DD

POLITECNICO DI MILANO 1863

Title: DREAM - Data-dRiven PrEdictive FArMing in Telangana

Description: Based on the project document from teachers

Document version: 1.2.0

Authors: Luca Bertelli, Matteo Savino, Giacomo Vinati

Release Date: 02/01/2022

Table of contents

1. INTRODUCTION

1.1 Purpose

This document is the Design Document (DD) of the DREAM web application. It contains a functional description of the system, and an accurate overview of all parts of the software through the component view, the deployment view and the run-time view.

1.1.1 Description

The main goals to be achieved with this application can be summarized in providing appropriate support for all the professionals involved in Telangana agricultural sector. The main actors involved are policy makers, farmers and agronomists.

More details can be found in the RASD document.

1.1.2 Goals

Goals	Description
Policy Makers related goals	
G1.1	Provide policy makers with data about farmers' production performances.
G1.2	Allow policy makers to identify farmers with best performances, especially the ones that faced adverse weather conditions.
G1.3	Allow policy makers to identify farmers with worst performances.
G1.4	Allow policy makers to identify whether agronomist initiatives increase farmers' production performances.
Farmers related goals	
G2.1	Provide quantitative data to each farmer based on location and type of production.
G2.2	Allow farmers to open discussions between them in the forum.
G2.3	Allow farmers to participate in forum discussions already opened.
G2.4	Allow farmers to send help requests to other farmers in the same area and the responsible agronomist.
G2.5	Allow farmers to send requests for suggestions to other farmers in the same area and the responsible agronomist.

G2.6	Allow farmers to receive help requests from their peers in the same area.
G2.7	Allow farmers to receive suggestion requests from their peers in the same area.
G2.8	Allow farmers to reply to help requests they received.
G2.9	Allow farmers to reply to suggestion requests they received.
G2.10	Allow farmers to insert their production data in the system.
G2.11	Allow farmers to insert into the system data about any problem they face.
Agronomists related goals	
G3.1	Allow agronomists to select their area of influence.
G3.2	Allow agronomists to receive help requests from farmers in their area.
G3.3	Allow agronomists to receive suggestion requests from farmers in their area.
G3.4	Allow agronomists to reply to help requests from farmers in their area.
G3.5	Allow agronomists to reply to suggestion requests from farmers in their area.
G3.6	Allow agronomists to get weather forecasts of their area of influence.
G3.7	Allow agronomists to get the list of farmers with the best performances in the area.
G3.8	Provide agronomists with a daily plan of the farms to be visited.
G3.9	Allow agronomists to update their daily plan.
G3.10	Allow agronomists to confirm the execution of the daily plan.
G3.11	Allow agronomists to specify the deviations from the original plan at the end of each day.
Other goals	
G4.1	Allow the DREAM system to acquire data concerning meteorological short-term and long-term forecasts.
G4.2	Allow the DREAM system to acquire data measured by the water irrigation systems.
G4.3	Allow the DREAM system to acquire soil humidity data obtained by sensors.

1.2 Scope

In this section, the phenomena related to the machine, which is the software to be developed, and to the world which is the real environment in which DREAMS will be used are enumerated. A phenomenon can be shared if it is controlled by the world and observed by the machine or vice versa.

1.2.1 World phenomena

World Phenomena	Description
WP1	Telangana government collects data about meteorological short-term and long-term forecasts.
WP2	Farmers collect data about the type of products they are producing.
WP3	Farmers collect data about the amount of product of each type they are producing.
WP4	Water irrigation systems collect the amount of water used by each farmer.
WP5	Sensors placed throughout the whole territory, collect data concerning the humidity of the soil.
WP6	Agronomists collect data from farms they visit.
WP6	Agronomists perform initiatives.
WP7	Policy makers judge agronomist initiatives.
WP8	The farmers performing well with respect to production are rewarded with special incentives.
WP9	Farmers in the same area share suggestions with each other.
WP10	Farmers in the same area help each other.
WP11	Agronomists give suggestions to farmers in their area.
WP12	Agronomists help farmers in their area.
WP13	Farmers have discussions with other farmers on agriculture related topics.

1.2.2 Shared phenomena

Shared Phenomena	Description	Control
Shared phenomena connected with Policy makers		

SP1.1	The system shows to policy makers data relevant to farmers and their production.	Machine Controlled
SP1.2	The system shows to policy makers an entire overview of farmers production performances.	Machine Controlled
SP1.3	Policy makers evaluate agronomist initiatives.	World controlled
Shared phenomena connected with Farmers		
SP2.1	Farmers insert in the system data about their production.	World Controlled
SP2.2	Farmers insert in the system data about any problem they face.	World Controlled
SP2.3	Farmers insert in the system requests for help.	World Controlled
SP2.4	Farmers insert in the system requests for suggestions.	World Controlled
SP2.5	Farmers answer requests for suggestions that they received.	World controlled
SP2.6	Farmers answer help requests that they received.	World controlled
SP2.7	Farmers create discussion posts.	World Controlled
SP2.8	The system shows farmers data relevant for their own production.	Machine Controlled
Shared phenomena connected with Agronomists		
SP3.1	Agronomists insert the area they are responsible for.	World Controlled
SP3.2	Agronomists receive requests for help from farmers in their area.	Machine Controlled
SP3.3	Agronomists receive requests for suggestions from farmers in their area.	Machine Controlled
SP3.4	Agronomists answer requests for help they received.	World Controlled
SP3.5	Agronomists answer requests for suggestions they received.	World controlled

SP3.6	The system shows an agronomist a daily plan to visit farms in his area.	Machine Controlled
SP3.7	Agronomists update the daily plan to visit farms in his area.	World Controlled
SP3.8	The system shows to agronomists data concerning weather forecasts in his area.	Machine Controlled
SP3.9	The system shows agronomists data about farmers' performances in their area.	Machine Controlled
SP3.10	Agronomists confirm the execution of their daily plan.	World Controlled
SP3.11	Agronomists register deviations from their original daily plan.	World Controlled

1.2.3 Machine phenomena

Machine Phenomena	Description
MP1	The system computes the production performances of farmers.
MP2	The system generates daily plans for agronomists so that they visit every farm in their area at least twice every year.
MP3	For each farm, the system maintains a counter for every visit in the current year.
MP4	The system updates the visit count to zero within 24 hours from the beginning of the new year.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Terms	Definition
Farmer	A person who owns or manages a farm.

Policy maker	A person responsible for or involved in formulating policies.
Agronomist	An expert in the science of soil management and crop production.
Weather Forecast	An analysis of the state of the weather in an area with an assessment of likely developments.
Information	Facts provided or learned about something or someone.
Daily Plan	It is the mapping of one’s daily activities.
Visit	Go to see and spend time with someone socially.
Initiative	An act or strategy intended to resolve a difficulty or improve a situation; a fresh approach to something.
Best practices	Professional procedures that are accepted or prescribed as being correct or most effective.
Problem	A matter or situation regarded as unwelcome or harmful and needing to be dealt with and overcome.
Discussion Forum	A web page where users can post comments about a particular issue or topic and reply to other users’ postings.
Sensor	A device which detects or measures a physical property and records or otherwise responds to it.
Peer	A person that belongs to the same category.

1.3.2 Acronyms

Acronyms	**Description**
DREAM	**D**ata-d**R**iven Pr**E**dictive F**A**r**M**ing in Telangana
RASD	Requirements Analysis and Specification Document
DD	Design Document
UML	Unified Modeling Language
IEEE	Institute of Electrical and Electronics Engineers
DMZ	Demilitarized zone
DB	Database
URI	Uniform Resource Identifier
API	Application Programming Interface
UAT	User acceptance testing

COTS	Commercial off the shelf
HTTP	Hypertext transfer protocol
JEE	Java Enterprise edition
AWS	Amazon Web Services
DoS	Denial of Service (attack)
DDoS	Distributed Denial of Service (attack)

1.3.3 Abbreviations

Abbreviations	Description
S2B	Software to be
PC	Computer
IT	Information technologies
e.g.	Exempli gratia
w.r.t.	With respect to
MP	Machine Phenomena
G	Goal
SP	Shared Phenomena
WP	World Phenomena
R	Requirements
D	Domain
NB	Note well
Webapp	Web Application
ETL	Extract, Transform, Load

1.4 Revision history

Version	Date	Description
0.0	9/12/2021	Project opening.
1.0	29/12/2021	First release.

1.1	31/12/2021	General review.
1.2	2/1/2021	Fixed typos.

1.5 Reference Documents

- Slides of the lectures
- Specification document “01.Assignment RDD AY 2021-2022”
- Hans van Vliet (2008), Software Engineering: Principles and Practice
- About the Unified Modeling Language Specification Version 2.5.1
- Telangana website: Telangana State Development Planning Society

1.6 Document Structure

The document is divided into six parts.

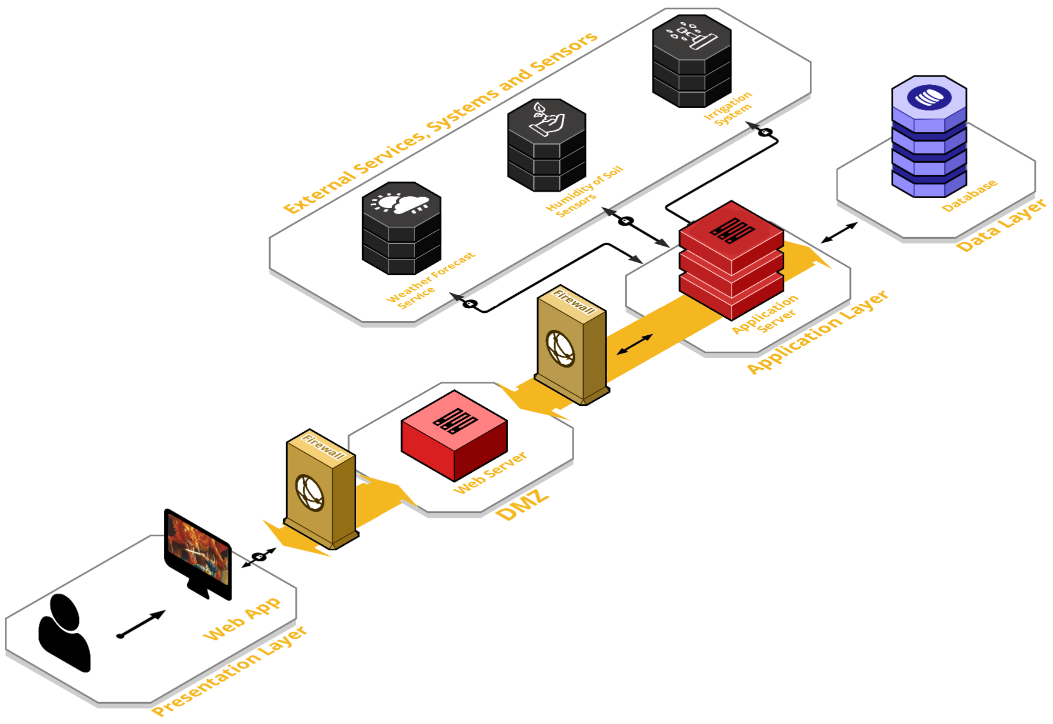
- **INTRODUCTION**: It gives an overview on the purpose and scope of the document by defining the main goals, phenomena. It also includes definitions, acronyms, abbreviations of the most used terms, the revision history and the reference documents to better underline how it has been developed.
- **ARCHITECTURAL DESIGN**: It describes the high-level architecture, highlighting the main components, the interfaces, their interaction with runtime views and other design decisions.
- **USER INTERFACE DESIGN**: It provides an overview on how the user interfaces of the software will look like.
- **REQUIREMENTS TRACEABILITY**: It explains how the requirements, defined in the RASD, map to the design elements that are defined in this document, describing the connection between the RASD and the DD.
- **IMPLEMENTATION, INTEGRATION AND TEST PLAN**: It Identifies the order in which we plan to implement the subcomponents of the S2B and the order in which we will integrate such subcomponents and test such integration.
- **EFFORT SPEND**: It includes information about the number of hours each group member has worked for the development of the document.

2. ARCHITECTURAL DESIGN

2.1 Overview

The web application is developed with a three-tier architectural style, which allows the distribution of functionalities across three independent systems: a presentation layer, an application layer, and a data layer. They will be discussed in detail in section 2.2. Moreover, the S2B will interact with some external services/systems.

The figure below represents a high-level description of the main components which constitute the S2B, or that interact with it.



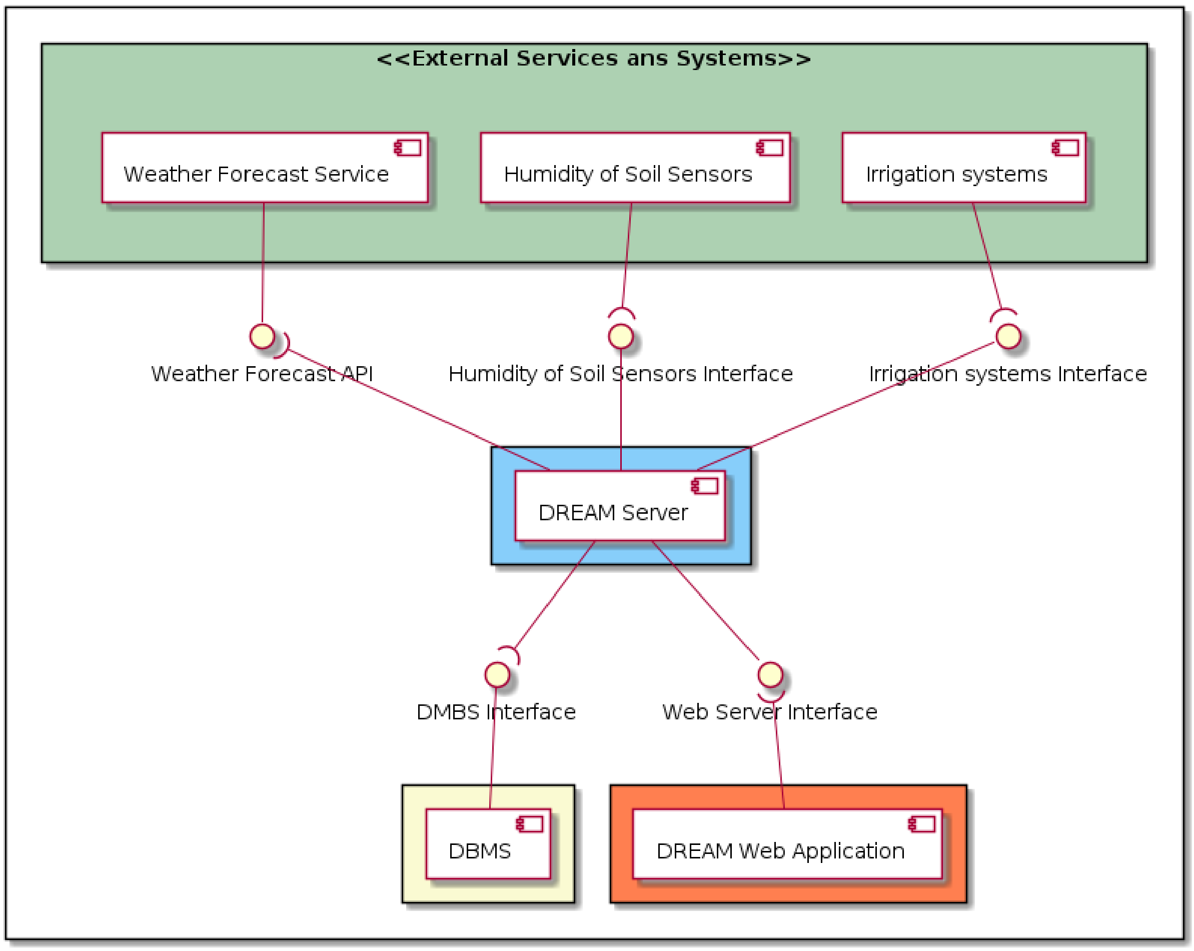
As we can see, a demilitarized zone (DMZ) is created thanks to the usage of two firewalls: the first one must be configured to allow traffic destined to the DMZ only, the second one only allows traffic to the DMZ from the internal network. In this way we can isolate the public services of the system (i.e.: the web server) from the local, private LAN machines in our network.

The main components are the following:

Web Application	The web application (or web app) is the application software that runs on the web server. It is accessed by the user through a web browser with an active network connection. The use of a web application makes the use of DREAM independent from the devices and the operating systems of users.
Firewall	It is a network security system that monitors and controls incoming and outgoing network traffic based on predetermined security rules. It typically establishes a barrier between a trusted network and an untrusted network, such as the Internet.
Web Server	It processes incoming network requests over HTTP(s), such as the REST API calls. It is the only entry point for the web app to access the system functionalities.

Application Server	It is the main back-end component on which the logic of the application takes place: it elaborates the requests coming from the App, interacts with the data layer and communicates with the external systems.
Database	It is the component responsible for data storage.
External services, systems and infrastructures	They are systems that interact with DREAM and provide functionalities and data not internally developed but needed in order to provide DREAM main functionalities.

2.2 Component view



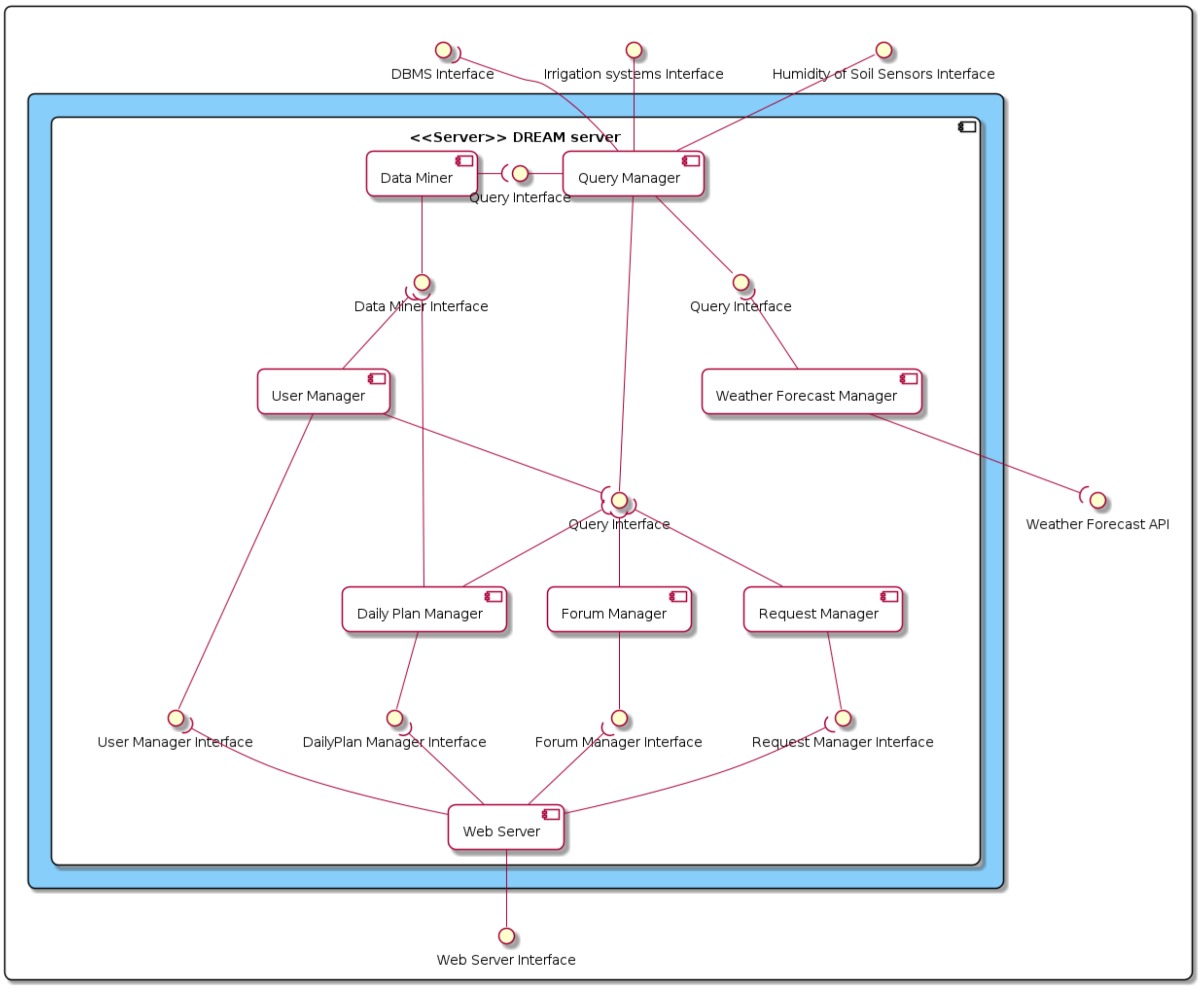
The component diagram gives a specific view of the system focusing on the internal structure and showing how the components interact. Colors have been used to highlight the different tiers: orange for the presentation layer, light blue for the application layer, yellow for the data access layer and green for the external services, systems, and infrastructures.

2.2.1 Presentation layer

The presentation layer represents the front-end of the system through which the user interacts with the software. It consists of just one component which is the DREAM web application. It is the software provided to the users through which they can exploit all the business functionalities. The web application can interact only with the web server in order to give to the users all the needed functions that are offered by the backend. It can make requests to different components by calling the proper interface offered by the web server.

The system shows a personalized view with information and functionalities based on the type of user currently logged in. In particular, the web application shows different graphical interfaces to policy makers, agronomists and farmers.

2.2.2 Application layer



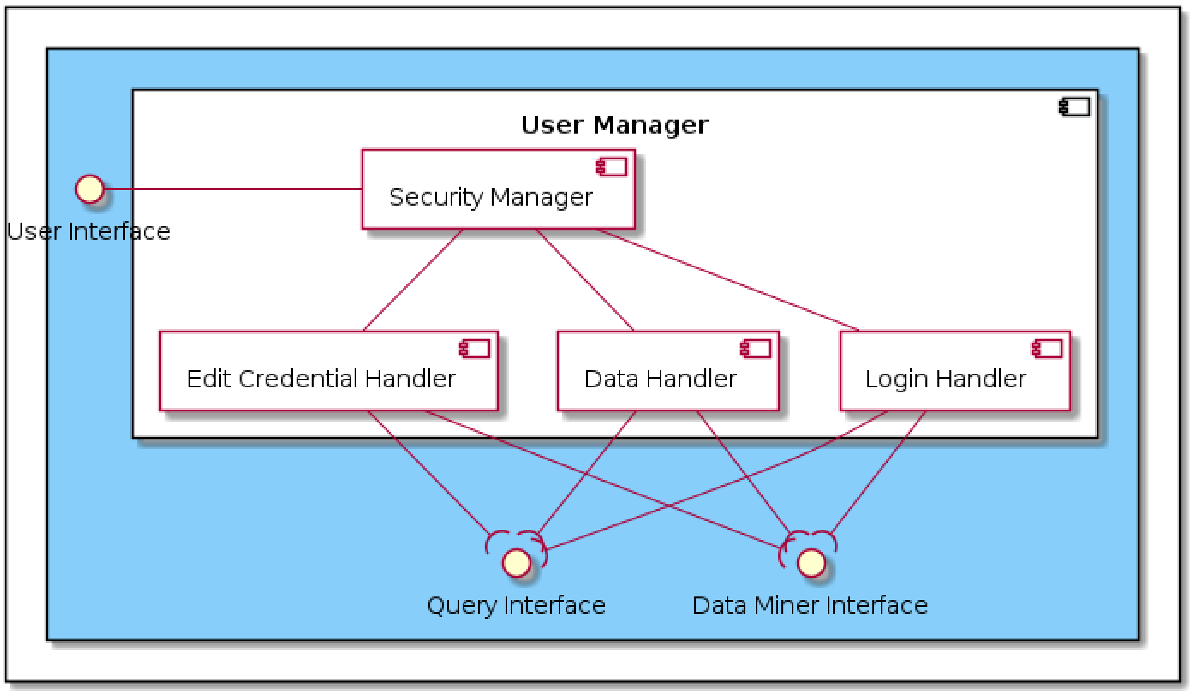
The application layer implements the business logic of the S2B. It is the tier that connects the presentation layer with the data layer, coordinating the flow of information between them and keeping a persistent connection with the DBMS. All the components are now described with their main functionalities. Further details are given in following sections, describing the methods related to interfaces and internal to the components.

2.2.2.1 User Manager

This is the component that manages all operations linked with users' accounts such as handling operations on data and granting authorization. The user manager can obtain data either by directly querying the database through the query manager or by means of the data miner obtaining pre processed data.

Its sub-components and their respective functionalities are listed below:

- Login Handler: it handles account authentication by checking credentials, querying the database through the Query Manager;
- Edit Credentials Handler: it allows users to change their credentials;
- Data Handler: it allows user to retrieve production data based on authorization levels
- Security Manager: It handles the encryption and decryption of data

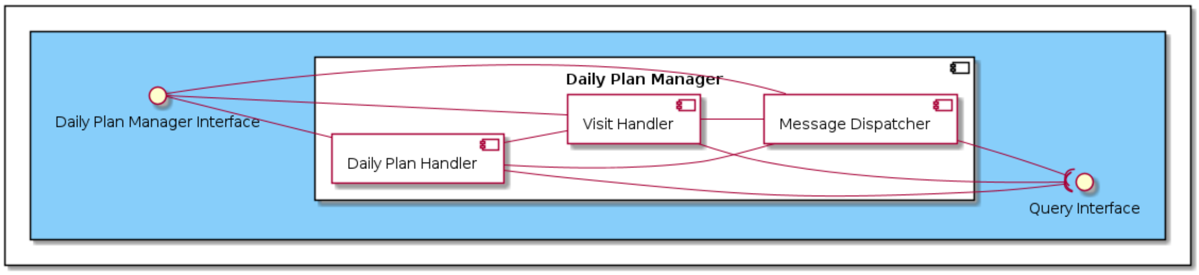


2.2.2.2 Daily Plan Manager

This component handles all operations linked to visits and daily plans for agronomists.

Its sub-components and their respective functionalities are:

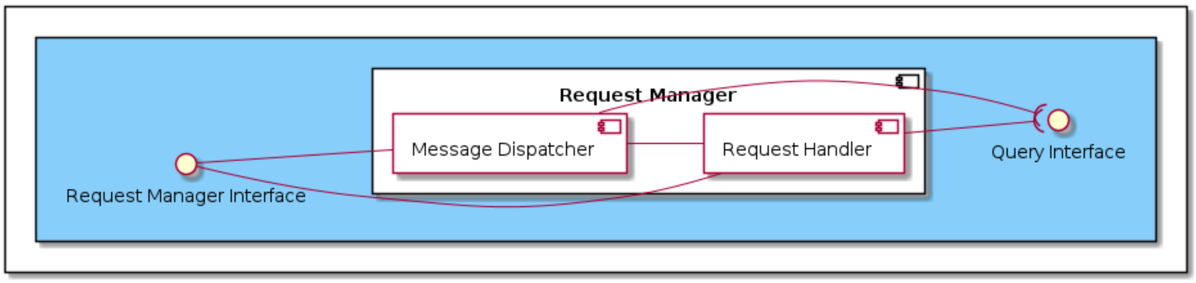
- Daily Plan Handler: It creates all the daily plans for agronomists.
- Visit Handler: It allows the creation of visits and evaluates their count for each farm in order to fulfill the annual visits requirement.
- Message Dispatcher: it allows the distribution of daily plans to all the agronomists.



2.2.2.3 Request Manager

This component handles all the operations linked to farmers and agronomist's help and suggestion requests.
Its sub-components and their respective functionalities are:

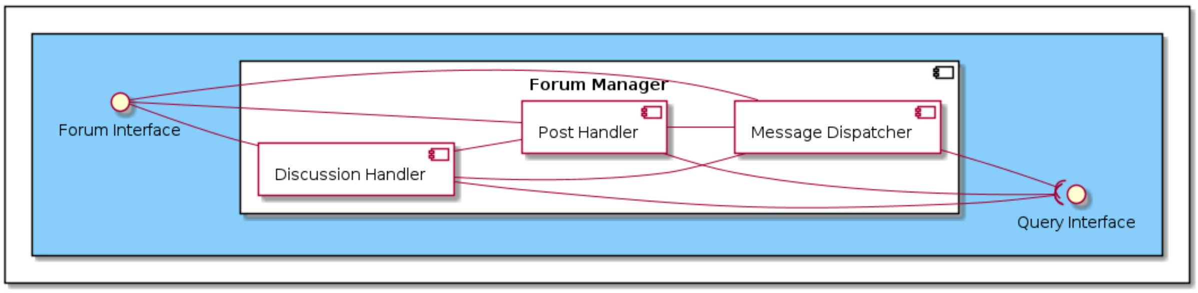
- Request Handler: It allows the creation of help/suggestions requests and of the associated replies.
- Message Dispatcher: It allows the distribution of the requests with the related replies to all the authorized users.



2.2.2.4 Forum Manager

This component handles all the operations linked to the farmers' forum.
Its sub-components and their respective functionalities are:

- Discussion Handler: It allows the creation of discussions by farmers and the association of posts for each discussion.
- Post Handler: It allows the creation of posts.
- Message Dispatcher: It allows the distribution of posts and discussions to all the authorized users.



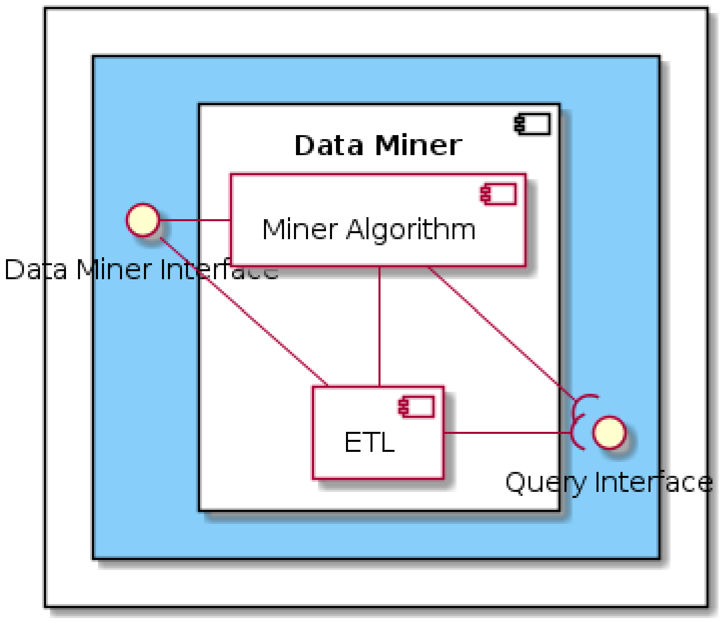
2.2.2.5 Weather Forecast Manager

This component is used to retrieve weather forecasts for a specific area from an external service through http calls. It is the unique component that interfaces with the Weather Forecast Service provided by Telangana government. The Weather Forecast Manager accesses local data through the Query Manager Interface to check that new information obtained is up to date. Then, if the check is successful it updates forecast data.

2.2.2.6 Data Miner

This component is used to mine statistics about performances, daily plans and users.
Its main components are:

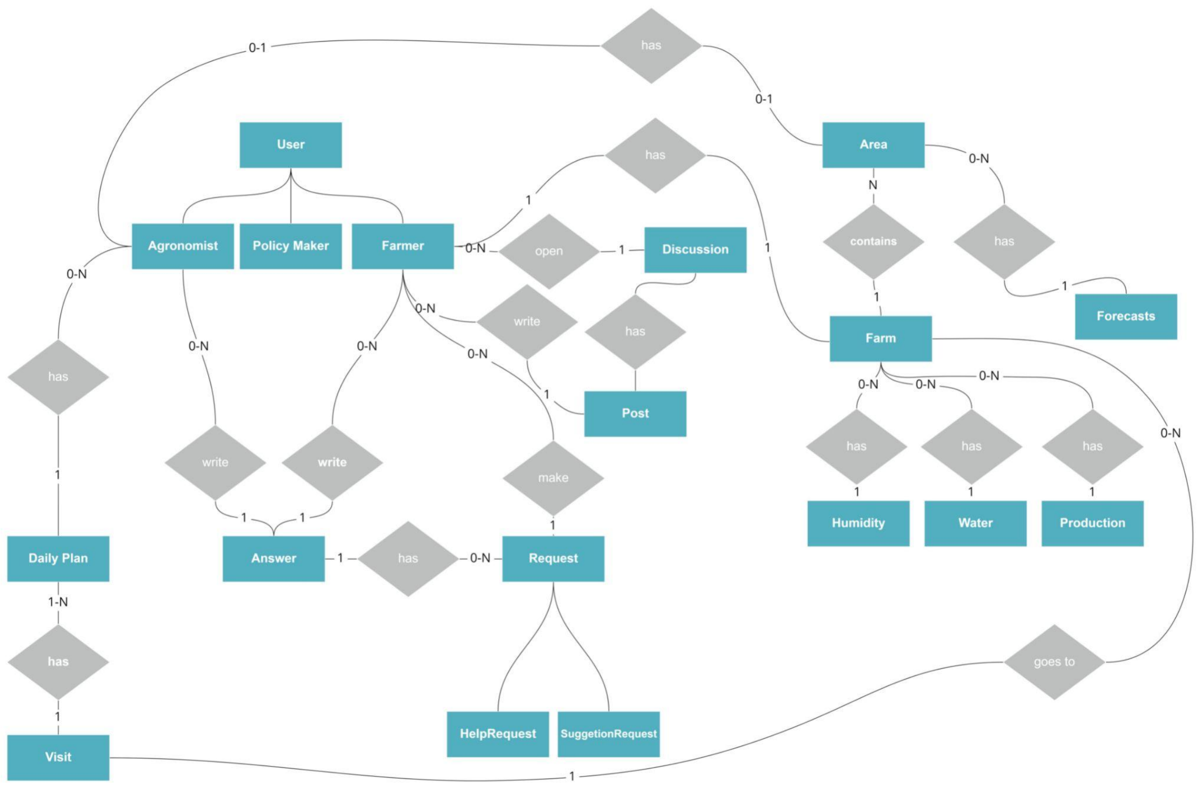
- ETL: It runs the ETL (Extract, Transform, Load) process. In short, it extracts the data from the database, transforms it by applying a series of rules and functions, and then loads the generated data into the Data Warehouse. In addition, it can send processed data to other components.
- Miner Algorithm: It provides to the ETL all the functions needed to transform the data coming from the database into new information.



2.2.2.7 Query Manager

This is the only component that interfaces with the data layer. It is linked to all the main components of the application server. They send requests to the Query Manager about reading, inserting or deleting data from the DBMS. Moreover it presents an external interface to allow the external sensors to send data (irrigation systems and humidity sensors).

2.2.3 Data access layer



The data layer is composed by a relational database that contains all the information. An entity relationship diagram has been designed to provide an overall view of the system.

2.2.4 External services and systems

2.2.4.1 External Services

- ***Weather forecast service:*** It is a web service that offers weather forecasts for telangana areas. The service is accessible through the use of specified APIs.

2.2.4.2 External Systems

- **Humidity of Soil Sensors:** They compose a multi-brand system of sensors deployed in the whole Telangana territory that is able to collect and send data relative to the humidity of soil. All data is following the SI metrics.

- **Irrigation System Sensors:** They compose a multi-brand system of irrigation implants deployed in the Telangana territory able to collect and send data relative to the amount of water used in a specified area. All data is following the SI metrics.

2.3 Deployment view

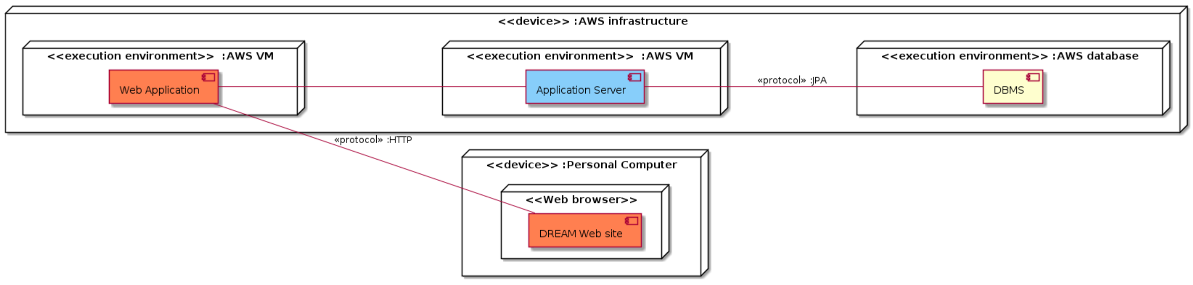
The deployment of the web application, for the server part, will be done exploiting the services provided by Amazon Web Services. AWS, if properly configured, handles server calls in a distributive way, and offers reliability, security, and scalability.

The benefits of using a service like AWS may be summarized as follows:

- ***Scalability on demand:*** AWS services are able to auto-scale according to the demands of the web application usage. In particular, since the DREAM system is developed as a web application, it is very easy to scale on demand.
- ***Flexibility:*** AWS is simple to adapt and customize.
- ***Load Balancing:*** AWS offers load balancing of the incoming traffic in order to distribute it equally among all the instances.

The AWS network will be organized as follows:

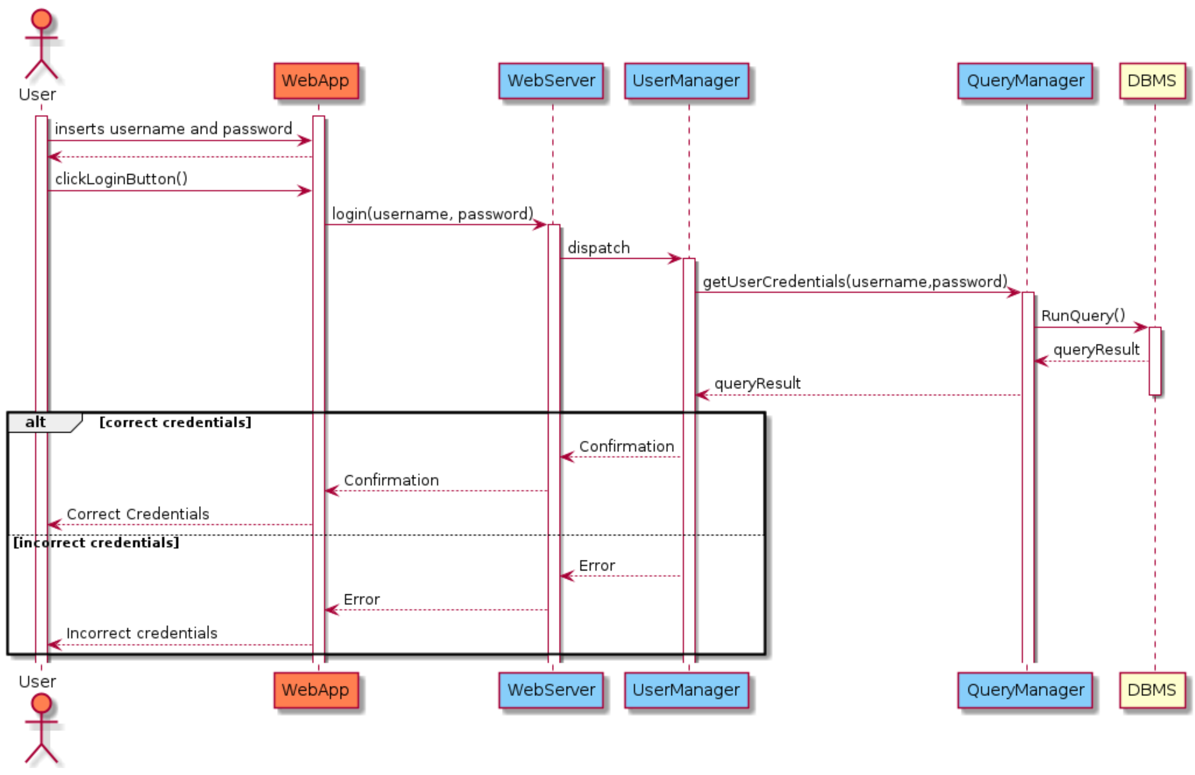
- ***DMZ and Web Server:*** It is realized thanks to the firewall offered by AWS, that also includes a DDoS protection added to mitigate possible attacks. The Web Server, conceivably composed of more machines, will be placed here managed by a (Public) load balancer which will be the only entry point of the system and will be used to properly distribute the incoming requests to different instances of the Web Server.
- ***Application Gateway:*** It makes routing decisions for distributing the traffic.
- ***Application Layer:*** The business methods are allocated on distributed machines inside an autoscaling group, a feature offered by AWS.
- ***Data Layer:*** For what concerns data management, we rely again on AWS with mySQL which is designed to provide scalability and high availability. All physical data are stored in multiple copies to ensure data redundancy, availability and reliability.


<<device>> :AWS infrastructure
<<execution environment>> :AWS VM
Web Application
<<execution environment>> :AWS VM
Application Server
«protocol» :JPA
<<execution environment>> :AWS database
DBMS
«protocol» :HTTP
<<device>> :Personal Computer
<<Web browser>>
DREAM Web site

2.4 Runtime view

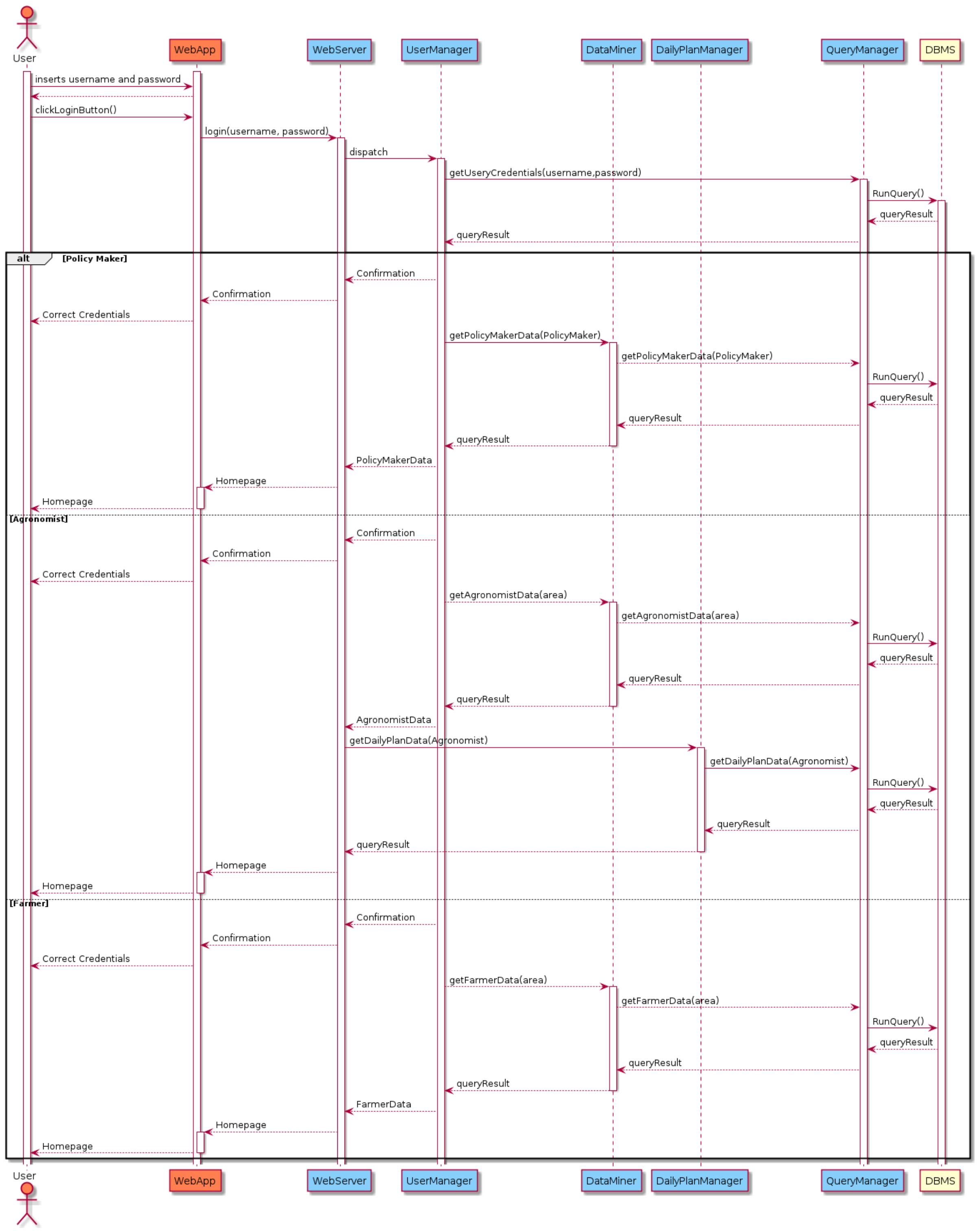
In this section, the sequence diagrams that represent the main interactions between the components of the system are presented. The methods are listed and described in section 2.5. All the components have been colored following the same scheme used in section 2.2 to highlight different tiers. The parameters of methods will be omitted in some cases to increase the readability of the sequence diagrams.

2.4.1 Login



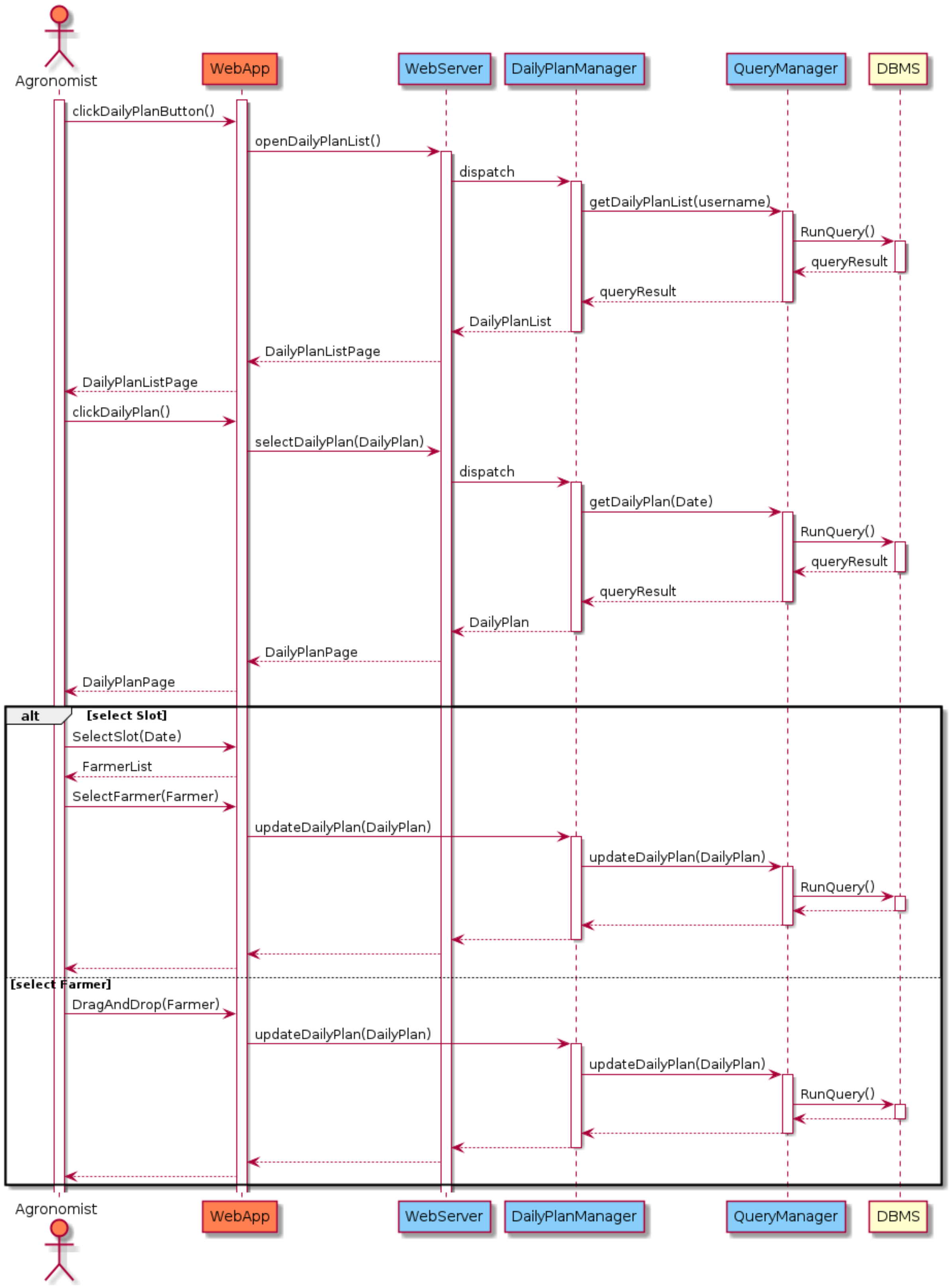
The diagram represents the login of a user. When the User Manager receives the data, it checks if there is an account associated to the given username and password by doing a query to the DBMS. If it is not, the system returns to the client an error, otherwise the login completes successfully and the system notifies the client the end of the process.

2.4.2 Get Homepage



After the completion of the login phase (shown for better reading in the sequence diagram), the loading of the page is differentiated on the basis of the user's authorization level. This level is rated by the User Manager. The collection of the proper data to the right user is done by the Data Miner, through the indications of the User Manager and communicating with the Query Manager. The Data Miner calls different methods based on the type of user (***getPolicyMakerData()***, ***getAgronomistData()***, ***getFarmerData()***). Such methods retrieve data with different spatial and temporal resolution, by the use of the Query Manager, that communicates directly with the DBMS interface with proper queries. To not add more complexity in the diagrams, these queries are omitted. In the agronomist's case even the Daily Plan Manager is invoked by the Web Server in order to get data about the Daily Plan.

2.4.3 Edit Daily Plan



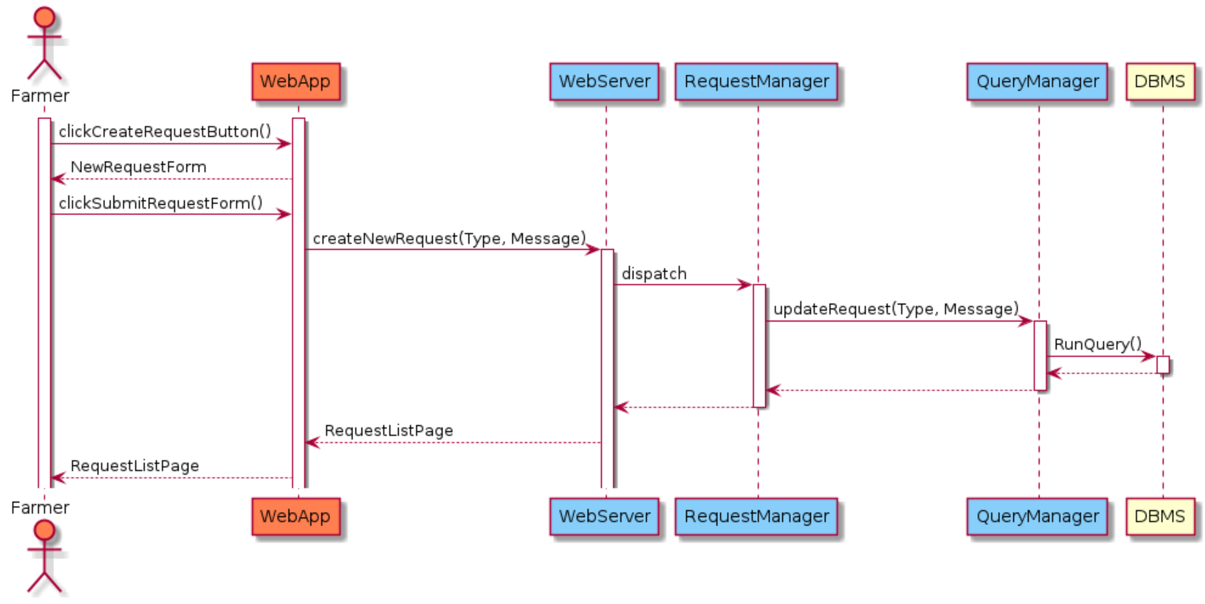
The act of editing a Daily Plan by an agronomist is organized in three distinct phases:

1. opening of the Daily Plan List - the agronomist clicks the Daily Plan Button and the system loads the list of Daily Plans; in particular the cascade of calls uses ***openDailyPlanList()*** method from WebApp, ***getDailyPlanList()*** method from Daily

Plan Manager, and queries from the Query Manager to the DBMS through the appropriate interface. To not add more complexity in the diagrams, these queries are omitted.

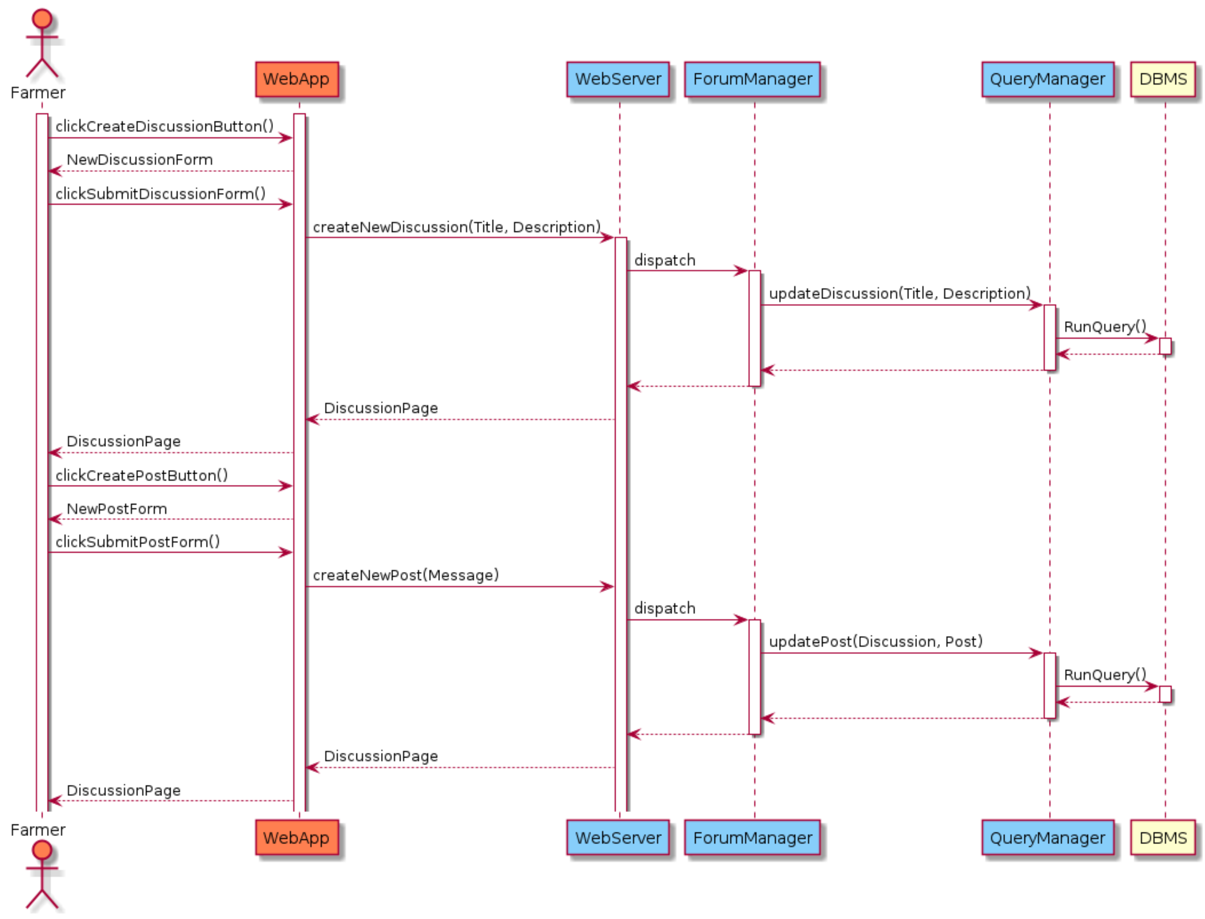
2. opening the a specific Daily Plan - the agronomist clicks a specific Daily Plan from a list and the system retrieves the relative list of visit; in particular the cascade of calls uses ***selectDailyPlan()*** method from WebApp, ***getDailyPlan()*** method from Daily Plan Manager, and queries from the Query Manager to the DBMS through the appropriate interface. To not add more complexity in the diagrams, these queries are omitted.
3. selecting either a slot or a farmer in order to change the order or the composition of the Daily Plan; in particular, after the proper selection from the agronomist, the method ***updateDailyPlan()*** is called from the webApp and the method updateDailyPlan is called from the DailyPlan Manager. At this point queries from the Query Manager to the DBMS through the appropriate interface. To not add more complexity in the diagrams, these queries are omitted.

2.4.4 Create Request



The flow of events necessary to create a request starts with a User (Agronomist or Farmer from the same area) that clicks the Create Request button. After the loading of NewRequestForm from the WebApp, the User fills the form and clicks the submit button. The cascade of calls is composed by a ***createNewRequest()*** method from the WebApp and from an ***updateRequest()*** from the Request Manager to the Query Manager. At this point queries from the Query Manager to the DBMS through the appropriate interface. To not add more complexity in the diagrams, these queries are omitted. Then the system uploads the RequestListPage over the browser.

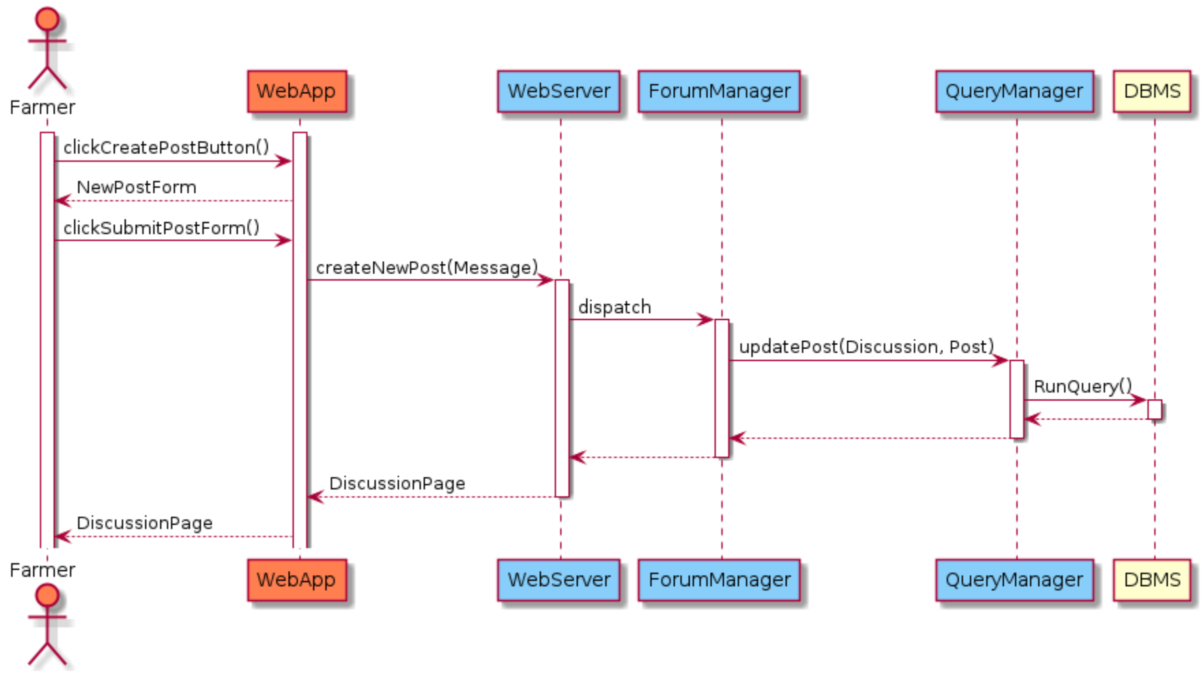
2.4.5 Create discussion



The flow of creation of a new discussion is organized in two distinct phases:

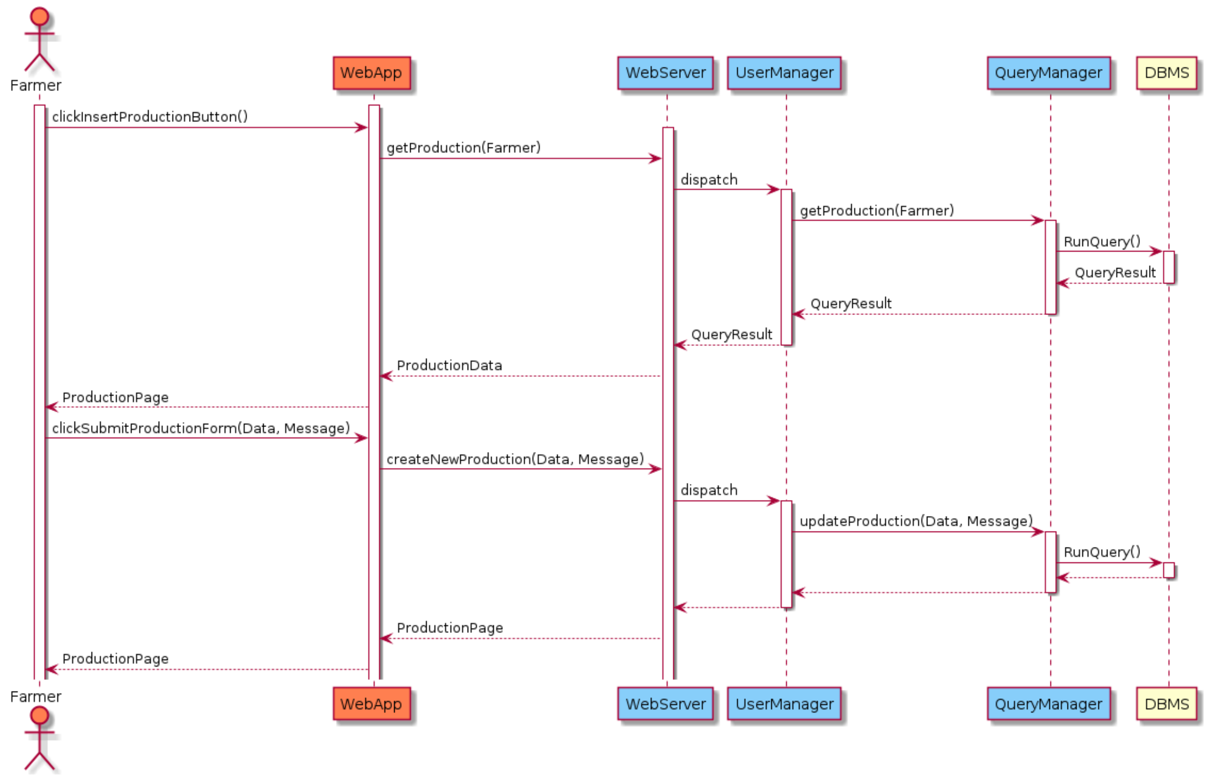
1. creation of a discussion - A farmer clicks the Create Discussion Button; the WebApp shows a form which, once completed, provides for the sending of data from the WebApp to the WebServer and from there to the Forum Manager, which takes care, through the Query Manager, of recording the data on the DBMS. The methods called are ***createNewDiscussion()*** and ***updateDiscussion()***, from the WebApp and from the Query Manager, respectively.
2. creation of the first post - A farmer clicks the Create Post Button; the WebApp shows a form which, once completed, provides for the sending of data from the WebApp to the WebServer and from there to the Forum Manager, which takes care, through the Query Manager, of recording the data on the DBMS. The methods called are ***createNewPost()*** and ***updatePost()***, from the WebApp and from the Query Manager, respectively.

2.4.6 Create Post



A farmer clicks the Create Post Button; the WebApp shows a form which, once completed, provides for the sending of data from the WebApp to the WebServer and from there to the Forum Manager, which takes care, through the Query Manager, of recording the data on the DBMS. The methods called are ***createNewPost()*** and ***updatePost()***, from the WebApp and from the Query Manager, respectively.

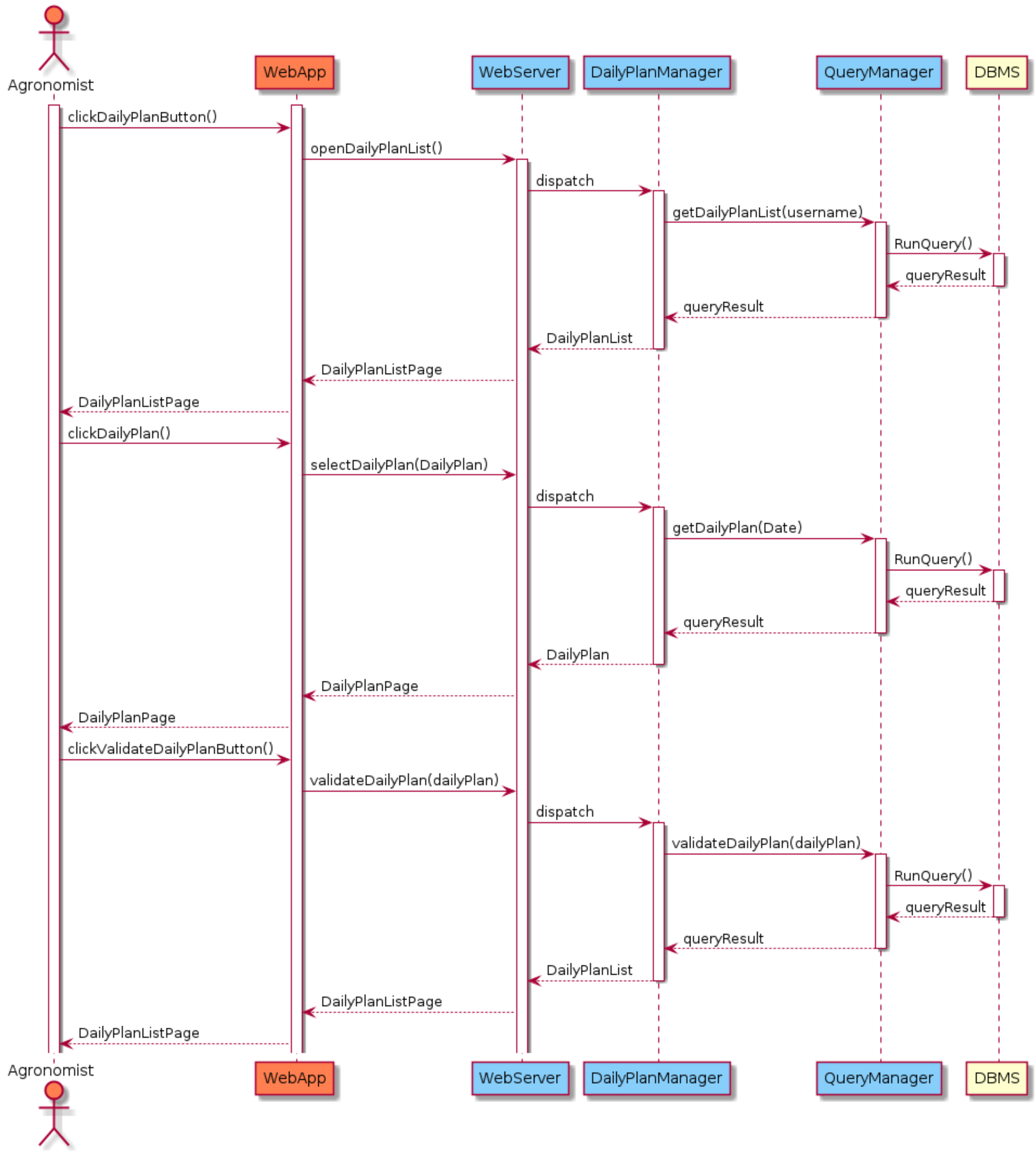
2.4.7 Farmer Inserts data about production and about problems encountered



The flow of events the let the production data to be stored is organized in two distinct phases:

1. A farmer clicks the Insert Production Button; the WebApp retrieves past production data from the WebServer and from there to the User Manager, which takes care, through the Query Manager, of getting the data from the DBMS. The methods called are ***getProduction()*** and ***getProduction()***, from the WebApp and from the Query Manager, respectively.
2. At this point the farmer fills the form with production data and comments and clicks the submit Button. The System provides for the sending of data from the WebApp to the WebServer and from there to the User Manager, which takes care, through the Query Manager, of recording the data on the DBMS. To not add more complexity in the diagrams, these queries are omitted.

2.4.8 Agronomist validates the daily plan



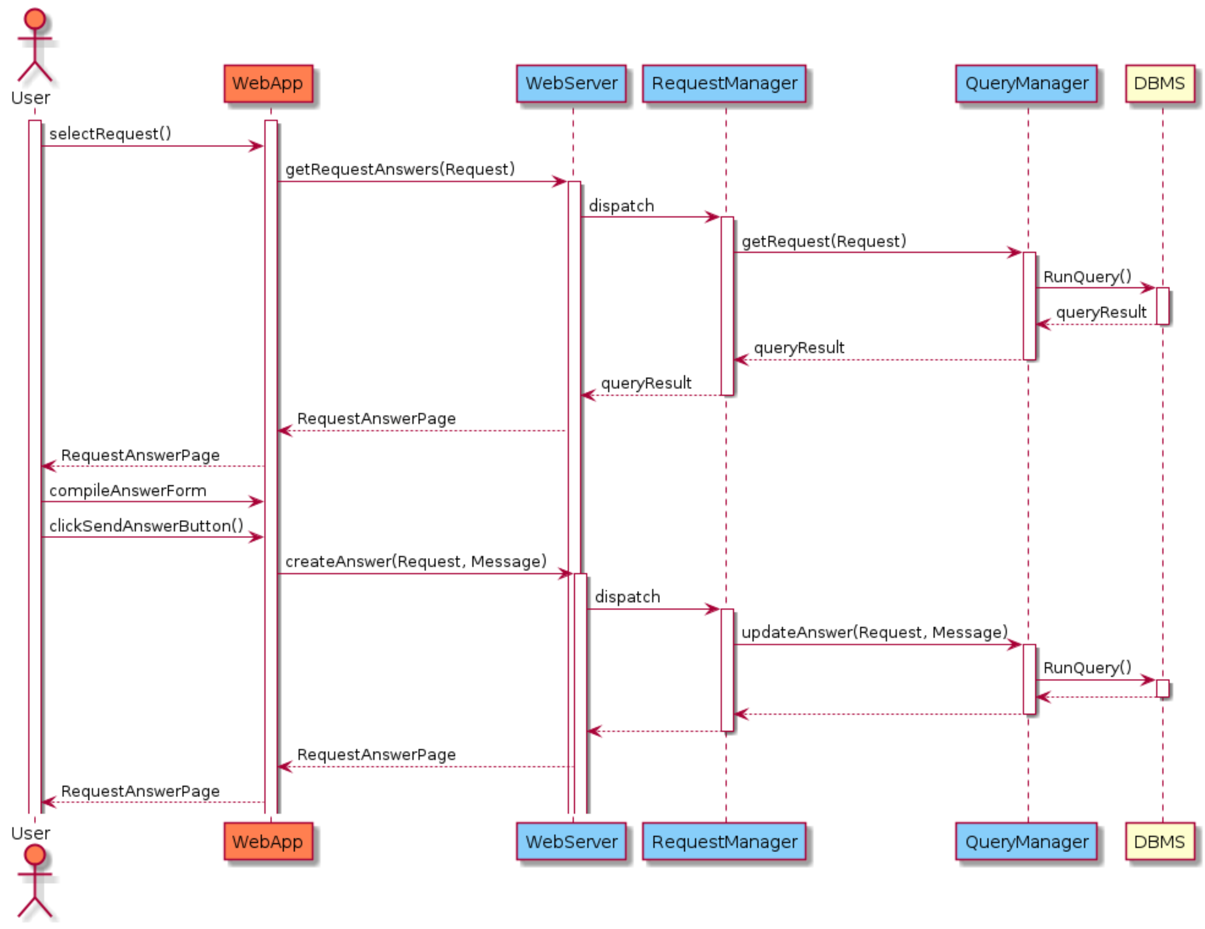
The act of validating a Daily Plan by an agronomist is organized in three distinct phases:

1. opening of the Daily Plan List - the agronomist clicks the Daily Plan Button and the system loads the list of Daily Plans; in particular the cascade of calls uses ***openDailyPlanList()*** method from WebApp, ***getDailyPlanList()*** method from Daily Plan Manager, and queries from the Query Manager to the DBMS through the appropriate interface. To not add more complexity in the diagrams, these queries are omitted.
2. opening the a specific Daily Plan - the agronomist clicks a specific Daily Plan from a list and the system retrieves the relative list of visit; in particular the cascade of calls uses ***selectDailyPlan()*** method from WebApp, ***getDailyPlan()*** method from Daily

Plan Manager, and queries from the Query Manager to the DBMS through the appropriate interface. To not add more complexity in the diagrams, these queries are omitted.

3. click the Validate Button - a ***validateDailyPlan()*** call from the WebApp to the WebServer activates the Query Manager with the ***validateDailyPlan()*** method; the DBMS is then called through the appropriate interface to edit the database. To not add more complexity in the diagrams, these queries are omitted.

2.4.9 Answer to help and suggestion requests

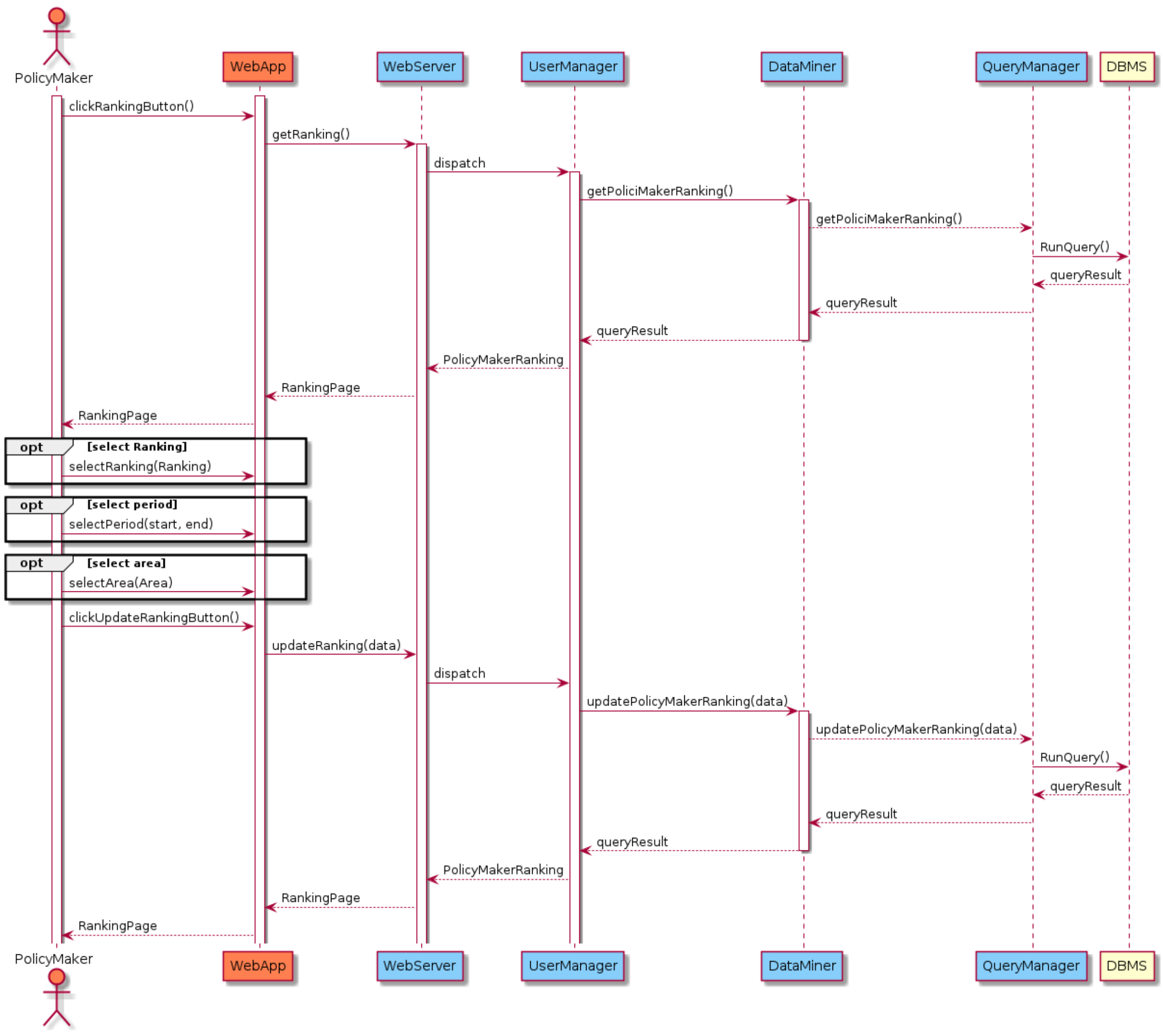


The flow of events necessary to answer a request can be divided into two phases:

1. the User (Agronomist or Farmer from the same area) selects a Request from a list and the system shows the sequence of answers already inserted - the cascade of calls is composed by a ***getRequestAnswer()*** method from the WebApp and from an ***getRequest()*** from the Request Manager to the Query Manager. At this point queries from the Query Manager to the DBMS through the appropriate interface. To not add more complexity in the diagrams, these queries are omitted. Then the system uploads the RequestAnswerPage over the browser.
2. the User inserts data into the proper form and DREAM updates the RequestAnswerPage - the cascade of calls is composed by a ***createAnswer()*** method from the WebApp and from an ***updateAnswer()*** from the Request Manager

to the Query Manager. At this point queries from the Query Manager to the DBMS through the appropriate interface. To not add more complexity in the diagrams, these queries are omitted. Then the system uploads the RequestAnswerPage over the browser.

2.4.10 Policy Maker Updates Ranking

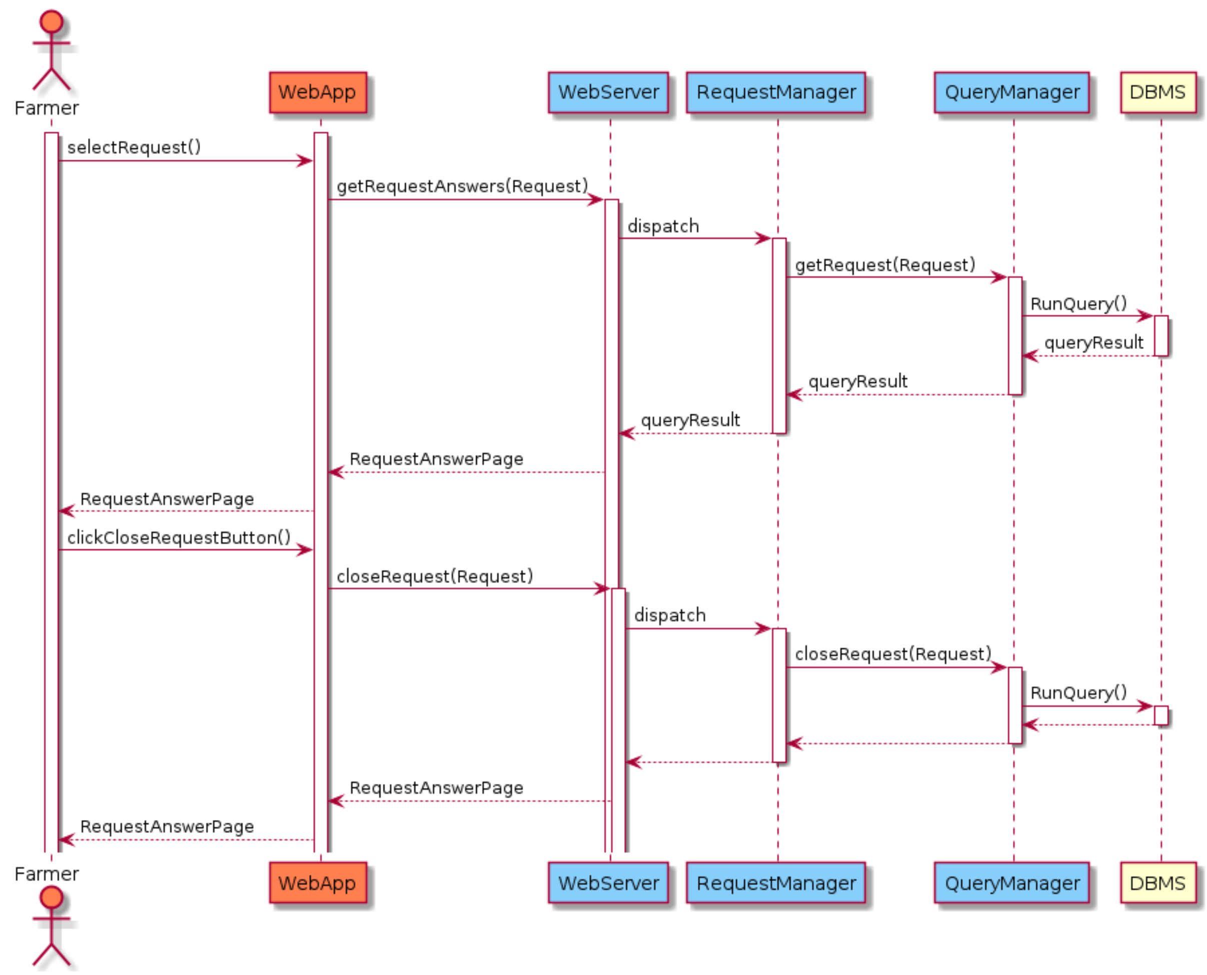


The flow of events necessary to Policy Maker to change the ranking of statistics is characterized by two steps:

1. the Policy Maker clicks the ranking button in the main page and load the Page - the WebApp calls the ***getRanking()*** method that starts a cascade of subsequent calls from the WebServer the UserManager, from the UserServer to the DataMiner (through the ***getPolicyMakerRanking()*** method) and from the DataMiner to the QueryManager (through the ***getPolicyMakerRanking()*** method). At this point queries from the Query Manager to the DBMS through the appropriate interface. To not add more complexity in the diagrams, these queries are omitted. Then the system uploads the RankingPage over the browser.

2. the Policy Maker choices between non exclusive options (selection of the type of ranking, selection of the geographical area of interest and/or selection of a specific time period) - once selection are performed the WebApp starts a sequence of calls of methods from the WebServer, the User Manager, the Data Miner and the Query Manager (the cascade of methods comprehends ***updateRanking()*** from WebApp, ***updatePolicyMakerRanking()*** from User Manager, ***updatePolicyMakerRanking()*** from Data Miner and queries from Query Manager omitted in the diagram).

2.4.11 Closure of a request

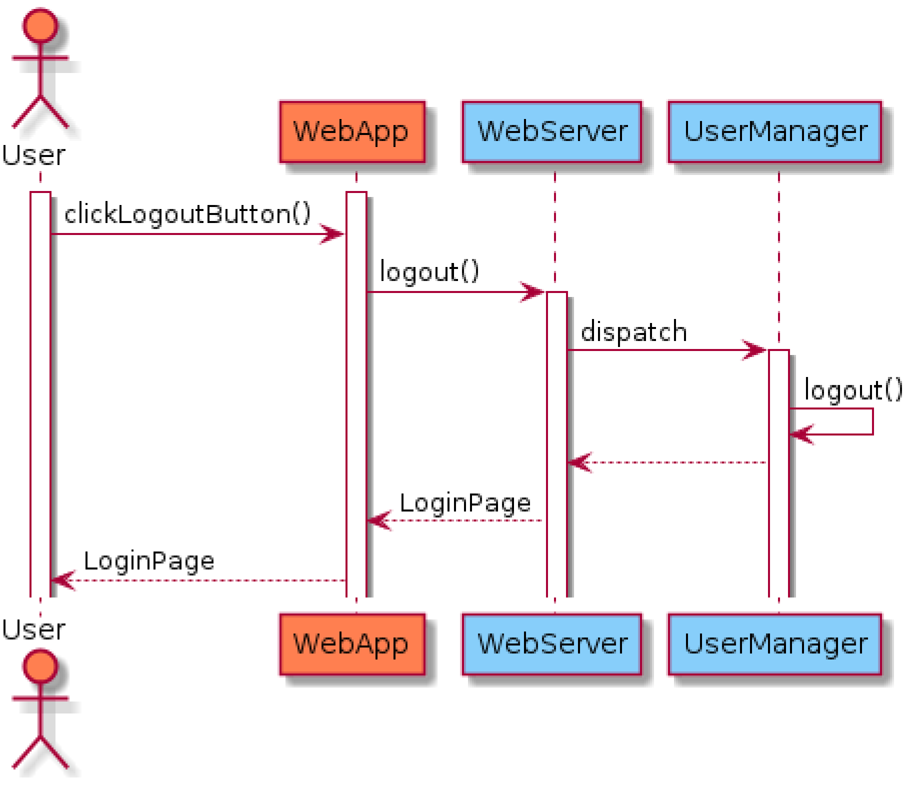


The flow of events necessary to close a request can be divided into two phases:

1. The Farmer selects a Request from a list and the system shows the sequence of answers already inserted - the cascade of calls is composed by a ***getRequestAnswer()*** method from the WebApp and from an ***getRequest()*** from the Request Manager to the Query Manager. At this point queries from the Query Manager to the DBMS through the appropriate interface. To not add more complexity in the diagrams, these queries are omitted. Then the system uploads the RequestAnswerPage over the browser.
2. The Farmer clicks the Close Request Button and DREAM updates the RequestAnswerPage - the cascade of calls is composed by a ***createAnswer()***

method from the WebApp and a ***closeRequest()*** method from the Request Manager to the Query Manager. At this point queries from the Query Manager to the DBMS through the appropriate interface. To not add more complexity in the diagrams, these queries are omitted. Then the system uploads the RequestAnswerPage over the browser.

2.4.12 Logout



The diagram represents the logout of a User. The User selects the Logout button and the system starts a cascade of calls to methods that let WebApp to redirect User to the Login Page.

2.5 Component interfaces

In this section, details about methods used in the sequence diagrams (presented in section 2.4) are provided. Moreover, some other methods have been added as they will be part of the software to be developed even if not seen in the runtime views.

2.5.1 Query Interface

getUserCredentials(username, password): retrieves from the db the user associated with a username and a password, returning an empty result if it doesn't exist.

getPolicyMakerData(policyMaker): retrieves from the db the data associated with a Policy Maker.
getAgronomistData(agronomist): retrieves from the db the data associated with an Agronomist.
getFarmerData(farmer): retrieves from the db the data associated with a Farmer.
getDailyPlanData(agronomist): retrieves from the db the first scheduled visit from a Daily Plan of an agronomist, returning an empty result if it doesn't exist.
getDailyPlanList(Username): retrieves from the db the data relative to a list of Daily Plans of an agronomist, returning an empty result if it doesn't exist.
getDailyPlan(Date): retrieves from the db the data relative to the list of visits of a Daily Plan of an agronomist, returning an empty result if it doesn't exist.
validateDailyPlan(DailyPlan): updates the db changing a status of a dailyPlan
updateDailyPlan(DailyPlan): updates the db changing the list of visits within a dailyPlan
getRequest(Request): retrieves from the db the message of a request, returning an empty string if it doesn't exist.
updateRequest(Answer): updates the db inserting a new answer into a request flow, if the request it's not already in a closed state.
updateRequest(type, message): updates the db inserting a new request into the system.
closeRequest(Request): updates the db closing a request already into the system.
updateDiscussion(title, description): updates the db inserting a new discussion into the system.
updatePost(discussion, post): updates the db inserting a new post to a discussion into the system.
getProduction(): retrieves from the db the production data of a farmer, returning an empty result if it doesn't exist.
updateProduction(data, message): updates the db inserting new production data to the list of farmer productions.
getPolicyMakerRanking(): retrieves from the db the performance data according to the standard ranking.
updatePolicyMakerRanking(data): retrieves from the db the performance data according to a particular ranking.
getMeanProduction(product, data): retrieves from the db the mean production data for a particular product, returning an empty result if it doesn't exist.
getProductionToWaterComsuptionRatio(data): retrieves from the db the mean production to water consumption ratio for a particular farmer and a specific product, returning an empty result if it doesn't exist.
getVisits(Farmer, period): retrieves from the db the visits from agronomists to a particular farmer in a specific period, returning an empty result if it doesn't exist.
getAgronomist(Farmer): retrieves from the db the agronomist associated with a particular farmer, returning an empty result if it doesn't exist.
getAgronomist(Area): retrieves from the db the agronomist associated with a single area, returning an empty result if it doesn't exist.
getFarmerList(Agronomist): retrieves from the db the list of farmers associated with a single agronomist, returning an empty result if it doesn't exist.
updateCredential(User, username, password): change the username and/or the password of a single user.
updateWeatherData(data): updates the db inserting a new Weather Forecast record for a specified area and period of time into the system.

getWeatherData(area): retrieves from the db the updated weather forecast record associated with a single area, returning an empty result if it doesn't exist.

2.5.2 Data Miner Interface

getPolicyMakerData(policyMaker): retrieves from the Query Manager the data associated with a Policy Maker according to a standard ranking.
getAgronomistData(area): retrieves from the Query Manager the data associated with an Agronomist.
getFarmerData(farmer): retrieves from the Query Manager the data associated with a Farmer.
getDailyPlan(data): retrieves from the Query Manager the list of visits of a single Daily Plan.
getPolicyMakerRanking(): retrieves from the Query Manager the data associated to a Policy Maker according to a prespecified ranking.
updatePolicyMakerRanking(data): retrieves from the Query Manager the data associated with a Policy Maker according to an inserted ranking.
getProductionDataRanking(Farmer, data): retrieves from the Query Manager the data associated with a farmer according to an inserted ranking.
getMissingFarmVisitsToday(Agronomist): retrieves from the Query Manager the list of not executed visits associated with an agronomist in a specific day.
getMissingFarmVisits(Agronomist): retrieves from the Query Manager the list of not executed visits associated with an agronomist in a specific day.

2.5.3 User Manager Interface

login(username, password): starts the process of login of a User.
getPolicyMakerRanking(PolicyMaker): retrieve the statistics for the Policy Maker according to a standard ranking.
updatePolicyMakerRanking(data): update the Policy Maker ranking according to specific parameters (period, etc).
logout(User): close the session for a user
updateCredential(User): start the update credential process

2.5.4 Daily Plan Manager Interface

getDailyPlanData(agronomist): retrieves from the Query Manager the first scheduled visit from a Daily Plan of an agronomist, returning an empty result if it doesn't exist.
updateDailyPlan(dailyplan)*:* starts the updating of a daily plan.
openDailyPlanList(agronomist): retrieves from the Query Manager the list of Daily Plans of an agronomist, returning an empty result if it doesn't exist.
selectDailyPlan(dailyplan): starts the process of opening the list of visits of Daily Plan.
validateDailyPlan(dailyplan): start the process that changes a Daily Plan status.

2.5.5 Forum Manager Interface

createNewDiscussion(title,description): creates a new discussion.
createNewPost(discussion, message): creates a new post in a specified discussion.

2.5.6 Request Manager Interface

createNewRequest(type,message): creates a new Request.
getRequestAnswer(request): retrieves from the Query Manager the list of answers to a specified request, returning an empty result if it doesn't exist.
createAnswer(request,message): creates an answer to an existing Request.
closeRequest(request): changes the status of a request from open to close.

2.5.7 Web Server Interface

login(username,password): allows the login of a user, dispatching the data to the User Manager.
openDailyPlanList(): retrieves the Daily Plan List for an agronomist, dispatching the data to the DailyPlan Manager.
selectDailyPlan(dailyplan): retrieves a Daily Plan from a Daily Plan List of an agronomist, dispatching the data to the DailyPlan Manager.
updateDailyPlan(dailyplan): updates a Daily Plan, dispatching the data to the DailyPlan Manager.
createNewRequest(type, message): creates a new Request, dispatching the data to the Request Manager.
createNewDiscussion(type,description): creates a new Discussion, dispatching the data to the Forum Manager.
createNewPost(message): creates a new post, dispatching the data to the Forum Manager.
getProduction(user): retrieves the production data of a farmer, dispatching the request to the User Manager.
createNewProduction(data,message): creates a new production entry, dispatching the data to the User Manager.
validateDailyPlan(dailyplan): updates a Daily Plan status, dispatching the data to the DailyPlan Manager.
getRequestAnswer(request): retrieves the answer to a request, dispatching the request to the Request Manager.
createAnswer(request,message): creates a new answer to a specified request, dispatching the data to the Request Manager.
getRanking(): retrieve data for Policy Maker data according to a standard Ranking.
updateRanking(data): retrieve data for Policy Maker data according to a specific Ranking.
closeRequest(request): updates a Request status, dispatching the data to the Request Manager.
logout(): allows the logout of a user, dispatching the request to the User Manager.

2.5.8 Irrigation System Interface

updateIrrigationData(id, data): inserts data sent from an irrigation system into the db, after checking the data format.

2.5.9 Humidity of Soil Interface

updateHumidityData(id, data): inserts data sent from a soil sensor into the db, after checking the data format.

2.5.10 Weather Forecast Manager Interface

getForecastData(area, Date): retrieves from an external forecast Service the updated weather forecast record associated with a single area, returning an empty result if it doesn’t exist.
checkForecast(Forecast): checks if a forecast is updated and returns a boolean result equal to false if the forecast is outdated.

2.6 Selected architectural styles and patterns

2.6.1 Architectural styles

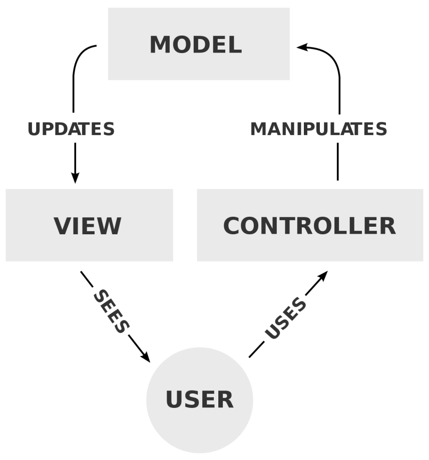
The web application will use the java servlets architecture with HTTP interactions. JEE offers all the features needed to exploit and enhance at best this type of architecture.

It should be noted that in the Runtime diagrams in section 2.4 the methods between the Web application and the Web server are only represented as interactions. However, in practice, all those methods will be mapped to the HTTP methods (GET, POST, PUT, DELETE).

2.6.2 Patterns

- ***MVC Design pattern:***

Model-View-Controller (MVC) is a pattern used in software engineering to separate the application logic from the user interface. As the name implies, the MVC pattern has three layers. The Model defines the business layer of the application, the Controller manages the flow of the application, and the View defines the presentation layer of the application.



- ***Adapter pattern:***

This pattern should be used in order to be able to communicate properly with all the different external APIs.

2.7 Other design decisions

2.7.1 Algorithms

In the following pseudocode, the ranking procedure, enforcing a lexicographic order, is shown. The code is divided in two steps to increase readability.

```
def next_lexicographic_permutation(x)
    return false if x.length<2
    # Step 1
    # Find the highest index(i) where x[i] < x[i+1]. If one isn't found,
then we are at the last lexicographic permutation.
```

```
     i = x.length-2

      while i >= 0 do
        if x[i] < x[i+1]
          break
        else
          i -= 1
        end
        return false if i < 0
      end
```

#Step 2
# Find the highest index j > i where x[j] > x[i]. We would not have been able to get to this spot if Step 1 above was not satisfied.

```
      j = x.length - 1

      while j > i do
        break if x[j] > x[i]
        j -= 1
      end
```

# Step 3
# Swap x[i] with x[j]

```
      swap(x, i, j)
```

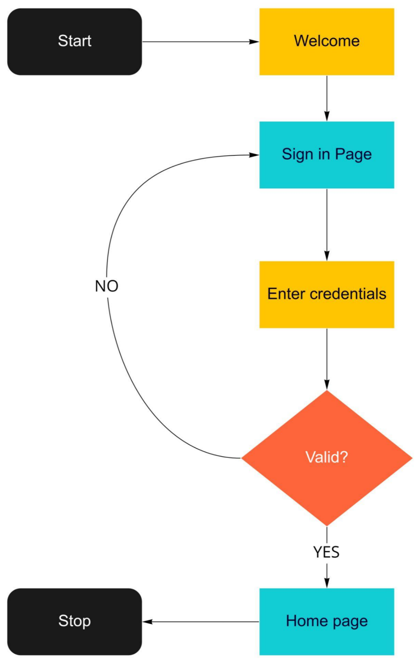
# Step 4
# Reverse the subarray from i+1 through to last element of array x

```
      reverse(x, i+1)

      return x
    end


    def swap(arr, x, y) # exchanges elements in the array in-place
(destructively).  This changes the original array
      temp = arr[x]
      arr[x] = arr[y]
      arr[y] = temp
    end


    def reverse(arr, i)
      if i>=arr.length-1 # either we are at the last element, or i is too
high to exist in the array
        return
      end

      j = arr.length - 1
      while i < j do
        swap(arr, i, j) # using our #swap helper method
        i+=1
        j-=1
      end
    end
```

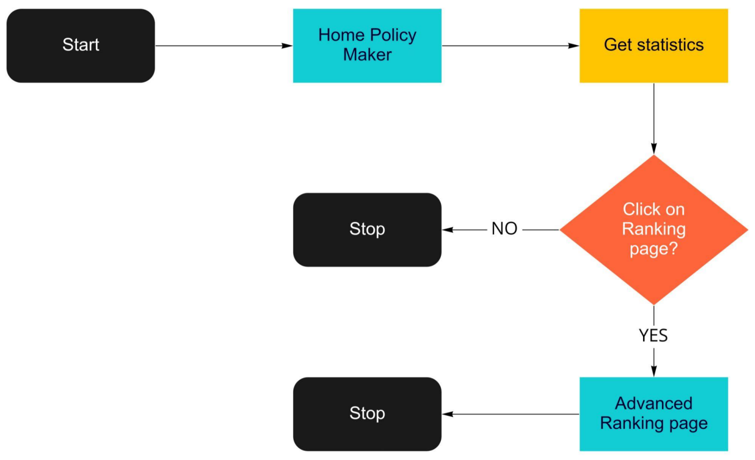
3. USER INTERFACE DESIGN

In the RASD we have shown a series of mockups that show the web application pages. Here we will extend the UI by providing the navigation flow between the screens by including a flow chart graph in this section.

3.1 Registration and Login flow

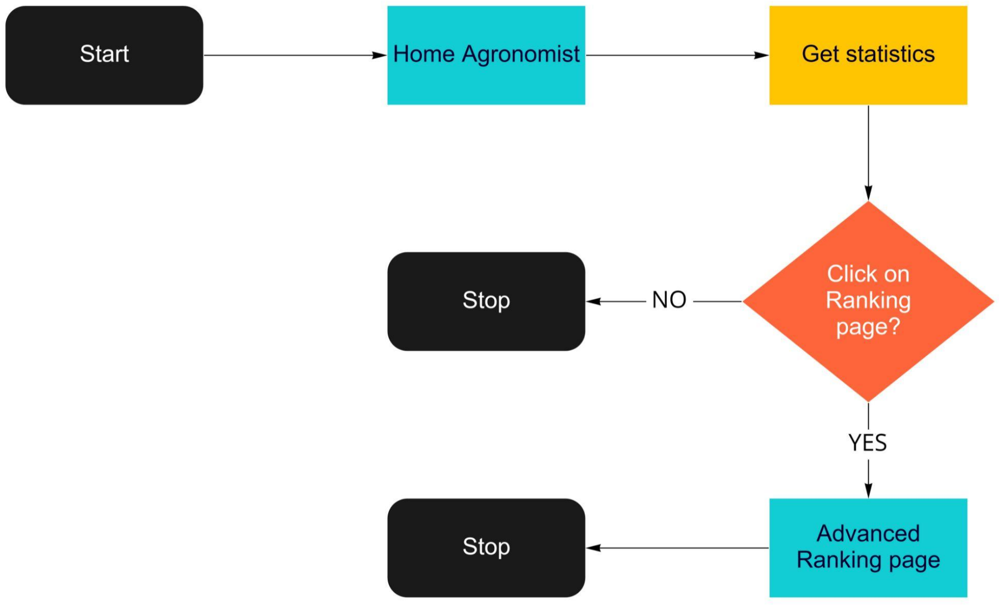


3.2 Policy makers flow

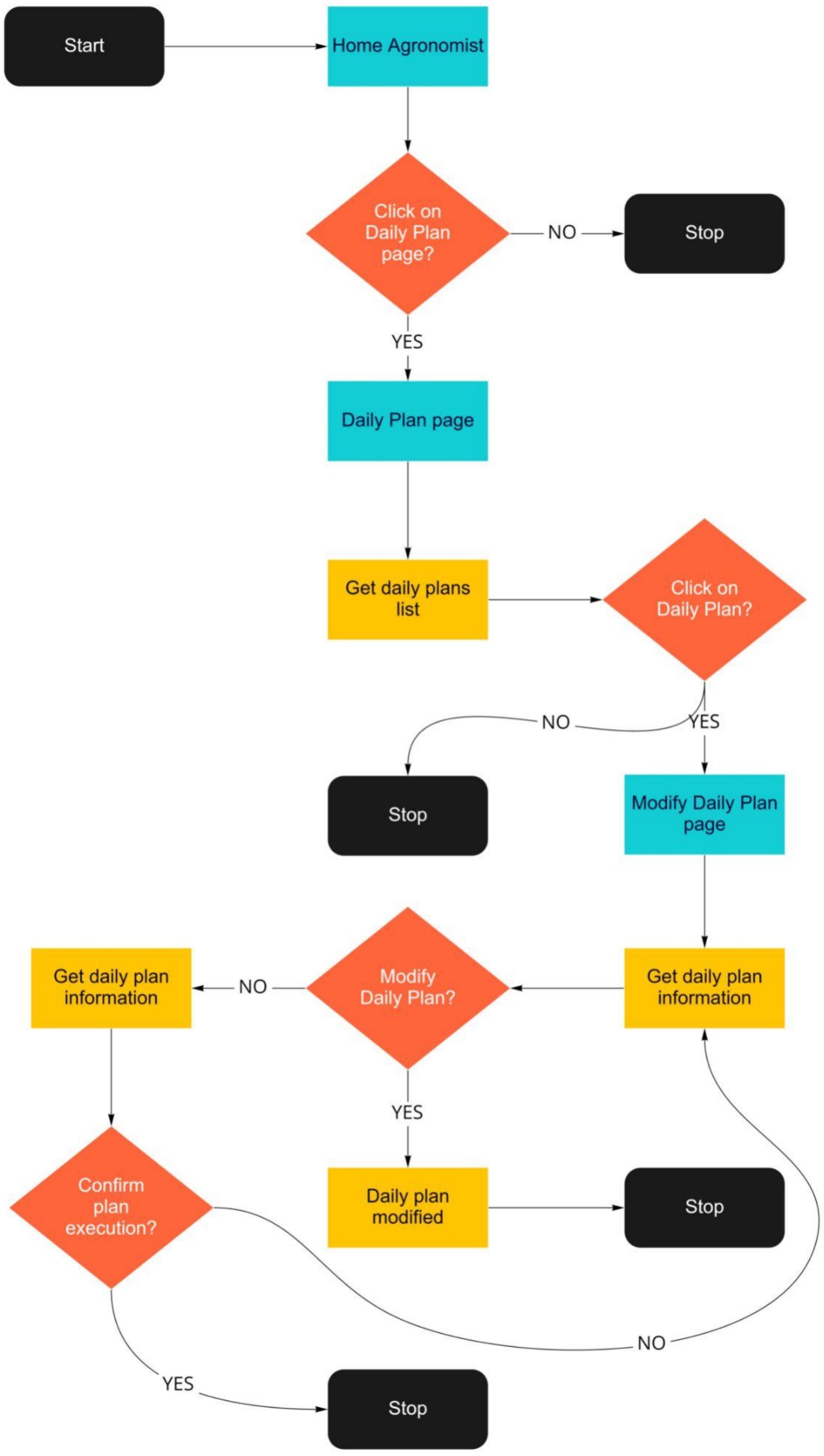


3.3 Agronomists flow

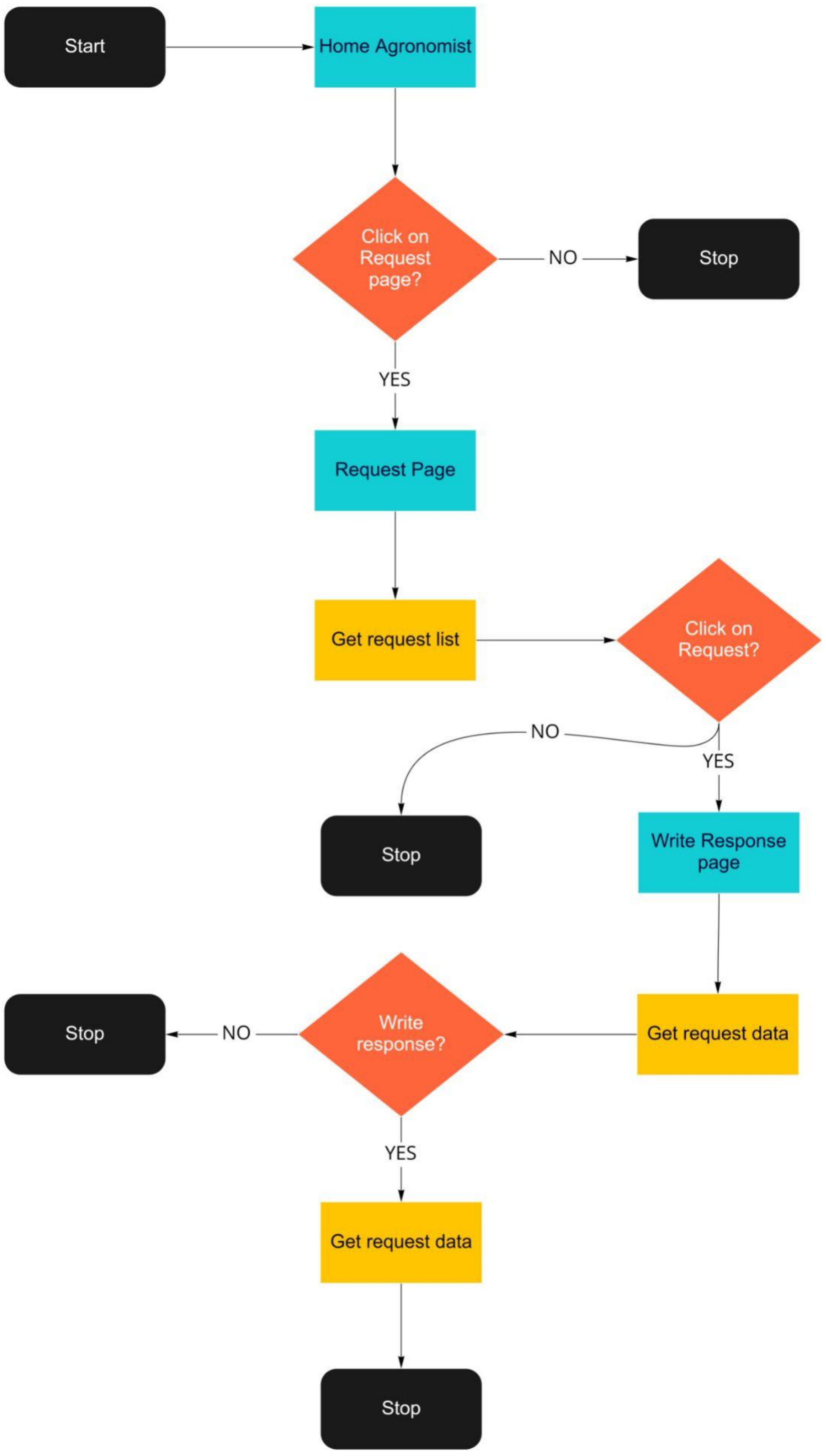
3.3.1 Ranking page



3.3.2 Daily plan management

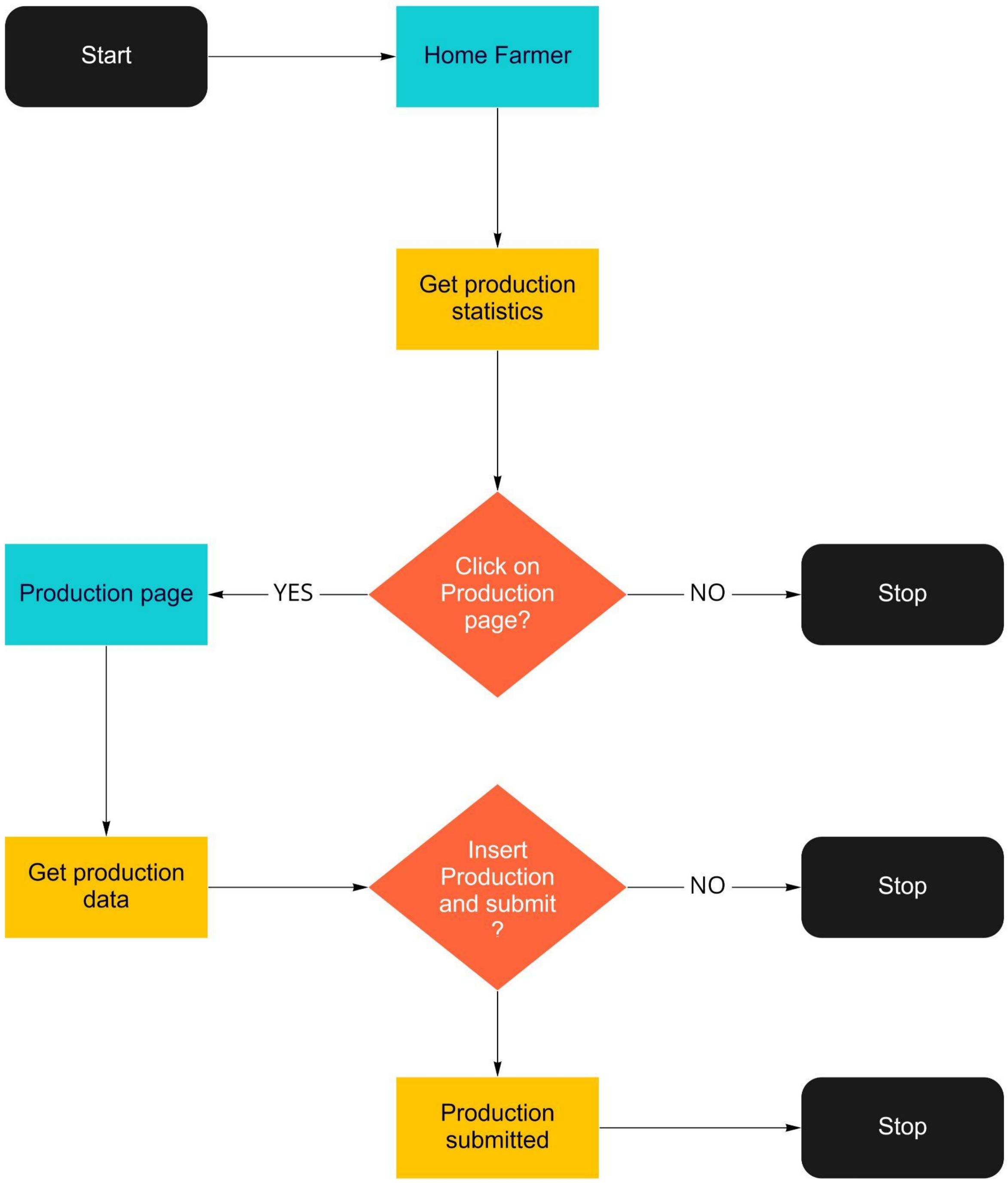


3.3.3 Requests management

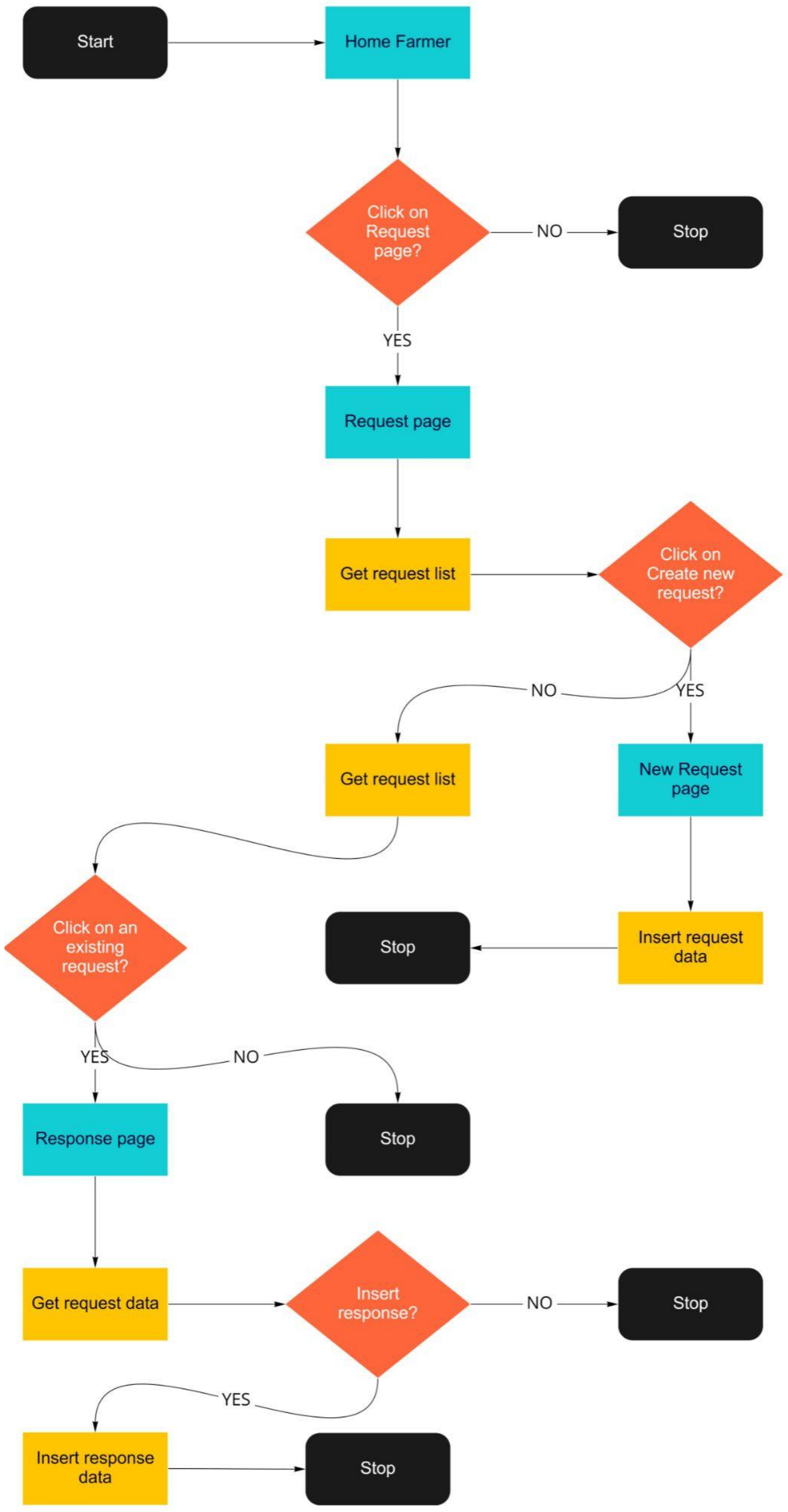


3.4 Farmers flow

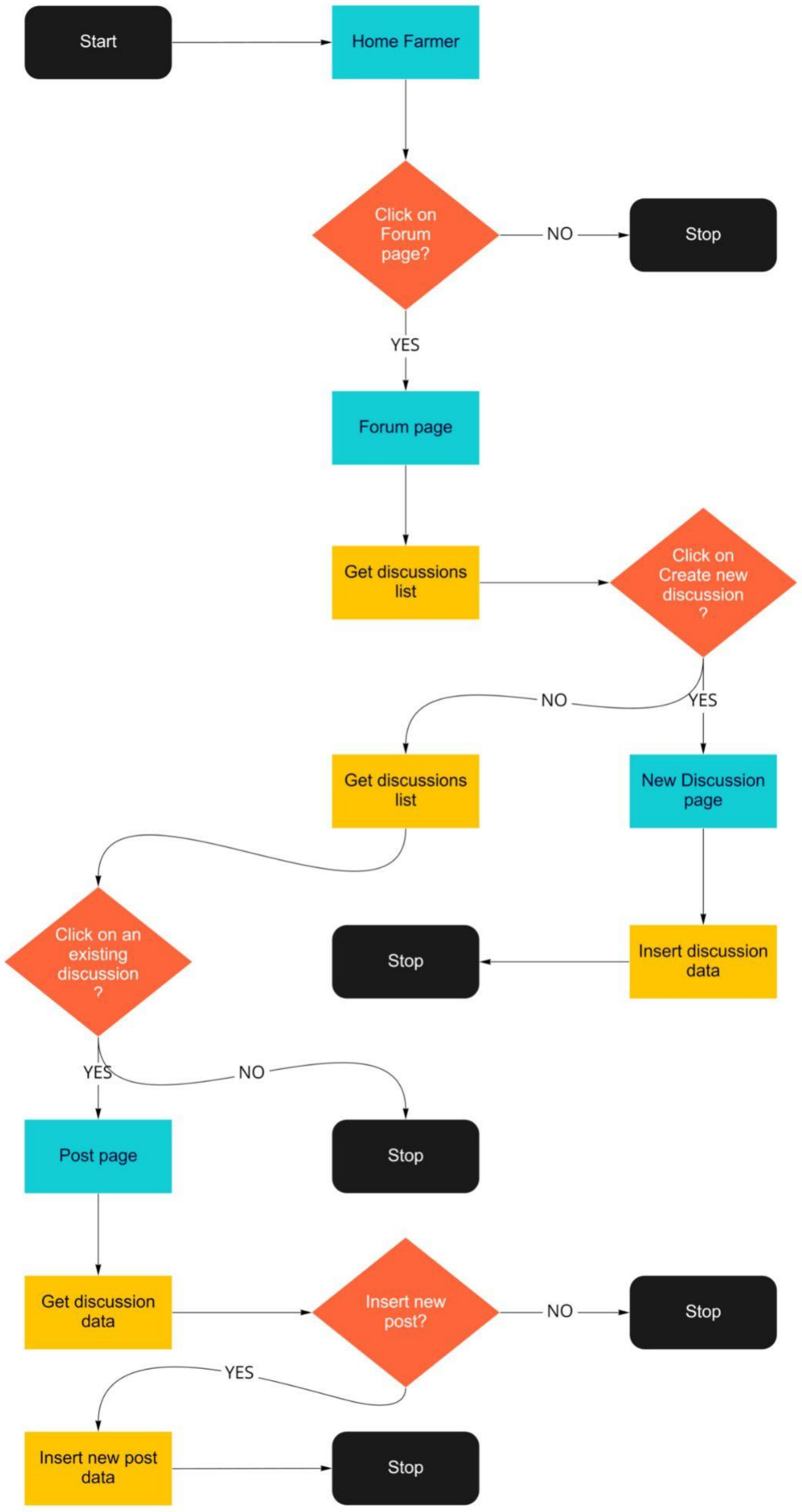
3.4.1 Production data management



3.4.2 Requests Management



3.4.3 Forum management



4. REQUIREMENTS TRACEABILITY

4.1 Table

In the following table the requirements described in the RASD (reported in the following page) are matched with the components described in Section 2.2.

		Components												
		Web Server	User Manager	Daily Plan Manager	Request Manager	Forum Manager	Weather Forecast Manager	Query Manager	Data Miner	Irrigation System	Humidity of Soil	Weather Forecast Service	DBMS	Web Application
Requirements	R1						✔	✔				✔	✔	
	R2							✔		✔			✔	
	R3							✔			✔		✔	
	R4	✔						✔					✔	✔
	R5	✔						✔					✔	✔
	R6	✔	✔					✔					✔	✔
	R7	✔	✔						✔				✔	✔
	R8		✔						✔					
	R9	✔	✔					✔					✔	✔
	R10	✔	✔	✔	✔	✔								✔
	R11						✔	✔	✔	✔	✔		✔	
	R12	✔	✔											✔
	R13	✔	✔											✔
	R14	✔							✔					✔
	R15	✔						✔					✔	
	R16	✔	✔					✔					✔	
	R17	✔	✔					✔					✔	
	R18	✔							✔					✔
	R19	✔	✔					✔					✔	
	R20	✔	✔					✔					✔	✔
	R21	✔	✔					✔					✔	✔
	R22	✔	✔					✔					✔	✔
	R23	✔				✔		✔					✔	✔
	R24	✔				✔		✔					✔	✔
	R25	✔				✔		✔					✔	✔
	R26	✔				✔		✔					✔	✔
	R27				✔			✔					✔	
	R28	✔			✔									✔
	R29	✔			✔			✔					✔	✔
	R30				✔			✔					✔	
	R31	✔			✔									✔
	R32	✔			✔			✔					✔	✔
	R33	✔			✔			✔					✔	✔
	R34	✔			✔									✔
	R35	✔			✔									✔
	R36	✔	✔											✔
	R37	✔	✔					✔					✔	✔
	R38	✔	✔					✔					✔	✔
	R39	✔			✔			✔					✔	✔
	R40	✔			✔			✔					✔	✔
	R41	✔	✔					✔					✔	✔
	R42	✔	✔					✔	✔				✔	✔
	R43	✔	✔											
	R44							✔	✔				✔	
	R45			✔				✔					✔	
	R46	✔		✔				✔					✔	✔
	R47	✔		✔				✔					✔	✔
	R48	✔		✔				✔					✔	✔
	R49	✔		✔				✔					✔	✔
	R50			✔				✔	✔				✔	

4.2 List of Requirements

List of the requirements identified in the RASD:

R1	The system must be able to retrieve the data of the weather forecasts.
R2	The system must be able to receive data from water irrigation systems.
R3	The system must be able to receive humidity data from soil sensors.
R4	The system must not allow the upload of data with invalid format.
R5	The system must not allow the upload of null data.
R6	The system must grant access to the S2B only if the user successfully types his username and the matching password.
R7	The system must provide to policy makers quantitative data about farmers' production performances.
R8	The system must normalize production data for field dimension.
R9	The system must be able to receive production and problem data from farmers.
R10	The system must not allow the user to alter or delete data and inputs previously inserted either by himself or by others if it’s not explicitly allowed by the S2B.
R11	The system must be able to compute a ranking of the farmers based on production and environmental data.
R12	The farmers must be able to insert his production data.
R13	The farmers must be able to insert the problems he faced during the production.
R14	The policy maker must be able to filter the farmers’ production data for areas.
R15	The policy maker is authorized to see the production data of all farmers.
R16	The policy maker is authorized to see the production data of all areas and their agronomists.
R17	The agronomist is authorized to see the production data of all farmers in his area.
R18	The user must be able to filter the farmers’ production data for time periods.
R19	The farmer is authorized to see the production data of his farm.
R20	The farmer must be able to obtain local short and long term weather

	forecasts.
R21	The farmer must be able to obtain humidity of the soil values collected by the sensors associated with his own field.
R22	The farmer must be able to obtain his water consumption from the irrigation systems.
R23	The farmer must be able to create new discussions in the forum.
R24	The farmer must be able to visualize all discussions in the forum.
R25	The farmer must be able to visualize the comments of forum discussions.
R26	The farmer must be able to add a new comment to a forum discussion.
R27	The system must be able to send the help requests to all farmers and to the agronomist of the area.
R28	The farmer must be able to send help requests to farmers and to the agronomist of the area.
R29	The system must be able to show all the help requests previously sent or received and all their included comments to the user of the area.
R30	The system must be able to send the suggestion requests to all farmers and to the agronomist of the area.
R31	The farmer must be able to send suggestion requests to farmers and to the agronomist of the area.
R32	The system must be able to show all the suggestion requests previously sent or received and all their included comments to the user of the area.
R33	The farmer must be able to mark as closed any of his request messages that are not already closed.
R34	The system must prevent any user from further responding to a close request.
R35	The farmer cannot close a request before receiving at least one response comment.
R36	Farmers must be able to insert their production data and the problems they faced for that harvest.
R37	The agronomist must select an area which he wants to be responsible for.
R38	The agronomist must be able to change his area of influence.
R39	The user must be able to reply to help requests.
R40	The user must be able to reply to suggestion requests.

R41	The agronomist must be able to obtain local short and long term weather forecasts for his area of influence.
R42	The system must be able to retrieve all the daily plans of an agronomist.
R43	The agronomist is not allowed to see other agronomists' daily plans.
R44	The system must be able to compute the daily plans of an agronomist.
R45	The system must be able to keep the visits count for every farm.
R46	The agronomist must be able to confirm the execution of his daily plan.
R47	The system must be able to show the areas that have not been assigned to an agronomist yet.
R48	The agronomist must be able to modify his own daily plan.
R49	The agronomist must be able to specify the deviations from the original daily plan.
R50	The system must be able to update the daily plans to maintain the two visits per farm every year constraint.

The non functional requirements are granted by the design choices:

Requirements	Design Choice
Reliability and Availability	Amazon Web Services (e.g.: replicated components, multiple copies of the persistent storage, DDoS Protection)
Security	DMZ, DBMS privileges, HTTPS protocol, encryption of sensitive data and session based authentication.
Maintainability	Modular development and Amazon Web Services
Portability	Being a web application, the system is intrinsically portable.
Scalability	Being a web application, the system is highly scalable because new features can be added just by creating new HTTP endpoints and the corresponding business methods in the backend.

5. IMPLEMENTATION, INTEGRATION AND TEST PLAN

We decided to adopt a bottom up approach for the implementation and integration phase. This strategy involves the fact that lowest level modules will be tested first and then they will be further used to facilitate the testing of the newly developed higher level modules. The process continues until all components are developed. Even if an early working prototype is not possible with this technique, eventually, it offers more advantages than disadvantages during the integration phase and enhances code reusability.

5.1 Implementation plan

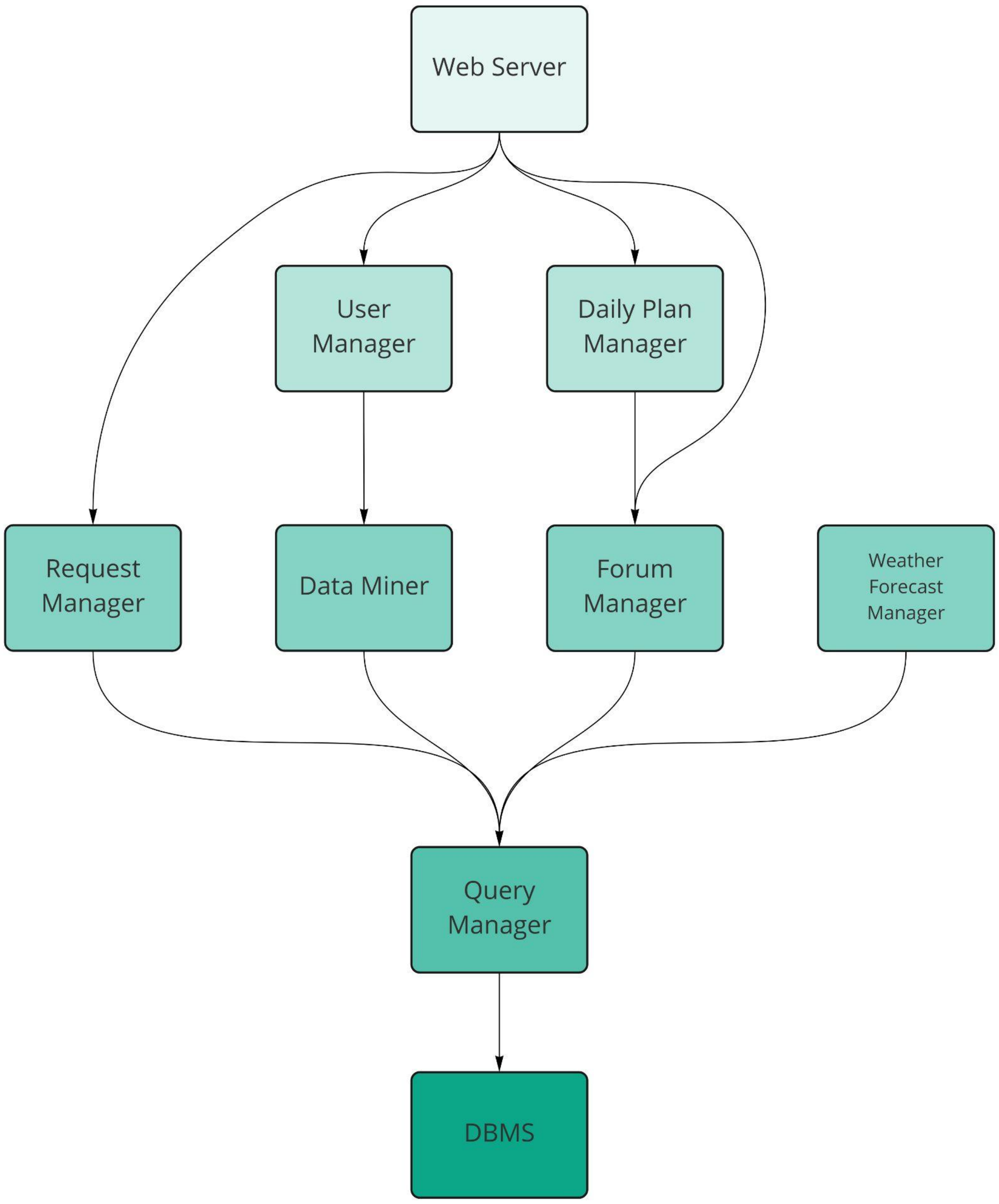
The implementation will be done from the lower components up to the top ones following the incremental bottom up approach. The DBMS won't be developed from scratch but an existing COTS will be chosen between a set of well tested and working implementations.

The components at the same level and with the same priority can be developed simultaneously. We included below the implementation order of the previously described components.

- The first level is composed of 4 components which are the **Data Miner**, the **Request Manager**, the **Forum Manager** and the **Weather Forecast Manager**. They are in the first level since they require only queries that are previously implemented by the Query Manager.
- The **User Manager** and the **Daily Plan Manager** realize the second level alone. These components rely on one or more components from the previous level.
- The **Web Server** is the last component to be developed since it is the only entry point for the client’s requests.

5.2 Integration plan

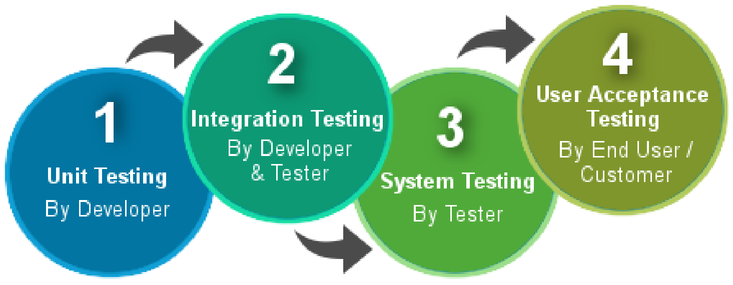
In the figure below, we included the flow of dependencies between components of the system. In the flow graph, the components pointed by the arrow are the ones required by the component from which the arrow starts.



Since the bottom-up approach has been adopted the system must be developed starting from the bottom to the top of the graph. That means that, during the integration process, the components above the level in development will be tested through driver modules, which are dummy programs that act as a substitute for the missing components.

5.3 Test plan

During software development mistakes can be made and thus, errors will emerge. To locate and fix those errors, different testing phases are required and all the phases need to be chained together.



1. Firstly, concurrently with the implementation phase, white box Unit Testing is performed in order to test each individual component using appropriate tools such as JUnit. Through the white box approach, the tester has a complete overview of the code, therefore allowing him to obtain the most suitable coverage. This type of testing can be completely automated.

2. Next, while the components are incrementally integrated into the system, the Integration Testing will be performed. For the system, we opted for the bottom up approach. Hence, testing will be started from low level components which will be incrementally integrated and coupled with other components at higher levels. If at a certain phase modules not yet developed are needed for testing then they will be simulated by drivers.In this way, it should be easier to localize any possible fault and no time is wasted waiting for all modules to be fully developed.

3. Once the system is completely integrated, it must be tested in its entirety to verify that functional and nonfunctional requirements are satisfied. System testing can be divided in subcategories, which are listed below.
    - **Functional testing**: It validates the software system against the functional requirements described in the RASD.
    - **Performance testing**: It determines how a system performs in terms of responsiveness and stability under a normal workload. Thanks to this it is possible to identify the presence of inefficient algorithms, possible query optimizations or network issues.
    - **Load testing**: It is performed to determine the system's behavior under peak workload.
    - **Recovery testing**: It is done to demonstrate that the software is reliable, trustworthy and that it can successfully recover from possible failures. It is performed by overwhelming the system resources or by removing them directly.

4. Eventually, the ***User Acceptance Testing (UAT)*** is performed by the end user or the client to verify or accept the software system before moving the software application

to the production environment. UAT is done in the final phase of testing after unit, integration and system testing is done. This phase is commonly known as beta testing.

6. EFFORT SPENT

6.1 Teamwork

Task	Hours
Initial briefing	1
Brainstorming	1
Purpose & Scope (and goals)	0.5
Architectural design from 2.1 to 2.4	2
Architectural design from 2.5. to 2.7	2
User interface design	0.5
Requirements traceability	1
Testing	1
Document Revision	5

6.2 Luca Bertelli

Task	Hours
Purpose & Scope (and goals)	0.5
Architectural design from 2.1 to 2.4	9
Architectural design from 2.5. to 2.7	9
User interface design	1
Requirements traceability	1.5
Testing	1
Document Revision	2

6.3 Matteo Savino

Task	Hours

Purpose & Scope (and goals)	1
Architectural design from 2.1 to 2.4	4
Architectural design from 2.5. to 2.7	4
User interface design	4
Requirements traceability	5
Testing	5
Document Revision	1

6.4 Giacomo Vinati

Task	Hours
Purpose & Scope (and goals)	0.5
Architectural design from 2.1 to 2.4	6
Architectural design from 2.5. to 2.7	6
User interface design	1
Requirements traceability	2
Testing	0.5
Document Revision	3

7. REFERENCES

- Slides of the lectures
- ISO/IEC/IEEE 29148-2018 - International Standard - Systems and software engineering - Life Cycle Process - Requirements Engineering
- Hans van Vliet (2008), Software Engineering: Principles and Practice
- Telangana website: Telangana State Development Planning Society
- Community paper: *Artificial Intelligence for Agriculture Innovation*
- About the Unified Modeling Language Specification Version 2.5.1
- PlantUML