POLITECNICO
MILANO 1863

Computer Science and Engineering

Software Engineering 2

Academic year 2021-2022

---

# Design Document


Data-dRiven PrEdictive FArMing in Telangana

---

*Authors:*

| | |
|---|---|
| Arslan Ali | 971503 |
| Elisa Servidio | 996387 |
| Federica Suriano | 953085 |


January -, 2022

Version 1.0

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Purpose

The goal of DD (Design Document) is to provide a more extensive overview of the architectural decisions, their communication interfaces, made to implement all the functionalities described in the RASD document. In the last chapter (5) it will also present implementation, integration and test plan.

## 1.2 Scope

As stated in the RASD document, the aim of the DREAM software product is to develop and adopt anticipatory governance models for food systems to strengthen data-driven statepolicy. It takes care of the acquisition and management of all data collected in order to support the work of farmers, agronomists and policy makers. The system aims to collect data not only from sensors located throughout the territory, but also from farmers. The analysis of the acquired data aims to improve the production of farmers. Low-performing farmers are identified by policy makers and helped by the best-performing ones. Everything is supervised by agronomists who take care of their own geographical areas of competence.

## 1.3 Definitions, Acronyms, Abbreviations

In the following section is clarified the meaning of some definitions, acronyms and abbreviations which will be use in the DD, in order to help the general understanding of the document.

### 1.3.1 Definitions

Table 1.1: Definitions

| | |
|---|---|
| *The system* | The whole system to be developed |
| *User* | A farmer/agronomist/policy maker who uses the application |
| *Unregistered user* | A farmer/agronomist/policy maker who is not yet registered into the application |
| *Application service* | Functionality offered by the system for certain users |
| *Policy maker* | The user of the application who decides about new policies for Telangana |
| *Farmer* | The user of the application who owns or manages a farm |
| *Address* | The address inserted by the farmer which corresponds to his/her farm's location. It is composed by city, zip code, street and number |
| *Performance* | Indicator of the progress of a farmer's activity up to a certain date. Its value is calculated as numerical score |
| *Score* | Performance rating computed by a function that depends on: type and quantity of harvested product, weather conditions, quantity of water consumed, soil moisture |
| *Well performing farmer* | A farmer who has a score higher than a certain threshold |
| *Bad performing farmer* | A farmer who has a score below a certain threshold |
| *Agronomist* | The user of the application dealing with the management of a certain mandal |

| | |
|---|---|
| *Mandal* | A local government area and administrative division. Telangana is subdivided into districts which are themselves subdivided into mandals |
| *Daily plan* | Application service that allows the agronomist to manage his/her daily work schedule. Specifically, it allows him/her to track and organize visits to farmers |
| *Discussion forum* | Application service which a farmer can use to exchange ideas and opinions about a topic |
| *Post* | Contributions of the participants to the discussion placed one below the other in sequence |
| *Thread* | It is composed of the topic followed by the posts left by the various participants in the discussion |
| *Help request* | Application service which a farmer can use to request for help to the agronomist or/and other well performing farmers |

### 1.3.2 Acronyms

Table 1.2: Acronyms

| | |
|---|---|
| *DREAM* | Data-dRiven PrEdictive FArMing in Telangana |
| *RASD* | Requirement Analysis and Specification Document |
| *UML* | Unified Modelling Language |
| *API* | Application Programming Interface |

### 1.3.3 Abbreviations

Table 1.3: Abbreviations

| | |
|---|---|
| *G.i* | i-th goal |
| *R.i* | i-th requirement |
| *D.i* | i-th domain assumption |
| *UC.i* | i-th use case |

## 1.4 Revision History

Table 1.4: Revision History

| Version | Date | Authors | Summary |
|---|---|---|---|
| 1.0 | 09/01/2022 | Arslan Ali<br>Elisa Servidio<br>Federica Suriano | First release |

## 1.5 Reference Documents

- Specification document: "R&DD Assignment A.Y. 2021/2022"
- Software Engineering 2 course slides A.Y. 2021/2022 - Politecnico di Milano
- *EEE Std 830-1998:* IEEE Recommended Practice for Software Requirements Specifications
- *ETH Zürich (2009),* Requirements Specification: `se.inf.ethz.ch/courses/2011a_spring/soft_arch/exercises/02/Requirements_Specification.pdf`
- UML offcial specification `https://www.omg.org/spec/UML/`

- *Enoch, Simon & Ge, Mengmeng & Hong, Jin & Alzaid, Hani & Kim, Dan. (2018).* A Systematic Evaluation of Cybersecurity Metrics for Dynamic Networks. Computer Networks.
- *Abdullah, Muhammad & Iqbal, Waheed & Bukhari, Faisal. (2019).* Containers vs Virtual Machines for Auto-scaling Multi-tier Applications Under Dynamically Increasing Workloads.

## 1.6 Document Structure

The DD is structured in the following five chapters:

- **Chapter 1 - *Introduction*:**
- **Chapter 2 - *Overview*:**
- **Chapter 3 - *User Interface Design*:**
- **Chapter 4 - *Requirement Traceability*:**
- **Chapter 5 - *Implementation, Integration and Test Plan*:**
- **Chapter 6 - *Effort Spent*:**

# Chapter 2

# Architectural Design

In this chapter are reported the main architectural design choices from a high level point of view. In the following sections are presented the chosen paradigm of the system and its components, followed by the description of how the system will be deployed. Moreover, components' sequence and interface diagrams are listed to describe the way components interact to accomplish specific tasks of the application. Eventually, the last two sections provide specifications regarding the selected architectural style and other design choices.

## 2.1 Overview

The software to be developed is a distributed application with a client-server multi-tiered architecture.
The general architecture of the system can be subdivided into the three logic layers:

- *Presentation Layer*: it handles the interaction with all kinds of user. It provides an interface to users accessing both the mobile application and the web application allowing them to exploit DREAM's services in the most efficient and intuitive way.
- *Application Layer*: it contains business and data access logic. It handles the functions to be provided for the users. It also manages communication between the end user interface and the database.
- *Data Layer*: it manages the persistent storage and retrieval of data accessing to the database.

The three logic software layers are built on top of a four tier architecture, which consists in four different hardware tiers that represent a machine (or a group of machines):

- *Client tier*: it consists of the client programs that are used to exploit system's services and the devices where those programs are installed, which in this case are pc for the web application and smartphone for the mobile applications.
- *Logic tier*: it controls the application's functionality by performing detailed processing. It dynamically generates content in various formats for the client tier from which it collects inputs and return appropriate results from the components in the logic tier. It also guarantees the interactions between the Client and the Data tier. It contains Web and Application Servers.
- *Data tier*: runs the DBMS and holds all of the sensitive application data. It contains database servers.

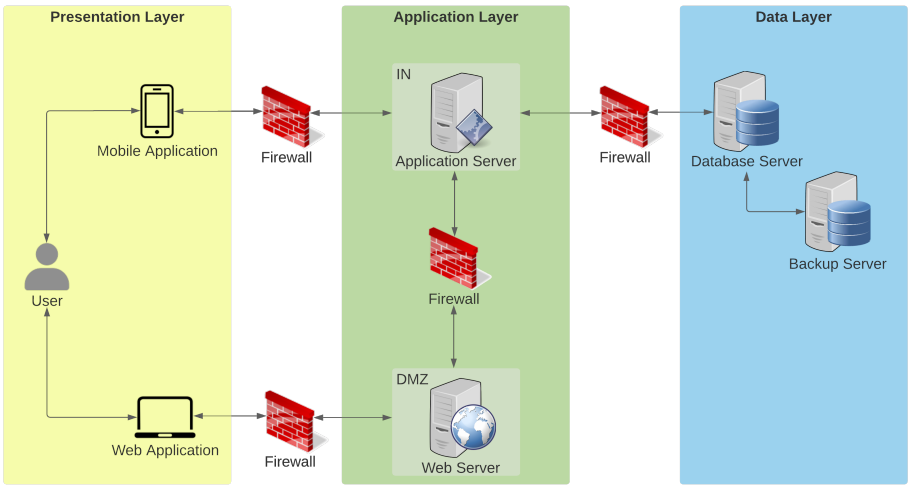


Figure 2.1: System Architecture

The Application Server is the core of the system's architecture as it is the central point between all the other components. It holds the business logic of the system with Web Server. Web and Application servers communicate to each other. It has persistent connection with Data tier and by communicating with its DBMS it can handle all the operations of reading, writing and uploading of data.

The service is supposed to be accessed through both a web application and a mobile application, and this is valid for all kinds of users. Users who

access through DREAM web application can communicate with the Web Server while users logged into the mobile application communicate directly with the Application Server.

To guarantee high level of security due to the sensitivity of treated data, firewalls are installed.
Specifically, there are firewalls between the Internet accessed by the customer to use the application and the Web Server and between the latter and the Application Server in order to create a demilitarized zone (DMZ), so that the external network can access only to the resources exposed in the DMZ.
In order to ensure secure access to the database and to delimit the internal network (IN), a firewall is placed between the Application Server and the Database Server.
Moreover, the Database Server periodically stores critical data in the Backup Server.

The described architecture has been chosen because tiering ensures the ability to scale applications, guaranteeing the system scalability and flexibility to future integrations and modifications. The servers that constitute each tier in fact may differ both in capability and number. If needed, it can be possible to add more servers (horizontal scaling or scaling out) or more powerful servers (vertical scaling or scaling up) to each singular tier.

### 2.1.1 Class Diagram

In the RASD document a Class Diagram has been showed to model the problem, including all the entities involved without going into too much detail.
For this reason, in this section an updated Class Diagram is introduced to represent the system from an implementation oriented point of view.

The following diagram differs from the one presented in the RASD in two aspects:

- the main methods of each kind of user are presented;
- all attributes' data types are specified.

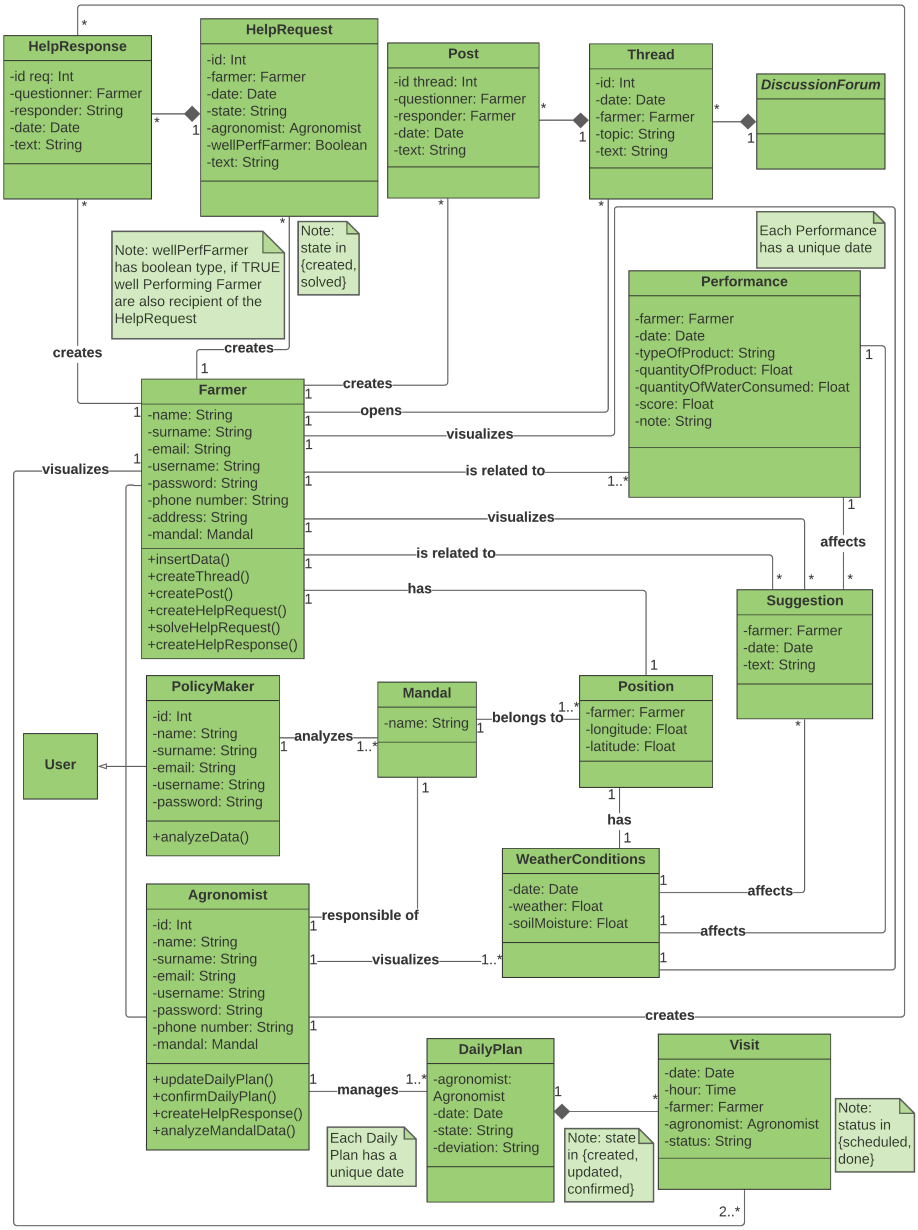


Figure 2.2: Class Diagram

## 2.2 Component View

The purpose of this section is to show the component diagram which is intended to represent the internal structure of the modeled software system in terms of its main components and the relationships among them.

The following legend is used:

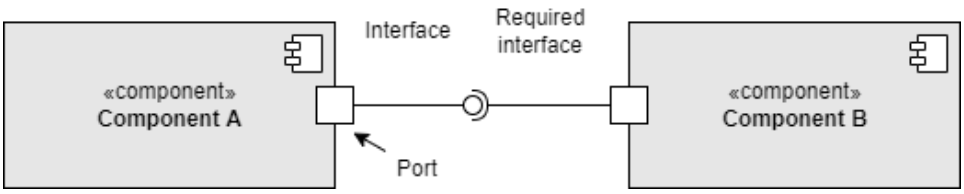


Figure 2.3: Component legend

All components within the diagram are described in detail below.

- **WebApplication**: it manages all user interactions with the web application and associates them with the required functions. It deals with managing all notifications addressed to a specific user.
- **MobileApplication**: it manages all user interactions with the mobile application and associates them with the required functions. It deals with managing all notifications addressed to a specific user.
- **AuthenticationService**: it is necessary for every type of user who wants to access the application (both from the mobile app and the web app) and use it according to the permissions he/she possesses, therefore according to the role with which he/she is registered. In order to verify username and password inserted by the user, the component must be connected to the DatabaseAccess component.
- **RegistrationService**: it manages the registration of a unregistered user; in particular, it allows to retrieve farmer's position thanks to the connection to GoogleMapsAPI component. It also accesses ID codes' DB managed by Telangana government to check validity of ID code inserted by policy makers and agronomist while registering into DREAM through TelanganaGovernmentService component. In order to store the new user in the database, the component must be connected to the DatabaseAccess component. RegistrationService component includes the following subcomponent.

  - **EmailManager**: it is necessary when a user who has not yet registered wants to register. To verify his/her email during registration, the system automatically sends a confirmation email to the user who wants to register and activates a 10-minute timer within which the user must confirm his/her registration.

- **PolicyMakerService**: This component includes the following subcomponents.

  - **TimeChartService**: it computes and displays Telangana's farmers data in a graph so that the policy maker can analyze the various parameters of interest over time. In order to collect all the data of interest, the component must be connected to the DatabaseAccess component.
  - **MapService**: it computes and displays Telangana's farmers performance map using some filter. In order to collect all the data of interest, the component must be connected to the DatabaseAccess component. The connection to GoogleMapsAPI component allows to process the map to be displayed.

- **AgronomistService**: This component includes the following subcomponents.

  - **HelpResponseService**: it allows an agronomist to manage the response to a help request received. In order to retrieve all the help requests with a certain agronomist as a recipient and store the new help response created by him/her, the component must be connected to the DatabaseAccess component.
  - **MandalMapService**: it retrieves and displays on a mandal map relevant personal data of farmers and their performance score to the agronomist responsible of the mandal. This component is closely linked to the TableService component, which is called up as soon as the agronomist selects a farmer of interest on the map. Obviously, in order to collect all the data of interest, the component must be connected to the DatabaseAccess component. The connection to GoogleMapsAPI component allows to process the map to be displayed.
  - **TableService**: it retrieves and displays the table of farmers' performance in descending order based on their performance score. Obviously, in order to collect all the data of interest, the component must be connected to the DatabaseAccess component.
  - **DailyPlanService**: it deals with the creation and updating and confirmation of the daily plan of each agronomist. Specifically,

it guarantees that for each farmer there are at least two visits per year by the responsible agronomist. To retrieve the scheduled plan, this component must be connected with DatabaseAccess component.

  - **WeatherForecastsService**: it collects the data related to the weather forecasts in the mandal of a certain agronomist according to the date selected by the agronomist. To retrieve weather forecasts, it needs to be connected to the external API TelanganaGovernmentService component. The connection to CopernicusClimateDataStoreService component allows to retrieve soil moisture in the mandal. These data are placed on the appropriate mandal map thanks to an external API component: GoogleMapsAPI.

- **FarmerService**: This component includes the following subcomponents.

  - **HelpRequestService**: it is necessary for the creation and subsequent management of help requests by the farmer. The connection to DatabaseAccess component allows it to retrieve the recipients to whom the request can be sent (mandal agronomist and well performing farmers).
    In addition, if a farmer who is well performing has received a request for help from another farmer, this component takes care of managing the function that allows the farmer to respond to the request.
  - **DiscussionForumService**: it manages the creation of threads by a farmer on the discussion forum, the creation of posts in response to a specific thread (always by other farmers) and the search by topic of threads already created. It also performs these functions thanks to the connection to the DatabaseAccess component.
  - **VisitService**: it retrieves the visits scheduled for each farmer on the related agronomist's daily plan. To retrieve them from the database, this component must be connected to the DatabaseAccess component.
  - **PerformanceService**: it takes care of updating a farmer's performance data every time he/she enters new data about his/her crop. It is connected to the DatabaseAccess component not only to store the computed performance but also to get farmer's position and his/her last performance date (to take into account the right time interval for which performance has to be computed).

It also is connected to external API TelanganaWaterIrrigationService to quantity of water consumed. Furthermore it is linked to WeatherConditionService component which allows to retrieve weather conditions and soil moisture.

  - **WeatherConditionService**: it collects the data related to the weather conditions in the position of a certain farmer. To do this, it needs to be connected to the external API TelanganaGovernmentService as well as to the CopernicusClimateDataStoreService component. It also has to be connected to DatabaseAccess component to retrieve farmer's position.
  - **SuggestionsService**: it takes care of calculating the suggestions of a farmer based on his/her relevant information regarding his/her production and the position of his/her farm, data obtained thanks to the connection to the DatabaseAccess and WeatherConditionService components.

- **DatabaseAccess**: it is responsible to get data from a data source. In fact, it manages the access to the database.
- **Database**: This component includes the following subcomponent.
  - **DBMS**: Database Management System refers to a software that allow users to access databases and manipulate, maintain, report, and relate data. It is used to reduce data redundancy, share data in a controlled manner, and reduce data integrity issues.
- **ExternalAPIs**: This component includes the following subcomponents.
  - **GoogleMapsAPI**: it allows the connection of the application to Google Maps APIs. All functions that require access to geographic data not yet saved in the internal database, will rely on this component.
  - **CopernicusClimateDataStoreService**: it is responsible for retrieving soil moisture data in the Telangana region.
  - **TelanganaGovernmentService**: it takes care of retrieving data relating to weather conditions in Telangana region. It also used to access ID codes' DB managed by Telangana government to check validity of ID code inserted by policy makers and agronomist while registering into DREAM.
  - **TelanganaWaterIrrigationService**: it is responsible for retrieving the amount of water consumed by each farmer in the Telangana region.

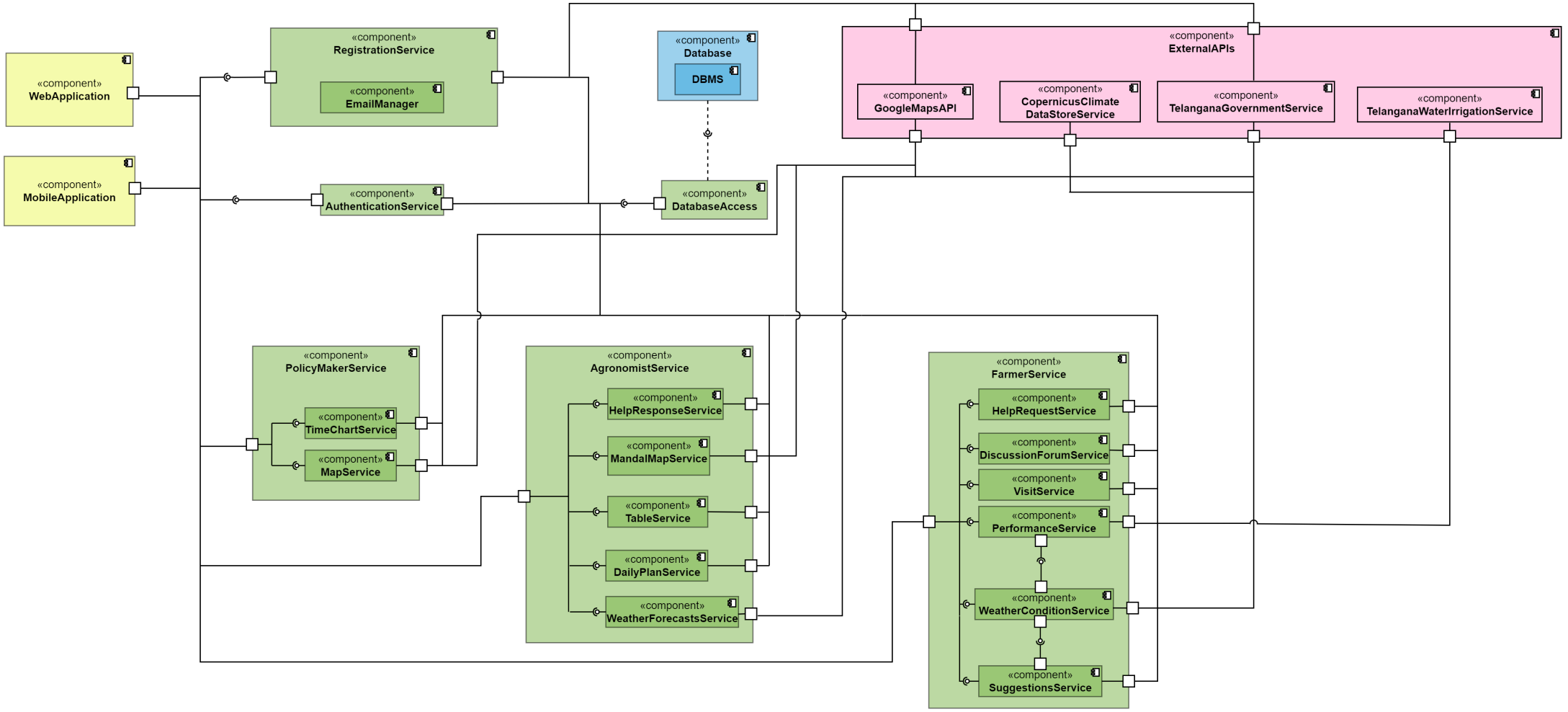


Figure 2.4: Component diagram

## 2.3 Deployment View

This section shows the physical distribution of the system, that is, the environment in which the system runs. Specifically, *Figure 2.4* shows the structure of the hardware components that execute the software.

Firewalls are inserted to ensure security between the different zones and load balancers are designed to distribute the workload.

As previously mentioned, the system is divided into 4 Tier, the components of which are described below:

- **Client Tier**: it consists of the client programs used to exploit system services and the devices on which these programs are installed.
  - **Smartphone**: users access the DREAM system and use the related services through a mobile application from a smartphone.
  - **Personal Computer**: users access the DREAM system and use the related services through the Web browser from a personal computer.
- **Logic Tier**: it consists of components that handle the interaction between client tier and the data tier. It contains Web and Application Servers.
  - **Web Server**: it provides content to the Web using the HTTP(S) protocol. It mainly works to serve the requested web pages continuously and without interruption. As long as the Web Server is up and running, the corresponding web pages and sites will be available to users on the network.
  - **Application Server**: it hosts and exposes business logic and processes. It provides the infrastructure and logical capabilities to support, develop and run applications in a distributed context. The Application Server communicates directly with the Mobile Application through HTTPS protocol, in order to create a lightweight API to request only the necessary data. The Application Server will receive this call, make the call to the external service, filter it, and return only what the application has requested. This will reduce the amount of data transiting over mobile Internet and thus increase application performance.
- **Data Tier**: it runs the DBMS and holds all of the sensitive application data. It contains database servers.
  - **Database Server**: it allows access to one or more Database Systems and is an essential support for Web Servers in managing

the delivery and storage of data. It communicates with the Application Server via the standard TCP/IP protocol and provides access to the physical database which is in charge of storing all persistent data of users who use DREAM.

**Firewalls** are also included in the system architecture to delimit safe areas within the system. Specifically, there are firewalls between the Internet accessed by the customer to use the application and the Web Server and between the latter and the Application Server in order to create a demilitarized zone (DMZ). The purpose of the DMZ is to add an additional level of security to an internal network, where a node belonging to an external network can only access the services made available, without putting at risk and compromising the security of the entire network. In order to ensure secure access to the database and to delimit the internal network (IN), a firewall is placed between the Application Server and the Database Server.

In order to distribute the load of the requests to the DREAM website among several servers, **load balancers** are introduced for the Web Server. In this way, the reliability and scalability of the entire architecture increases because requests will be shared equally among the different servers. If one of the servers is unable to respond to the request, the user can still use the service based on one of the active servers. Similarly, the Web Server may need to query the database to generate responses. Web Servers send database queries to the load balancer which balances the inbound database workload on the database servers.

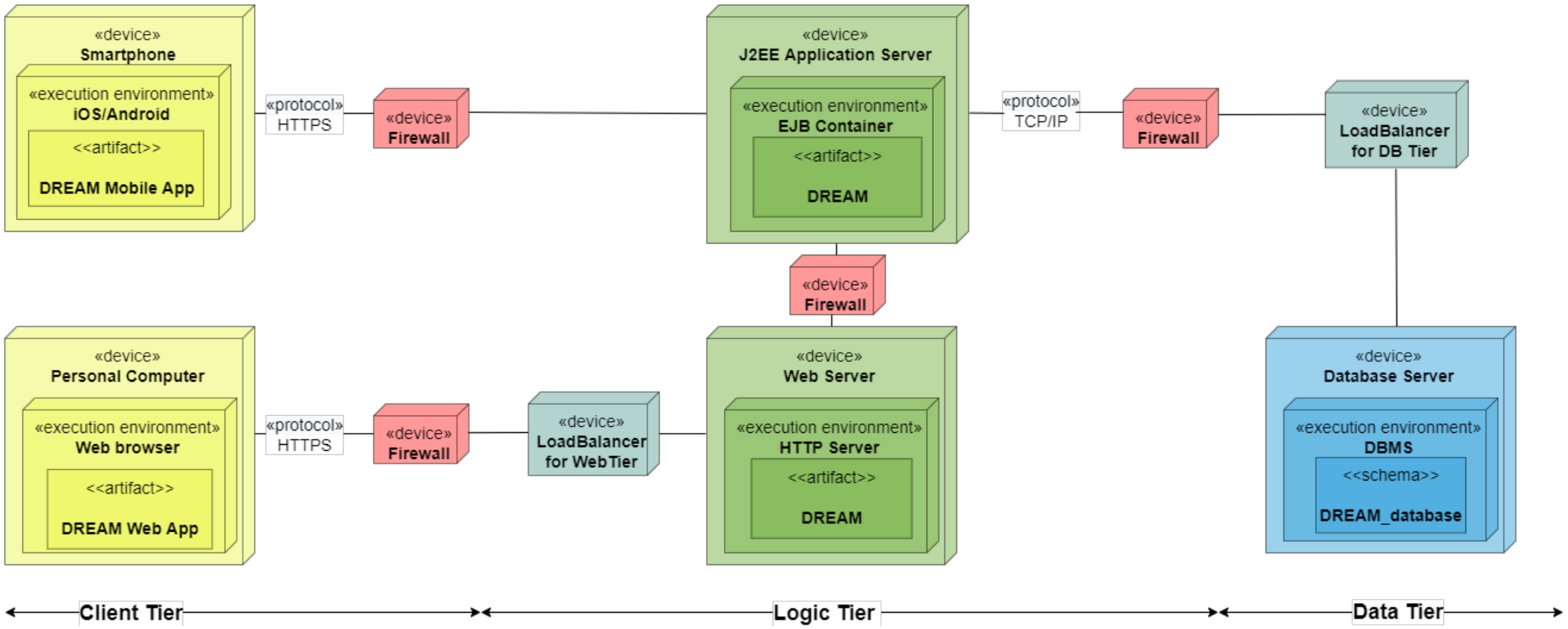


Figure 2.5: Deployment diagram

## 2.4 Runtime View

### 2.4.1 Farmer signs up

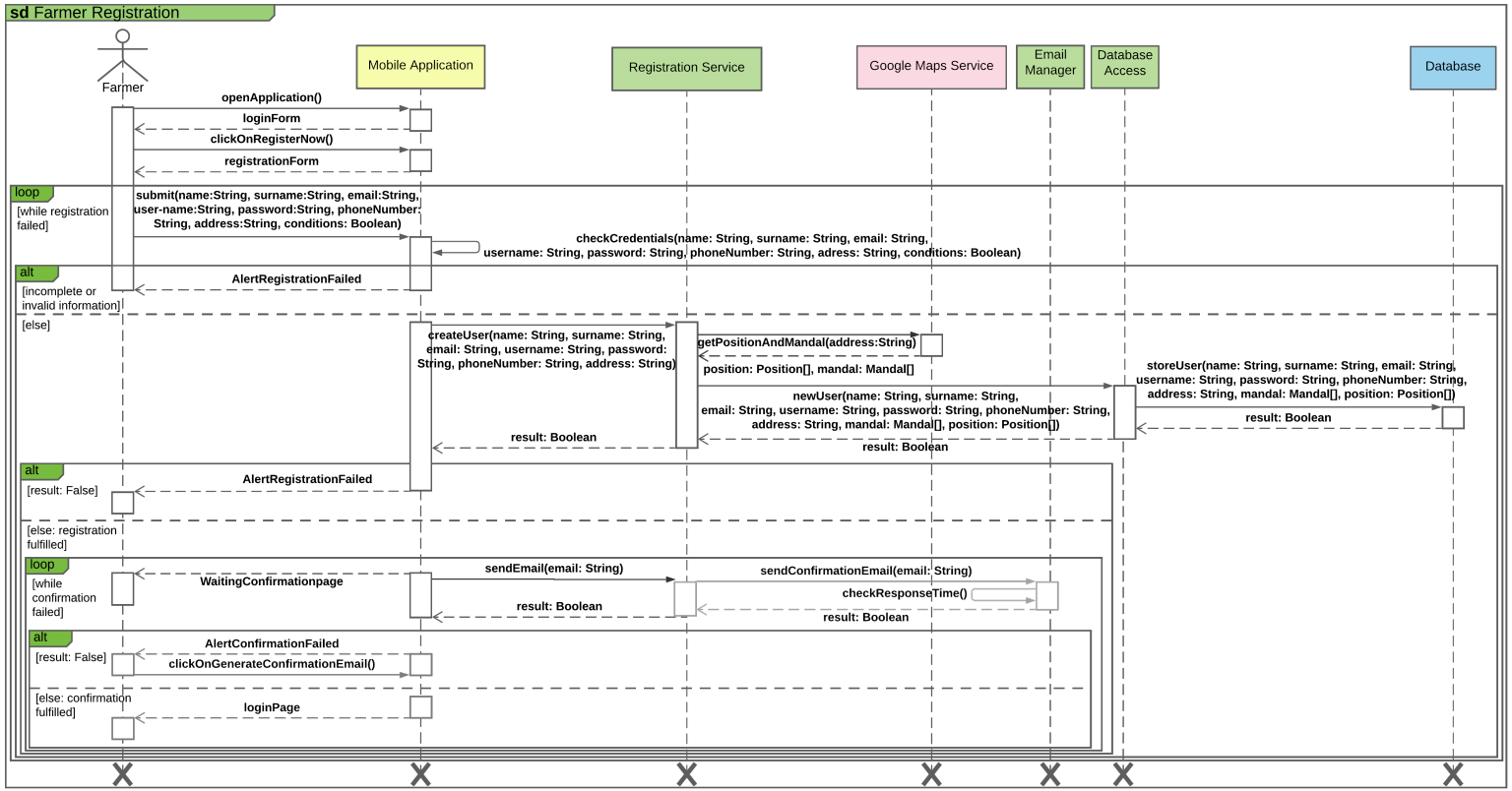


Figure 2.6: Farmer Registration - Sequence Diagram

### 2.4.2 Farmer signs in

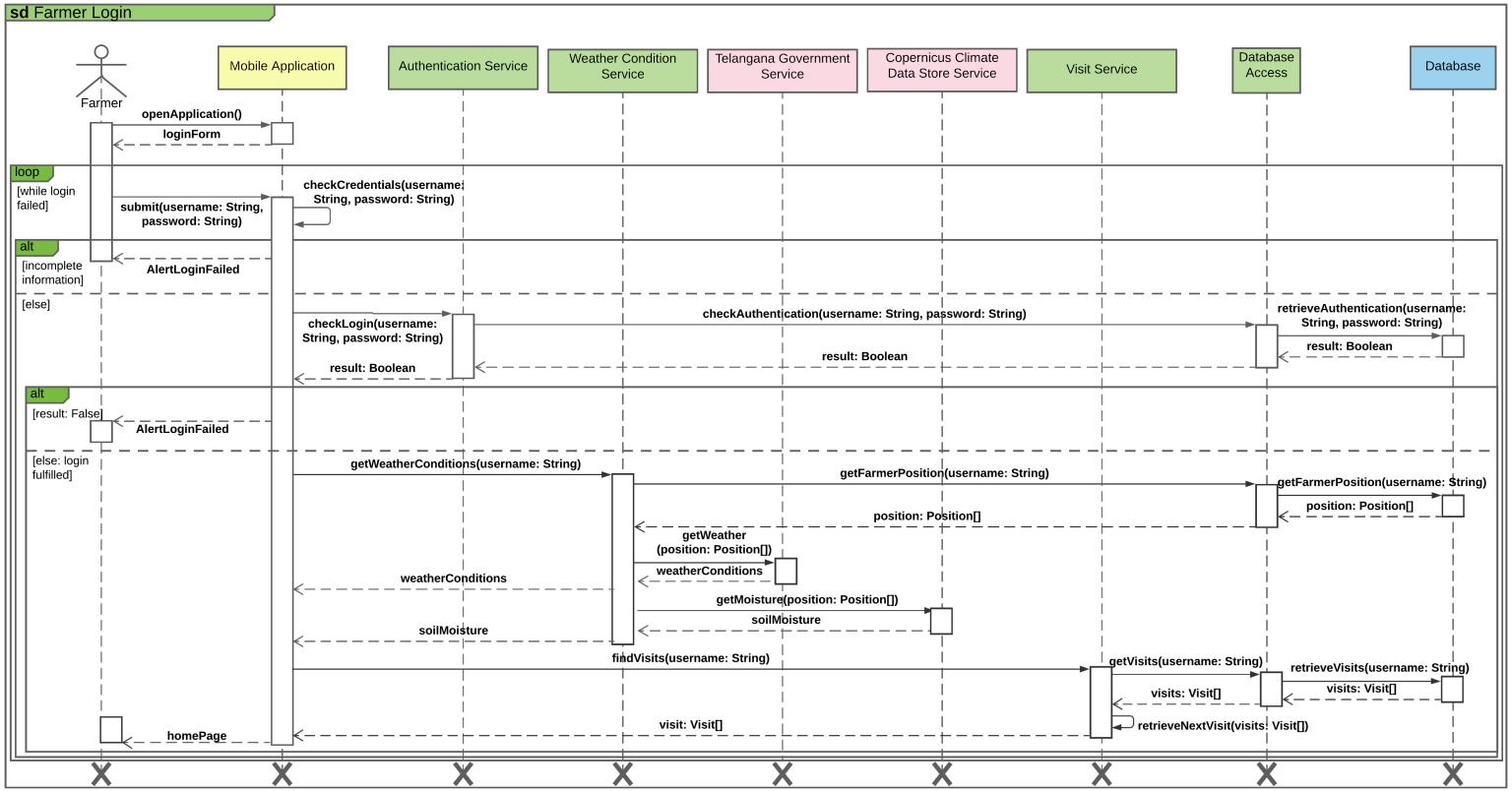


Figure 2.7: Farmer Login Sequence Diagram

### 2.4.3 Agronomist visualizes and analyzes farmers' data

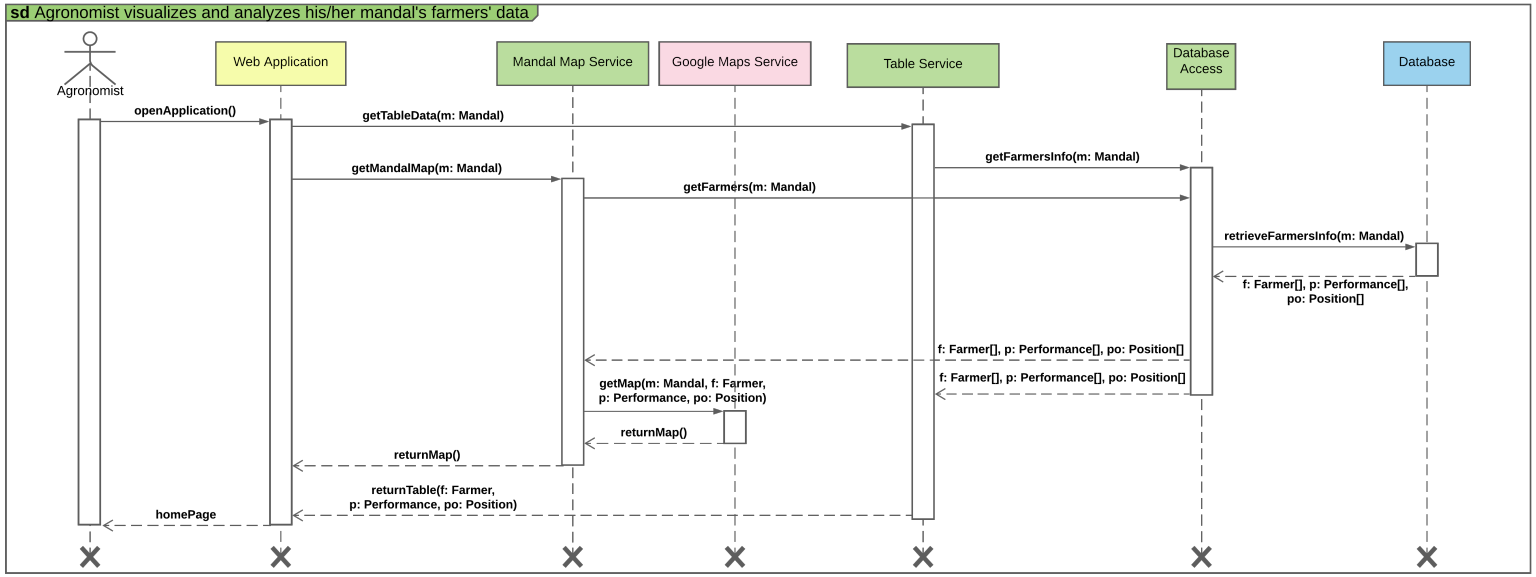


Figure 2.8: Agronomist visualizing and analyzing farmers' data Sequence Diagram

### 2.4.4 Agronomist visualizes weather conditions

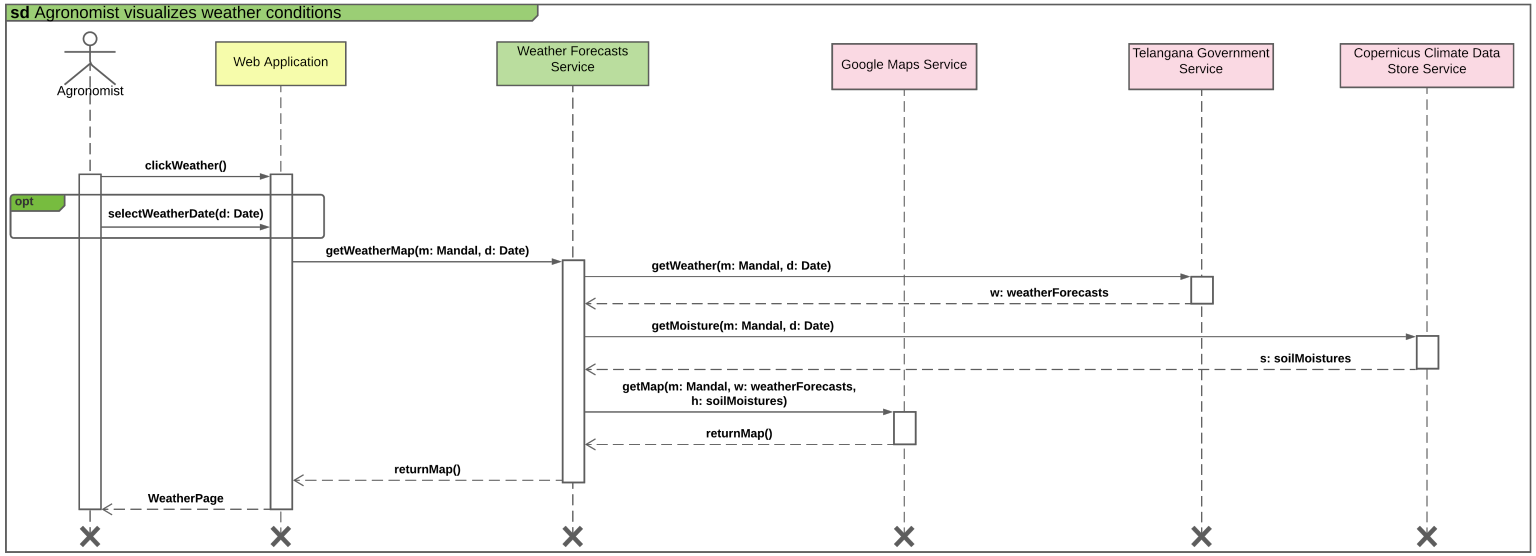


Figure 2.9: Agronomist visualizes weather conditions

### 2.4.5 Policy Maker visualizes and analyzes farmers' data

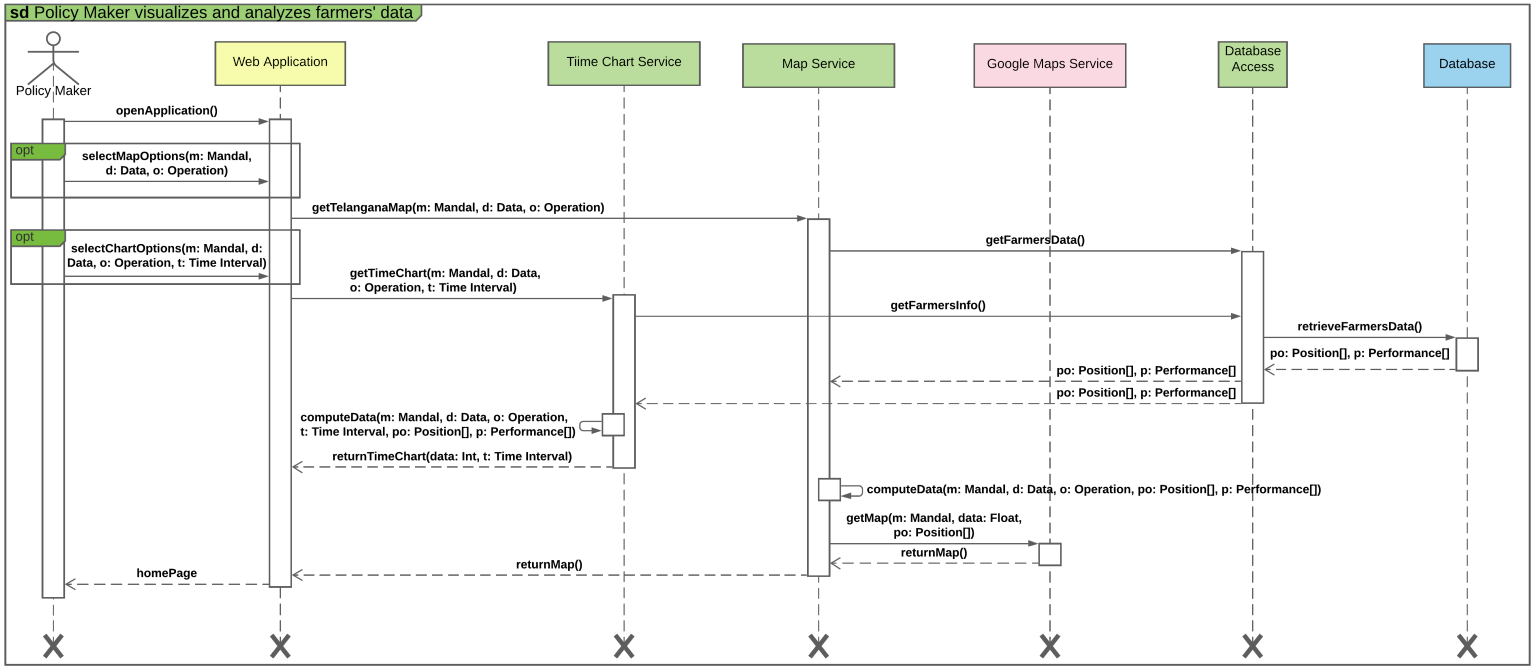


Figure 2.10: Policy Maker visualizes and analyzes farmers' data

### 2.4.6 Farmer inserts data about his/her production

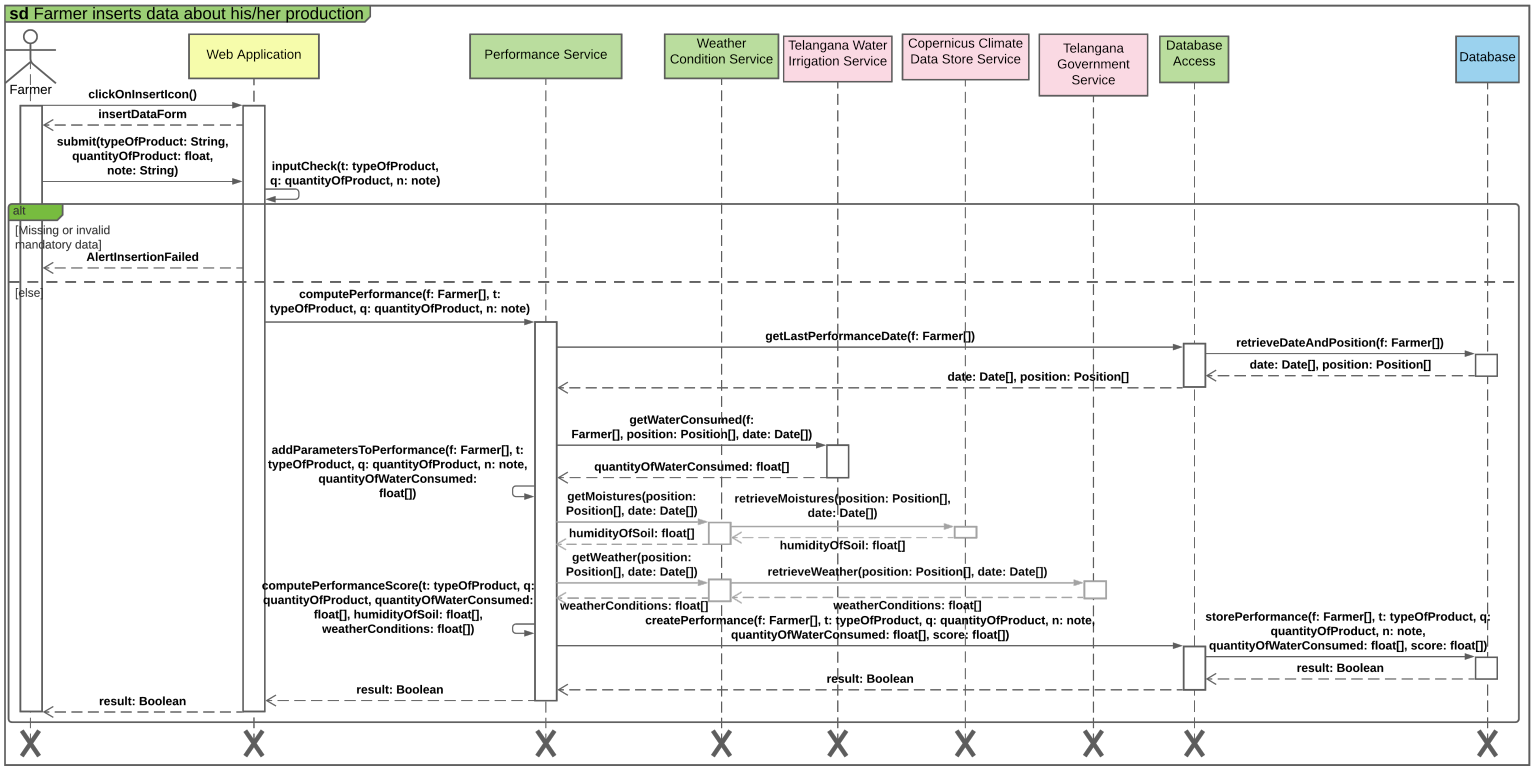


Figure 2.11: Farmer inserts data about his/her production

### 2.4.7 DREAM sends suggestions to farmer

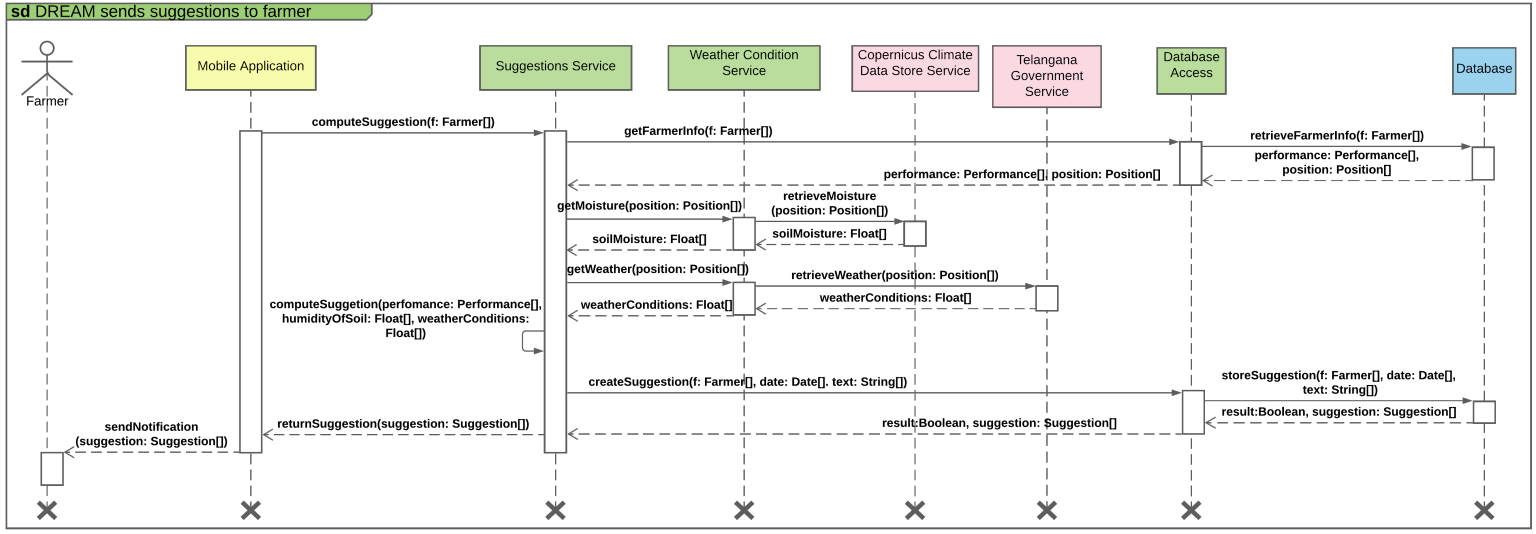


Figure 2.12: DREAM sends suggestions to farmer

### 2.4.8 Agronomist updates his/her daily plan

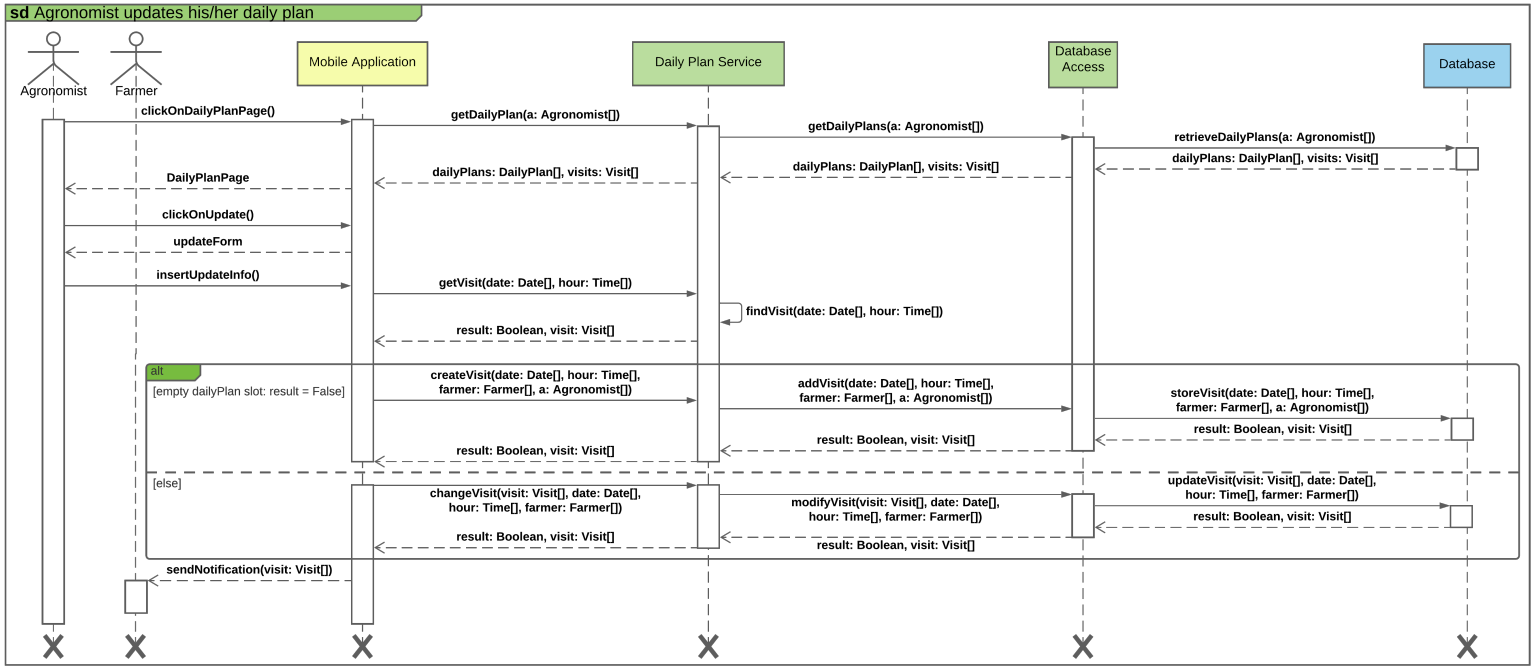


Figure 2.13: Agronomist updates his/her daily plan

### 2.4.9 Agronomist confirms his/her daily plan

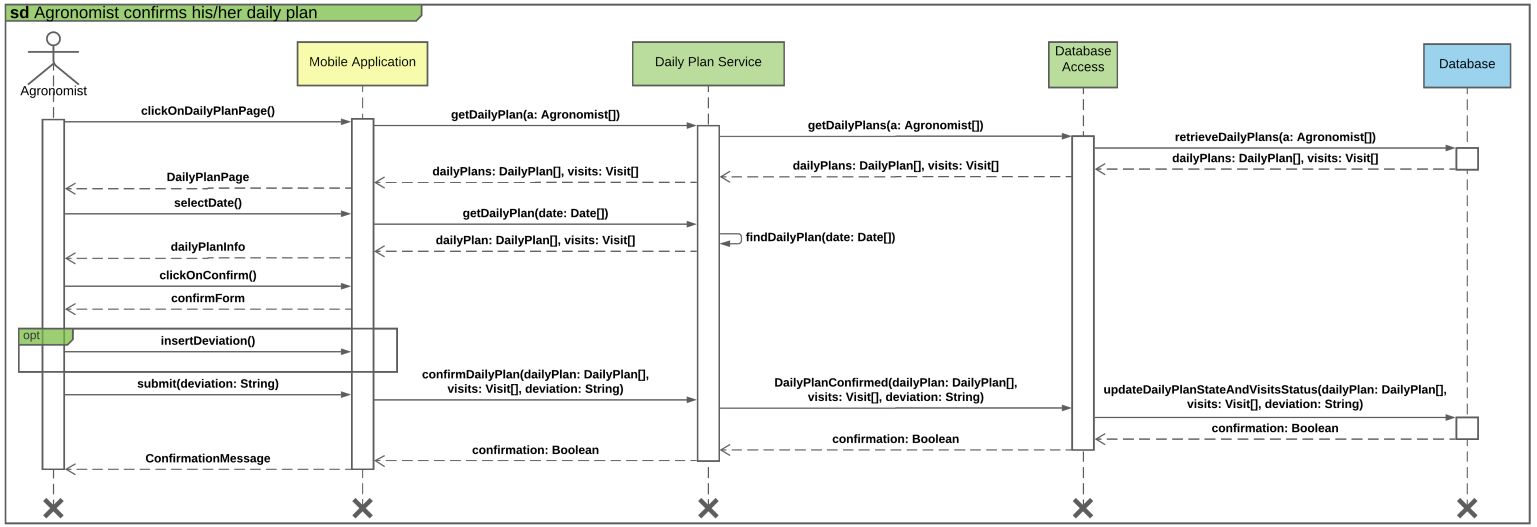


Figure 2.14: Agronomist confirms his/her daily plan

### 2.4.10 Farmer searches for a thread on the discussion forum

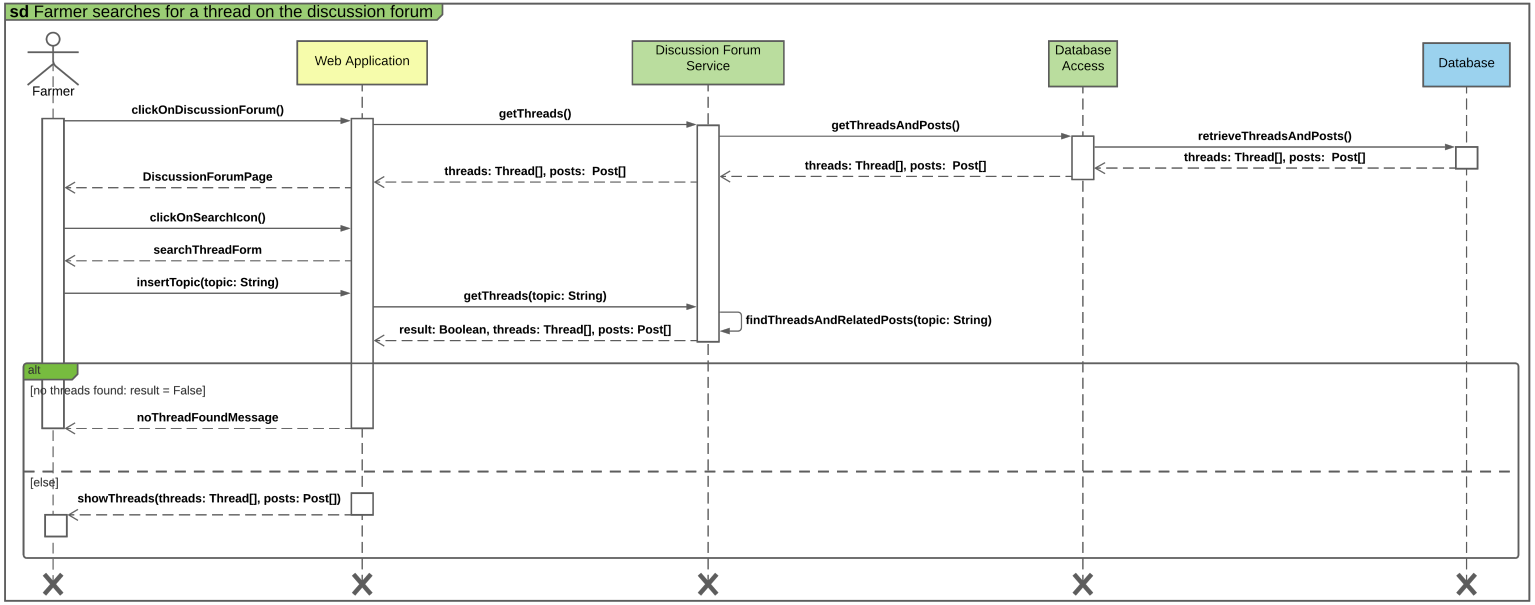


Figure 2.15: Farmer searches for a thread on the discussion forum

### 2.4.11 Farmer creates a new thread on the discussion forum

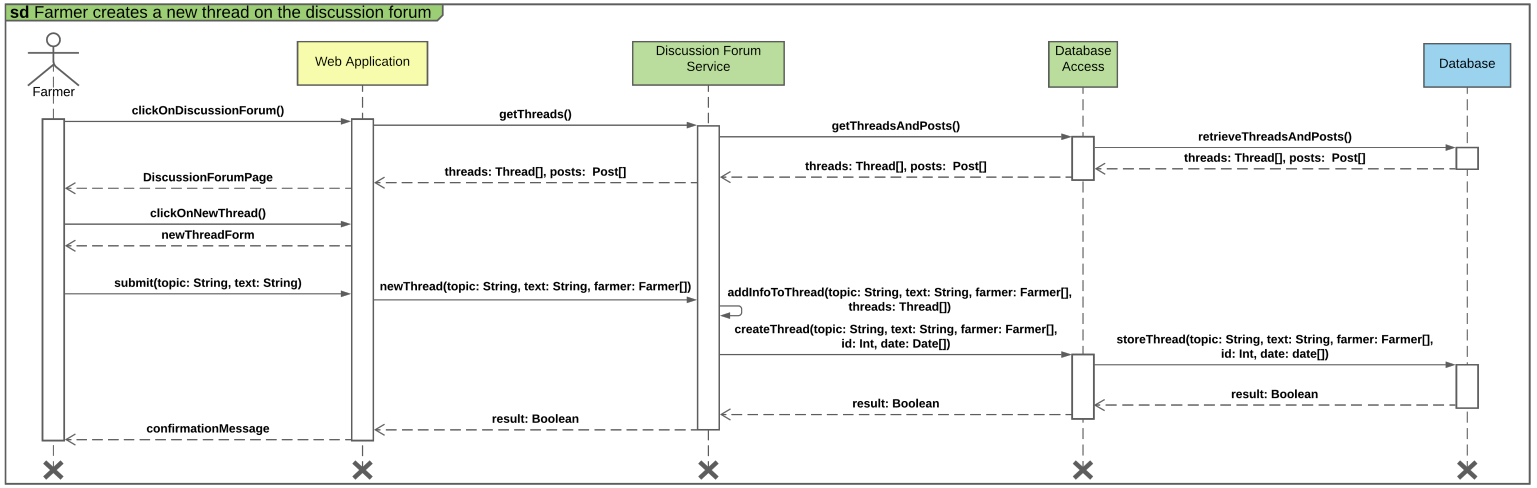


Figure 2.16: Farmer creates a new thread on the discussion forum

### 2.4.12 Farmer replies on the discussion forum

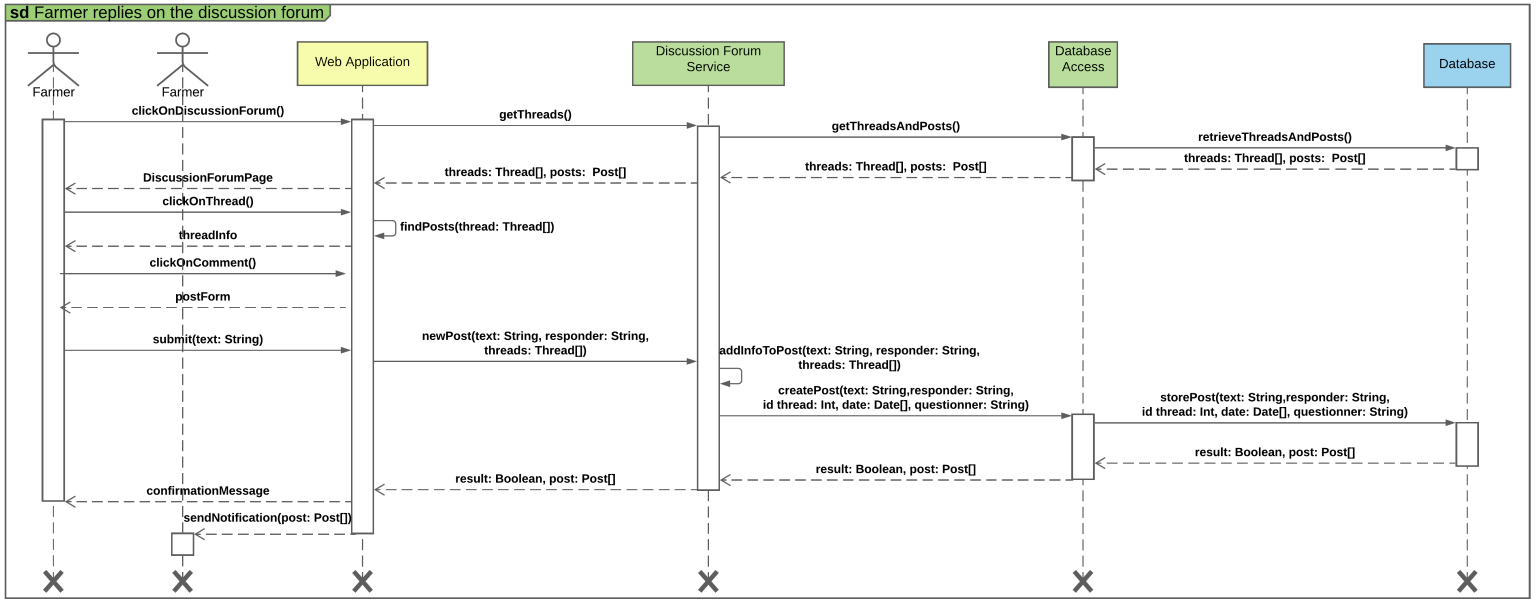


Figure 2.17: Farmer replies on the discussion forum

### 2.4.13 Farmer creates a help request

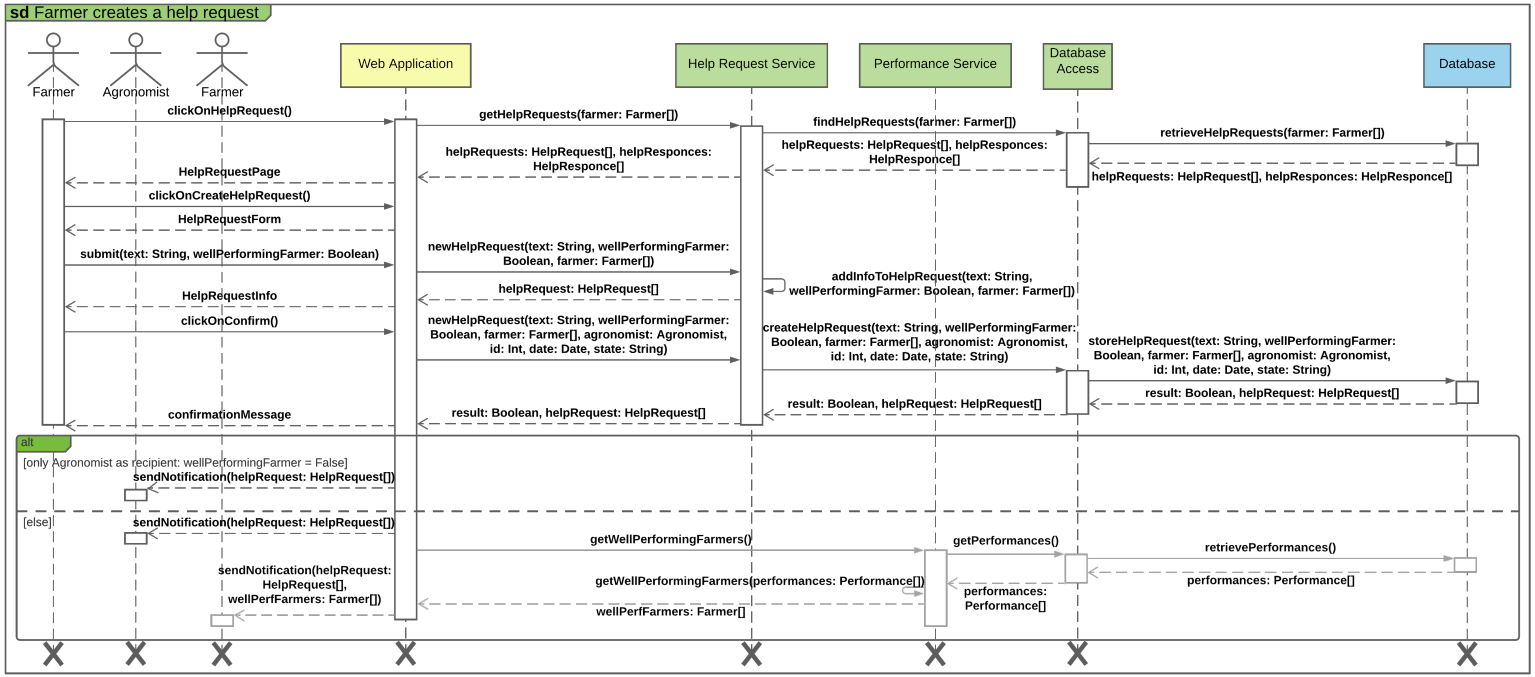


Figure 2.18: Farmer creates a help request

### 2.4.14 Agronomist replies to a help request

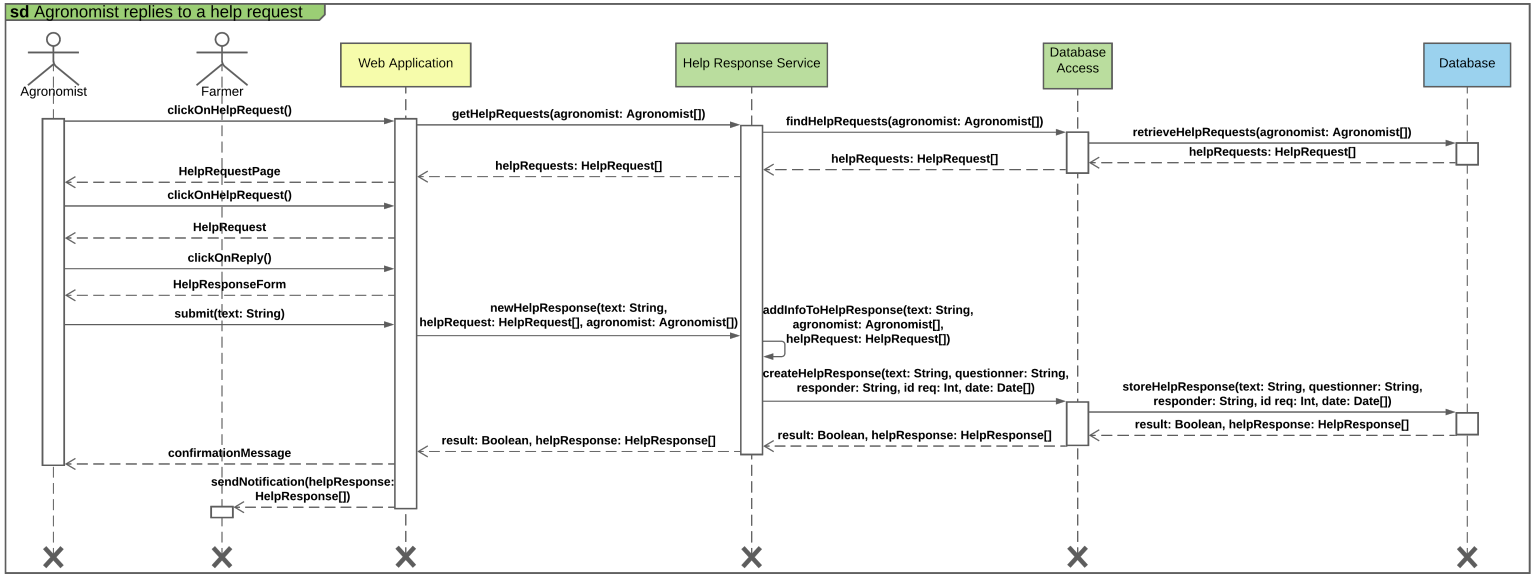


Figure 2.19: Agronomist replies to a help request

### 2.4.15 Farmer solves a help request

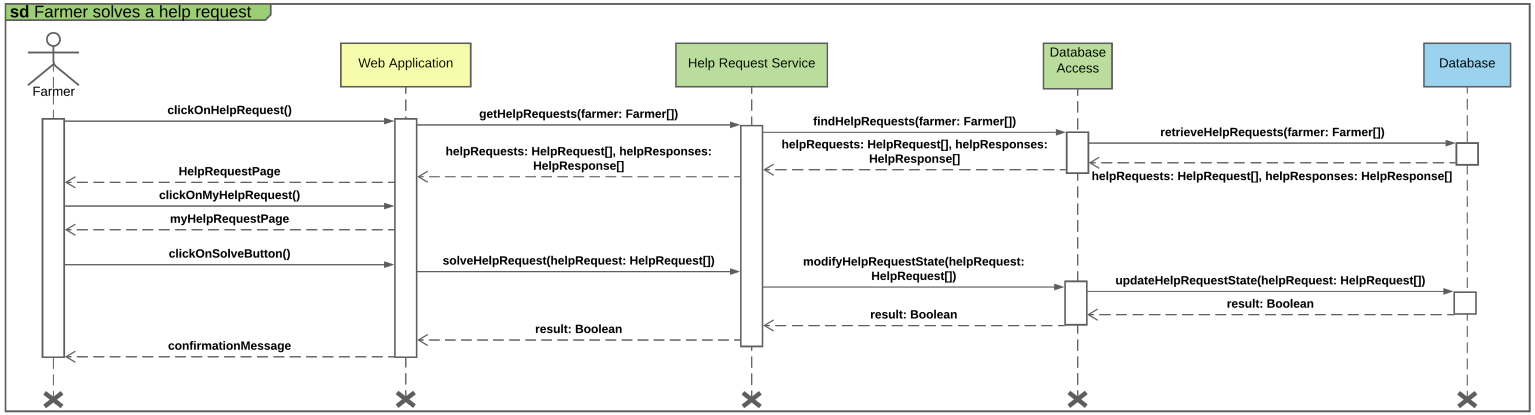


Figure 2.20: Farmer solves a help request

## 2.5 Component Interfaces

As shown in *Section 2.4*, many method are invoked on the interfaces. Their interactions guarantee the execution of the most important processes required to fulfill DREAM's goals.
The following diagram is presented to better understand how interfaces interact with each other.

The methods listed in the previous diagram are a logical representation of what component interfaces have to guarantee the users.
Certainly, considering a future implementation, they will be subjected to many modifications, according to choices and needs of developers.

## 2.6 Selected Architectural Style and Patterns

This section clarifies the architectural style and architectural pattern adopted in the design of the DREAM system. The main difference between the two is that an architectural style is the application design at the highest level of abstraction, it is a name given to a recurrent architectural design; while an architectural pattern is a way to implement an architectural style, it is a way to solve a recurring architectural problem related to it.

### 2.6.1 Architectural Style: Client-Server

In the design of the DREAM application a three tier architecture has been used. It is a client-server architecture in which the functional process logic, data access, computer data storage and user interface are developed and maintained as independent modules on three levels:

- a presentation tier;
- a logic tier;
- a data tier.

In this type of architecture there is the thin client, that is the device that requests the resource, equipped with a user interface in charge of the presentation-level functions. Then, there is the application server, also called middleware, in charge of providing the resource and communicating with the server database, which stores the data used by the application.

The main advantage of the three-tier architecture is the logical and physical separation of functionality. This allows each tier to run on a separate operating system and server platform that best suit its functional requirements.

Compared to one or two tier architecture, this allows for **faster development** as each tier can be developed simultaneously by several teams.

Furthermore, this separation of levels allows programmers to use the latest and best languages and tools for each level.

Moreover, any tier can be scaled independently of the others as needed and if one tier is interrupted, it will be less likely to impact the availability or performance of the other tiers, so this architecture also has a positive impact on **scalability** and **reliability** as well.

Since the presentation layer and the data layer cannot communicate directly, a well-designed application layer can function as a kind of internal firewall, preventing some malicious attacks and ensuring **greater security**.

### 2.6.2 Architectural Pattern: Model-View-Controller Structure

DREAM application can be structured with the Model-View-Controller (MVC) pattern. The MVC structure consists of three building blocks: model, view, and controller.

- the **Model** provides the methods to access the data useful for the application;
- the **View** displays the real user interface for the presentation of the data contained in the model and deals with the interaction with users;
- the **Controller** receives the user's commands through the view and implements them by accessing the Model and defining the corresponding View to be presented

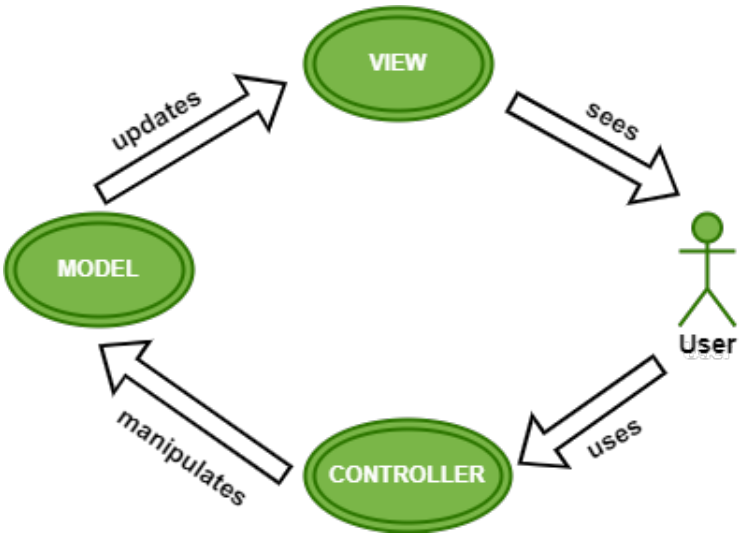


Figure 2.21: Model-View-Controller structure

This pattern was chosen primarily because application development becomes fast. In fact, it is easier for multiple developers to collaborate and work together given the nature of the architecture itself. As a result, it also becomes easier to update the application and debug as we have multiple levels properly written in the application.

## 2.7 Other Design Decisions

This section describes design decisions that have not been described in the other sections of the document in order to clarify their operation and purpose.

### 2.7.1 ID code for policy maker and agronomist from external database

DREAM application implies the interaction with two particular kinds of user: the policy maker, an institutional figure, and the agronomist, a professional figure. Agronomists and policy makers must be authorized through a personal and unique id code to be able to exploit DREAM's functionalities according to their role.

It is assumed that those unique id codes are created by Telangana Government and stored persistently in one of its databases accessible from the application only thanks to the external API TelanganaGovernmentService.

This means that DREAM has only the permission to visualize the data in the external database to check that the entered id codes are correct, but it has no permission to modify or update or delete it.

### 2.7.2 Internal database replication

The replication of the database that contains the persistent data of the system is used with the aim of improving the reliability and performance of the DREAM application.

This technique consists of copying a database instance to exactly another location. It is used in distributed database management systems environments where a single database must be used and updated in multiple locations at the same time.
A full replica of the entire database is created at each site of the distributed system. This choice improves system availability because it can continue to work properly as long as at least one site is active.

Incremental backup method is expected, with only the changes made after the last backup archived instead of backing up the whole target.

Since a single update must be performed on different databases to keep the copies consistent, the update process of the databases can be slow. However, it is necessary because the DREAM application is based precisely on the collection of data that are fundamental for the analysis of farmers' performances.

# Chapter 3

# User Interface Design

### Farmer

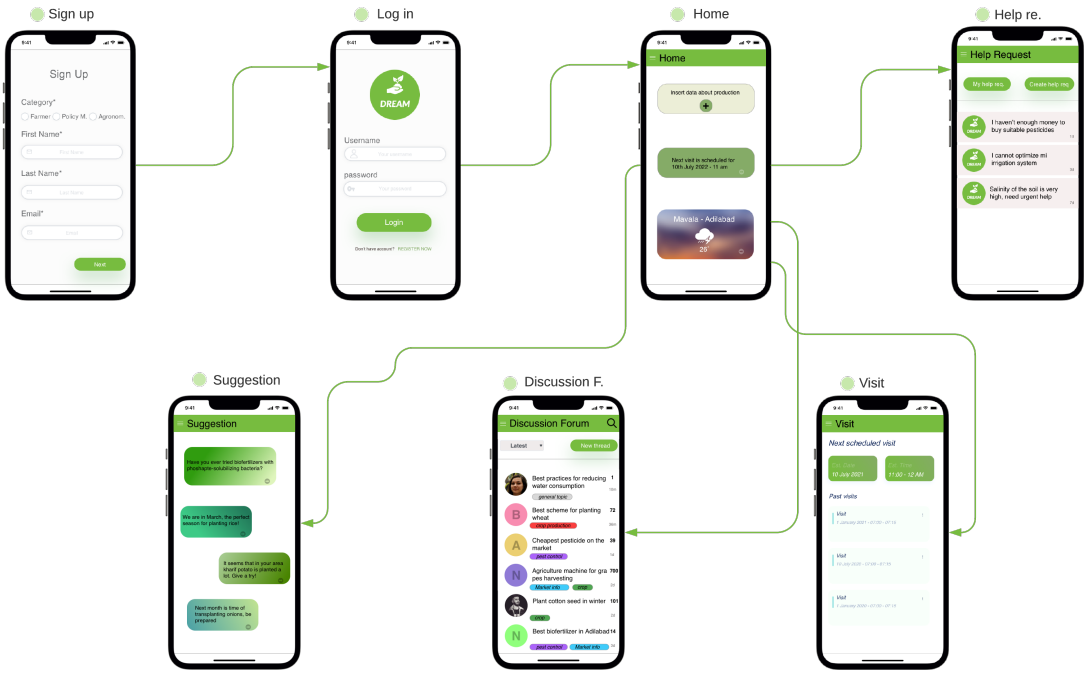


Figure 3.1: Farmer

## Agronomist

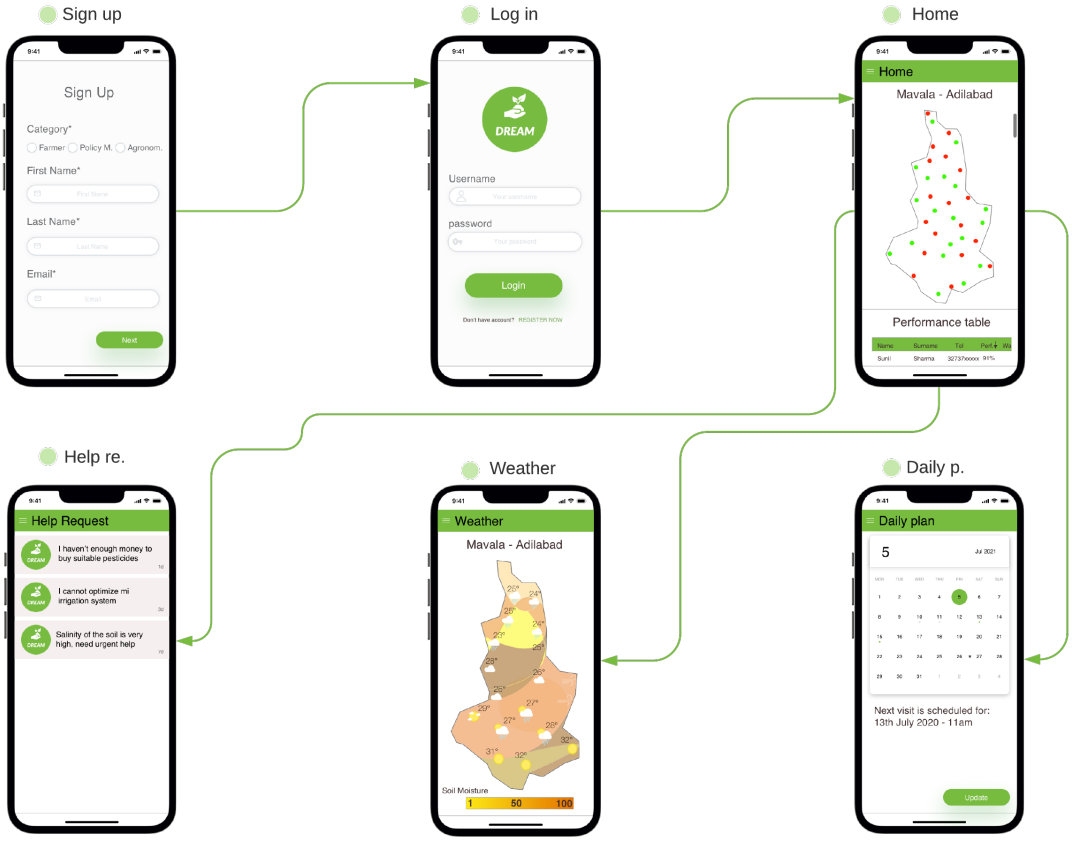


Figure 3.2: Agronomist

# Chapter 4

# Requirement Traceability

In order to check whether all requirements listed in the RASD have been considered designing DREAM application, they are mapped below with the related components described in *Section 2.2*.
Furthermore, a good mapping allows to verify if a correct modularity is reached, which allows to manage a complex software for implementation and maintenance purposes.

***R.1*** **Unregistered user must be able to register to DREAM**

- MobileApplication/WebApplication
- RegistrationService
- EmailManager
- GoogleMapsAPI
- TelanganaGovernmentService
- DatabaseAccess
- Database

***R.2*** **User must be able to login to DREAM by filling out the login form**

- MobileApplication/WebApplication
- AuthenticationService
- DatabaseAccess
- Database

***R.3*** **The system must be able to associate each farmer to the corresponding mandal according to his/her position**

- MobileApplication/WebApplication
- RegistrationService

- GoogleMapsAPI
- DatabaseAccess
- Database

***R.4*** **Farmer must be able to fill out the form for data insertion**

- MobileApplication/WebApplication

***R.5*** **The system must be able to add parameters to data inserted by farmers regarding their production**

- MobileApplication/WebApplication
- PerformanceService
- WeatherConditionService
- TelanganaWaterIrrigationService
- CopernicusClimateDataStoreService
- TelanganaGovernmentService
- DatabaseAccess
- Database

***R.6*** **The system must be able to return whether a data insertion has been done correctly by the user or not**

- MobileApplication/WebApplication

***R.7*** **The system must be able to classify farmers into bad and well performing farmers according to their performance score value**

- PerformanceService
- DatabaseAccess
- Database

***R.8*** **The system must be able to show the map with performance score regarding farmers**

- MobileApplication/WebApplication
- MapService
- MandalMapService
- GoogleMapsAPI
- DatabaseAccess
- Database

***R.9*** **The policy maker must be able to select options which he/she wants to update the map with**

- MobileApplication/WebApplication

***R.10*** **The system must be able to show the updated map according to the option chosen by policy maker.**

- MobileApplication/WebApplication
- MapService
- GoogleMapsAPI
- DatabaseAccess
- Database

***R.11*** **The policy maker must be able to select options for the time chart**

- MobileApplication/WebApplication

***R.12*** **The system must be able to show the time chart according to the option chosen by policy maker.**

- MobileApplication/WebApplication
- TimeChartService
- DatabaseAccess
- Database

***R.13*** **The system must be able to compute data according to the chosen operation**

- TimeChartService
- MapService

***R.14*** **The system must be able to show a table with farmers' data in descending order based on their performance score**

- MobileApplication/WebApplication
- TableService
- DatabaseAccess
- Database

***R.15*** **The system must be able to show the map of mandal's agronomist with weather conditions and soil moisture of the current date**

- MobileApplication/WebApplication

- WeatherForecastsService
- GoogleMapsAPI
- TelanganaGovernmentService
- CopernicusClimateDataStoreService

***R.16*** **The agronomist must be able to visualize weather forecasts up to seven days after the current date**

- MobileApplication/WebApplication
- WeatherForecastsService
- GoogleMapsAPI
- TelanganaGovernmentService
- CopernicusClimateDataStoreService

***R.17*** **The system must be able to show the daily plan**

- MobileApplication/WebApplication
- DailyPlanService
- DatabaseAccess
- Database

***R.18*** **The agronomist must be able to update the daily plan**

- MobileApplication/WebApplication
- DailyPlanService
- DatabaseAccess
- Database

***R.19*** **The agronomist must be able to confirm the daily plan**

- MobileApplication/WebApplication
- DailyPlanService
- DatabaseAccess
- Database

***R.20*** **The system must be able to show weather conditions and soil moisture regarding farmer's position**

- MobileApplication/WebApplication
- WeatherConditionService
- TelanganaGovernmentService
- CopernicusClimateDataStoreService

- DatabaseAccess
- Database

***R.21*** **The farmer must be able to visualize all visits scheduled for him/her**

- MobileApplication/WebApplication
- VisitService
- DatabaseAccess
- Database

***R.22*** **The system must be able to send notifications to the farmer/agronomist**

- MobileApplication/WebApplication

***R.23*** **The farmer and the agronomist must be able to visualize notifications' details**

- MobileApplication/WebApplication

***R.24*** **The system must be able to compute suggestions according to the information inserted by the farmer and his/her position**

- MobileApplication/WebApplication
- SuggestionsService
- WeatherConditionService
- TelanganaGovernmentService
- CopernicusClimateDataStoreService
- DatabaseAccess
- Database

***R.25*** **The farmer must be able to request for help**

- MobileApplication/WebApplication
- HelpRequestService
- PerformanceService
- DatabaseAccess
- Database

***R.26*** **The system must be able to show to well performing farmers a list of the help requests created and not yet solved with them as recipient**

- MobileApplication/WebApplication

- HelpRequestService
- DatabaseAccess
- Database

***R.27*** **The well performing farmer must be able to select a help request to read its details**

- MobileApplication/WebApplication

***R.28*** **The system must be able to show the details of the help request selected by the well performing farmers**

- MobileApplication/WebApplication

***R.29*** **The well performing farmer must be able to answer to a help request**

- MobileApplication/WebApplication
- HelpRequestService
- DatabaseAccess
- Database

***R.30*** **The system must be able to notify the agronomist when a new help request is created and not yet solved**

- MobileApplication/WebApplication

***R.31*** **The agronomist must be able to visualize notification's details**

- MobileApplication/WebApplication

***R.32*** **The system must be able to show to the agronomist a list of the help requests created and not yet solved with him/her as recipient.**

- MobileApplication/WebApplication
- HelpResponseService
- DatabaseAccess
- Database

***R.33*** **The agronomist must be able to visualize help requests' details**

- MobileApplication/WebApplication

***R.34*** **The agronomist must be able to answer to a help request**

- MobileApplication/WebApplication
- HelpResponseService
- DatabaseAccess
- Database

***R.35*** **The farmer must be able to solve a help request**

- MobileApplication/WebApplication
- HelpResponseService
- DatabaseAccess
- Database

***R.36*** **The system must be able to show a list of existing threads only visualizing the associated topic**

- MobileApplication/WebApplication
- DiscussionForumService
- DatabaseAccess
- Database

***R.37*** **The farmer must be able to visualize a certain thread with related posts**

- MobileApplication/WebApplication

***R.38*** **The farmer must be able to search for a topic on the discussion forum**

- MobileApplication/WebApplication

***R.39*** **The farmer must be able to open a thread on the discussion forum**

- MobileApplication/WebApplication
- DiscussionForumService
- DatabaseAccess
- Database

***R.40*** **The farmer must be able to answer on the discussion forum**

- MobileApplication/WebApplication
- DiscussionForumService
- DatabaseAccess
- Database

# Chapter 5

# Implementation, Integration and Test Plan

In this chapter it is illustrated the approach that will be adopted, in particular in section 5.1 and 5.2 the Implementation Plan and Integration Strategy are described, while in section 5.3 it is presented the Testing Plan.

## 5.1 Implementation Plan

In order to choose which component will be implemented first, there are two main approaches:

- Bottom-Up
- Top-Down

The approach that has been chosen for this task, and also for the Integration and Testing, it the first one, Bottom-up. This approach permits to performs testing and integration in parallel with the implementation in a quite easy way: when a component is implemented, programmers perform Unit testing immediately, and as the system grows through the Integration testing it is checked that the different modules interacts correctly. When the entire system is completed, using this incremental process, System testing is performed, that will ensure that every requirement has been satisfied. The main disadvantage of this strategy is that it is not possible to release a "early version" of the product, however this issue is compensated by a higher execution speed that the Bottom-Up approach ensures. In this kind of approach, in order to establish which component will be implemented first, it's important to exploit dependencies between components: if a component rely on another one, necessarily the latter must be integrated before the former. Adopting the method described above, we start by choosing components which are completely independent from the other ones, or that, as in this case, only

rely on external components. In parallel this two components could be implemented: **Database** (with it's subcomponent **DBMS**) and **ExternalAPIs** (with it's subcomponents **GoogleMapsAPI**, **CopernicusClimateDataStoreService**, **TelanganaGovernmentService**, **TelanganaWaterIrrigationService**). Then, the next component that should be implemented is **DatabaseAccess**, as stated in previous chapters it is responsible to get data from a data source, in our case with **Database**. Going on, **AuthenticationService** and **RegistrationService** could be implemented in parallel, since they manages login and registration to DREAM. Finally high level components, that belongs to the three main actors of the application should be implemented in this order: **FarmerService** (with it's subcomponents **HelpRequestService**, **DiscussionForumService**, **PerformanceService**, **VisitService**, **WeatherConditionService**, **SuggestionsService**, **NotificationFarmerService**), then **AgronomistService** (with it's subcomponents **HelpResponseService**, **MandalMapService**, **TableService**, **DailyPlanService**, **WeatherForecastsService**, **NotificationAgronomistService**) and finally **PolicyMakerService** (with it's subcomponents **TimeChartService** and **MapService**). In conclusion, to sum up, the components of DREAM will be implemented in the following order:

1. Database and ExternalAPIs
2. DatabaseAccess
3. AuthenticationService and RegistrationService
4. FarmerService (and it's subcomponents)
5. AgronomistService (and it's subcomponents)
6. PolicyMakerService (and it's subcomponents)

## 5.2 Integration Strategy

This section describes the integration plan for the system to be developed. As stated in the previous section, each component must first undergo *Unit Testing*. The aim of this section is to verify the interactions between the modules of DREAM, it is also exploited the system's coexistence with others and tests the interface between modules, we firstly test components individually *Unit Testing* and then combined to make a system *System testing*. Every time that it is needed to test a component which is used by other ones that have not been integrated yet, an artificial environment is necessary for each integration test; the environment consists of driver programs and test data. It is also important to point out that in our system there are some external components, such as: **Database** and **ExternalAPIs**, and so they are directly exploited without having to implement them, however they're

integrated as soon as possible in order to test whether their interfaces can communicated properly. The following diagrams illustrate the integration process at the various levels of dependency. (Note that it is not possible to apply a pure bottom-up approach due to the mutual dependency relationship between *ExternalAPIs* and some high level components).

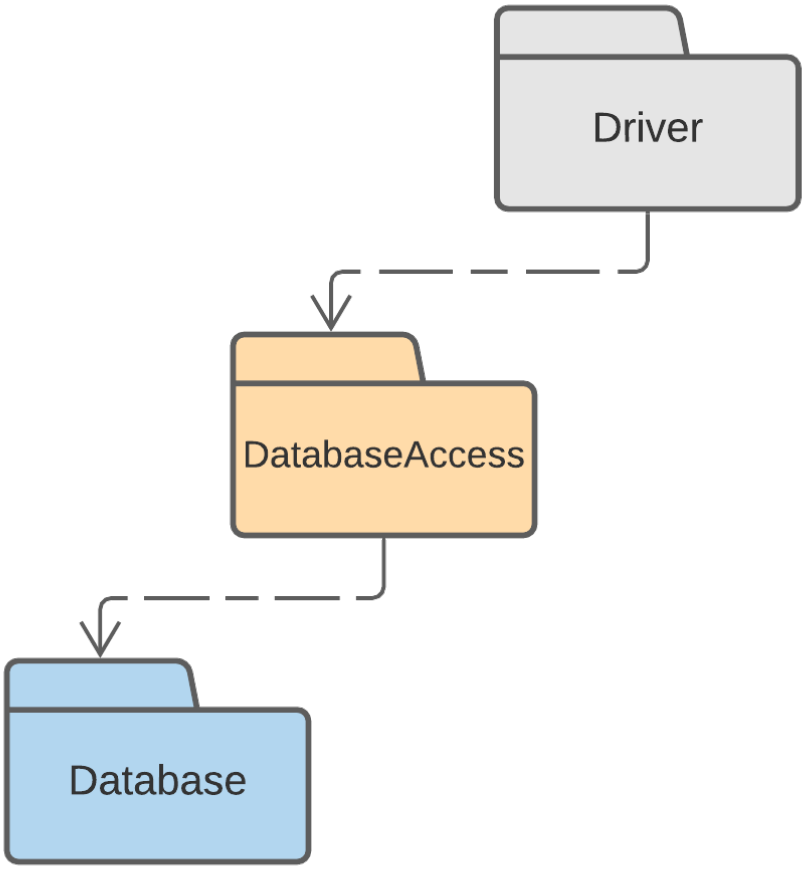


Figure 5.1: DB

Figure 5.1 shows the integration of Database and DatabaseAccess components.

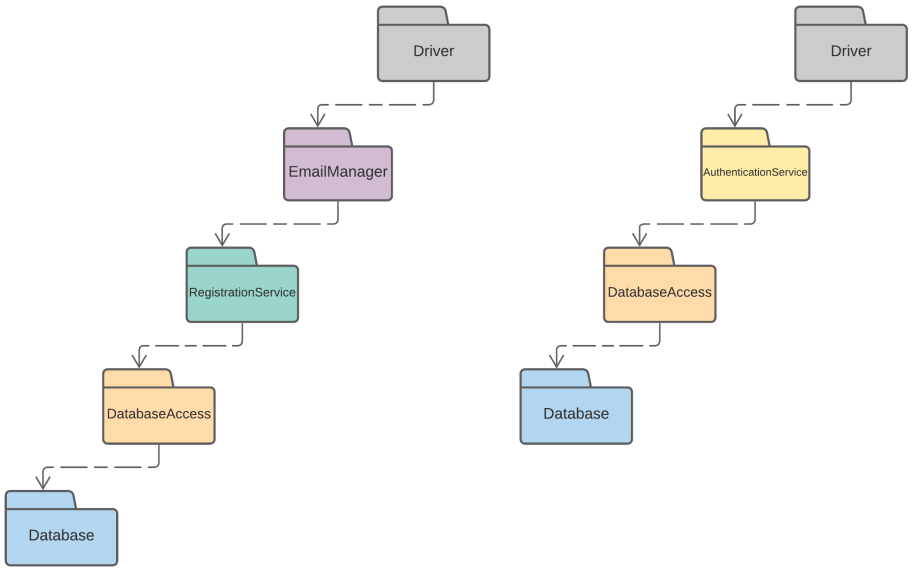


Figure 5.2: Login and Reg.

After that, login and registration services are integrated, by adding **RegistrationService** and **AuthenticationService**.

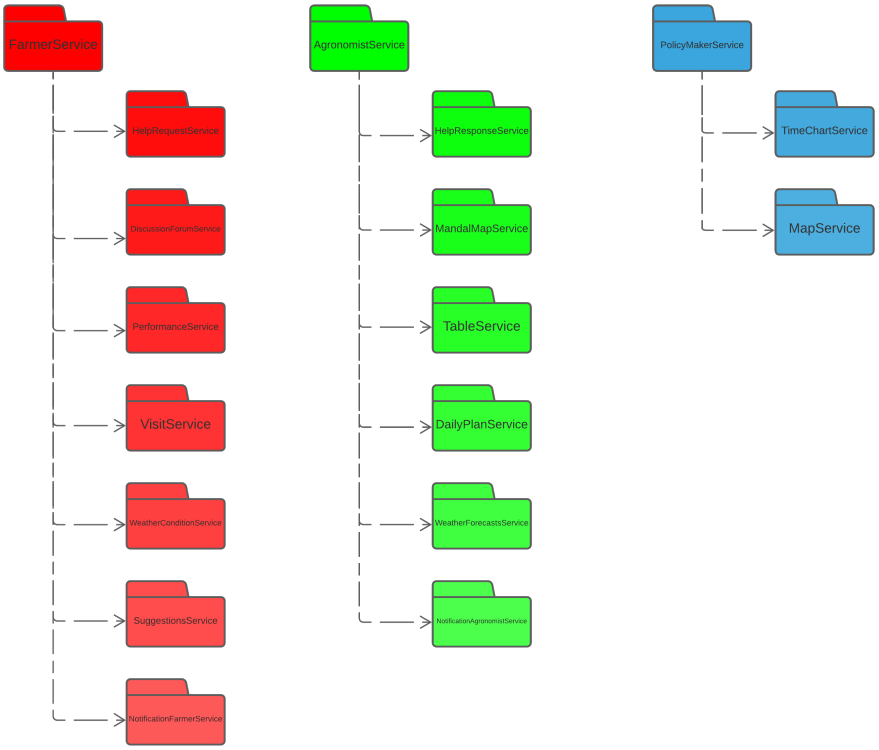


Figure 5.3: Three actor services

Figure 5.3 shows the integration of components for each of the three actors of DREAMS.

And Finally it's possible to put together all these components and perform a *System Testing*.

## 5.3 Testing Plan

Once all components have been integrated, programmers could perform *System Testing*. The aim of this kind of testing if to verify functional and non-functional requirements. Specifically, *DREAM* will be subjct of the following tests:

- **Functional Testing:** It ensures that the requirements and the specifications defined in the RASD are properly satisfied by the System.
- **Performance Testing:**The main purpose is to identify and eliminate the performance bottlenecks in the software application affecting response time, utilization, throughput.
- **Load Testing:** It aims at detecting bugs such as memory leaks, mismanagement of memory and buffer overflows. It also identifies the maximum operating capacity of the application.
- **Stress Testing:**It verifies stability and reliability of software application. The goal is measuring software on its robustness and error handling capabilities under extremely heavy load conditions and ensuring that software doesn't crash under crunch situations. It even tests beyond normal operating points and evaluates how software works under extreme conditions.

# Chapter 6

# Effort Spent

### 6.0.1 Ali Arslan

Table 6.1: Effort spent - Ali Arslan

| Task | Hours |
|---|---|
| Introduction | |
| User Interface Design | |
| Implementation Plan | |
| Integration Strategy | |
| Testing Plan | |
| Document revision | |

### 6.0.2 Servidio Elisa

Table 6.2: Effort spent - Servidio Elisa

| Task | Hours |
|---|---|
| Overview | 1.30 |

| | |
|---|---|
| Runtime View | 8.00 |
| Component Interfaces | 2.00 |
| Other Design Decisions | 0.20 |
| Requirement Traceability | 1.30 |
| Document revision | |

### 6.0.3 Suriano Federica

Table 6.3: Effort spent - Suriano Federica

| **Task** | **Hours** |
|---|---|
| Component View | 5.00 |
| Deployment View | 5.00 |
| Architectural Style and Patterns | 1.00 |
| Other Design Decisions | 0.20 |
| Requirement Traceability | 1.30 |
| Document revision | |