . . 43
- 5.3 System testing . . . 48

6 Effort Spent **49**

7 References **49**

1 Introduction

1.1 Purpose

The purpose of this document is to provide a full technical description of the system described in the RASD document. In this design document we discuss about both hardware and software architectures in terms of interaction among the components that represent the system. Moreover, there are mentions about the implementation, testing and integration process. This document will include technical language so it is primarily addressed to programmers, but stakeholders are also invited to read it in order to understand the characteristics of the development.

1.2 Scope

The scope of this design document lays in the definition of the system behavior, in both general and critical cases, and in the design of the system architecture by describing the logical allocation of the components and the interaction between them. This document also extends in part to the implementation and testing plan, where one possible course of action is explained, user interface design of user applications and requirements traceability relating to the RASD.

1.3 Definitions, acronyms, abbreviations

Definitions

- **Production entry**: the pair sown quantity and harvested quantity of a certain crop that the farmer inserts as his/her production.
- **Full production entry**: a production entry that carries both sown quantity and harvested quantity, even if inserted in different times of the year, as it normally should, since it is expected that the farmer will insert the sown quantity of crop during the sowing season and the harvested quantity during harvest. Every basic production entry should become a valid production entry.

Acronyms

- **DREAM**: *Data-driven predictive farming*
- **RASD**: Requirement Analysis and Specification Document
- **DD**: Design Document
- **API**: Application Programming Interface
- **DBMS**: Database Management System
- **UML**: Unified Modeling Language
- **GPS**: Global Positioning System
- **IT**: Information Technology
- **GUI**: Graphic User Interface
- **UI**: User Interface
- **HTTPS**:HyperText Transfer Protocol Security
- **HTML**: HyperText Markup Language
- **CSS**: Cascade Style Sheet
- **JS**: JavaScript

1.4 Revision history

- Version 1.0: initial version
- Version 1.1: update after the implementation
  - update high-level-architecture: GoogleMaps and OpenWeather now communicates with the Web Server
  - update component diagrams: LocationModule, MapServiceManager and WeatherManager now are in the Web Server, this changes are reflected also in the component interfaces diagram
  - update deployment diagram: change hosting platform and load balancer

- delete EditProfile sequence diagram, update SignUp and CheckWeatherForecast sequence diagrams

1.5 Reference documents

- Specification document: "Assignment RDD AY 2021-2022"
- Requirements Analysis Specification Document (RASD)
- UML documentation: https://www.uml-diagrams.org/
- ArchiMate documentation: https://pubs.opengroup.org/architecture/archimate3-doc/
- Slides of the lectures

1.6 Document structure

- **Section 1** gives a brief description of the design document, it describes the purpose and the scope of it including all the definitions, acronyms and abbreviations used.
- **Section 2** delves deeply into the system architecture by providing a detailed description of the components, the interfaces and all the technical choices made for the development of the application. It also includes detailed sequence, component and ArchiMate diagrams that describe the system in depth.
- **Section 3** contains a complete description of the user interface (UI), it includes all the client-side mockups with some graphs useful to understand the correct execution flow.
- **Section 4** maps the goals and the requirements described in the RASD to the actual functionalities presented in this DD.
- **Section 5** presents a description of the implementation, testing and integration phases of the system components that are going to be carried out during the technical development of the application.

2 Architectural Design

2.1 Overview

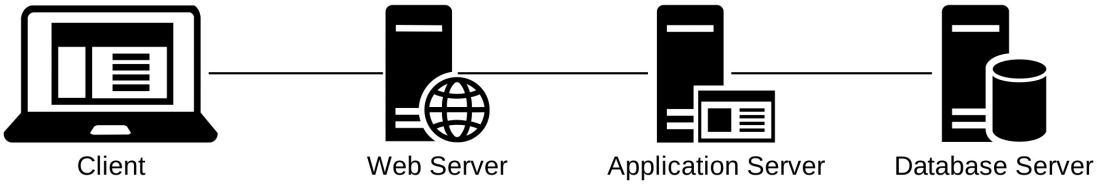


Figure 1: Four-Tier Architecture Scheme

The system is a distributed application that follows the common client-server paradigm. The architecture of the application is structured in three logic layers:

- **Presentation Layer (P)**: it manages the presentation logic and handles the user actions. It is characterized by a GUI (Graphic User Interface) that allows the user to interact with the application in a simple and effective way.
- **Logic or Application Layer (A)**: it manages all the functionalities that has to be provided to the users, it is also responsible of data exchange between the client and the data sources.
- **Data Layer (D)**: it manages the access to data sources, it gets data from the database and moves it through the other layers. It is essential to guarantee a high level of abstraction from the database in order to provide an easy to use model.

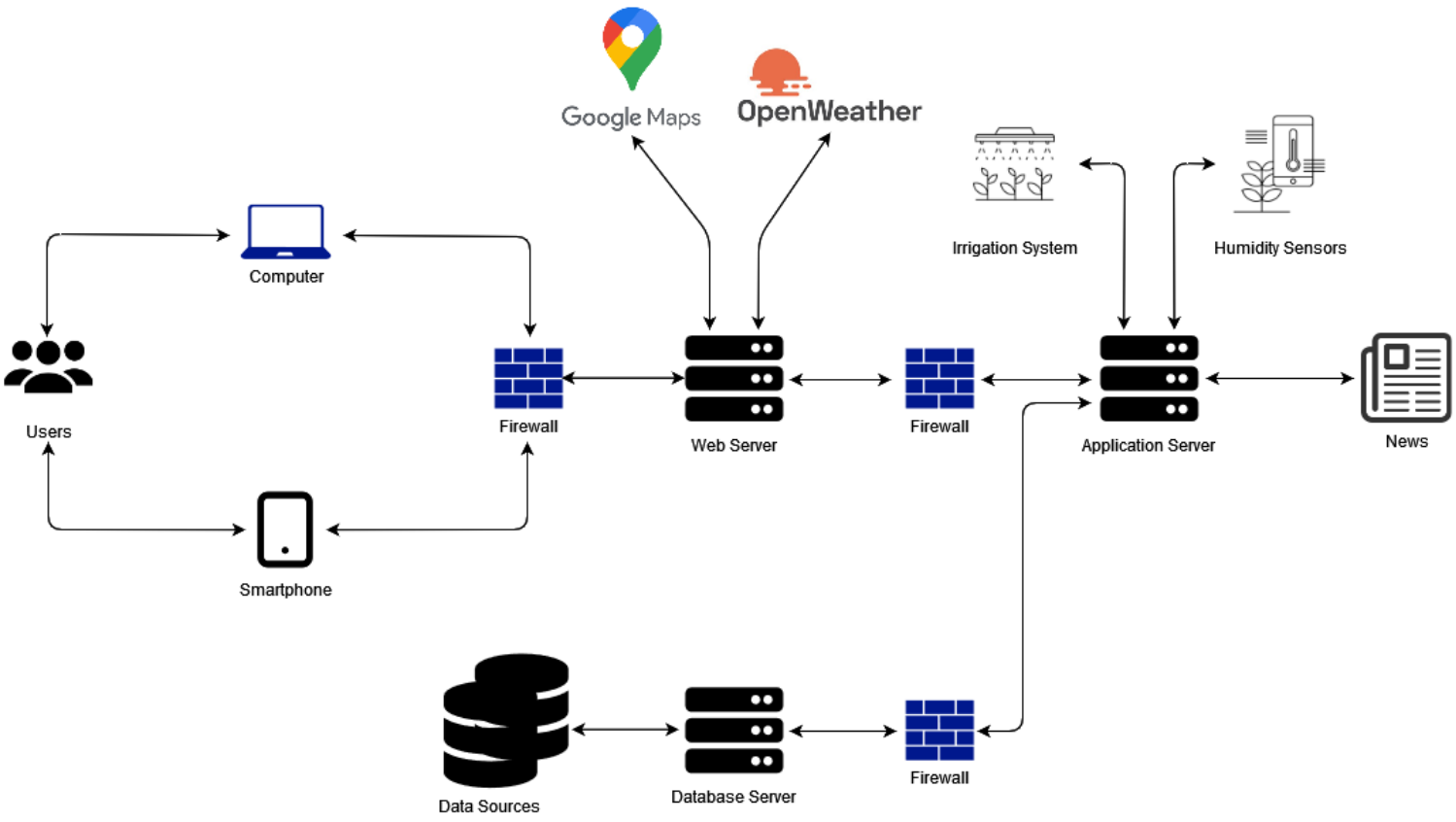


Figure 2: High Level Architecture

The system, as shown in *Figure (1)*, is based on a four-tier architecture (Client, Web Server, Application Server and Database Server), this ensures more flexibility and high scalability. The tiers are separated by firewalls in order to guarantee a higher level of security to the whole system. A thin client is used to prevent heavy computation loads client side, in this way all the heavy operations are executed at server side. The client's devices can be a personal computer, a mobile device or any kind of IT device able to connect to the Web Server through a web browser. In fact, the four-tiers architecture has been specifically chosen to allow the widest variety of commercial devices to access the application, without the need of developing an OS-specific one

for each of them. The Web Server manages the HTTP requests by the users and forwards them to the Application Server. It also manages all the external APIs including GoogleMapsAPI and OpenWeatherAPI. The first one enable the geolocalization while the other one allows to retrieve data like weather information directly from it in order to show them to the user. The Application Server communicates with the Database Server and manipulates data following its business logic. It also manages the external services such as the news, the water irrigation service and the humidity service. Finally the Database Server manages all the internal data sources and communicates directly with the Application Server. All the components will be described in depth in the following sections

2.2 Component view

General Component View

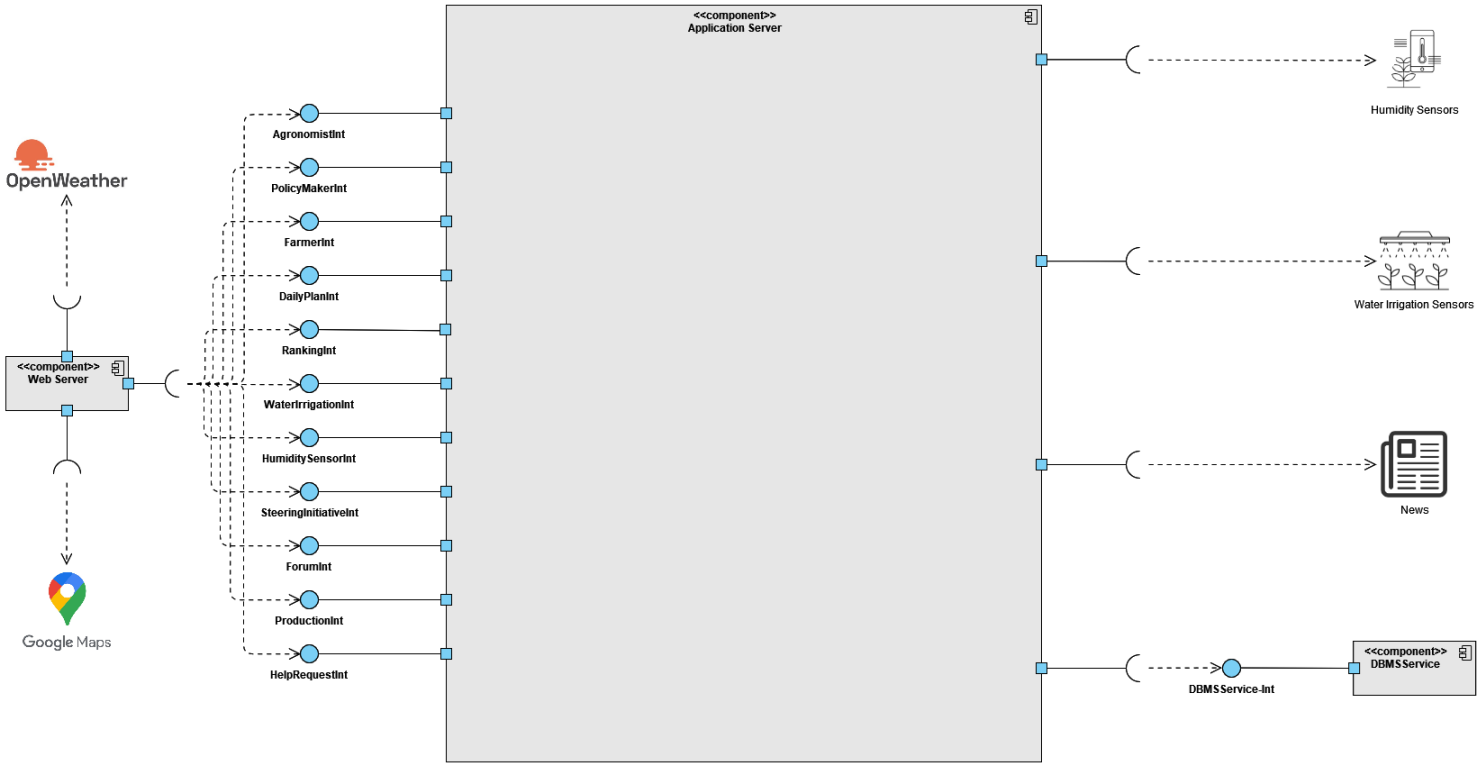


Figure 3: General Component Diagram

This image gives a high level representation of the components of the system. On the left are shown the provided interfaces between the Web Server and the Application Server that in this scheme is represented as a "black box", a complete description of it is provided in the next section. All the interfaces basically represent the main functionalities requested by the client application. On the right-hand side, there are the external requested interfaces. Among these there are the news, the water irrigation system and the humidity system. Finally, the DBMS interface manages the DBMS

service and it's responsible of the communication between the Application Server and the Database Server.

Application Server Component View

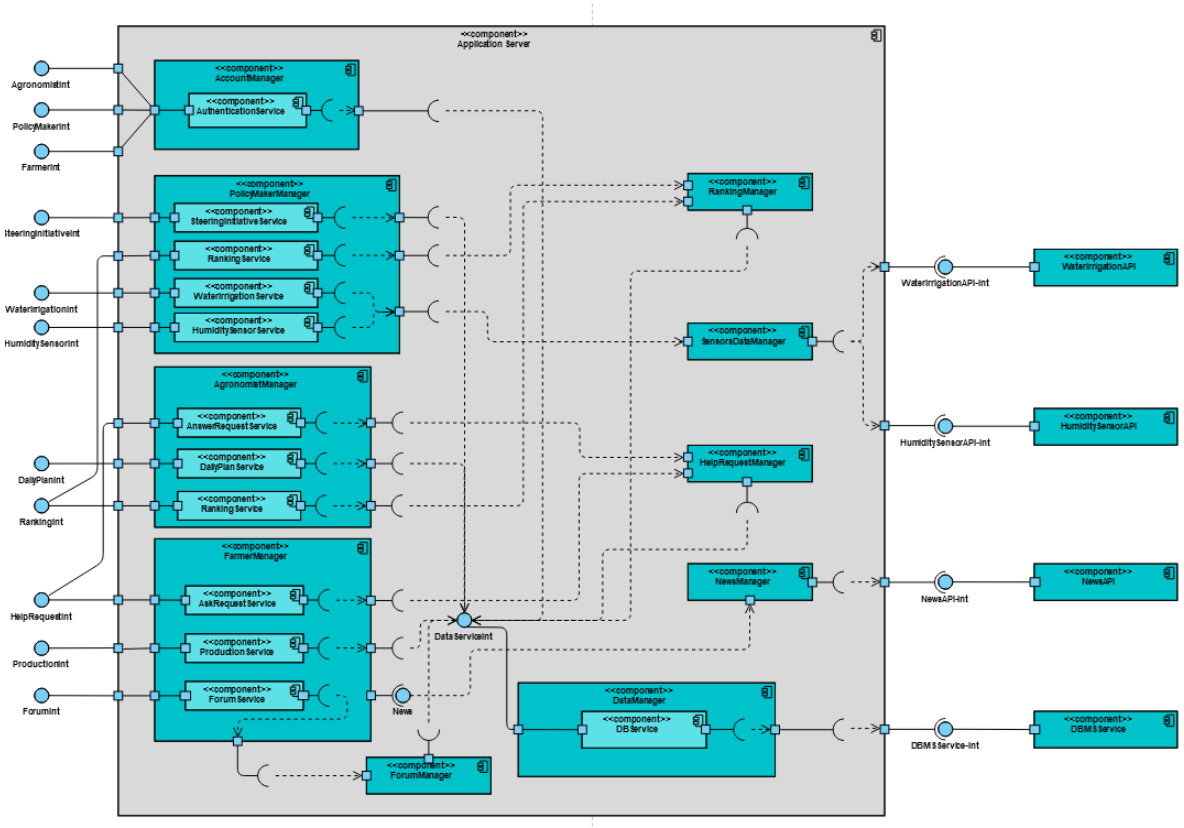


Figure 4: Application Server Component Diagram

The following component diagram gives a detailed view of the Application Server. It shows the internal structure and the interaction between the components. External elements in the diagram are represented in a simplified way.

- **AccountManager**: this component handles all the basic requests made by the client. The *AuthenticationService* manages the authenti-

cation process that is composed by the log in, the log out and the sign up services. Once a user is logged in, all the specific functionalities are provided by the component that manages this type of user.

- **PolicyMakerManager**: it manages the policy maker services, these include: download of steering initiatives' reports submitted by agronomists, farmers' ranking, visualization of water irrigation sensors data and humidity sensors data. The steering initiative service and the ranking service are linked through an interface to the internal database, on the other side the water irrigation service and the humidity sensor service are considered external services and are managed by the *SensorsDataManager*.

- **AgronomistManager**: it manages the agronomist services, these include: answer help requests, creation and updating of daily plans and farmers' ranking. The daily plan service and the ranking service are linked to the internal database, instead the answer request service communicates directly with the *HelpRequestManager* component.

- **FarmerManager**: it manages the farmer services, these include: submit help requests, insert production data and participate in the forum. It includes one external interface that communicates with the *NewsManager* component. In this case, only the production service is linked directly to the internal database, instead the ask request service is linked to the *HelpRequestManager* component, while the forum service exposes an interface to connect to the *ForumManager* component.

- **ForumManager**: this component manages the forum section, in particular it is responsible of the management of all the topics with the related messages between farmers.

- **DataManager**: it provides access to the external interface of the database, it manages queries and interacts with the DBMS. It includes one component that expose an interface that allow other internal components to communicate with the database. This component is connected to the *DBMSService* external component, establishing the connection between the Application Server and the Database Server.

- **HelpRequestManager**: it manages the help requests between farmers and agronomists. It communicates with the *FarmerManager* com-

ponent for submitting a farmer's help request, on the other way it communicates with the *AgronomistManager* component for the ones to answer. Essentially, it works as the dispatcher of messages between the two type of user. Finally, it provides an interface that communicates with the internal database in order to store the exchanged messages.

- **RankingManager**: it manages the ranking system for both the policy maker and the agronomist. It is responsible for calculating the final score used to compare the farmers in the ranking. More details on how to calculate it can be found in *Section (2.7)*.

- **NewsManager**: this component manages the service related to the news, it provides updated suggestions about crops and fertilizers for the farmers. To do this, it interacts with the external component *NewsService*.

- **SensorsDataManager**: this component handles two external APIs, the first one is provided by the water irrigation system and the other one provided by the humidity sensors. Both services are used by the *PolicyMakerManager*.

Web Server Component View

Regarding the Web Server the main components are:

- **LocationModule**: this module provides the interface that allows the geolocalization of the farm based on the GPS (functionality available for farmer users) or on the address provided by the farmer. In order to provide this service, it communicates with the *MapServiceManager*.

- **MapServiceManager**: this component communicates directly with the external API provided by *Google Maps*, it provides information regarding the district and allows the user to visualize the map of a specific location. Mainly it adapts the data received by the API in a comprehensible way for the other components.

- **WeatherManager**: this component manages the service related to the weather forecast. It interacts with the external API provided by *Open Weather* in order to get the weather data tailored to the user location.

2.3 Deployment view

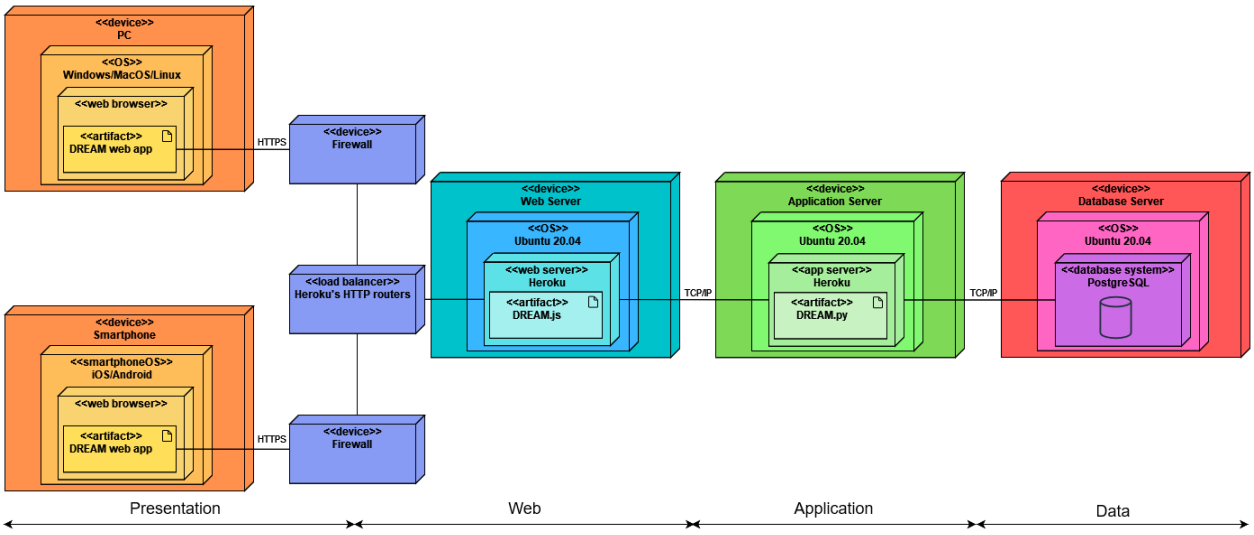


Figure 5: Deployment Diagram

The deployment diagram in *Figure (5)* shows the most important components necessary for the correct behaviour of the system. The devices shown in the diagram are:

- **PC/Smartphone**: they are the client machines, they can be used by all the type of users. One allows the user to connect to the Web Server through a web browser as long as the HTTPS protocol is used.
- **Firewall**: first level of security, it guarantees safety access to the internal network of the system.
- **Load balancer**: the main goal is to prevent an overload of a server, this device distributes the incoming requests equally across multiple servers. This increase availability and reliability of the application. The one selected are the *Heroku HTTP routers* provided by the hosting service. More details in section 2.7.1.

- **Web Server**: it manages the HTTPS requests from the clients and route them to the *Application Server* for processing. It is responsible of the incoming responses from the application server, they are managed by serving an HTML file to the client in addition to CSS and JS sheets. The result page is built with a client-side scripting. It also manages the external APIs such as the Google Maps API and the OpenWeatherAPI. Finally, web servers are duplicated to ensure high scalability and to guarantee good performance in case of a single point of failure.

- **Application Server**: it runs the core functionalities of the system, here there is the application logic. It also manages some external services such as the news, the water irrigation service and humidity service. Just like the *Web Server*, for the same reasons, it is duplicated. Finally, it establishes the connection with the data tier through the DBMS interface. The *Application Server* and the *Database Server* are connected through an internal LAN, there are no firewalls between them in order to increase the average connection speed.

- **Database Server**: it represents the data tier, it receives the data insertions and data queries from the *Application Server* and manages them. PostreSQL has been selected as the main database infrastructure, this allows to easily manage in a secure way the data storage and requests.

2.4 Runtime view

Here the runtime views of some relevant use cases of the system are represented through sequence diagrams. In the diagrams, the part regarding the user is omitted because it has been deemed as redundant to the understanding of the interaction. In later diagrams, some parts, like the login phase or returning to home page, were omitted for the the same motivations.

Sign Up

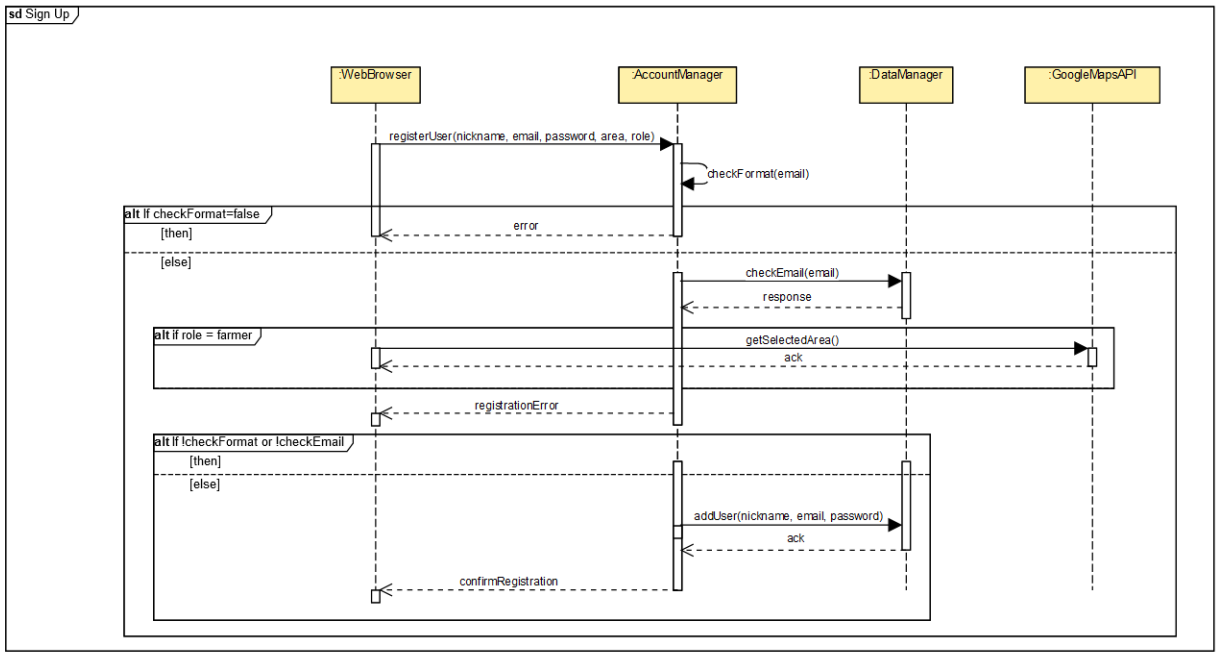


Figure 6: Sign Up

Sign up is pretty standard: user registers himself/herself by inserting first name, last name, email, password, job role and district (if the user isn't a Policy Maker). If the user is a farmer, in a subsequent page, he/she chooses also the crop type the farm deals with. Farmer can select manually their address without the use of GPS.

At the end of the process, if every check results successful, the user is registered to the database by the *DataManager* component.

Log In

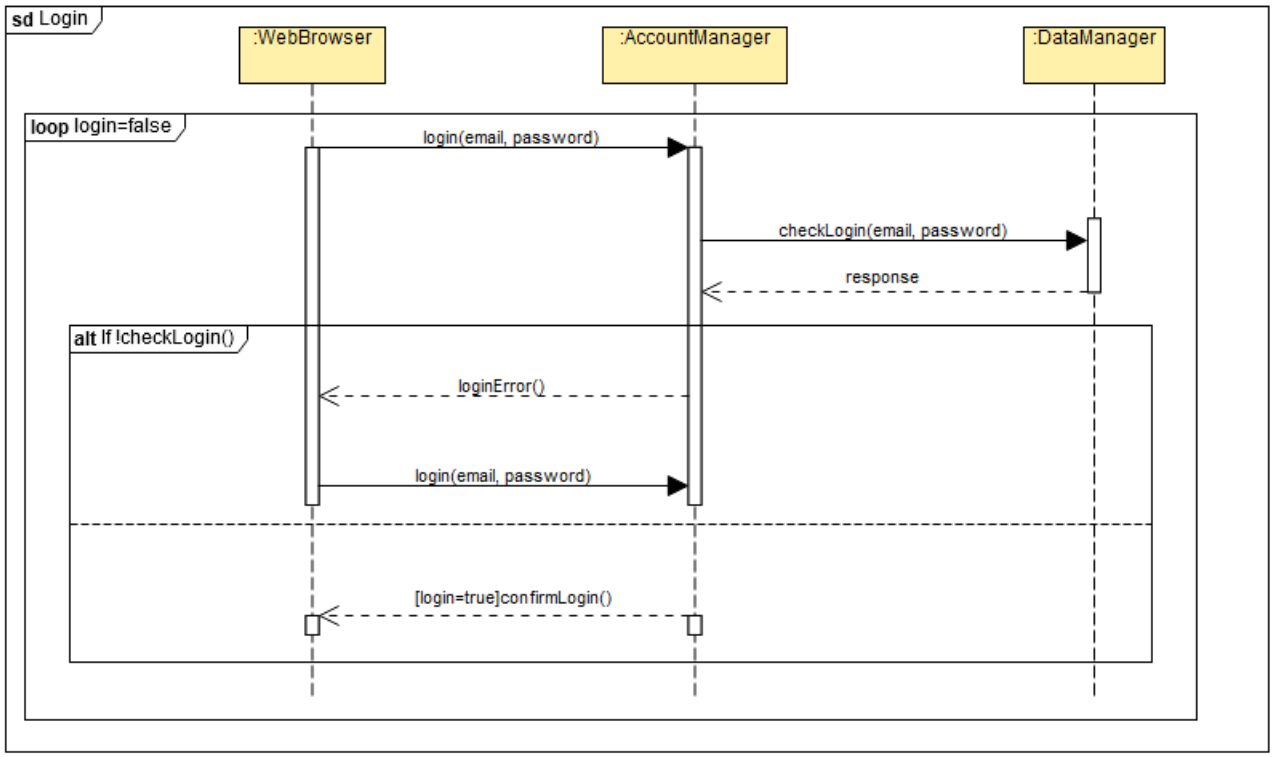


Figure 7: Log In

The Log In phase simply consists in the user action of inserting their email (unique key in the database) and password in the login fields, then the system checks if the pair corresponds to an user entry in the database. In case of success, the user can log in the system and use its functionalities available for that particular type of account chosen during the Sign Up phase.

View Steering Initiative Reports

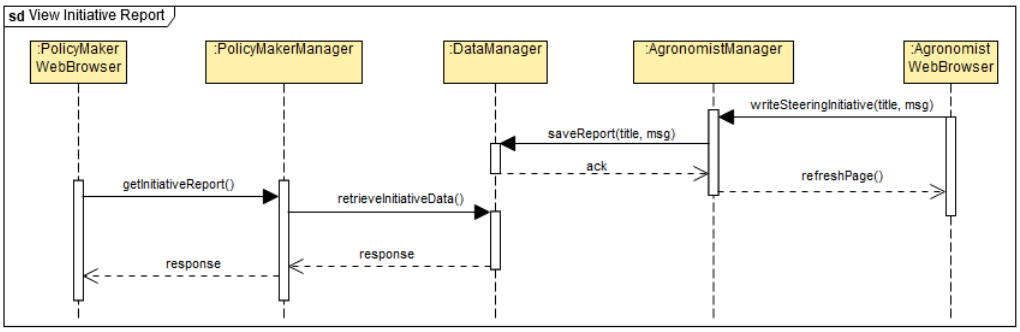


Figure 8: View Initiative Report

This sequence diagram represents the interactions between components that occur when displaying the steering initiatives (previously uploaded in the database by an agronomist) to a policy maker user.

Check Soil Humidity Data

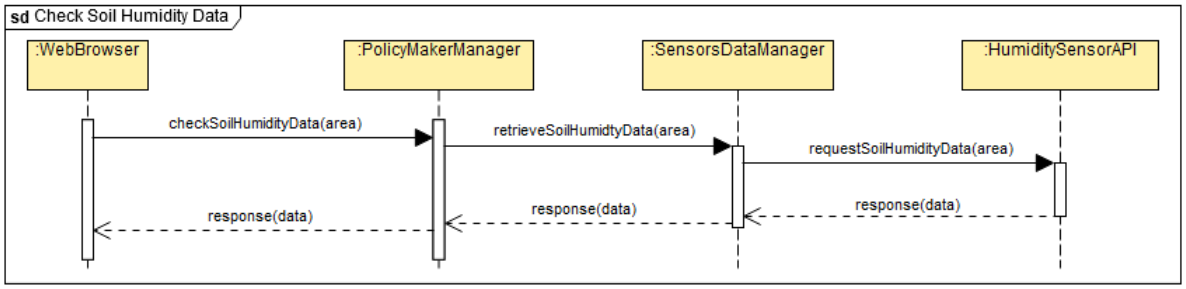


Figure 9: Check Soil Humidity Data

This sequence diagram represents the interactions between components that occur when displaying the sensors data regarding soil humidity to a policy maker user. The business logic running on the Application Server retrieves these data from an external source, furthermore it handles them just temporarily since they are not modified by the *DREAM* system and they're fetched on user demand.

Check Water Irrigation Data

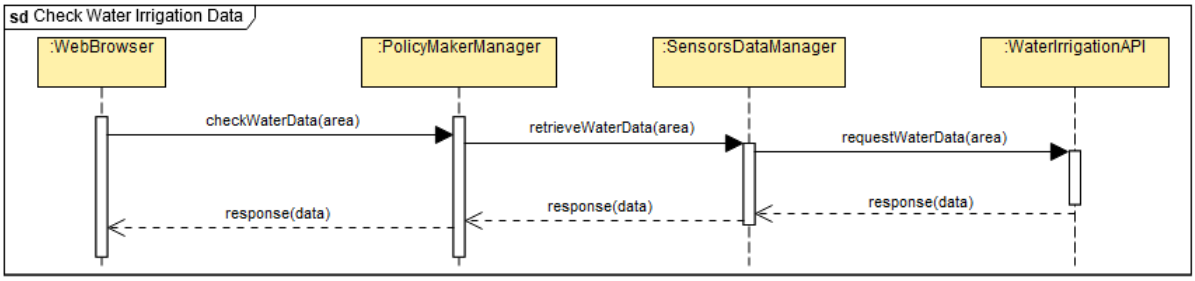


Figure 10: Check Water Irrigation Data

This sequence diagram represents the interactions between components that occur when displaying the sensors data regarding the water irrigation to a policy maker user. The function is the same of the above mentioned case.

View Farmers Ranking

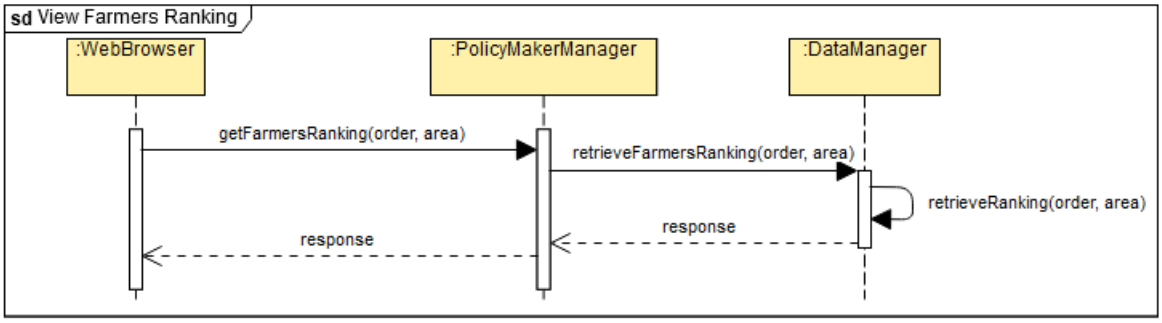


Figure 11: View Farmers Ranking

This sequence diagram represents the interactions between components that occur when displaying the farmers ranking to policy maker and agronomist users. It will be explained better later in the document. The function includes an area parameter to specify of which area the user wants the ranking. In order to achieve code reusability, this function is invoked by either policy makers or agronomists. When a policy maker wants to see the ranking the parameter `area` will be set to `all` in order to obtain the ranking comprehensive of all Telangana's farmer registered in the app. Instead, when an agronomist wants to use this functionality, `area` will be automatically taken to his/her respective responsibility area. By doing this, the function becomes more modular.

View Specific Farmers Informations

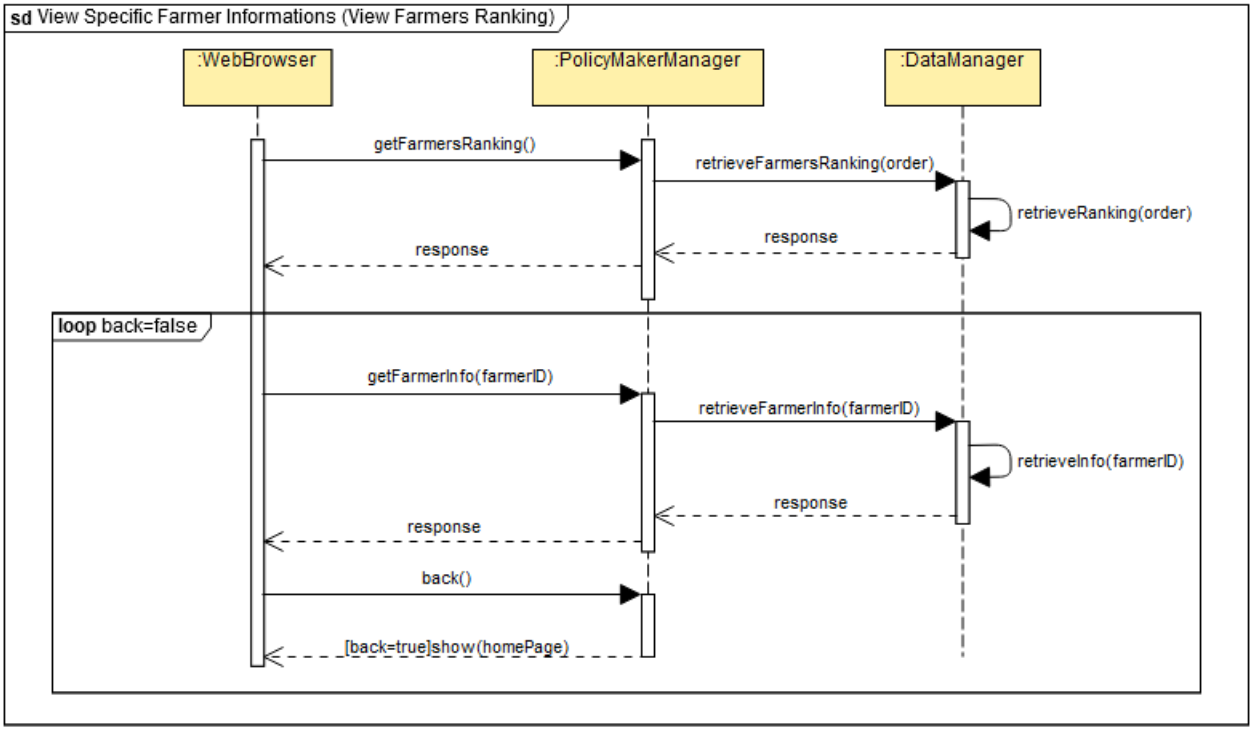


Figure 12: View Specific Farmer Informations

This sequence diagram represents the interactions between components caused by a search for a specific farmer information summary.

Insert Production Data

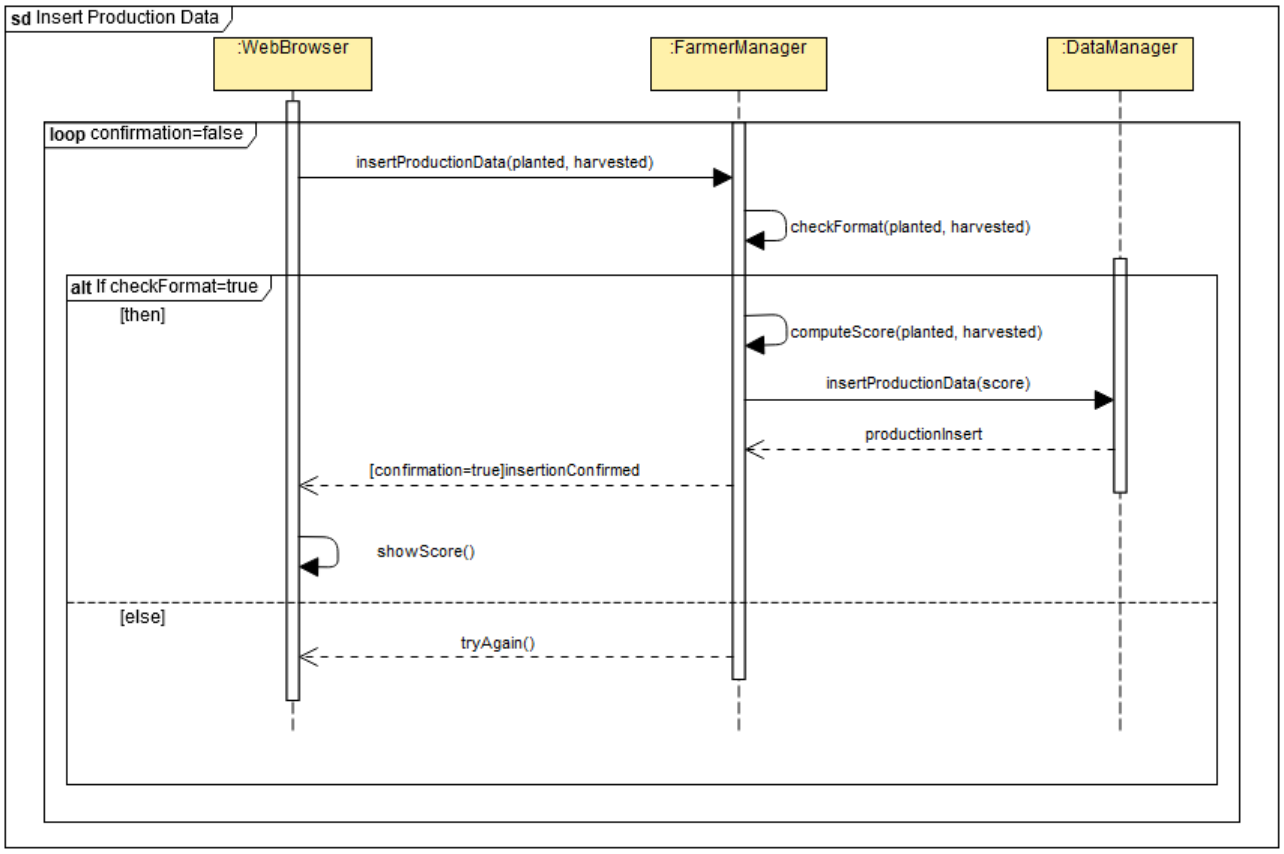


Figure 13: Insert Production Data

This sequence diagram represents the interactions between components caused by an insertion of production data from a farmer user. Each new production entry cause the score associated to the farmer to be calculated again and updated. So the score shall be an attribute of each farmer profile. Since a farmer's rank is directly proportionated to the his/her score, the recomputation can lead to a shifting in the general ranking too.

Check News

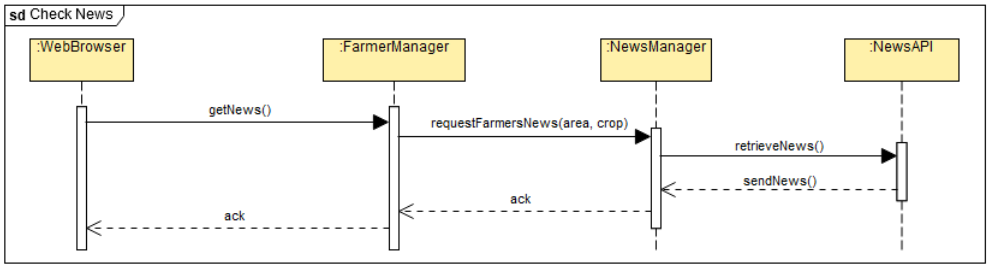


Figure 14: Check News

This sequence diagram represents the interactions between components that occur when displaying the news regarding farmers' interests to them, such as news concering the same area or crop type of the farmer. The *NewsManager* uses the *NewsAPI* component in order to get always up-to-date news.

Forum

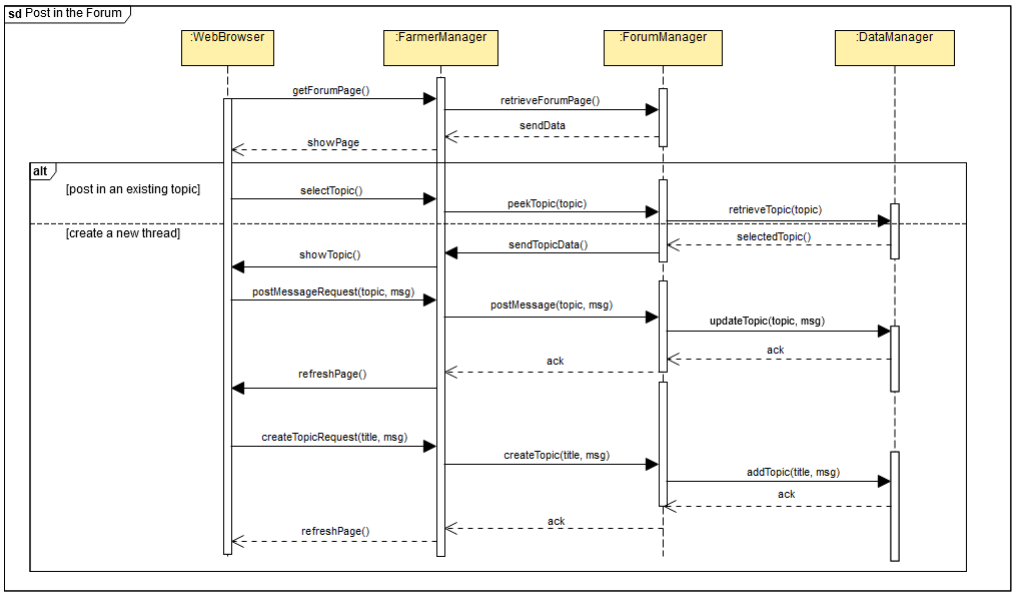


Figure 15: Forum

This sequence diagram represents the interactions between the components of the forum. It is split into two part: the one that allows the user to make a post in an already existing topic and the one that allows the user to create a new topic.

Check Weather Forecast

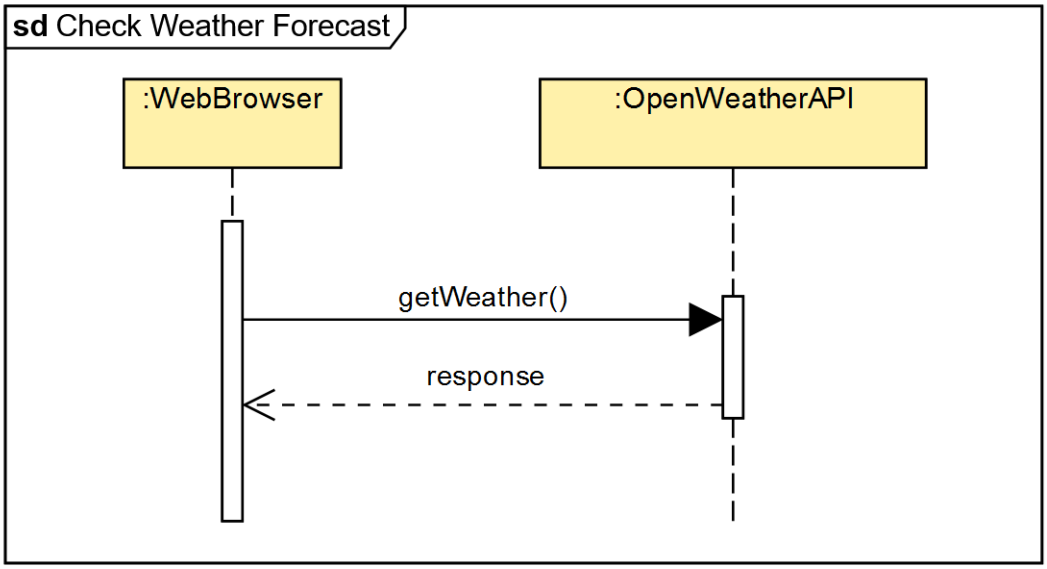


Figure 16: Check Weather Forecast

This sequence diagram represents the interactions between components that occur when displaying the weather forecast to the users. The weather report is tailored to the user area. The *WeatherManager* component uses the *OpenWeatherAPI* component in order to get always up-to-date forecast.

Help Request

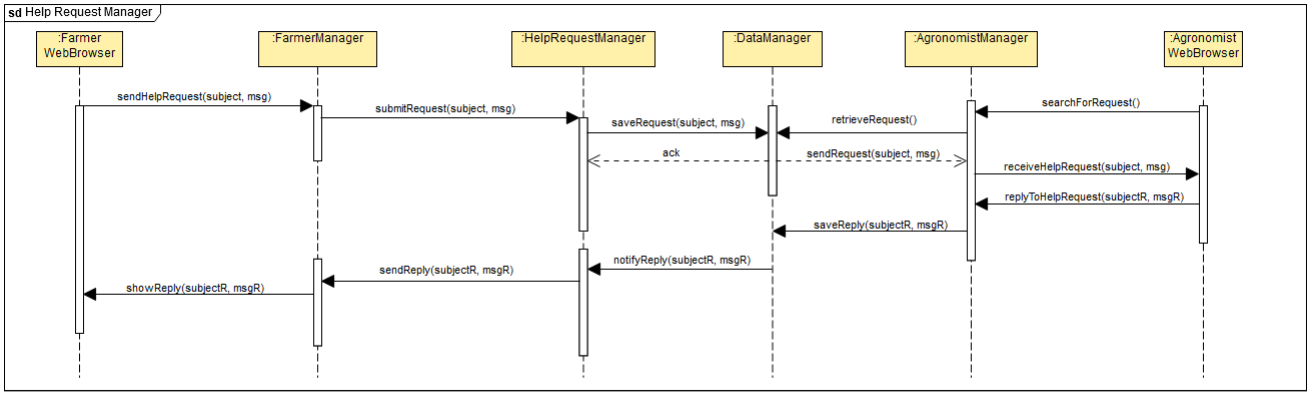


Figure 17: Help Request

This sequence diagram represents the interactions between the components needed to manage the help requests that the farmers send to the agronomists. When a farmer sends an help request, the system saves it in the database. When the agronomist is online, the system checks if there are some unread messages in the database and sends them to the agronomist. The agronomist replies to the request and saves it into the database. When the farmer is online the system checks if there are some unread replies in the database and displays them to the farmer. Since the exchange is necessarily asynchronous because of the fact that the receiver of the message might be offline in the moment of the delivery, the messages have to be stored on the database and the unread ones have to be fetched each time the user is back online.

Daily Plan Management

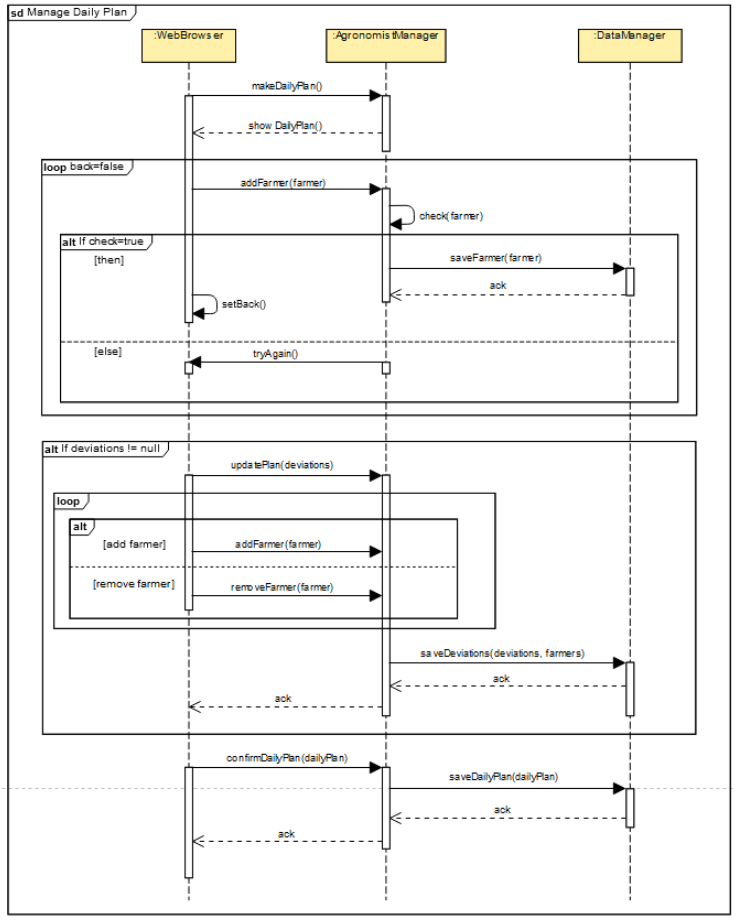


Figure 18: Daily Plan

This sequence diagram represents the interactions between components in managing the daily plan. In the diagram, there are both the creation and the updating of the daily plan.

2.5 Component interfaces

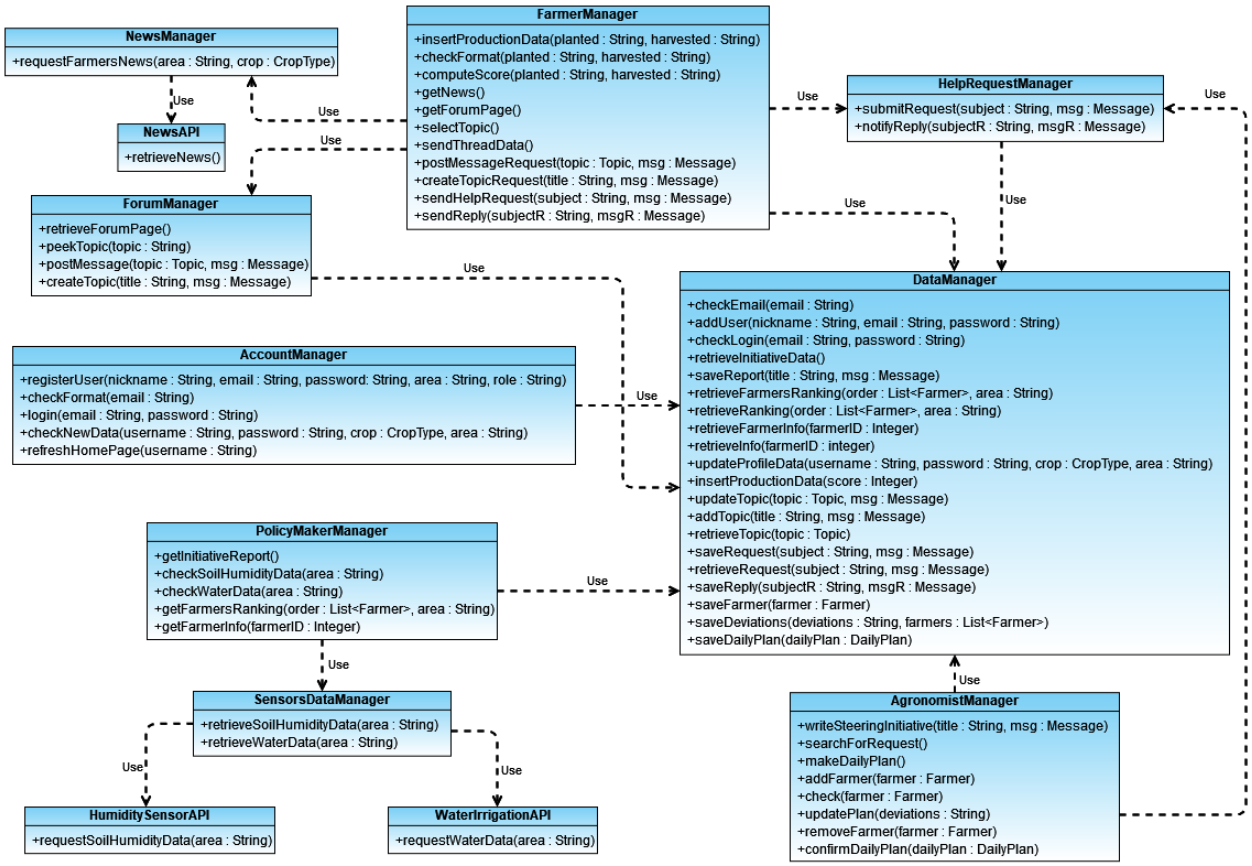


Figure 19: Component Interfaces Diagram

This diagram above describes in detail the interfaces and the corresponding methods offered by each component, it also shows the interaction between them. The diagram refers to the component diagrams in *Figure (3)* and *Figure (4)*. Please note that the methods described in *Figure (19)* don't represent exactly the final version that will be used during the implementation, they just provide a logical representation.

2.6 Selected architectural styles and patterns

- **Four tier architecture**
  This kind of architecture divides the system in four different modules: client, presentation, applications and data tier. As descibed in the previous sections, the architecture is based on this architectural style.

- **Thin client**
  The thin client approach increases performance and security since every piece of data and business logic code are not stored locally but in a server.

- **Scalability**
  A four tiers application guarantees that a scaling architecture approach is adopted only for the most critical components. The result obtained maximize the performances but also minimize the costs.

- **Model View Controller**
  Model-View-Controller [1] (MVC) is a software design pattern used for developing user interfaces that divides the program logic into three interconnected elements. This separate internal information representations from the ways they're presented to and accepted from the user. These three components are:

  - Model: the 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: The view defines how the app's data should be displayed.
  - Controller: it contains logic that updates the model and/or view in response to input from the users of the app.

2.7 Other design decisions

2.7.1 Servers availability and response time

In order to guarantee the availability of 99.998% promised in the RASD, replicas of the web servers and the application servers are needed, this prevents the system to crash in case of a point failure. Servers replication is done by the hosting service Heroku. The response time described in the RASD is 0.5 seconds, the load balancer will allow to reach this goal by distributing the incoming requests from users among the replicas. The database server is replicated as well for the same reasons. So that, in case of failure the system can continue to work, thanks to the distributed nature of the system.

Furthermore, in case of failure of the whole system, the status can be restored thanks to backups. So a backup routine has to be planned. Periodically, in an automated way, the system shall perform a backup of the system and replace the older copy with it.

2.7.2 Farmers ranking

The ranking system has been thought as an aid for policy makers and agronomists to understand how the farmers are performing. However, there are multiple factors that play a role in a farmer's performance, some of them might be difficult to quantify. Furthermore, DREAM should highlight the climate resiliance aspects, which is not a trivial task.

In the first version of the app, the following formula is proposed in order to achieve a fair and scientific way to compute the score associated to each farmer.

$$score = \frac{harvested\ qty}{sown\ qty} \cdot \left(1 + \frac{hostile\ weather\ days}{365}\right) \tag{1}$$

The formula can be broken down into two parameters, η and δ:

$$\eta = \frac{harvested\ qty}{sown\ qty} \tag{2}$$

$$\delta = \frac{hostile\ weather\ days}{365} \tag{3}$$

η is the main term that represents the productivity of a farmer, that is how efficient he/she has been given the quantity of sown seeds (in kilograms) and

the return in harvested crop (in kilograms). The data taken into consideration for η's computation is the one collected in 12 months, from March to March of the previous year year[1], resetting the score to 0 every March. In the computation, only full production entries (with an harvested quantity of the crop) are taken into consideration.

Whereas δ weighs up the harshness of the climate conditions endured by the farmer in the last year. The variable *hostile weather days* is a counter of the days in which the climate has not been very lenient in the farm's region and could have ruined the plantation. Days are considered "hostile weather days" in case the amount of rain fallen during that day has been greater than 115.60 mm/hour (so in condition of very heavy rainfall or worse)[2], particularly expected during monsoon seasons; as well as strong heat waves and prolonged periods of drought. Also specific crop disease that might have afflicted the region for a certain time during the year may be considered as challenging days to take in consideration. However, at the time of this document, just rainfalls data are available for the DREAM app, so only heavy rain days will be taken into account in computing the δ parameter.

The role of δ is to favor farmers that have been particularly hitten by climatic problems but still have managed to produce, since this demonstrate that they've been able to carry out usefull practices that helped them staying productive despite the natural adversities.

A new δ parameter is computed everyday, for each one of the 10 Telangana's district. The δ is valid for just that day and is computed collecting all the data from today till the 12 months prior, creating a sliding window of a solar year.

An upgrade of a farmer's production entry into a full one triggers the recomputation of their score, since their η parameter changed. The new score will be affected by their district's δ computed that particular day in which the score has been triggered.

> **Example**
> Let's suppose it's the 15th of June and farmer Pajeet (in Medak district) has been using DREAM for 3 months. He inserts just one production entry stating that he has sown 10kg of rice the 15th of June in his land.

[1]March has been chosen as the reset month after comparing the sowing to harvesting period of all the most popular crop grown in India. In a solar year, the first to be sown are cucumbers and pumpkin, to the last to be harvested are wheat and mustard, all of these in March.

His score is 0 since no full production entry has been inserted, that is the entry of Pajeet is still missing the harvested amount since the crop has not been harvested yet. From January to May, there have been several days of heavy rain, 23 days to be specific.

After five months, Pajeet harvests his rice field that has produced 10000kg of rice, so he upgrade the rice production entry (begun the 15/06) on his profile the 15th of November. This insertion cause his score to be re-computed because now the status of his rice prodcution entry has become "full". Since June, there have been 8 days of very heavy rainfalls in Medak: 7 days in Septemper and 1 day the 20th of October. Up to the 15th of November the δ parameter for Medak district is $\delta_{year} = \frac{23+8}{365} = 0.085$. His score is updated the 15th Nov as follow: $score_{year} = \frac{10000}{10}(1 + 0.085) = 1085$.

The 27-28th of November, it rains heavily in Medak again. The 29th of November, the policy maker Chaytanya queries the farmer ranking. There is Pajeet in the ranking, his score is still 1085.

The rank is directly proportional to the farmer's current score. If farmers obtain the same score, they are going to be ranked in the same position as well.

2.7.3 Farmers location

In order to locate a farm the user has two options:

- Address: farm can be located through the address inserted by the user during the sign up phase. The address is verified by the *LocationModule*.
- GPS coordinates: if the address hasn't been inserted by the user or, if the *LocationModule* doesn't find the address, the user can manually specify the coordinates with the help of the *MapService*.

2.7.4 Agronomist daily plan

An agronomist has to have a ready daily plan. During the daytime it can be manually updated until the end of the day where the system automatically confirms the plan. An empty plan means that the agronomist hasn't visited any farmer.

3 User Interface Design

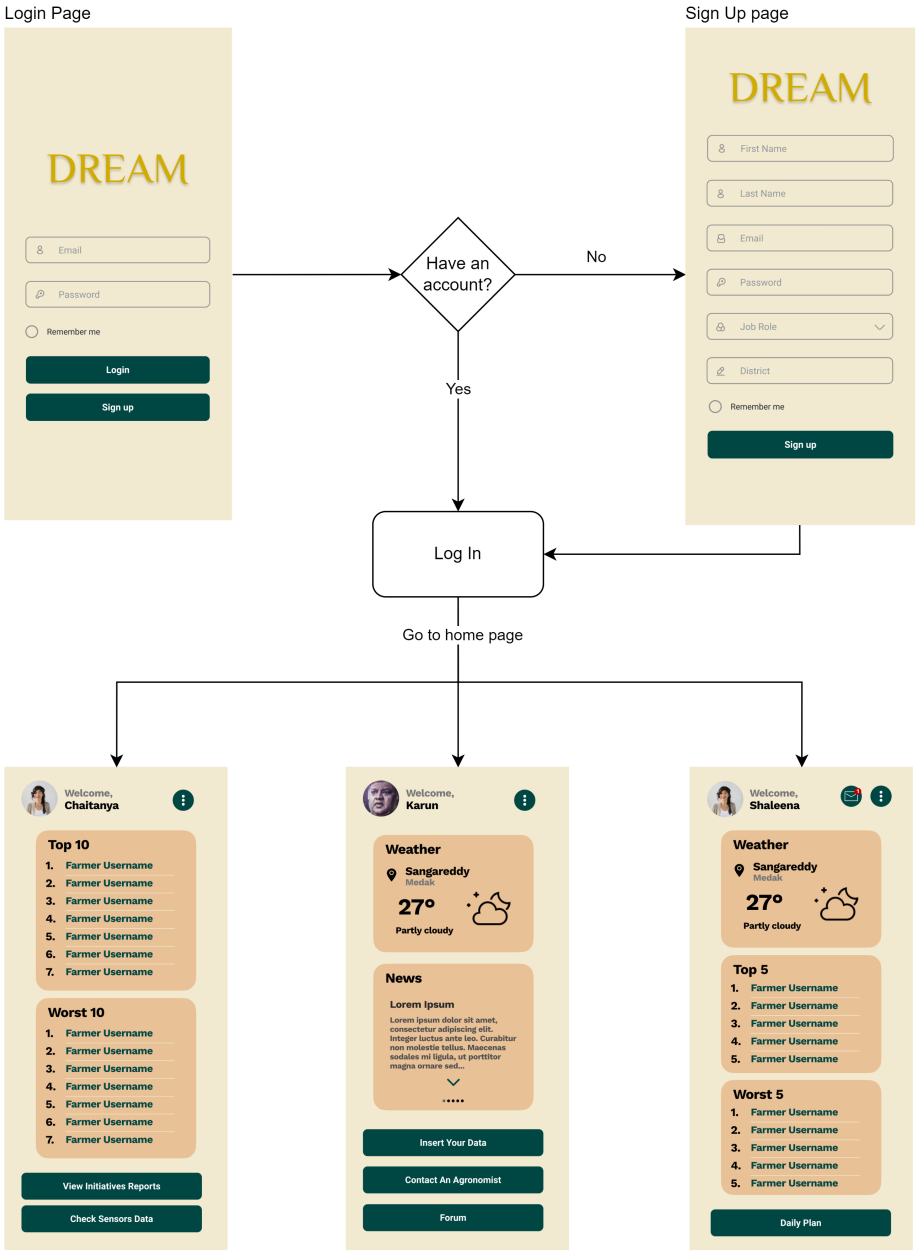


Figure 20: Sign Up and Log In

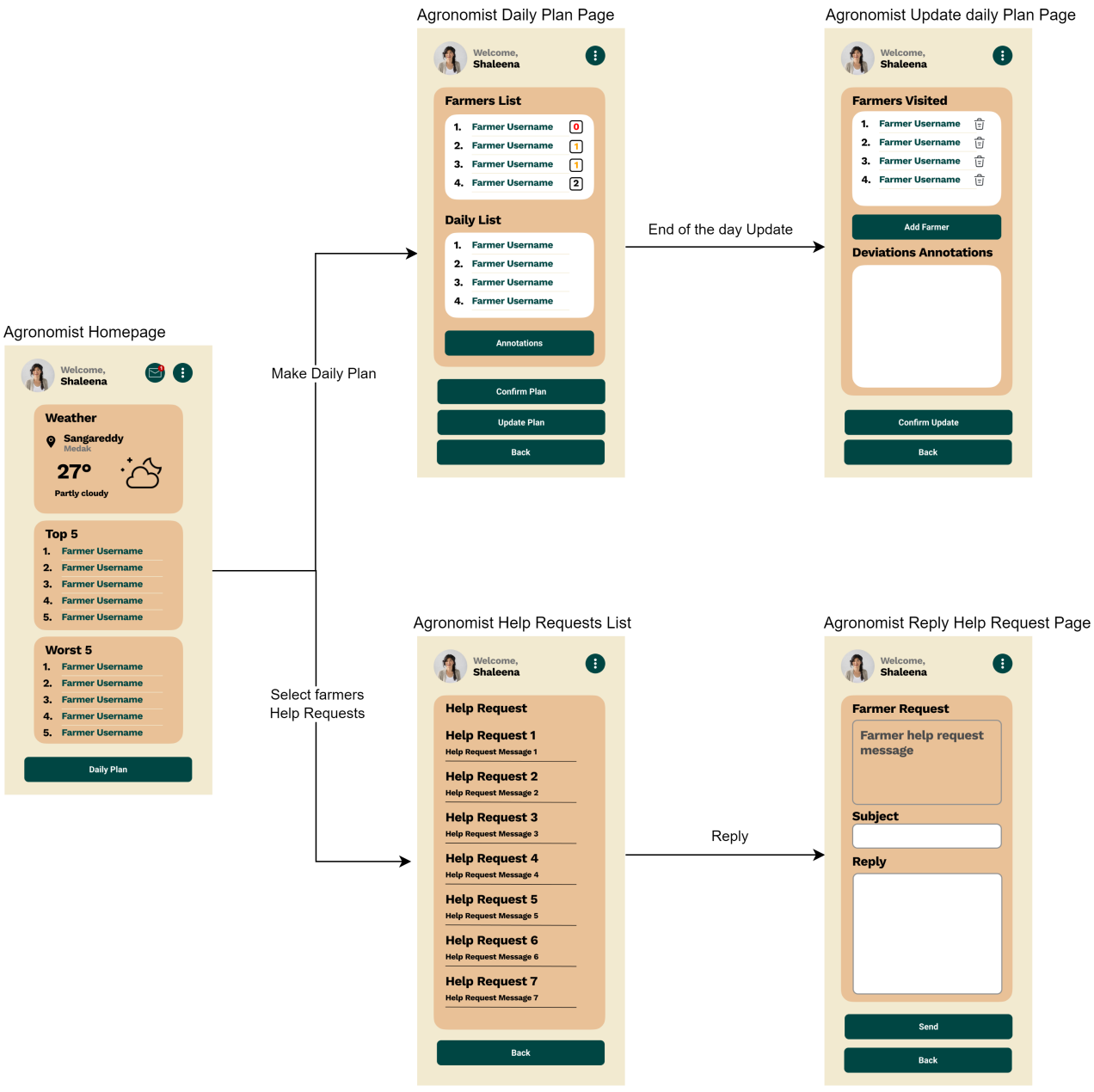


Figure 21: Agronomists Interactions

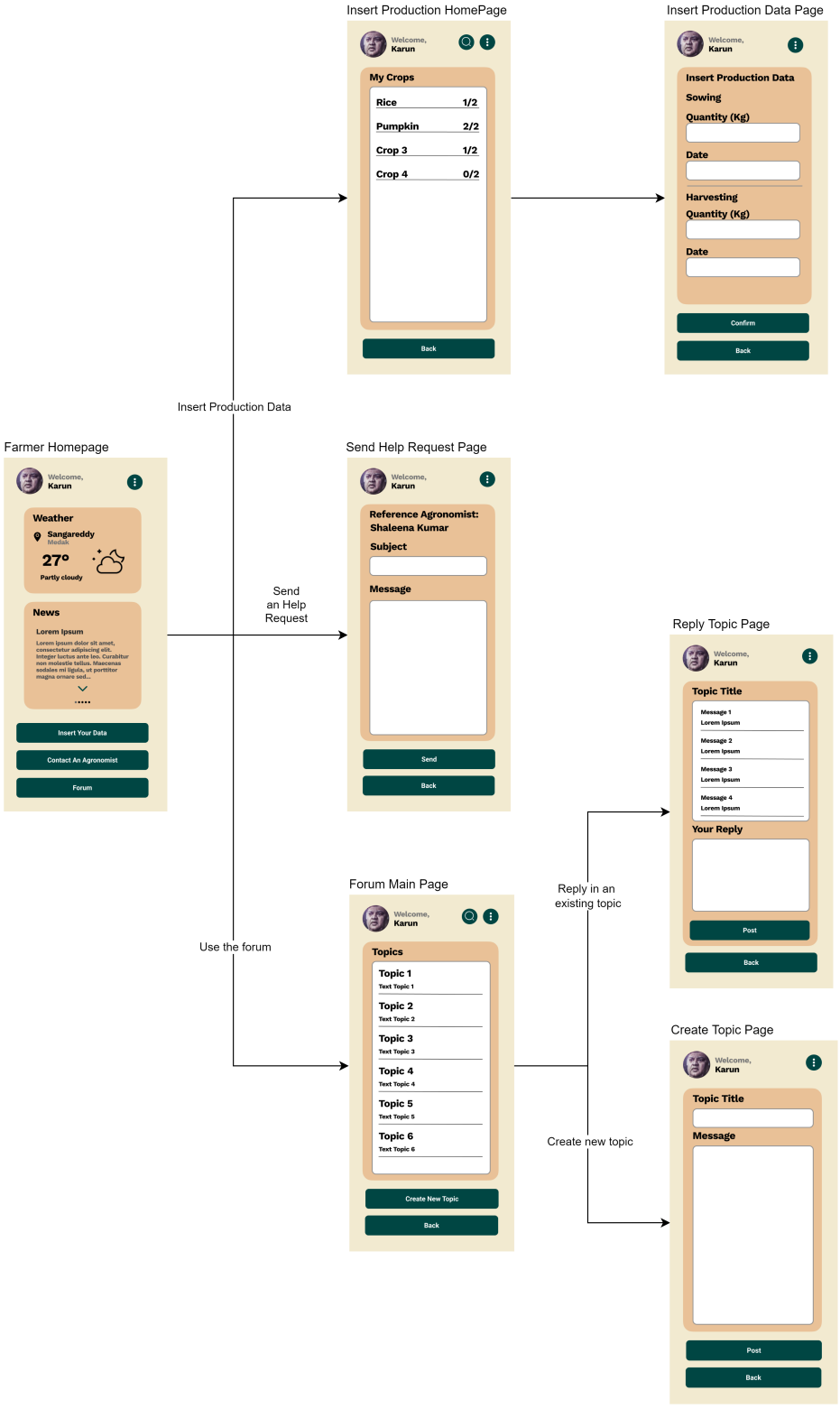


Figure 22: Farmers Interactions

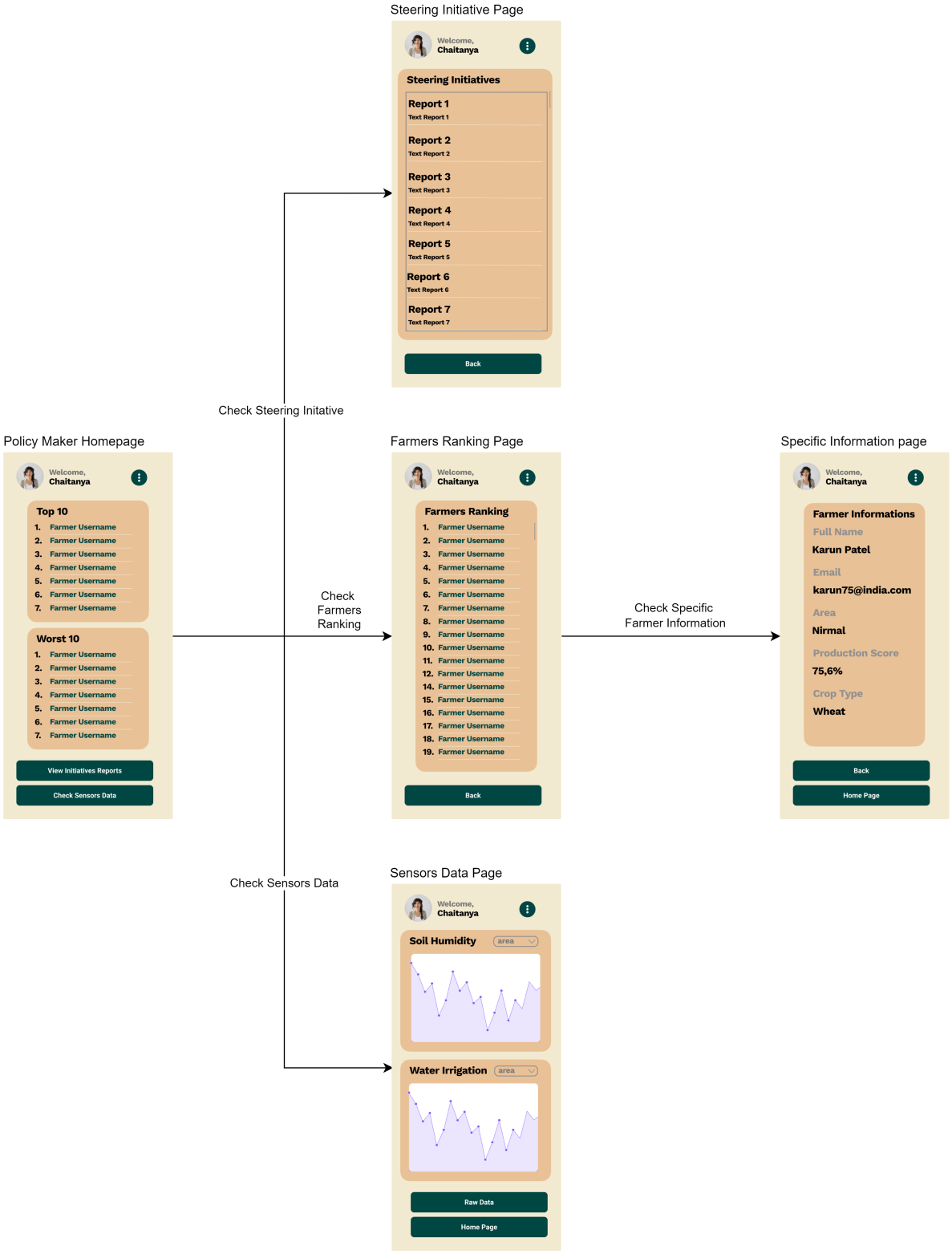


Figure 23: Policy Maker Interactions

4 Requirements Traceability

Requirements	Components
R1) The system must allow only registered and logged-in users to use the app R2) The system shall allow users to be identified by an email of their choosing R3) The system must suggest to the users the area in which they are during the registration phase, if the IT device is equipped with GPS technology. In any case, the final decision will be up to the registering user	• AccountManager • LocationModule • MapServiceManager
R4) The system must inform the user to try later if they are experimenting connectivity issues	• AccountManager

R5) The system must allow Policy Makers to see up-to-date statistic data about water irrigation systems and soil humidty sensors R6) The system must allow Policy Makers to retrieve all agronomists' steering initiative reports uploaded to the database R7) The system must give Policy Makers up-to-date ranking of the best and worst performing farmers	• PolicyMakerManager – WaterIrrigationService – HumiditySensorService – SteeringInitiativeService – RankingService • DataManager • SensorsDataManager • RankingManager
R8) The system must use a fair and scientific score to rank the Farmers in order to represent the real situation	• RankingManager
R9) The system must let Farmers insert their production information every day	• FarmerManager – ProductionService • DataManager

R10) The system must provide the contact of the agronomist appointed to the area of the farmer requesting professional help	• FarmerManager – AskRequestService • AgronomistManager • DataManager • HelpRequestManager
R11) The system must allow every user registered as Farmer to access the forum, to create new discussions and to post replies to already existing ones	• FarmerManager – ForumService • ForumManager • DataManager
R12) The system must notify agronomists of unresolved requests of help from Farmers	• AgronomistManager – AnswerRequestService • HelpRequestManager

R13) The system must allow the agronomists to add a new schedule for the day each day R14) The system must suggest to the agronomists which farms to visit while planning the daily plan based upon the last visit day following a FIFO policy R15) The system must suggest to the agronomists only the farms among the ones in their competence area R16) The system must allow the Agronomists to update their schedule during the day and to confirm the execution before the end of the day R17) The system must register the already uploaded schedule as definitive if the user doesn't confirm the daily plan for day x before 23:59 of the day x	• AgronomistManager – DailyPlanService
R18) The system must give Agronomists up-to-date ranking of the best and worst performing farmers in their responsibility area	• RankingManager • AgronomistManager – RankingService

R19) The system must show weather forecasts relevant to the area concerning the Farmer or the agronomist	• WeatherManager • FarmerManager • AgronomistManager
R20) The system must present news concerning crop only if relevant to the Farmer's own crop type	• FarmerManager • NewsManager

Here, we present a summary of the table above for a more immediate visualization.

AccountManager:	AcM
LocationModule:	LM
MapServiceManager:	MSM
PolicyMakerManager:	PMM
AgronomistManager:	AgM
FarmerManager:	FaM
ForumManager:	FoM
DataManager:	DM
HelpRequestManager:	HRM
RankingManager	RM
NewsManager:	NM
WeatherManager:	WM
SensorsDataManager:	SDM

Table 2: Components' legend

	AcM	LM	MSM	PMM	AgM	FaM	FoM	DM	HRM	RM	NM	WM	SDM
R1	x	x	x										
R2	x	x	x										
R3	x	x	x										
R4	x												
R5				x				x					x
R6				x				x					x
R7				x				x					x
R8								x					
R9						x		x					
R10					x	x		x	x				
R11						x	x	x					
R12					x				x				
R13					x								
R14					x								
R15					x								
R16					x								
R17					x								
R18					x					x			
R19					x	x						x	
R20						x					x		

Table 3: Component and requirement mapping

5 Implementation, Integration and Test Plan

5.1 Implementation Plan

Multiple components will be implemented at the same time, in order to parallelize the development when possible. The general plan is to follow a bottom-up approach, so that core and basic functionalities with very few dependencies can be tested as soon as their incapsulating component is done. By doing so, the application will be built up with solid and tested foundations that will ease the further testing of bigger and complex components. In any case, unit testing will be performed on each component on the go, in order to find flaws out in advance. This will positevely impact the necessary actions

to fix the faults since they will be done in an earlier stage.

The implementation's order of the component will be as follow:

1. DataManager, MapServiceManager, WeatherManager, SensorDataManager, NewsManager

2. RankingManager, HelpRequestsManager, ForumManager, LocationModule

3. PolicyMakerManager, AgronomistManager, FarmerManager, AccountManager

Each group is composed by independent modules so they can be easily developed in parallel. Furthermore, it is expected that external services (e.g. GoogleMapsAPI, OpenWeatherAPI, WaterIrrigationAPI, HumiditySensorsAPI, NewsAPI and DBMS Service) work properly since they're not a responsibility of the DREAM app.

The *DataManager* component should be implemented first since all the remaining components of the application server depend directly or indirectly on it for handling the communication with the database to store and retrieve data. *MapServiceManager*, *WeatherManager*, *SensorDataManager* and *NewsManager* components are independent from all the other ones because the former just communicates with the external services, by modifying the API's information so that it can be comprehensible by the Application server, and adapting the requests to the API's protocol.

The second group depends on the first one but at the same time incapsulates building blocks for the third one. *RankingManager*, *HelpRequestsManager* and *ForumManager* directly exploit the *DataManager* module to retrieve information, store data or carry out consistency and security checks. Moreover, *LocationModule* also use the *MapServiceManager* to access the map service that ease the registration to the farmers in localizing their farm.

Finally, *PolicyMakerManager*, *AgronomistManager*, *FarmerManager* and *AccountManager* are the component that mostly depend on all the lower level ones, as can be seen in figure 4.

5.2 Integration Strategy

Considering both the overall system's architecture and the implementation plan, the chosen integration strategy is the bottom-up approach. System

integration begins with the integration of the lowest level modules and uses test drivers to drive and pass appropriate data to the lower-level modules. As and when the code for the other module gets ready, these drivers are replaced with the actual module.

This approach allows to start the integration and testing without necessarily waiting for the completion of the development and the unit testing of each system's component. Being the low-level modules and their combined functions often invoked by other modules, it is more useful to test them first so that meaningful effective integration of other modules can be done. Moreover, starting at the bottom of the hierarchy means that the critical modules are built and tested first and therefore any errors in these modules are identified early in the process.

Each integration in the same level (defined by the groups of the previous section) is independent and there is no specific order in which to complete them. In this way, the integration process and its testing are more flexible.

5.2.1 Integration and Testing

In this section it is defined the order of the integration between components. Test drivers will be used to simulate higher components not yet implemented. As already said, bottom-up approach is going to be followed in implementing the components. So firstly, *DataManager* (fig. 24), *MapServiceManager* and *WeatherManager* (fig. 25), *SensorDataManager* and *NewsManager* (fig. 26) are implemented using drivers for higher-up components yet to be built.

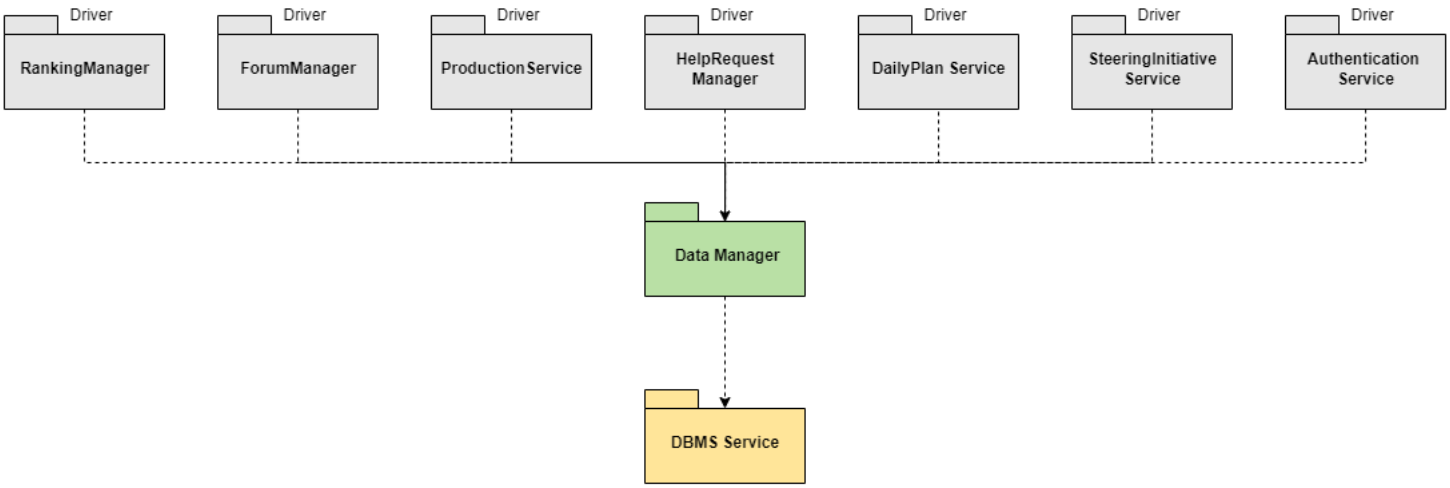
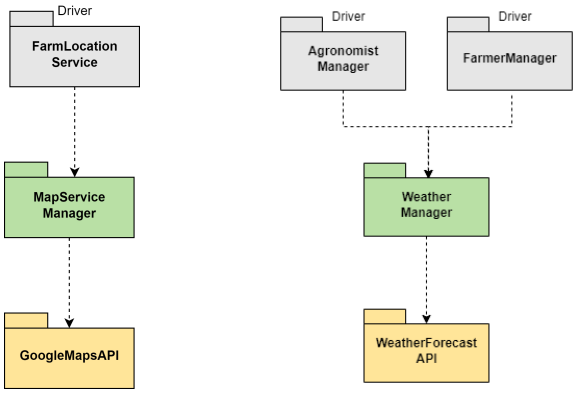


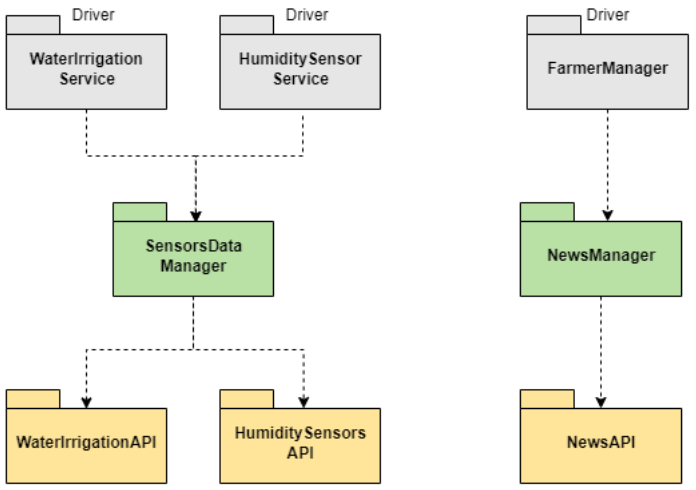
Figure 24: Integration of DataManager component



(a) MapServiceManager

(b) WeatherManager

Figure 25: Integration of MapServiceManager and Weather Manager components

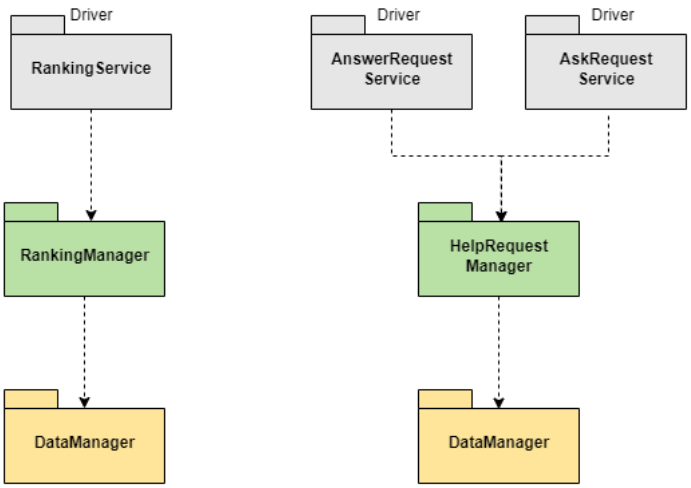


(a) SensorDataManager

(b) NewsManager

Figure 26: Integration of SensorDataManager and NewsManager components

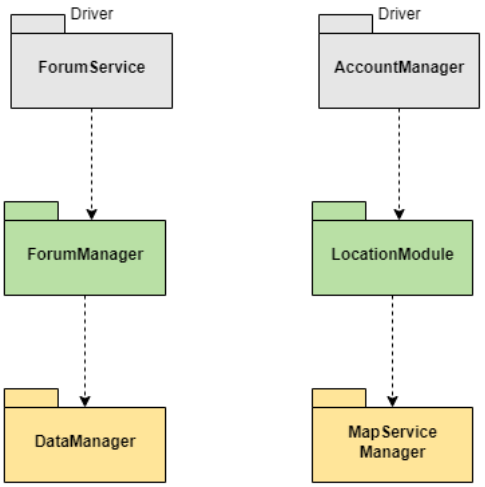
After the first group implementation and testing, it's time for the implementation of the middle tier: *RankingManager* and *HelpRequestManager* (fig. 27), *ForumManager* and *LocationModule* (fig. 28).



(a) Ranking-Manager

(b) HelpRequestManager

Figure 27: Integration of RankingManager and HelpRequestManager components



(a) Forum-Manager

(b) Location-Module

Figure 28: Integration of ForumManager and LocationModule components

Finally, the last group, composed by the components with most dependencies, doesn't require drivers to do integration testing, they just rely on the already built modules underneath. The integration testing to carry out for *PolicyMakerManager*, *AgronomistManager*, *FarmerManager* and *AccountManager* are presented in the following figure respectively: 29, 30, 31, 32.

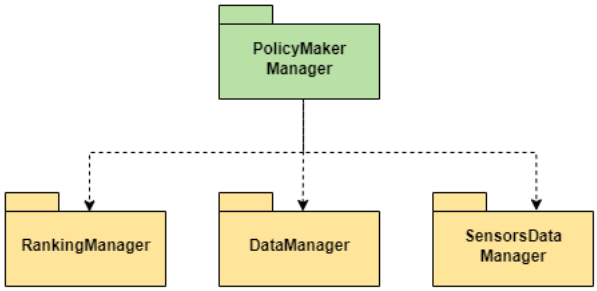


Figure 29: Integration of PolicyMakerManager components

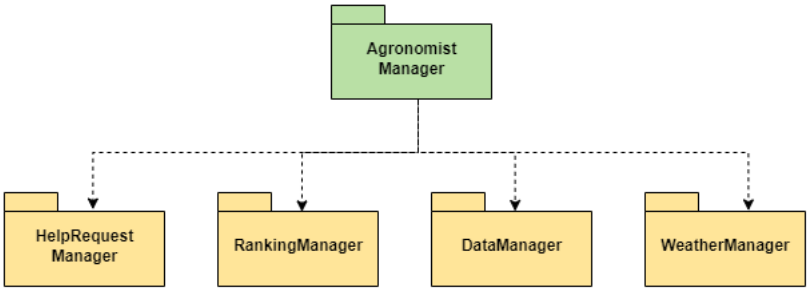


Figure 30: Integration of AgronomistManager components

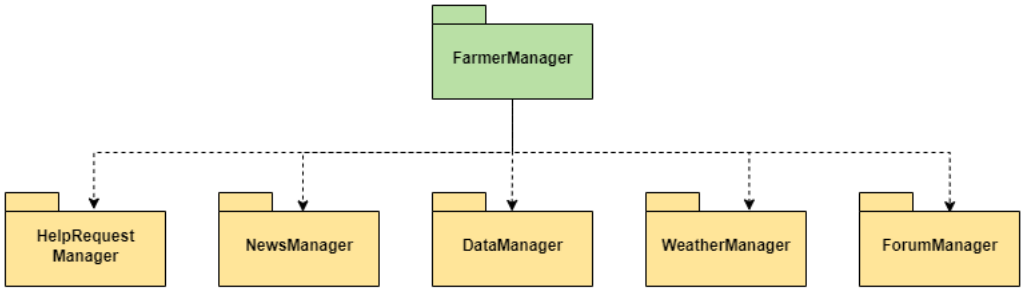


Figure 31: Integration of FarmerManager components

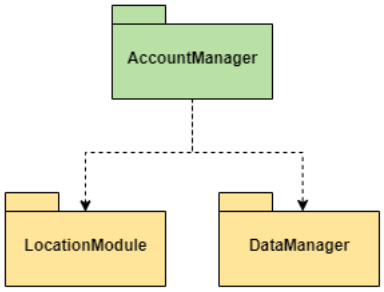


Figure 32: Integration of AccountManager components

5.3 System testing

Once the whole application has been implemented and tested, the application server can be integrated in the whole distributed system. This brings new testing phases to carry out: stress and load testing. The purpose of these is to achieve data regarding availability. Load testing is especially important because of the high expect demand of users that will use the system. On the same note, performance testing would allow us to know the response time of the most frequent operations and requests such as retrieving sensors' data, ranking visualization, posting on the forum, weather forecast and news visualization, sending an help request and replying to one, retrieving of agronomists' reports, etc. Performance testing is a key aspect to fulfilling the RASD non-functional requirement of needing a response time of 0.5s. Moreover, the testing should consider important algorithms such as the retrieval of data from the database (queries and insertions) or response time from external APIs.

6 Effort Spent

Student	Time for S.1	S.2	S.3	S.4	S.5
Ottavia Belotti	1h	7h	2h	4h	7h
Alessio Braccini	2h	8h	8h	1h	2h
Riccardo Izzo	2h	15h	2h	1h	1h

7 References

References

[1] MDN Web Docs Glossary: Definitions of Web-related terms -> MVC `https://developer.mozilla.org/en-US/docs/Glossary/MVC`

[2] Rainfall state in Telangana: `https://www.tsdps.telangana.gov.in/aws.jsp`