POLITECNICO
MILANO 1863

M.Sc. Computer Science and Engineering
Software Engineering 2 Project

# DREAM – Data dRiven PrEdictive FArMing in Telangana

# Design Document


Censuales Simone, Giovia Giuseppe, Meli Giuseppe

9th January 2022

Version 1.0

GitHub Repository - https://github.com/Censu08/CensualesGioviaMeli

## Contents

# Chapter 1

# Introduction

## 1. Purpose

The main purpose of the Design Document (DD) is to provide a technical and more specialized analysis of the *Software-to-be*, with the description of the principal architectural components, including the communication interfaces between parties, as well as their interaction.

This document presents a guide toward the implementation process by means of developing steps and integration of all the parts intended to take place in the overall procedure. Furthermore, it represents an important support for the application developers.

## 2. Scope

DREAM objective is the development of management model, which is able to predict the food requirement of the entire nation and build a resilient food system, capable of exploiting digital goods, adopting new technological methodologies and techniques, monitoring the productions of every farmer located in the region.

In doing so, the System allows Policy Makers to identify farmer performance and steering initiative outcomes, carried out by agronomist. Moreover, DREAM Application let every agronomist manage his/her personal daily plan and keep under control the situation of farmer's productions. Finally, the System allows farmers to send help requests and exchange information with other farmers.

## 3. Definitions, Acronyms, Abbreviations

### 3.1 Definitions

- **Telangana's Policy Makers:** a person who has access to the DREAM Application data and who manages farmers and interacts with agronomists
- **Farmers:** a person who has access to the DREAM Application data and is able to manage data about his production, create and interact with discussion, send help request to agronomist and other farmers, participate to steering initiative carried out by agronomist
- **Agronomists:** a person who is able to access DREAM Application data, answer help request, create steering initiative and visit farmer according to his/her daily plan
- **DREAM System:** refers to the whole system to be developed
- **DREAM Services:** refers to the functionalities offered by the DREAM System
- **DREAM Application:** refers to the application that makes DREAM Services available everywhere. In this document, this term is intentionally used in a generic way
- **External Services:** refers to weather forecast services and geo-localization services that the System accesses from third parties
- **Sensors:** refers to specific devices located in the ground, sending data about soil humidity and water irrigation to the System

### 3.2 Acronyms

- **RASD:** Requirement Analysis and Specification Document
- **DD:** Design Document
- **UML:** Universal Modelling Language, it is a way of designing application based on its graphic class representation
- **GPS:** Global Positioning System
- **ER**: Entity – Relationship

### 3.3 Abbreviations

- **[R.n]**: Requirement number #n
- **[C.n]**: Constraint number #n

# 4. Revision history

| Version | Date | Authors | Summary |
|---|---|---|---|
| 1.0 | 09/01/2022 | Censuales Simone<br>Giovia Giuseppe<br>Meli Giuseppe | First release |

# 5. Reference Documents

- Specification document: Assignment RDD A.Y. 2021/2022.pdf
- Software engineering 2 course slide
- Previous project example (given by professors):
    - Specification document: Project assignment A.Y. 2020/2021.pdf
    - DD to be analyzed.pdf
- IEEE Standard on requirement engineering (ISO/IEC/IEEE 29148)

# 6. Document Structure

The document contains the following sections:

- **Introduction:** A brief yet exhaustive introduction to the main purposes and scope of the software-to-be
- **Architectural Design:** A static and dynamic representation of the high-level components, including their interactions, supported by the use of diagrams
- **User Interface Design**: A general overview of how the User Interface(s) of the System will look like
- **Requirements Traceability**: An explanatory section in which the requirements defined in RASD map to the design elements pointed in this document
- **Implementation, Integration and Test Plan**: Identification of the order in which the subcomponents are intended to be implemented, and further tested, in the Software-to-be
- **Effort spent:** section that shows the total amount of effort spent by each member of the group on every topic of the document
- **References:** references to all the document used as a model for the realization of the project.

# Chapter 2

# Architectural Design

## 1. Overview

The figure 1 shows how the main components, that generate the System, interact with each other through a high-level representation.

The architecture is organized in 4 tiers, in order to promote scalability of components, reparability and maintainability.

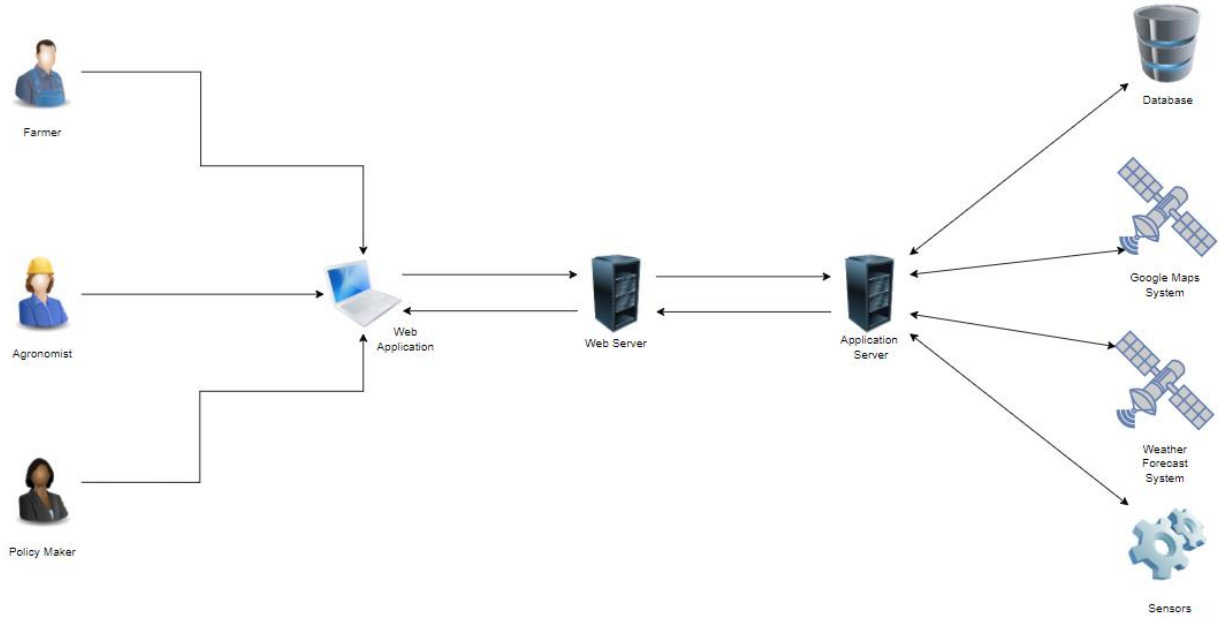


*Figure 1: 4-tier System Architecture*

A description of each component of the System Architecture is now provided:

- **Web Application**
  The web application through which every type of user (Farmer, Agronomist and Policy Maker) can access DREAM Services, using any of the most recent internet browser. The communication with the services offered by the System is made possible through the exchange of requests with the Web Server component
- **Web Server**
  It represents one of the two backend component of the architecture and allows the interaction between the web application, accessible by each user, and the application server
- **Application Server**
  It is the second, and principal, backend component of the architecture and it has the job to elaborate the requests arriving from the web application side, also managing the interaction with the Data Tier and the web server. It is even responsible for the interaction with the external service systems

- **Database**
  This is the component in which all the data are stored. Data are made available through the interaction with the application server
- **Google Maps System**
  It is the component responsible for geolocating the various farmer inside the Telengana region, so that agronomists know which farmer to visit
- **Weather Forecast System**
  It is the component that allows DREAM application to show weather forecast information to Farmers and Agronomists
- **Sensors**
  They are devices located inside the ground of farm of each farmer and allows the collection of specific parameters measured, through the communication with the application server.

# 2. Component View

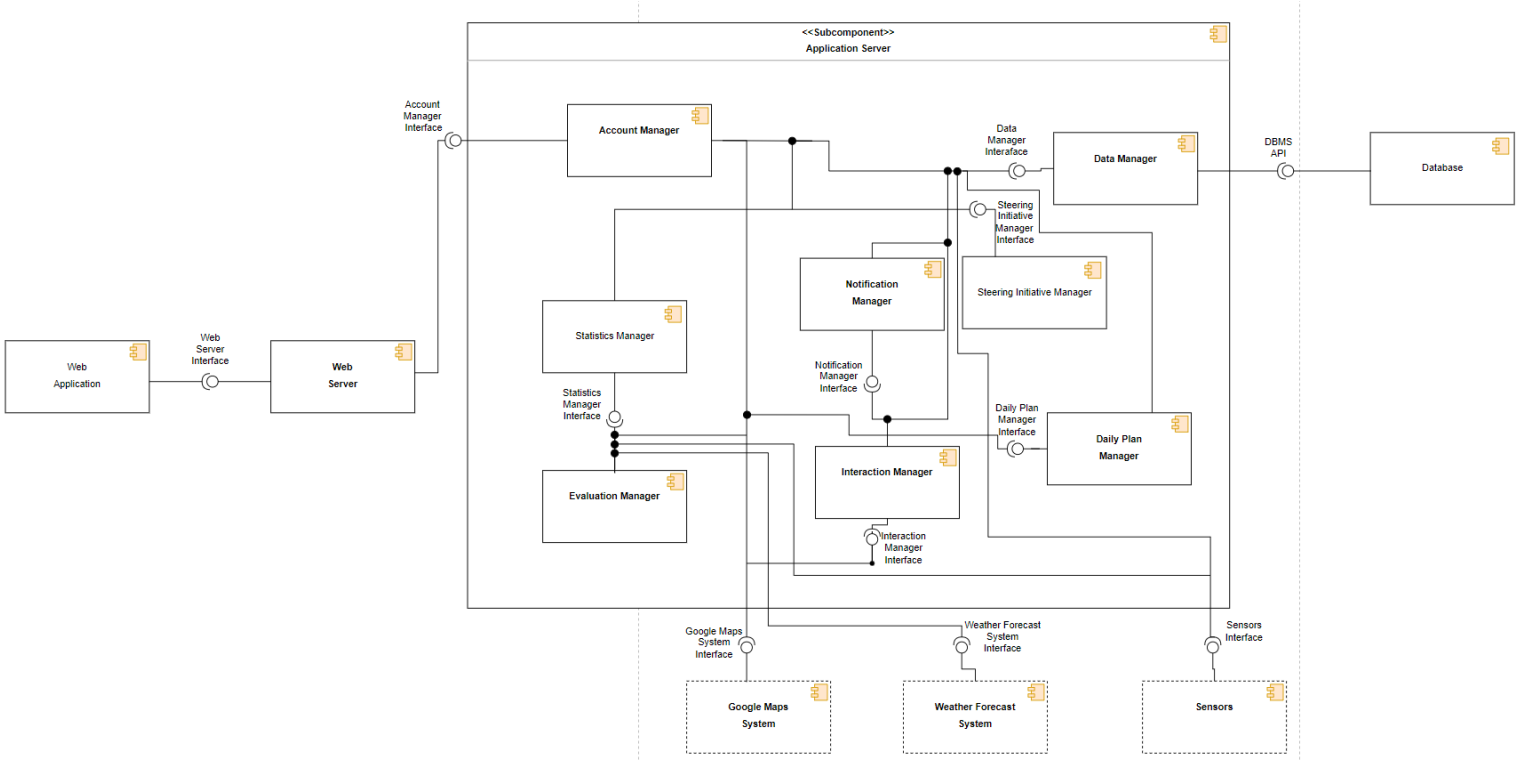


*Figure 2: Component View DREAM Application*

## 2.1 Application Component

The Application Component is constituted by every component which allows a high-level interaction between Users and System or rather the front-end of the System. So, with graphical interfaces, Users can interact with DREAM Application. The Application Component is always connected to the Application Server dedicated to all the computational tasks; the only logic incorporated in the Application Component is reduced to basic checks. The Application Component consists of the following component:

- **Web Application**
  It is the application component dedicated to Users that allows them to keep access data, functionalities under control and, if possible, interact with the data Application

## 2.2 Web Server Component

The Web Application is assured to the Application Server thanks to the Web Server Component. It receives some requests (through HTTP) from the User's browser and forwards them to the Application Server to store all necessary data required to generate the dynamic web pages.

## 2.3 Application Server Component

- **Daily Plan Manager**
  It handles all operation related to Agronomists' daily plan. It communicates with the Account Manager and with Data Manger to acquire information to generate a specific Agronomist's daily plan. Furthermore, this component is responsible for daily plan creation and for daily plan updating.
- **Account Manager**
  This component manages all the operations concerning the Registration of an account, the Login operation, the Update of data and the account Deletion. It interacts with the Data Manager component to verify the correctness of all the operations mentioned above, with the Google Maps System to retrieve localization information and with Interaction Manager to allow user interact with each other.

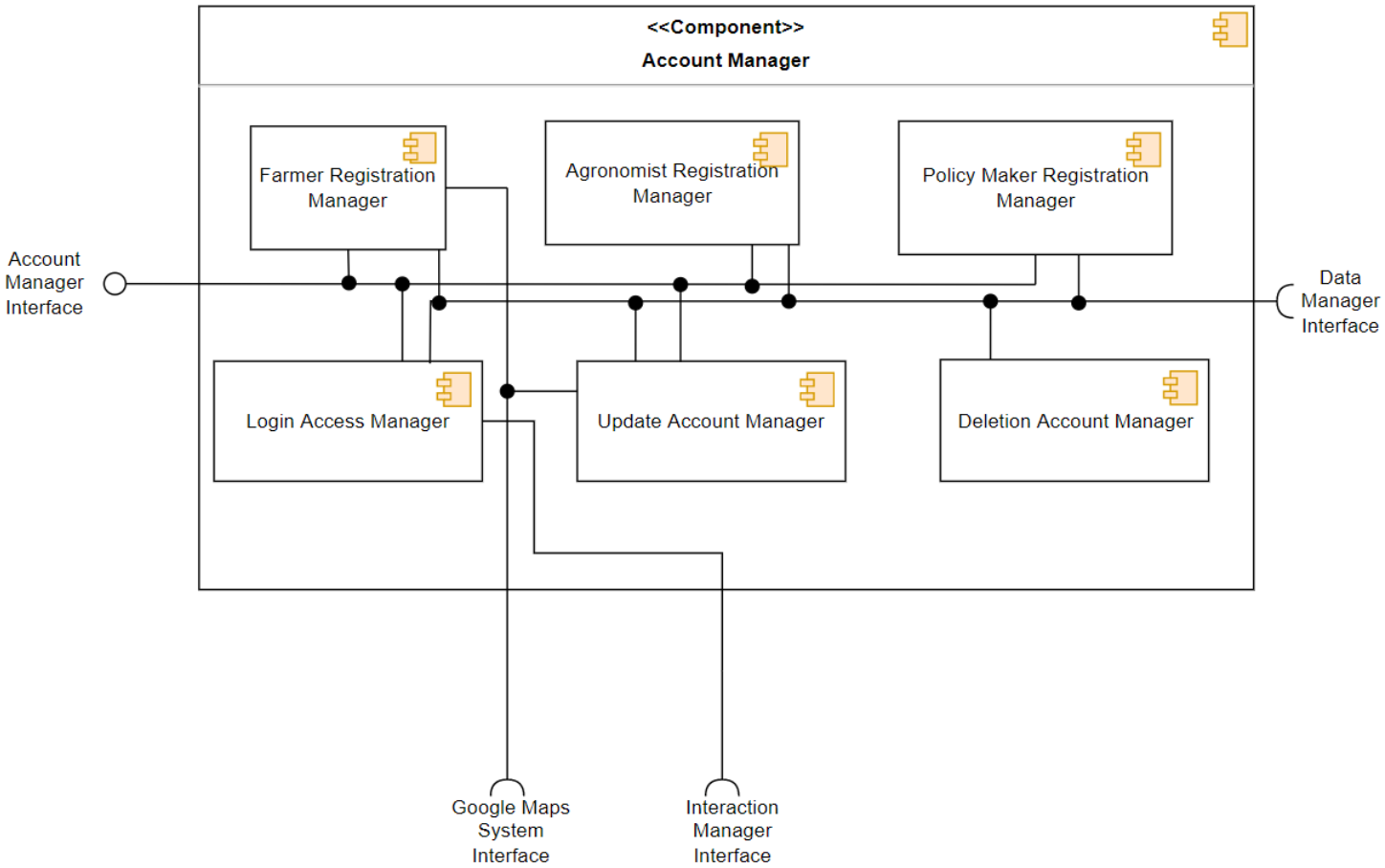


*Figure 3: Account Manager Diagram*

It is composed by the following sub-components:

- **Farmer Registration Manager**: it manages the registration of a new Farmer and, more specifically, interacts with the *Google Maps System* in order to take the position of the farm location, associating it with a sub-region inside Telengana
- **Agronomist Registration Manager**: it manages the registration of a new Agronomist

- **Policy Maker Registration Manager**: it manages the registration of a new Policy Maker
- **Login Access Manager**: it is the sub-component that checks the credentials to assess the authentication
- **Update Account Manager**: it is the sub-component that handles the data updates of all the accounts
- **Deletion Account Manager**: it is the sub-component that handles the deletion of an account

- **Steering Initiative Manager**
  This is the component responsible for the management of a steering initiative. Its interface is directly accessible through the web application of every type of user. In order to store and retrieve data related to farmer production, it interacts with the Data Manager component. Finally, it communicates with both Interaction Manager component, to let Farmer insert data about initial and final production, and Evaluation Manager, which allows Policy Maker to evaluate the steering initiative outcome.

- **Statistics Manager**
  This component is responsible for generating the performance index for every Farmer. Its interface is accessible by the evaluation manager to retrieve the performance index. It communicates with the Data Manager, the Weather Forecast System, the Sensors and the Evaluation Manager.

- **Data Manager**
  This is the component responsible for the interaction with the Data layer of the architecture. It manages and forwards the request coming from previous components to access and delete data stored in the database.

- **Evaluation Manager**
  It is responsible for computing the performance index and associate it to the farmer, as well as empower the Policy Maker to report the farmers as best or worst performing. It communicates with the Data Manager to retrieve the production information relative to every Farmer

- **Interaction Manager**
  It handles all the operations related to the communications between Users. It communicates with the Data Manager to access, store, and delete messages and conversations. It communicates with the Notification manager too, used to notify agronomists about new help requests.

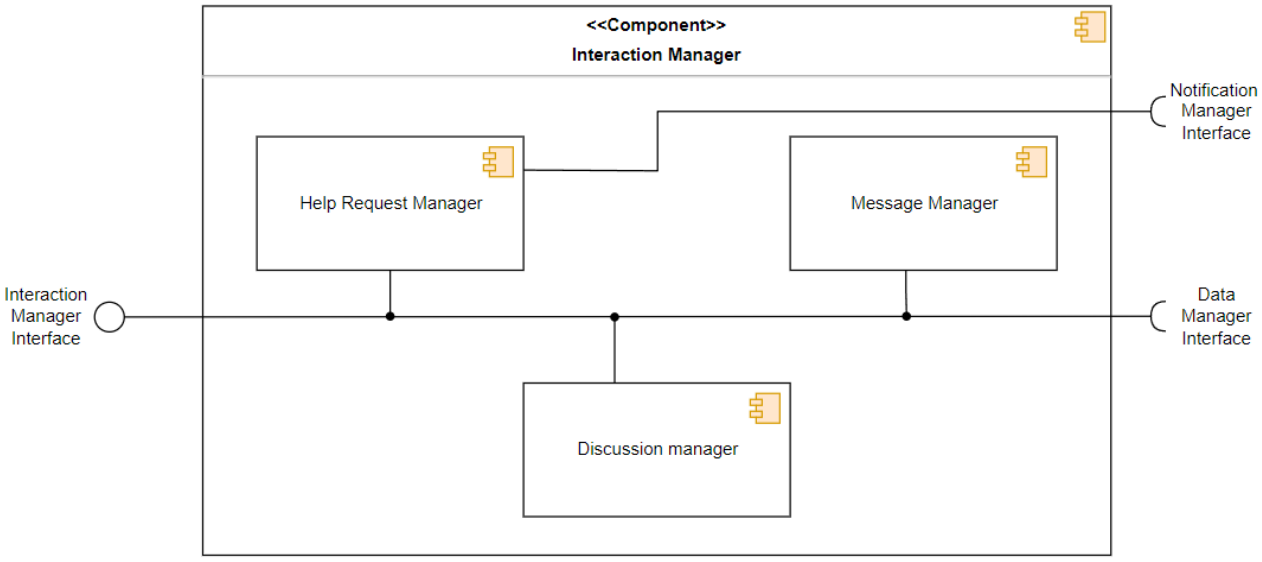


*Figure 4: Interaction Manager*

- **Help Request Manager**
  It manages all the operations related to the help requests.
- **Message Manager**
  It manages all the messages exchanged between Farmers and Agronomists about some help request.
- **Discussion Manager**
  This component handles all the discussions between farmers.

- **Notification Manager**
  It handles the notification to the Agronomist when a new Help Request is created by one of the Farmers in his region. To manage these interactions the component needs to interface with the Interaction Manager and the Data Manager. It needs to interface with the web application too to allow the DREAM application to function properly.

## 2.4 Data Component

Below it is reported a representation of the Entity-Relationship (ER) diagram of the *Database* of the architecture, which compose the *data layer* and is managed by a DBMS that performs and optimizes the queries.

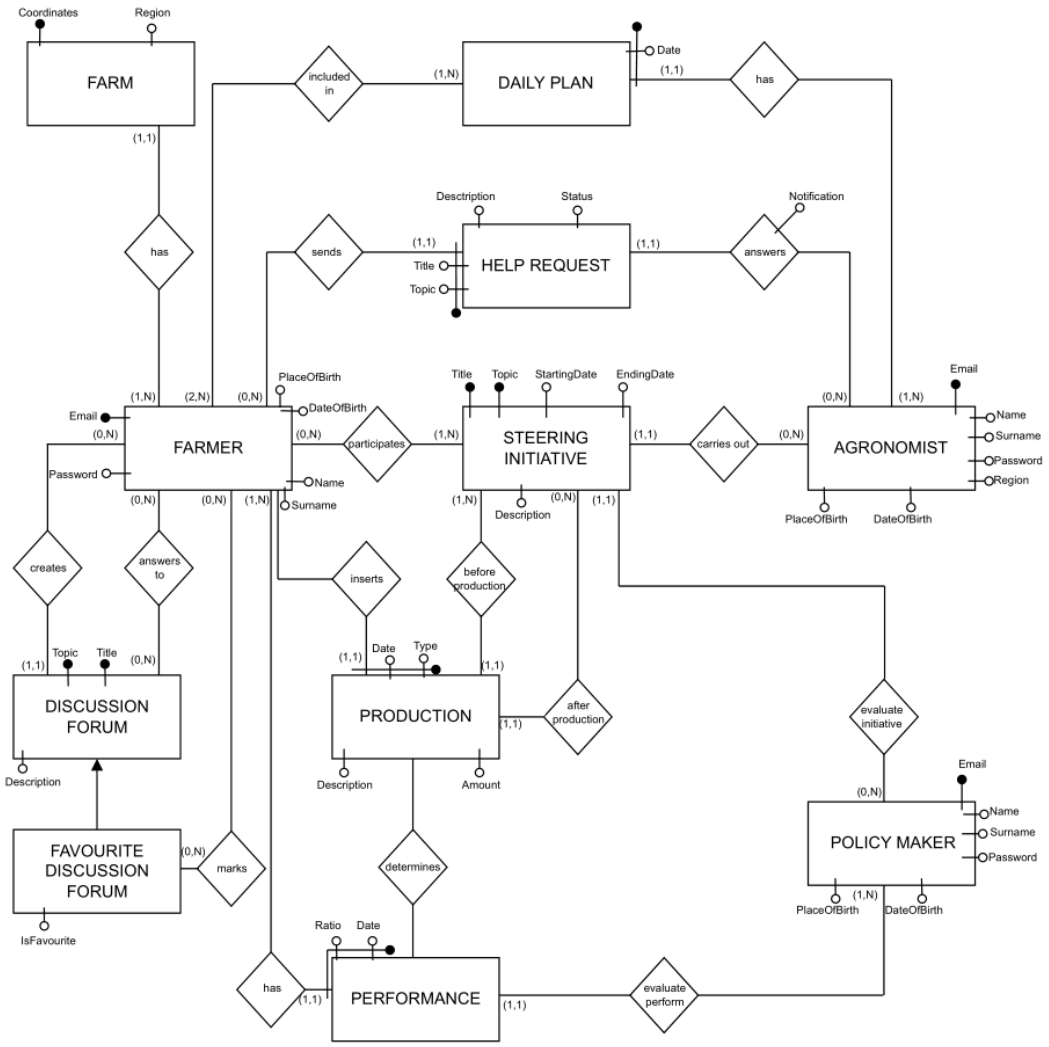


*Figure 5: DREAM Entity-Relationship Diagram*

Some relevant aspect of the Entity – Relationship diagram are:

- *PRODUCTION*, *HELP REQUEST*, *PERFORMANCE* and *DAILY PLAN* entities are identified as *weak – entity* kind, as their primary keys concern also other entitiess' primary keys. So, for example, to identify a unique *Production*, the System will refer to its Date and Type, as well as FarmerEmail of the Farmer user that previously inserted that production in the System. In this way, two versions of each of the over-mentioned entity will differ basing on the user that interacted with the application. And this is the case of each of the weak – entities.
- *beforeProduction* and *afterProduction* relationships link *STEERING INITIATIVE* entity with two different *PRODUCTION* entities. These two must have the same ProdType and the same FarmerEmail, so that it is possible to retrieve information about the same Farmer initial production related to a Steering Initiative, and its final production, provided that their ProdDate is different. It is trivial that the *beforeProduction* and *afterProduction* relationships must point to the same *STEERING INITIATIVE*.

From the ER diagram the following *Logical Schema* is determined:

***Farmer*** (Email, Password, Name, Surname, DateOfBirth, PlaceOfBirth, FarmCoordinates)

***Agronomist*** (Email, Password, Name, Surname, DateOfBirth, PlaceOfBirth, Region)

***PolicyMaker*** (Email, Password, Name, Surname DateOfBirth, PlaceOfBirth)

***Farm*** (Coordinates, Region)

***DailyPlan*** (Date, AgronomistEmail, FarmerEmail)

***HelpRequest*** (Topic, Title, FarmeEmail, Description, Status)

***SteeringInitiative*** (Topic, Title, StartingDate, EndingDate, Description, AgronomistEmail)

***DiscussionForum*** (Topic, Title, Description, CreatesFarmerEmail, AnswerFarmerEmail, IsFavourite*)

***Production*** (Date, Type, FarmerEmail, Description, Amount)

***Performance*** (Ratio, Date, FarmerEmail, PolicyMakerEmail)

***EvaluateInitiative*** (Topic, Title, PolicyMakerEmail, Status)

***EvaluatePerform*** (Ratio, FarmerEmail, PolicyMakerEmail, Status)

***Answers*** (Topic, Title, FarmeEmail, AgronomistEmail , notification)

***Participates*** (FarmerEmail, SteeringTopic, SteeringTitle)

***BeforeProduction*** (SteeringTopic, SteeringTitle , ProdDate, ProdType, FarmerEmail)

***AfterProduction*** (SteeringTopic, SteeringTitle , ProdDate, ProdType, FarmerEmail)

## 2.5 External Service Systems

- **Google Maps System**
  Google Maps System is an external system that communicates with the Application Server via APIs, providing a map view through which it's possible to localize farmer's farm.

- **Weather Forecast System**
  Weather Forecast System is an external system that communicates with the Application Server via APIs, providing weather information and data to Agronomists and Farmers.

- **Sensors**
  Sensors is an external system that communicates with the Application Server via APIs, providing relevant data about soil, collected by devices from loggers placed in different Telengana locations by DREAM.

# 3. Deployment View

In the Deployment view section, it is explained the configuration of processing nodes and the components that interact in the DREAM System. Deployment diagram for the entire System is shown below:

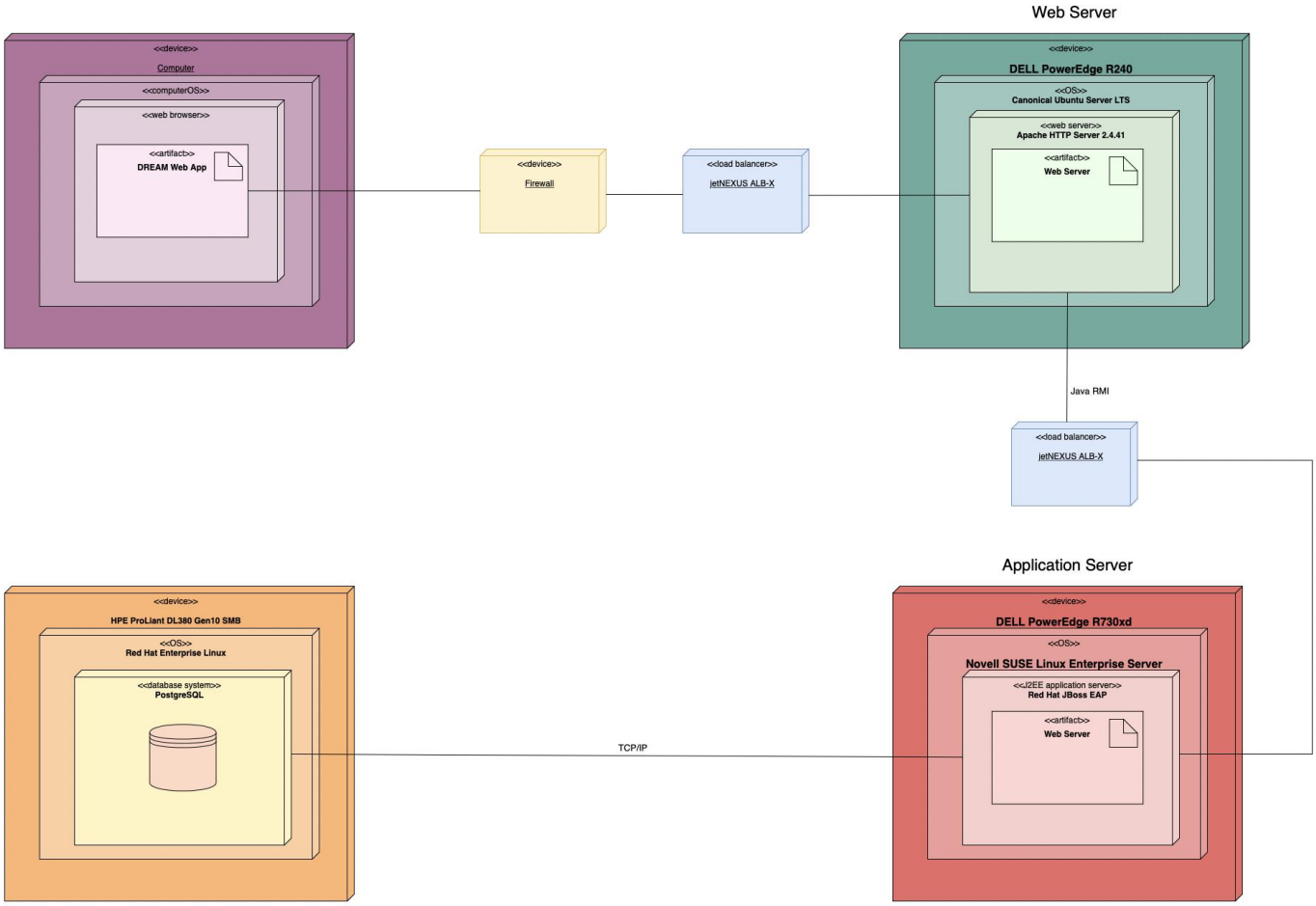


*Figure 6: Deployment View DREAM Application*

The data flow from the main device to the server component is managed by *firewall* and *load balancers*.

In particular, *firewalls* control the filtering of packets coming from the Internet; secondly, these information inside packets travel through the *load balancer*, which has the job of distributing the load among the resources available. In this way, the System guarantees optimal performance and reliability.

Furthermore, the *Application Server* interacts with the *Web Server*, while managing the business tier logic, so that the communication with *J2EE Application* components and with *Database Server* is enhanced through *PostgreSQL* APIs.

# 4. Runtime View

Now the dynamic nature of DREAM System is fully presented through the exhibition of the most relevant interaction between the processes that take part in the interaction between components.

- **Agronomist Registration**

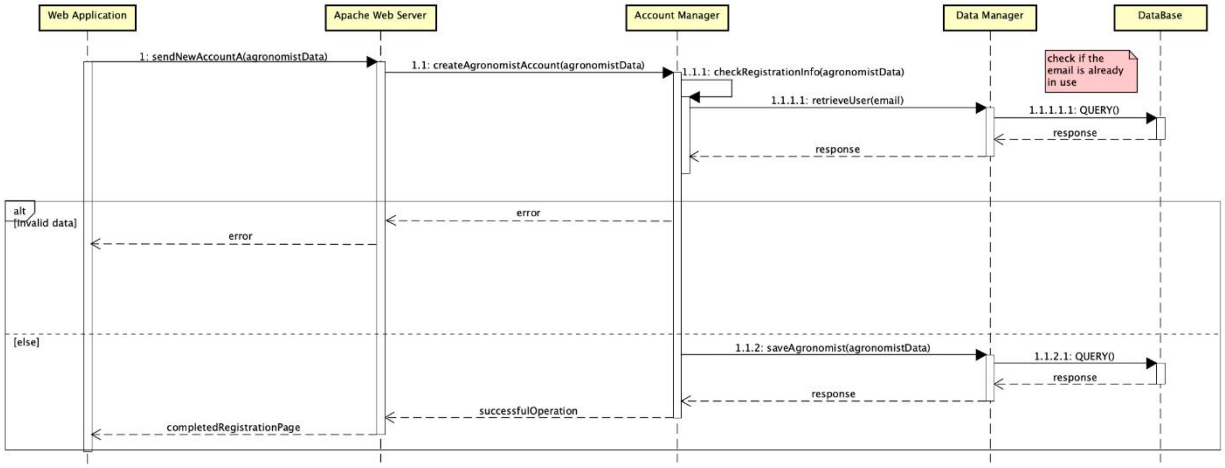


*Figure 7: Agronomist Registration Sequence Diagram*

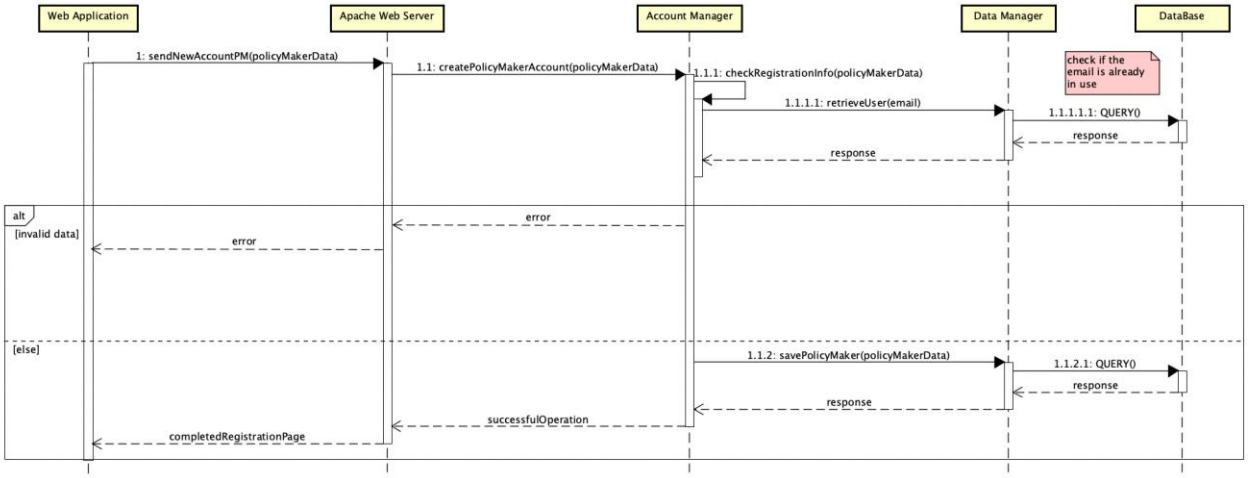


*Figure 8: Policy Maker Registration Sequence Diagram*

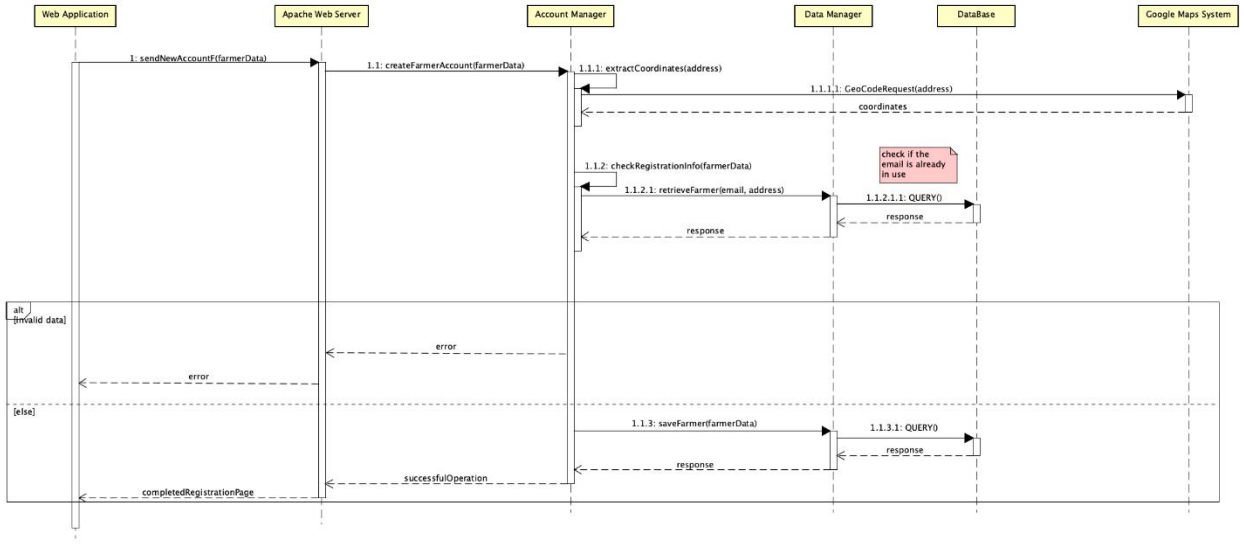


*Figure 9: Farmer Registration Sequence Diagram*

- **User Login**

In the following diagrams it is possible to witness the workflow that takes place during the operation of *LogIn* of each type of user (in order: *Farmer, Agronomist* and *Policy Maker*).

After the initial form is submitted, the *Web Application* sends the request to the *Account Manager* through the *Web Server* component. In case of positive verification outcome of the credentials , made possible through the interaction of the *Data Manager*, the *Web Server* component redirects the User to its personal Homepage; otherwise, and *Error Page* will be shown.

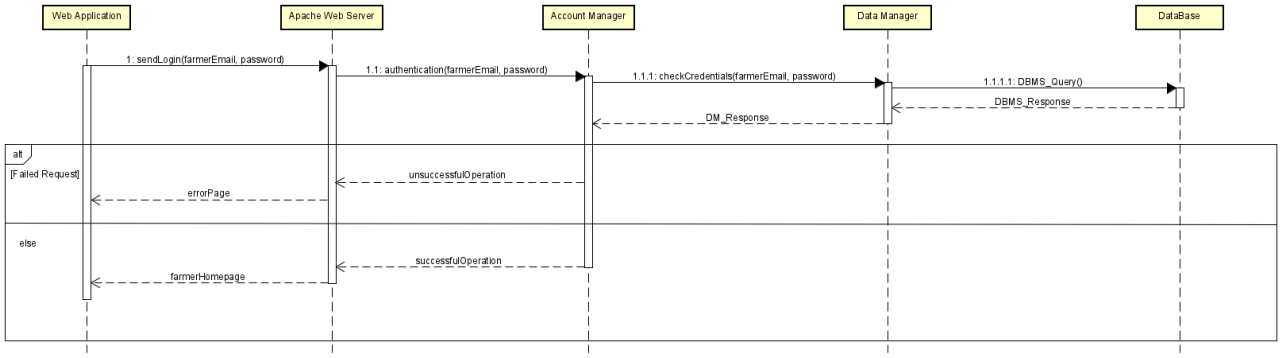


*Figure 10: Farmer LogIn Sequence Diagram*

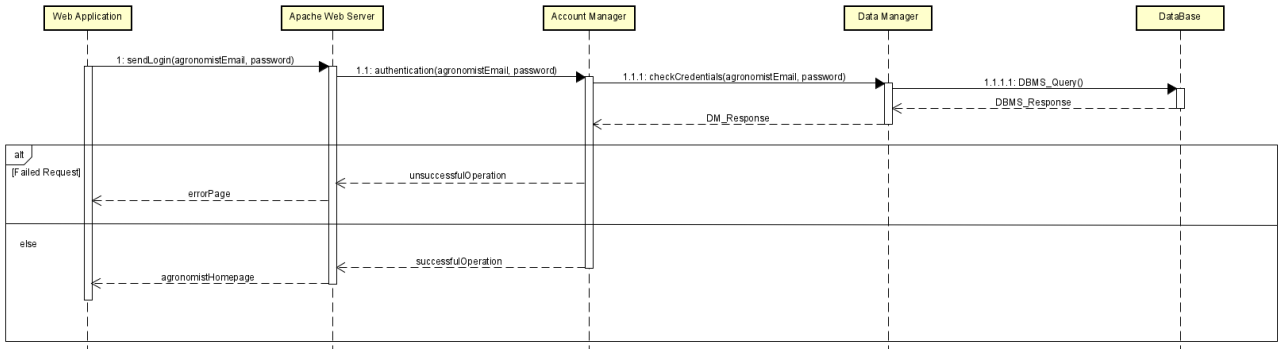


*Figure 11: Agronomist Registration Sequence Diagram*

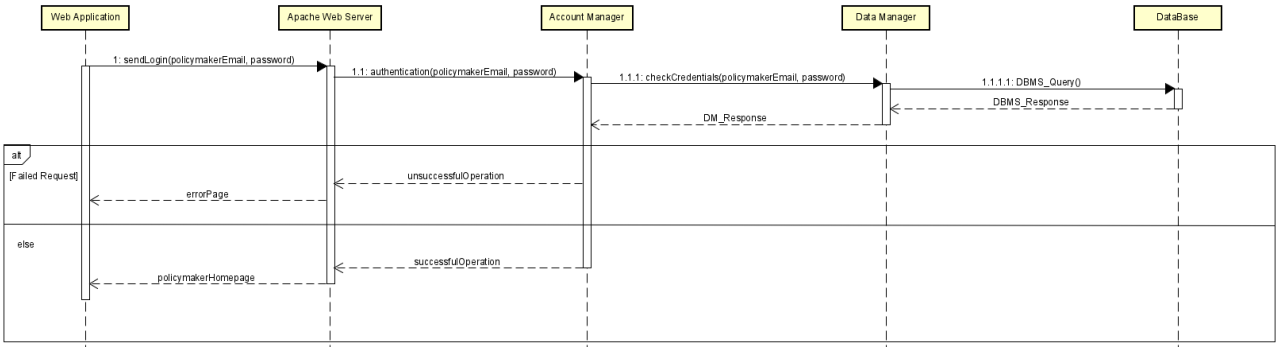


*Figure 12: Policy Maker Registration Sequence Diagram*

- **Help Request Creation**

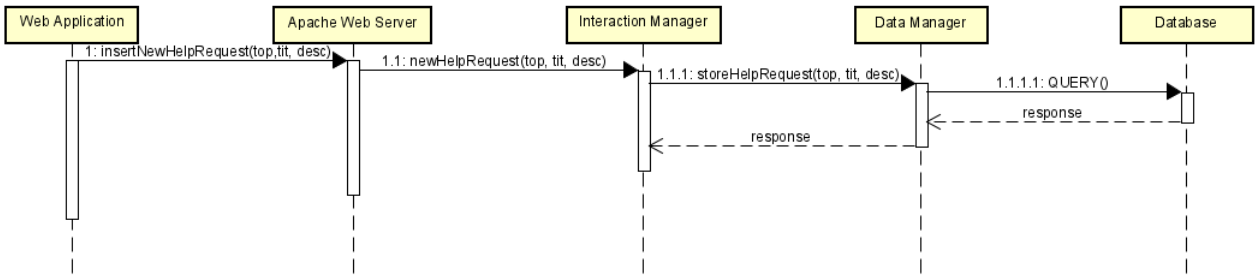


*Figure 13: Farmer Help Request Sequence Diagram*

- **Discussion Creation**

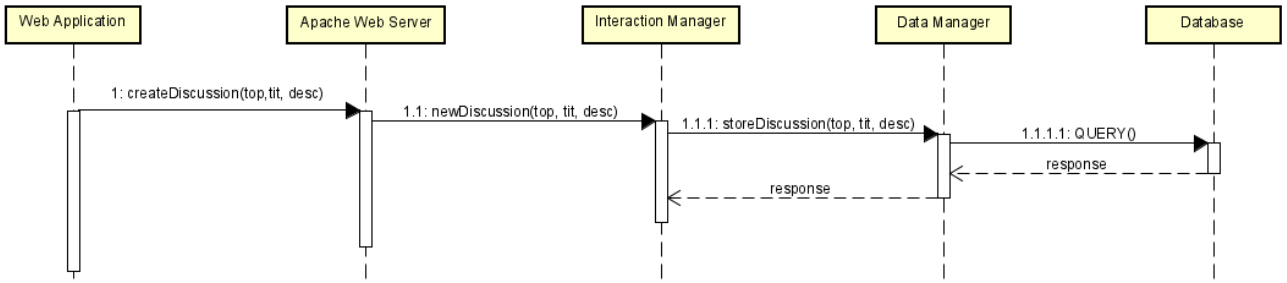


*Figure 14: Farmer Discussion Creation Sequence Diagram*

- **Farmer Performance Evaluation**

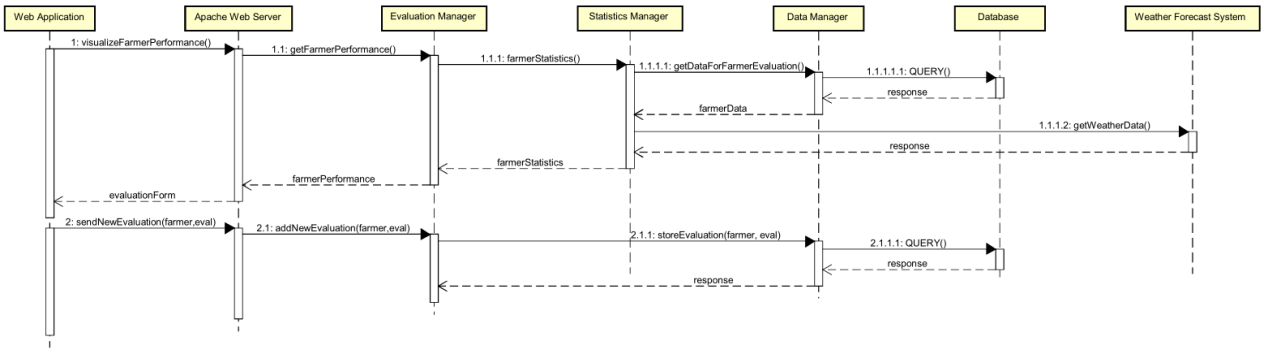


*Figure 15: Policy Maker Registration Sequence Diagram*

# 5. Component Interfaces

## 5.1 Data Manager Interfaces

- **retrieveFarmer**(email, address)
  This method returns the data about User whose email is passed as parameter to verify that email and address inserted in registration phase are not already in database.
  If none is found, the null value is returned.
- **retrieveUser**(email)
  This method returns the data about User whose email is passed as parameter to verify that email inserted in registration phase is not already in database.
  If none is found, the null value is returned.
- **saveFarmer**(farmerData)
  This method stores the data inserted by Farmer during registration to the DREAM Application, to create a new Farmer account.
  Throws an exception in case an error occurs.
- **saveAgronomist**(agronomistData)
  This method stores the data inserted by Agronomist during registration to the DREAM Application, to create a new Agronomist account.
  Throws an exception in case an error occurs.
- **savePolicyMaker**(policyMakerData)
  This method stores the data inserted by Policy Maker during registration to the DREAM Application, to create a new Policy Maker account.
  Throws an exception in case an error occurs.
- **storeEvaluation**(farmer, eval)
  This method stores the evaluation inserted by Policy Maker during an evaluation Farmer phase.
- **saveDiscussion(**top, tit, desc**)**
  This method stores the discussion details inserted by Farmer in Database through the Data Manager.
- **saveHelpRequest(**top, tit, desc**)**
  This method stores the Help Request inserted by Farmer in Database through the Data Manager and it stores Help Request to appropriate Agronomist.
- **checkCredentials**(email, password)
  This method verifies the validity, the correctness and the existence of the data entered during the login phase by Farmer.
  Throws an exception in case an error occurs.

## 5.2 Statistics Manager Interfaces

The Statistics Manager is a component accessible by Web Server, dedicated to provide and manage Farmer's statistics.

- **farmerStatistics**()
  This method retrieves data related to farmer's performance index

### 5.3 Account Manager Interfaces

The *Account Manager* component presents a unique external interface which makes available the interaction with the *Web Server* component. Its most relevant methods are:

- **generateFarmerAccount(**farmerData**)**
  If the inserted data are valid, the method generates a new account for Farmer. If not, it throws an error message
- **generateAgronomistAccount(**agronomistData**)**
  If the inserted data are valid, the method generates a new account for Agronomist. If not, it throws an error message
- **generatePolicyMakerAccount(**policymakerData**)**
  If the inserted data are valid, the method generates a new account for Policy Maker. If not, it throws an error message
- **authenticate(**email, password**)**
  It manages the login operation of every type of user

### 5.4 Notification Manager Interfaces

The Notification Manager is a component accessible by Web Sever dedicated to manage notification feature. Its most relevant methods is:

- **notify()**
  This method sends a notification to a farmer

### 5.5 Interaction Manager Interfaces

The Interaction Manager is a component useful for creating interactions between Farmers and Agronomists

- **storeDiscussion(**top, tit, desc**)**
  This method allows *Interaction Manager* forward discussion main information to the *Data Manager*, which will store them inside the Database

### 5.6 Web Server Interfaces

The *Web Server* component presents an external interfaces used by the *Web Browser* component of each type of user to access the *Web Application*. Its most important methods are:

- **createAccount(**userData**)**
  It manages the Registration request. If the operation is successful, the user is correctly directed to its personal homepage, otherwise an error message will show up
- **newDiscussion(**top, tit, desc**)**
  It manages the invitation of the new discussion to the *Interaction Manager* component, which will forward them to the *Data Manager*.
- **sendLogin(**email, password**)**
  It manages the Login request. If the operation is successful, the user is correctly directed to its personal homepage, otherwise an error message will show up

### 5.7 Google Maps System Interfaces

The *Google Maps System* furnishes those APIs that allows user to access its methods. Below are lists those used by DREAM System:

- **coordinateRequest(**address**)**
  It returns the coordinates (in terms of latitude and longitude) of the specified address
- **getRegion(**address**)**
  It returns the geographical region of the specified address, later used to understand whether the Farmer's farm is located inside an Agronomist region

### 5.8 Database Interfaces

The *DBMS* component presents an external interface through which it is possible to query and to access data inside of it

# 6. Selected Architectural Styles Interfaces

This section explains the main architectural choices and solution adopted in order to solve the most relevant architectural issues.

## 6.1 4-Tiers CLIENT-SERVER

The DREAM System is structured according to the *Client-Server* architecture for distributed applications, with a 4 tiers structure:

- The *presentation tier* (client-side) is the component responsible for making requests to the server, which handles them and returns a response. The DREAM System is accessible through the most common web browsers, which communicate with the *web tier* to perform the action requested

- The *web tier* is the component responsible for the interaction between the browser and the *business tier*. Its main job is to gather all the actions registered through the web browser and give back proper results arriving from the *business tier*.

- The *business tier* is the component which contains the core functionality of the architecture and provides the business logic for the system.

- The *data tier* is the component of the database server side of the architecture. It allows the *business tier* to access resources inside of it

## 6.2 Stateless Components

The *Application Server* component has been designed not to maintain any kind of state between requests, as every single one of them is provided of all information needed for the arrangement of the service. This choice is particularly suited for high-volume applications, as it allows to maintain optimal performance without loading the server with session information.

## 6.3 Data Access Component

The *Data Manager* component is responsible for storing and accessing data elements, without showing the complexity of these operations to any other component of the architecture. This approach allows to be able to only repair or replace the data manager component in case of failure of either the component itself or its interface

## 6.4 Provider Adapter

The *provider* interfaces are grouped together and linked to a unique interface, through which any other component can access its methods.

# 7. Other design decisions

## 7.1 Password Storing and User Authentication

DREAM System adopts the most recent and secure method currently available in internet to access System functionalities. For every type of user (Farmer, Agronomist and Policy Maker), DREAM login requires the email (used as unique identifier) and the password (hashed using the Secure Hash Algorithm SHA-256 and then stored in the *Database* component), both previously chosen by the user during the registration operation.

## 7.2 Relational Database

The Relational *DBMS* is the system component responsible for the management of data. It has been structured in order to ensure the following properties:

- Atomicity
- Confidentiality
- Integrity
- Durability

the so called ACID properties.

As a relational database, it is transactional and, ever after each *isolated* transaction has been performed, the state of the DBMS remains consistent and resilient to failure.

# Chapter 3

# User Interface Design

## 1. UX Diagrams

All the interfaces of the DREAM System have already been shown in the RASD.
Now it will be presented a graph that shows the interaction between all the pages, represented by the nodes, connected through links, represented by arrows.

### 1.1 Web Application Flow Graph

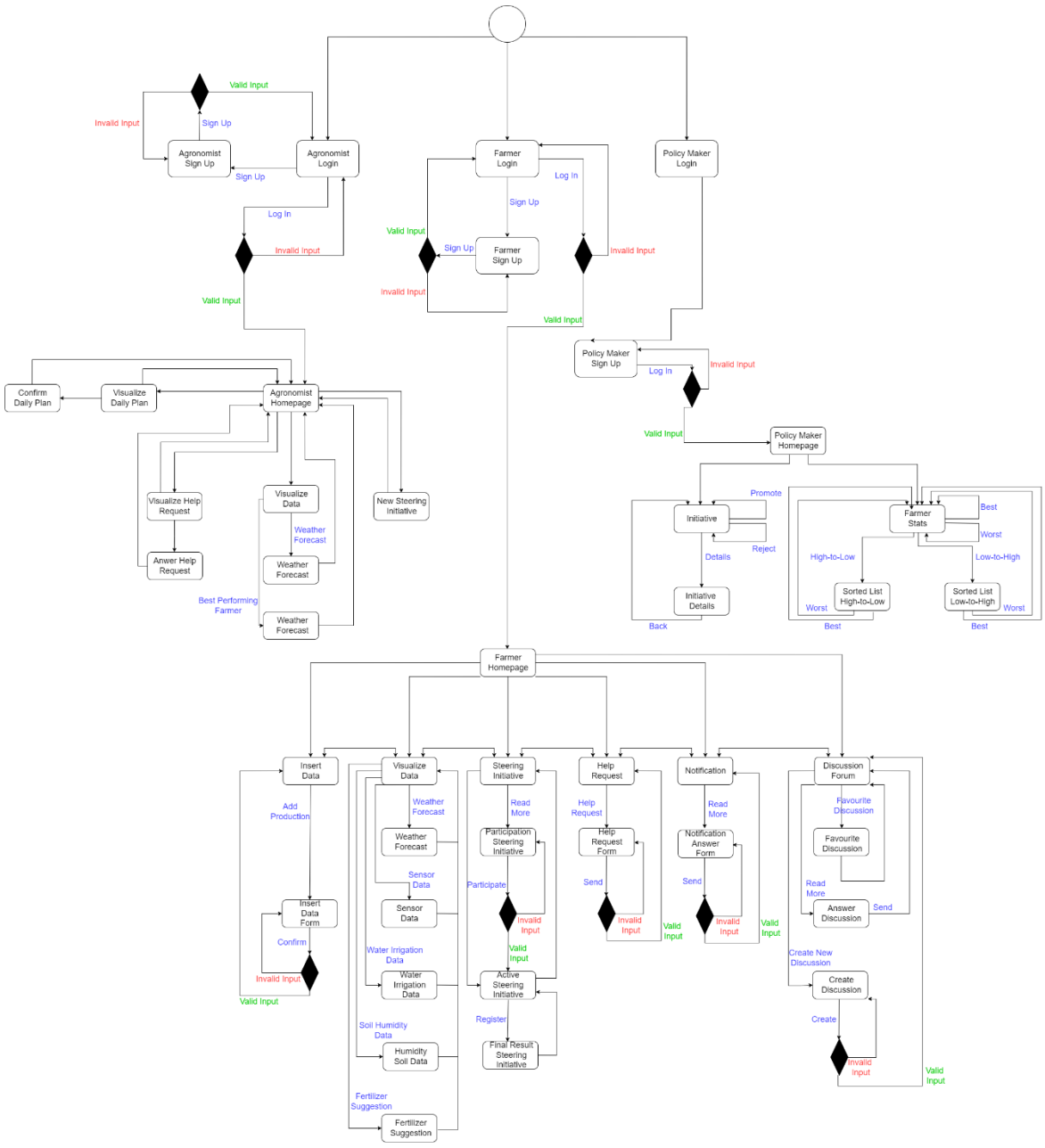


*Figure 16: Web App Flow Graph*

# Chapter 4

# Requirements Traceability

This section is dedicated to illustrate the relations between the DREAM Application components and the requirements described in the concerned section of RASD. So, the System is composed of the following components

**[C.1]** Web Application

**[C.2]** Web Server

**[C.3]** Account Manager

**[C.4]** Notification Manager

**[C.5]** Evaluation Manager

**[C.6]** Statistics Manager

**[C.7]** Interaction Manager

**[C.8]** Steering Initiative Manager

**[C.9]** Daily Plan Manager

**[C.10]** Data Manager

**[C.11]** Database

**[C.12]** Google Maps System

**[C.13]** Weather Forecast System

**[C.14]** Sensors

| \ | [C.1] | [C.2] | [C.3] | [C.4] | [C.5] | [C.6] | [C.7] | [C.8] | [C.9] | [C.10] | [C.11] | [C.12] | [C.13] | [C.14] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| [R.1] | X | X | | | | | | | | X | X | | | |
| [R.2] | X | X | | | | | | | | X | X | | | |
| [R.3] | X | X | | | | X | | | | X | X | | | |
| [R.4] | X | X | | | X | | | | | X | X | | | |
| [R.5] | X | X | | | X | | | | | X | X | | | |
| [R.6] | X | X | | | | | | | | X | X | | | |
| [R.7] | x | X | | | | | | X | | X | X | | | |
| [R.8] | X | X | | | | | | | | | | | X | |
| [R.9] | X | X | | | | | | | | X | X | | | X |
| [R.10] | X | X | | | | | | | | X | X | | | |
| [R.11] | X | X | | | | | | | | X | X | | | |
| [R.12] | X | X | | | | | | | | X | X | | | |
| [R.13] | | | | | | | | | | X | X | | | |
| [R.14] | X | X | | | | | | | | X | X | | | |
| [R.15] | | | | | | | | | | X | X | | | |
| [R.16] | X | X | | X | | | | | | X | X | | | |
| [R.17] | X | X | | | | | X | | | X | X | | | |
| [R.18] | X | X | | | | | X | | | X | X | | | |
| [R.19] | X | X | | | | | X | | | X | X | | | |
| [R.20] | X | X | | | | | | | | X | X | | | |
| [R.21] | X | X | | | | | | | | X | X | | | |
| [R.22] | X | X | | | | | | | | X | X | | | |
| [R.23] | X | X | | | | | | X | | X | X | | | |
| [R.24] | X | X | | | | | | X | | X | X | | | |
| [R.25] | X | X | | | | | | X | | X | X | | | |
| [R.26] | | | X | | | | | | | | | | | |
| [R.27] | | | X | | | | | | | | | | | |
| [R.28] | | | | | | | | | | X | X | | | |
| [R.29] | X | X | | | | | X | | | X | X | | | |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| [R.30] | X | X | | | | | X | | | X | X | | | |
| [R.31] | X | X | | | | | X | | | | | | | |
| [R.32] | X | X | | | | | X | | | X | X | | | |
| [R.33] | X | X | | | | | | | | | | | X | |
| [R.34] | X | X | | | | | | | | X | X | | | |
| [R.35] | X | X | | | | | | | | X | X | | | |
| [R.36] | X | X | | | | | | | X | X | X | | | |
| [R.37] | | | | | | | | | | X | X | | | |
| [R.38] | X | X | | | | | | | X | X | X | | | |
| [R.39] | X | X | | | | | | | X | | | | | |
| [R.40] | X | X | | | | | | | X | X | X | | | |

| **Requirement** | | **Components** |
|---|---|---|
| ID | Description | |
| [R.1] | The system allows policy maker to visualize list of farmers | Web Application<br>Web Server<br>Data Manager<br>Database |
| [R.2] | The system allows policy maker to visualize farmer's statistics | Web Application<br>Web Server<br>Statistics Manager<br>Data Manager<br>Database |
| [R.3] | The system allows policy maker to sort farmers depending on their performance | Web Application<br>Web Server<br>Statistics Manager<br>Data Manager<br>Database |
| [R.4] | The system allows policy maker to report the best performing farmers | Web Application<br>Web Server<br>Evaluation Manager<br>Data Manager<br>Database |
| [R.5] | The system allows policy maker to report the worst performing farmers | Web Application<br>Web Server<br>Evaluation Manager<br>Data Manager<br>Database |
| [R.6] | The system allows policy maker to visualize list of steering initiative | Web Application<br>Web Server<br>Data Manager<br>Database |

| | | |
|---|---|---|
| [R.7] | The system allows policy maker to promote/reject steering initiative | Web Application<br>Web Server<br>Steering Initiative Manager<br>Data Manager<br>Database |
| [R.8] | The system allows farmer to visualize weather forecast data | Web Application<br>Web Server<br>Weather Forecast System |
| [R.9] | The system allows farmer to visualize sensor's collected data | Web Application<br>Web Server<br>Data Manager<br>Database<br>Sensors |
| [R.10] | The system allows farmer to visualize suggestion for crop | Web Application<br>Web Server<br>Data Manager<br>Database |
| [R.11] | The system allows farmer to visualize suggestion for fertilizer | Web Application<br>Web Server<br>Data Manager<br>Database |
| [R.12] | The system allows farmer to visualize the list of all products | Web Application<br>Web Server<br>Data Manager<br>Database |
| [R.13] | The system must correctly save and store any data inserted by the farmer | Data Manager<br>Database |
| [R.14] | The system allows farmer to visualize the list of requests previously done | Web Application<br>Web Server<br>Data Manager<br>Database |
| [R.15] | The system must correctly display the notification badge | Data Manager<br>Database |
| [R.16] | The system allows farmer to visualize the list of notification | Web Application<br>Web Server<br>Notification Manager<br>Data Manager<br>Database |
| [R.17] | The System allows farmer to visualize the list of all discussions forum | Web Application<br>Web Server<br>Interaction Manager<br>Data Manager<br>Database |
| [R.18] | The System must allow Farmer to create discussion forum | Web Application<br>Web Server<br>Interaction Manager<br>Data Manager<br>Database |
| [R.19] | The System must allow Farmer to answer to discussion forum | Web Application<br>Web Server<br>Interaction Manager<br>Data Manager<br>Database |
| [R.20] | The System must allow Farmer to choose his favorite discussion | Web Application<br>Web Server |

| | | |
|---|---|---|
| | | Data Manager<br>Database |
| [R.21] | The System allows farmer to visualize the list of a filtered discussions forum | Web Application<br>Web Server<br>Data Manager<br>Database |
| [R.22] | The System allows farmer to visualize the list of all steering initiative | Web Application<br>Web Server<br>Data Manager<br>Database |
| [R.23] | The System must allow Farmer to participate his favorite steering initiative | Web Application<br>Web Server<br>Steering Initiative Manager<br>Data Manager<br>Database |
| [R.24] | The System must allow Farmer to visualize his personal steering initiative | Web Application<br>Web Server<br>Steering Initiative Manager<br>Data Manager<br>Database |
| [R.25] | The System must allow Farmer to read more about a steering initiative | Web Application<br>Web Server<br>Steering Initiative Manager<br>Data Manager<br>Database |
| [R.26] | The System must ask in the registration form if the new user is an agronomist | Account Manager<br>Web Server Manager<br>Web Application Manager |
| [R.27] | The System must allow agronomist to insert the region they are responsible of during the registration process | Account Manager<br>Web Server Manager<br>Web Application Manager |
| [R.28] | The System must store the data about the agronomist's region | Data Manager<br>Database |
| [R.29] | The System must allow agronomist to visualize the help requests addressed to them | Web Application<br>Web Server<br>Interaction Manager<br>Data Manager<br>Database |
| [R.30] | The System must allow agronomist to visualize all previous interactions regarding a help request | Web Application<br>Web Server<br>Interaction Manager<br>Data Manager<br>Database |
| [R.31] | The System must allow agronomist to enter some text | Web Application<br>Web Server<br>Interaction Manager |
| [R.32] | The System must allow agronomist to send some text as an answer | Web Application<br>Web Server<br>Interaction Manager<br>Data Manager<br>Database |
| [R.33] | The System must allow agronomist to visualize weather forecasts | Web Application<br>Web Server<br>Weather Forecast System |

| | | |
|---|---|---|
| [R.34] | The System must allow agronomist to visualize list of farmers | Web Application<br>Web Server<br>Data Manager<br>Database |
| [R.35] | The System must allow agronomist to identify best performing farmers | Web Application<br>Web Server<br>Data Manager<br>Database |
| [R.36] | The System must allow agronomist to visualize his daily plan | Web Application<br>Web Server<br>Daily Plan Manager<br>Data Manager<br>Database |
| [R.37] | The System must compile agronomists daily plan for the current day before 6:00 AM | Data Manager<br>Database |
| [R.38] | The system must allow agronomists to add a visit to the daily plan | Web Application<br>Web Server<br>Daily Plan Manager<br>Data Manager<br>Database |
| [R.39] | The system must allow agronomist to check which visit they did | Web Application<br>Web Server<br>Daily Plan Manager |
| [R.40] | The system must allow agronomist to confirm the daily plan | Web Application<br>Web Server<br>Daily Plan Manager<br>Data Manager<br>Database |

# Chapter 5

# Implementation, Integration and Test Plan

## 1. Development Process

The System's architecture is composed by, as already said, 4 tiers that offer the possibility to be implemented together at the same time, speeding up the process in order to give a full-functional first iteration of the product to the final users. The *External Services* can be integrated at the end of the developing process, when all the parts constituting the final application are gathered together.

To perform the implementation and the integration, it is highly suggested a *bottom-up approach*: at every step of the procedure, the already existing elements are utilized to build new and more powerful ones, which will be further used as the procedure goes on in next implementing steps. Finally, the last iteration of each elements will be combined together, forming the final DREAM System.

The following image shows the interaction and dependencies between all the component forming the *Application Server*.

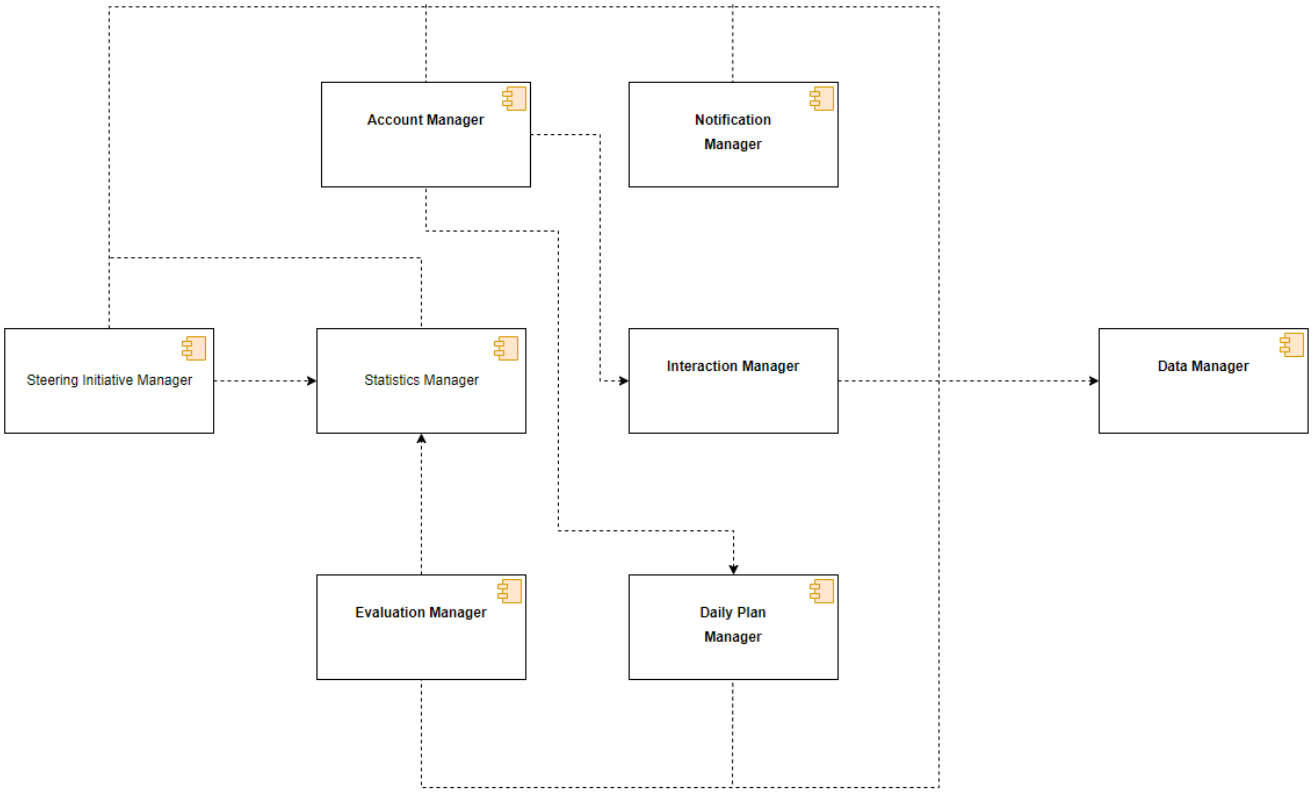


*Figure 17: Dependency Diagram*

The following table shows the implementation complexity of each component that has been previously taken into account.

| Component | Implementation Complexity |
|---|---|
| **[C.1]** Data Manager | Medium |
| **[C.2]** Account Manager | High |
| **[C.3]** Notification Manager | Low |
| **[C.4]** Evaluation Manager | Low |
| **[C.5]** Statistics Manager | High |
| **[C.6]** Interaction Manager | Medium |
| **[C.7]** Steering Initiative Manager | High |
| **[C.8]** Daily Plan Manager | High |

# 2. Implementation Plan

The implementation order for the development of the Application Server is the following one:

- **Data Manager:** this is the first module to be implemented. Indeed, all the other components of the Application Server rely on it to communicate with the Database. Therefore, it is a module of crucial importance but characterized by a not high level of complexity as it only implements the queries to the database.
- **Statistics Manager:** it could be implemented in parallel with the Account Manager as they are mutually independent. The two components have the same levels of complexity, but this one is essential for the correct functioning of the Evaluation Manager. Given the high complexity of the latter, higher priority will be given to the development of the Statistics Manager.
- **Account Manager:** the implementation of this module is essential for the correct functioning of the DREAM System, the other modules depend totally on this.
- **Daily Plan Manager:** being one of the most complex modules, this is one of the first to be implemented
- **Interaction Manager:** this component is then implemented. It represents a critical point for the success of the entire project, and, at the same time, it requires to interact with the components listed above.
- **Evaluation Manager:** this component can be implemented just after the Statistics Manager one, being dependent on it.
- **Steering Initiative Manager:** this component manages an aspect of the system that is marginal and is not needed for any other modules. This is a difficult module to implement and it is highly dependent on the other ones, so it will be developed as one of the last.
- **Notification Manager:** once the above modules are completed, particularly the Help Request, the implementation of the Notification Manager takes place. This choice is due to the fact that it only handles additional or marginal functions of the System.

# 3. Integration Sequence

This section describes the integration plan for the system to be. *Unit Testing* is the first phase for each component. Integration testing is done incrementally. Every time a component who's dependent on other ones which have already been integrated is completed, it is integrated with them.

The following diagrams show the integration of the various component.

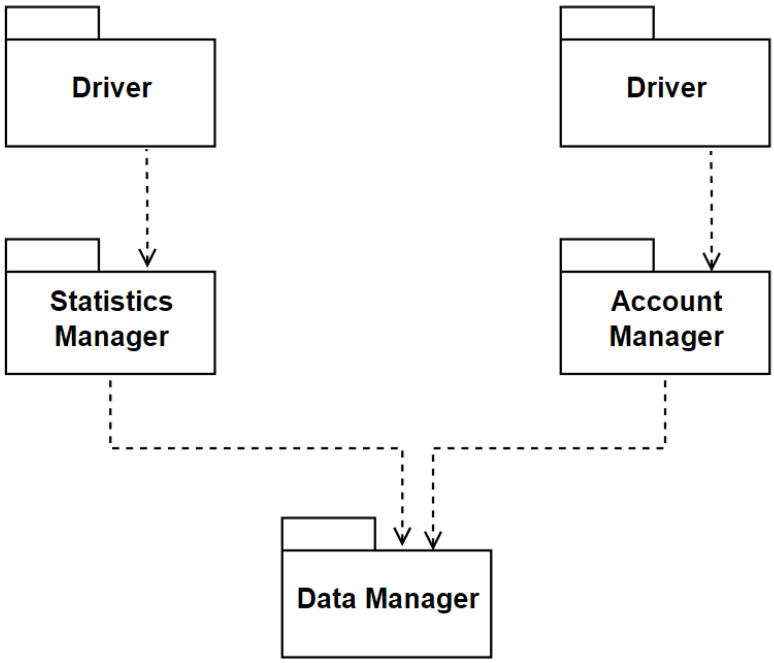


*Figure 18.1: component integration*

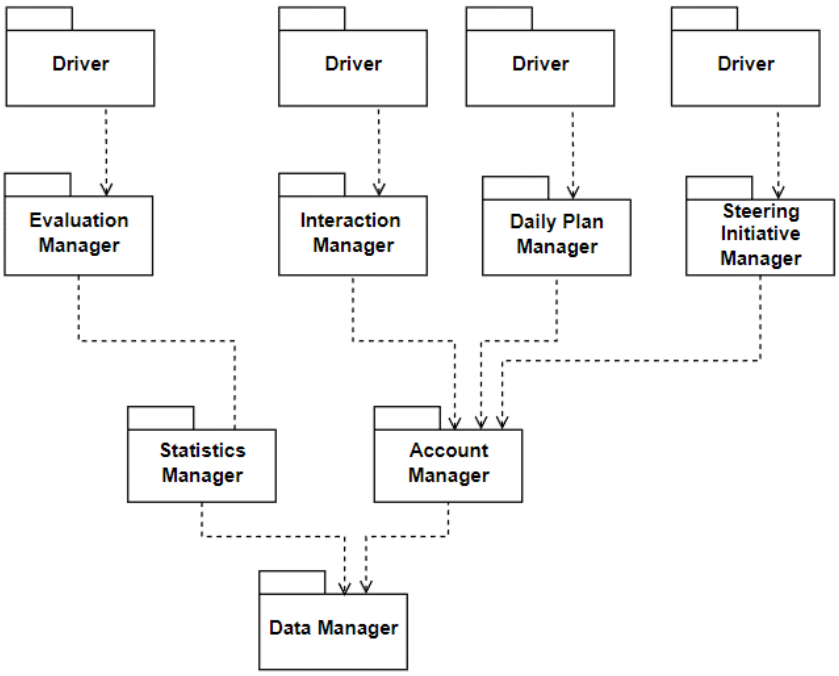

*Figure 18.2: component integration*

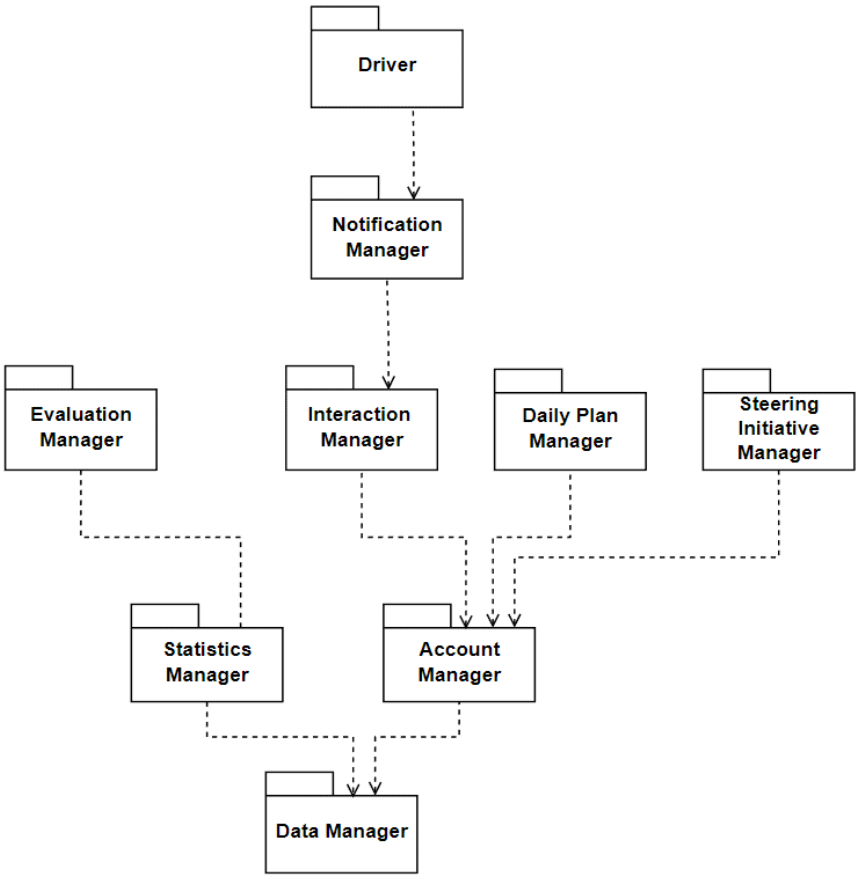


*Figure 18.3: component integration*

# 4. System Testing

This step is the last one. Its main objective is to check the correctness of functional and non-functional requirement in a testing environment, which has to be as similar as possible to the production environment.

The testing step DREAM System has taken into account are:

- *Functional Testing*: it aims at assuring that every requirement and specifications documented in the RASD are correctly fulfilled by the System
- *Performance Testing*: it identifies and tries to overcome the main performance bottlenecks concerning System's:
    - Response time
    - Throughput
    - Service utilization
- *Load Testing*: it concerns the practice of predicting the expected usage of the DREAM System, simulating multiple concurrent access. Its main objective is to identify the memory performance and test the maximum operating capacity of the application
- *Stress Testing*: this kind of test is the most extreme one and it aims at verifying reliability and robustness of the System, as well as its capacity of handling error under intense load conditions without crashing

# Chapter 6

# Effort Spent

The following tables summarize the effort spent by each member of the team to produce the DD document.

**Censuales Simone**

| **Description of the task** | Hours |
|---|---|
| Document Introduction: Purpose and Scope Summary | 0.5 |
| Identification of External Services, Patterns and Architectural styles | 1 |
| Definition of the General Architecture and Components Identification | 3 |
| Deployment of the System: definition and diagram | 4 |
| General definition of Runtime Views: identification and description | 7 |
| Runtime View Refinements | 4 |
| Description of the Patterns and Architectural Choices adopted | 3 |
| UX Diagrams: definition and realization | 1.5 |
| Full Revision of Chapter 2 | 1 |
| Requirements Traceability Definition | 0.5 |
| Full Revision of Chapters 3 and 4 | 0.5 |
| Definition of the Development Process | 2 |
| Revision of the Document and further improvements | 1 |
| Fixes to the Runtime View section | 1 |

**Giovia Giuseppe**

| **Description of the task** | Hours |
|---|---|
| Document Introduction: Purpose and Scope Summary | 0.5 |
| Identification of External Services, Patterns and Architectural styles | 1.5 |
| Definition of the General Architecture and Components Identification | 2.5 |
| Deployment of the System: definition and diagram | 4 |
| General definition of Runtime Views: identification and description | 4 |
| Runtime View Refinements | 6 |
| Description of the Patterns and Architectural Choices adopted | 3 |
| UX Diagrams: definition and realization | 2 |
| Full Revision of Chapter 2 | 1 |
| Requirements Traceability Definition | 0.5 |
| Full Revision of Chapters 3 and 4 | 0.5 |
| Definition of the Development Process | 1.5 |
| Revision of the Document and further improvements | 1 |
| Fixes to the Runtime View section | 1 |

**Meli Giuseppe**

| **Description of the task** | Hours |
|---|---|
| Document Introduction: Purpose and Scope Summary | 0.5 |
| Identification of External Services, Patterns and Architectural styles | 1 |
| Definition of the General Architecture and Components Identification | 3.5 |
| Deployment of the System: definition and diagram | 3.5 |
| General definition of Runtime Views: identification and description | 6 |
| Runtime View Refinements | 5 |
| Description of the Patterns and Architectural Choices adopted | 3 |
| UX Diagrams: definition and realization | 1.5 |
| Full Revision of Chapter 2 | 1.5 |
| Requirements Traceability Definition | 0.5 |
| Full Revision of Chapters 3 and 4 | 0.5 |
| Definition of the Development Process | 2 |
| Revision of the Document and further improvements | 1 |
| Fixes to the Runtime View section | 0.5 |

# Chapter 7

# References

M.G. Rossi, Lecture Slides, Politecnico di Milano
M.G. Rossi, Project Assignment AY 2021-2022, Politecnico di Milano
Uml Diagrams. uml-diagrams.org
Cloud computing patterns. cloudcomputingpatterns.org
Oracle - Tiered Applications. docs.oracle.com
Google Maps Platform. developers.google.com/maps