POLITECNICO
MILANO 1863

Dream DILETTA QUARTICELLI
ALESSANDRO PINDOZZI

Design Document

Deliverable:	DD
Title:	Design Document
Authors:	DILETTA QUARTICELLI ALESSANDRO PINDOZZI
Version:	1.0
Date:	9 February 2021
Download page:	https://github.com/apindozziPolimi/PindozziQuarticelli
Copyright:	Copyright © 2021, DILETTA QUARTICELLI ALESSANDRO PINDOZZI– All rights reserved

Contents

List of Figures

1 Introduction

1.1 Purpose

Agriculture is the basic source of India's economic, in fact 58% of rural households depend on it. In past decade, it is observed that there is not much crop development in agriculture sector consequently food price are increasing. Moreover the climate change defeats more and more the agriculture sector which impacts productivity. The climate change is not the only problem, the population will increase in 2050 in order to the food demand will increase.

Among these problems, also the COVID-19 crisis had a negative impact in this sector, hurting farms of all sizes in India. It has exposed the vulnerability of India's food system and accentuated the need for agricultural market reforms and digital solutions to connect farmers to markets, to create safety nets and ensure reasonable working conditions (as is written in the article How Indian agriculture should change after COVID-19).

Telangana is the 11th largest state in India with a geographical area of 112,077 km2 and 35,193,978 residents (data from 2011) and is one of the fastest-growing states in India posing average annual growth rate of 13.90% over the last five years . Agriculture form a backbone of Telangana's Economy.

In the light of the above-mentioned problems, the farmers must cultivate more, to deal with the problem of the population increase, and better to address the problem of the climate changes and the crisis due to the COVID-19 pandemic. For this reason Telangana's government wants to find a way to design develop and demonstrate anticipatory governance models for food systems using digital public goods and community-centric approaches to strengthen data-driven policy making in the state.

Figure 1: Logo of DREAM

In order to do this, the system "DREAM" will support the work of different stakeholders from farmers to policy makers. Farmers can use it to insert information about their productions, to receive suggestions, to check weather conditions everyday because DREAM interacts with Weather.com, in order to take better decisions. They also can ask for help to the agronomists, or chat with other farmers in the forums they create to share their experiences and help each other. The policy Maker, instead can check the work of every farmer, seeing the productions' information they insert in the system. Therefore DREAM is able to compute a ranking of the farmer for the production of a certain product, based on the amount of product collected, the size of the field, the age of the farmer and the weather conditions that the farmer faced during the period of this production. In this way the policy makers can identify the farmers who are performing well, and who are more resilient to adverse weather conditions, in order to send them incentives and encourage them ton give advice to the others. In the same way they can identify the farmer who are most in difficulty, and contact the agronomists who can help them. So DREAM give them all the cards they need in order to improve the food system and to take future decisions.

The purpose of this document is to give more technical details than the RASD about DREAM application system. While the RASD presented a general view of the system and what requirements and functions the system is supposed to execute, this document aims to present the implementation of the system including components, run-time processes, deployment and algorithm design. It also presents in more details the implementation and integration plan, as well as the testing plan.

1.2 Scope

DREAM is offered to all the farmers and the policy maker of Telangana, in order to create a continuous collaboration between these parties.

A farmer, after registering and logging in, will be able to insert his/her fields' information, like the location of the field, the size and information about what he/she produces, such as the type of products, the amount, the fertilizers used. Anyway what is meant when referring to field's information and production information is better explained in the subsection 1.3. DREAM allow farmers to check the weather forecasts, in a certain location and for certain dates, so that the farmer can better manage his/her productions. The farmer may need help or suggestions. In the system a differentiation is made between help and suggestion. When the farmer asks for help, inserting data about the location of his/her field and the type of problem he/she faced, the system sends a message to the agronomists, interacting with their own platform. The agronomist will receive the request for help and will response by communicating a date in which they will physically go to the field indicated by the farmer in order to help him. The farmer will receive the message thanks to the system, which provides a section for the notifications. The suggestions, instead, are given directly by DREAM, which is able to provide suggestions about the best products and fertilizer to use in each location. So the farmer only has to enter the location of the field for which he needs suggestions and DREAM will respond immediately. The last important feature of the system is the forum. They can interact each other, either to ask for help or to give advice to other farmers. A farmer can create his/her own forum, or join a forum created by other farmers, in order to chat with them. Each forum has a topic, so that a farmer who join a forum already knows what is being talking about in it.

DREAM offers also functionality to Policy makers. A policy maker, already registered and logged into the system, can see the information of each farmer's fields and each farmer's production. Moreover, for each production DREAM provides data about the average quantity of water used by the farmer and the average soil humidity level (obtained through the interaction by API with the water irrigation system and the soil humidity sensors), as well as information on weather conditions (like the amount of rainfall) during that production. Another important feature offered to the policy maker is Ranking: the system computes a ranking for each farmer who has cultivated the product in order to allow policy makers to see who are the best and worst farmer. In this way the policy makers can decide to send incentives to the best farmers, and messages to the agronomist in order to help the worst ones. Pay attention that DREAM does not provide an online payment system, but allow policy makers to send to the farmers notifies that they will receive an incentive (for example via bank transfer). Therefore, in the notify sent to the farmer, he/she is encouraged to create a forum by which he can explain why and how he is performing so well, so that he can help other farmers. Also in this case, when a message is sent to the agronomists, it is indicated the farmer who need help, the location of the field and production, and the agronomist will receive the request by another platform, as explained previously for the case of the request for help sent by the farmers. Last but not least, policy makers can check the work of the agronomist who help the farmers. In fact, when they finish their intervention they write a report on their platform, which is sent to DREAM and provided to the policy makers. In this way they can see who are the agronomists and the farmers who have collaborated, what the intervention was about, and what improvements are visible at the end of the intervention.

1.3 Definitions and Abbreviations

Definition	Descriptions

Field's information	Information about field: • the size • the location • the details of the field
Farmer's information	Information of the farmer: • name • surname • birth date • sex
Production	Product cultivated in a single field in a period of time.
Production's information	Information of the cultivated product: • product: the type of production cultivated in the field • planted date • collected date • amount of product collected • amount of product planted • fertilizers used
Farmer's Notification	There are two kinds of notification a farmer can receive: • The policy maker sends to a farmer a message which notify he/she will receive an incentive and is encouraged to create a forum to give advice to other farmer. • The agronomist sends a message in response to the request for help of the farmer where he/she explains when he/she will come to help the farmer.
Suggestions	The suggestions are tips given to the farmer directly by the system, interacting with a database created by us with the help of some agronomists which contains the best products to cultivate and fertilizers to use depending on the location. There are two types of suggestions a farmer can ask for: • suggestions about products: the system shows the best product to cultivate in a certain location • suggestions about fertilizers: the system shows the best fertilizer to use in a certain location

Request for help	It means that a farmer needs help and the request, which can be sent either by a farmer or by a policy maker, is addressed to an agronomist, not to the system or to other farmers.
Agronomists Platform	This is a platform which is external to DREAM. It is used by the agronomist to receive the requests for help sent by the farmers or by the policy maker in order to help the farmers and to notify the farmers about the date of the intervention.
Weather.com	This is a platform which is external to DREAM and provides the information about the weather conditions.
Ranking	The system compute a ranking for each product, in order to allow the Policy Makers to identify the best and the worst farmers. Obviously different products imply different rankings. The ranking is based on the Score.
Score	A value calculated by the system which is assigned to each Production, in order to insert it in the corresponding Ranking in a position based on this value. More information about it are in the subsection
Secret Code	This is a secret code, a Nonce, which is given by us to the Policy Makers when the final version of the DREAM will be released. They will use it in order to Register as PolicyMaker into the system. It is a way to not allow Guests who are not real Policy Maker to register in the system as PolicyMaker and see all the information provided by the farmers and to improve the security of the system.
API	Application Programming Interface.
MVC	Model View Controller is a design pattern used for GUIs.
HTTP	HyperText Transfer Protocol.

1.4 References:

- **Algorithm** ref 32 :
  - DISKIN, Patrick. Agricultural productivity indicators measurement guide. Food Security and Nutrition Monitoring (IMPACT) Project, 1997.
  - Yan Li, Kaiyu Guan, Gary D. Schnitkey, Evan DeLucia, Bin Peng. Excessive rainfall leads to maize yield loss of a comparable magnitude to extreme drought in the United States. Global Change Biology, 2019; DOI:10.1111/gcb.14628
- Slides – "Design part I and II.pdf"
- Slides – "Architecture and Design in Practice"
- Slides – "Verification and Validation"
- A Scalable Distributed System for Precision Irrigation, 2020 IEEE International Conference on Smart Computing (SMARTCOMP)

1.5 Architectural Design

- Introduction : Recap of purpose and scope already described in the RASD. It describe also the definition and the abbreviations used in the whole document and the references used.
- Architectural Design : The High-level description of the system and the interaction with third part. Moreover there is specific description of component of the systems and the deployment used, from a general overview and then going more in details whit the component view and the physical deployment and their interaction. Several run time are used to explain in detail all the use cases and the sequence diagrams mentioned in RASD.

- User Interface Design: Provide specific explanation on how the user interface of your system looks like.
- Requirements treacebility : Explain how the different component map to the requirements defined in the RASD.
- Implementation, Integration and Test Plan: describe how to implement the subcomponent of the system and how they are tested.
- Effort Spent: information of effort done by each member of the group

2 Architectural Design

2.1 Overview

All the choices made in terms of architectural design are aimed at implementing a distributed system, which can therefore be used by numerous users at the same time and which can manage large quantities of data from different sources (users, sensors, external platforms).

There are three logic software layers, and three corresponding hardware ones, which provide the needed functionalities:

- Presentation Layer: it manages the interactions between the User and the system.
- Application Layer: it handles the business logic of the application and its functionalities
- Data Layer: it manages the information interacting with the database

Thus, this is the so called three-tier architecture. The following image presents an informal view of the system which reflects the division just described.

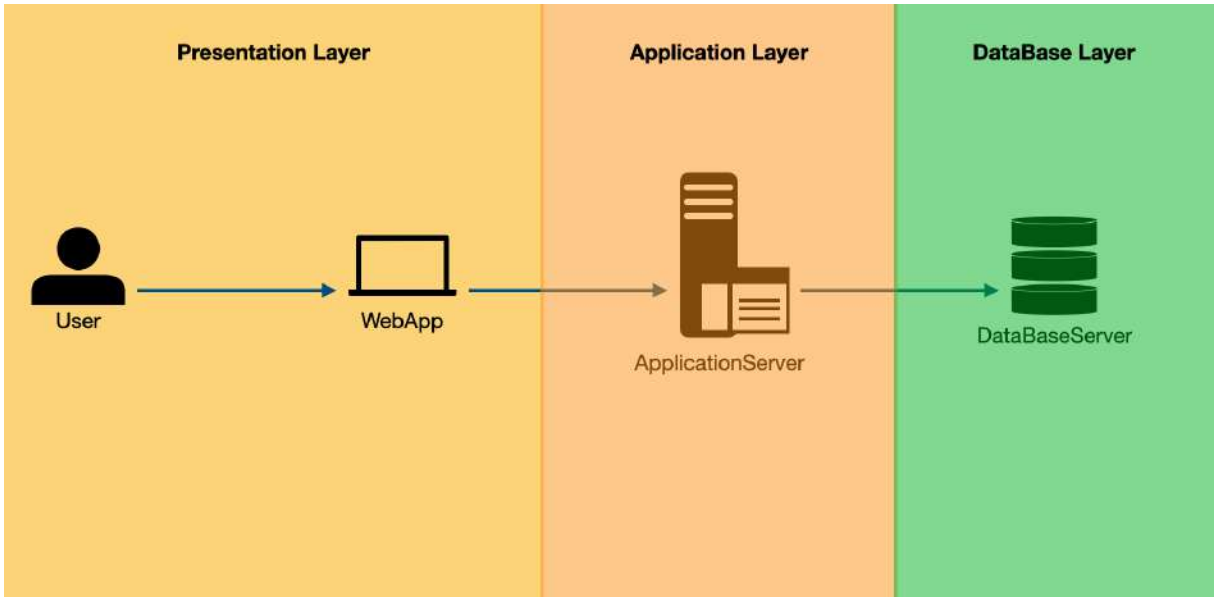

Figure 2: Three-Tier Architecture

As you can see, the User interacts with the Presentation Layer, through the WebApp, where he/she can use all the functionalities DREAM offers.

All the data provided by the User, or more in general every action he/she performs in the Presentation Layers are then managed by the Application Layer, which is the heart of the system. In fact all the information collected in the Presentation tier is processed often relative to other information in the Data tier. Moreover, it interacts also with all the external parts, in fact, all the data provided by the Weather platform, the AgronomistsPlaform, the humidity soil sensors and the water irrigation system are collected at first and then managed by the Application tier before they are stored in the database.

The Data layer is where the information processed by the application is stored.

Hence, in a three-tier system, all communication passes through the application tier. The presentation tier and the data tier cannot communicate directly with each other. This brings several advantages:

- Faster development: since each tier can be developed simultaneously by different teams.
- Improved scalability: any tier can be scaled independently of the others as needed.

- Maximum reliability: an outage in one tier is less likely to impact the availability or performance of the other tiers.

- Improved security: Since the presentation tier and data tier cannot communicate directly, a well-designed application tier can act as a kind of internal firewall, preventing malicious exploits.

Obviously what is shown in 3 is a high-level and not realistic representation of the architecture of the system, because it has a high risk of failure. In this case, both the ApplicationServer and the DataBaseServer are single points of failure (SPOF). A SPOF is any non-redundant part of a system that, if dysfunctional, would cause the entire system to fail. For this reason we have to relay on redundancy, deploying multiple copies of them in order to to make the system more tolerant. To manage the flow which comes in these copies of the ApplicationServer and of the DataBaseserver we need some load balancers, which are a software solution that helps to move packets efficiently across multiple servers.

It is also important to mention the SystemManager who has access to the entire system. He/She can be an engineer or a technician who intervene in case of faults or if small changes are needed.

The Figure below show a more detailed but view of the architecture of the system.

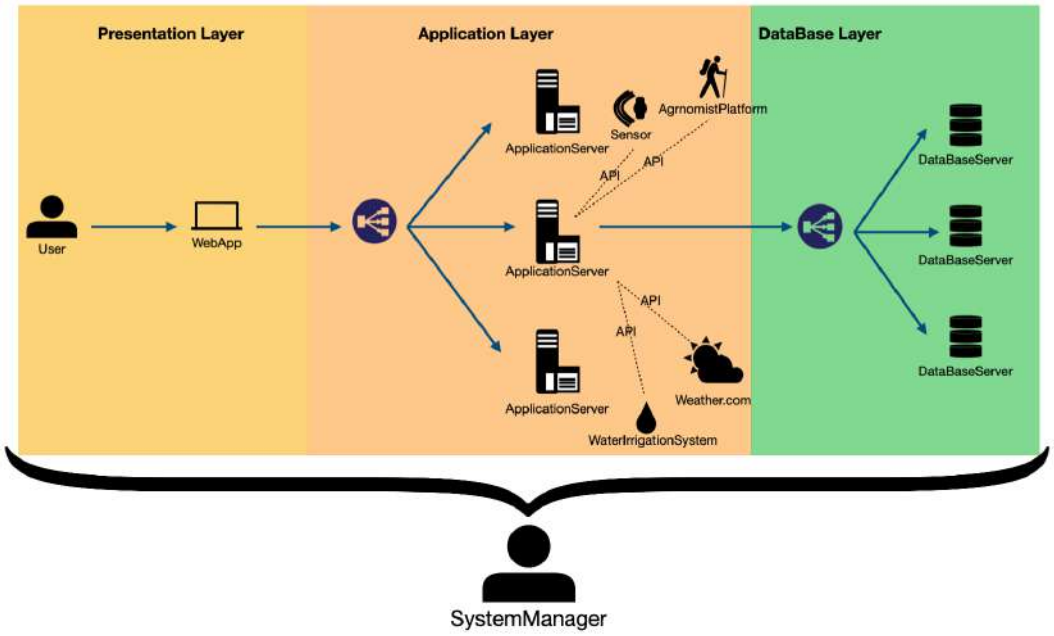

Figure 3: High-Level System Architecture

2.2 Class Diagram

It is presented an update view of class diagrams, compered to the diagram in the RASD more attributes are added. It describes the structure of a system by showing the system's classes, their attributes that should be used as a reference for the other diagrams and the implementation of the system.

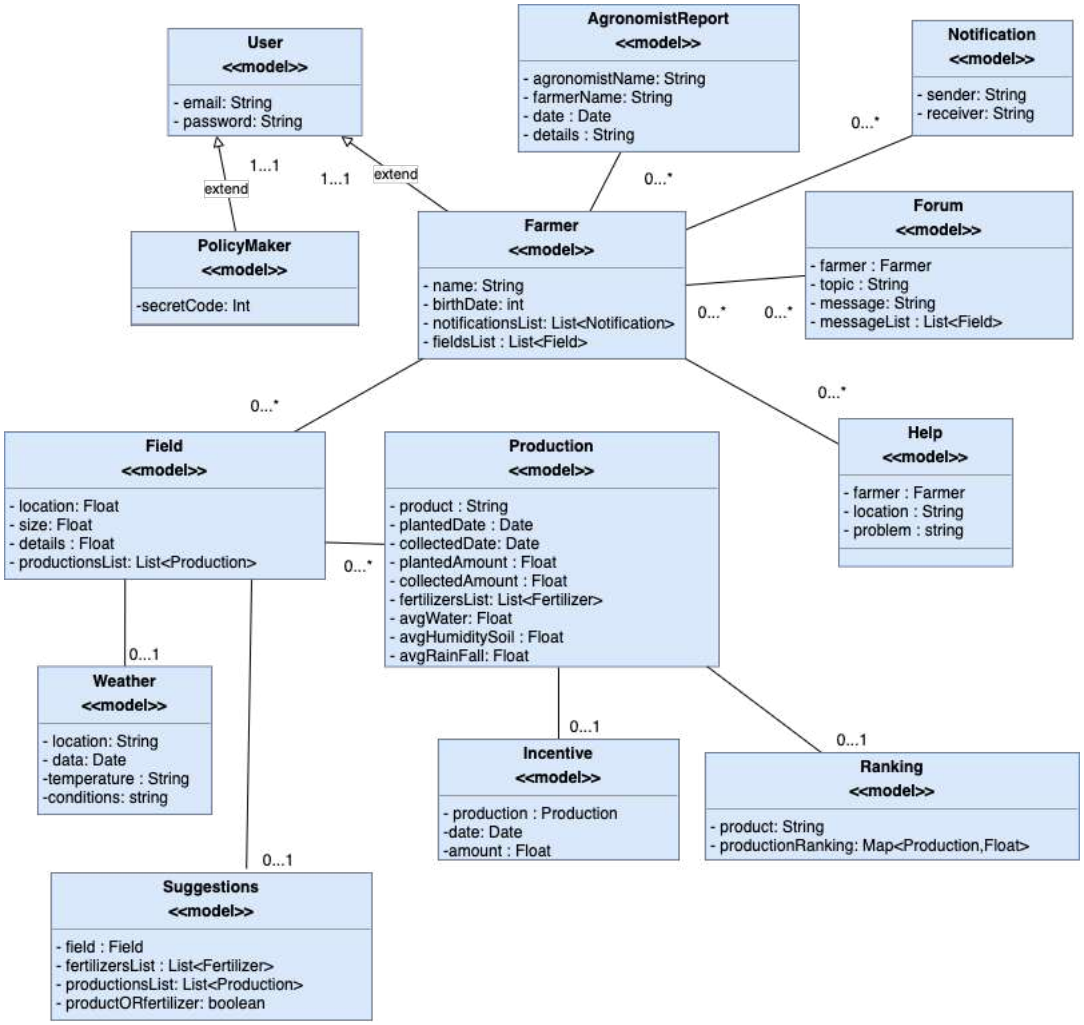

Figure 4: Class Diagram

2.3 Component view

This section describes the architecture from a software component point of view, where the server respects the three-tier subscription as shown in the High Level System architecture [3].

- The presentation layer is made of one component: the web application which interacts with the user such as the farmer and the policy maker
- The application layer is made of several components, in order to get more comprehensibility they are divided in two component, each of them included several subcomponents
  - The FarmerWebService component supports the farmer in his requests and allow to access to the functionalities of the system
  - The PolicyMakerWebService component supports the policy maker in order to see the ranking, the agronomistReport and all the functionalities of the system
- The DataBase layer unifies the communications between the application layer and the databases. In particular there are three databases in order to augment the system scalability through nodes replication. Moreover, replicating the shared disk configuration increases both scalability properties and faults tolerance.

- An external component : The SystemManager has the access to the entire system in order to control and to resolve the problem of the system

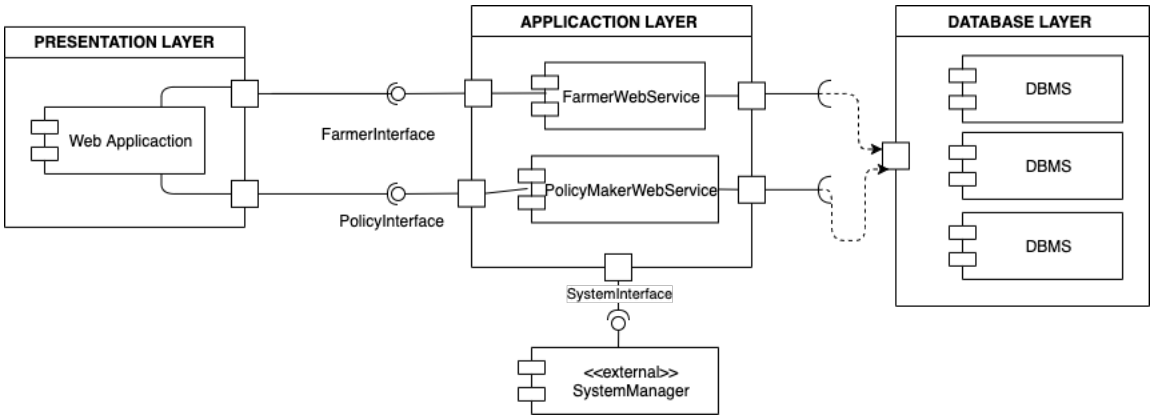

Figure 5: component diagrams

The Application layer is divided in FarmerWebService and PolicyMakerWebService, the first one is related to the farmer and the second one to the policy maker. Both of them have several subcomponent enfact they are called with the same name, we have chosen to divide FarmerWebService and PolicyMakerWebService in order to organize well the diagrams and to explain well the interaction between them. The FarmerWebService is divided in several subcomponents:

- **Registration Manager**: responsible for the registration of the Farmer, it controls taht the farmer's information is correctly inserted and connect to the Email external component trough an API to send the confirmation registration link to the email in order to complete the registration process. The registration manager interacts with the Model in order to store the data in the database.
- **LoginManager**: responsible for the Authentication of the Farmer
- **Field Manager**: provides functionalities concerning field,it uses the FieldInterface to add new Field or to check the Field owned by the Farmer
- **Notification Manager**: responsible to display the notification sent by the AgronomistPlatform or the PolicyMaker. The subcomponent connects to the Model and requests to display the notification received by the Farmer.
- Suggestion Manager: is in charge of showing the farmer suggestions of the type of fertilizer or type of product he can grow in the field of his choice. The subcomponent connects to the Model and request to the DataBase the information.
- **Help Manager**: responsible to connect with the Agronomist Platform which will receive the request for help.
- **Forum Manager**: is in charge of showing all the forums, which connects to the Model. The Model performs a query on the database in order to get all the forums stored (already created by other farmers). The Model sends the query results to the Forum Manager, which in turns sends the forum lists to the FormuInterface. It manages also the creation of new Forum. And keeps the messagesList of the forums synchronised with the database (through the Model).
- **Production Manager**: provides functionalities concerning the view of the productions and the creation of a new Production. The Production Manger checks if all the field are inserted, then it sends the information to the Model, which sends the query to the database and saves all. After that

the Production Manager connects whit the DataProcessing which in turn sends the Data concerning the score of the production and the average water used to the Model that queries the information in the DBMSs.

- **Weather Manager**: provides information of weather, it connects with an external component Wheater.com trough an API and shows the data received by the external component.

- **Help Manager**: deals whit the interaction between the Agronomist Platform and the farmer. When a farmer has some problems, he writes an messages to the agronomist platform. It interact with the platform trough an API, in response the agronomist platform will send an message where it is specifies the date of arrival

- **DataProcessing Engine**: responsible to connect with several external components such as Sensor Humidity Soil, Weather Irrigation and Weather.com trough APIs. It is in charge to collect Data to calculate the score of each production and to calculate the average of humidity and water used. In this way the system is scalable and faster. The Production Manager can shows the productions added in the system without waiting the response of the DataProcessing Engine.

- **Model**: provides a single point of access to the data tier. The interaction with the DBMs is performed by the Model trough quering and pushing. It is important in order to decouple the data tier to the application tier in an appropriate and scalable way. This subcomponent is in charge to create the specific object of the class that compose the Data Base, all other subcomponent connect with the DataBase Tier layer trough this subcomponent and update or add the data trough it.

The are several external component:

- **Agronomist Platform**: it is an external component which provides help to the farmer and receives request trough an API from Help Manager and sends the information about his or her arriving to the Model trough an API.

- **Sensor Humidity Soils**: external component which provides Data concerning the Humidity of the soil during a production, in particular the DataProcessing Engine request the Data between the planted date and collected date of a production and then the DataProcessing calculate the average of it to calculate the ranking.

- **Water Irrigation**: is an external component that provides data of the water irrigation of particular production. In the same way as the sensor humidity soil, the DataProcessing Engine connects to the Water Irrigation trough an API and calculate the average of water used in that production during the planted date and collected date.

- **Weather.com**: external component provides weather conditions of a particular day

- **Email**: external component provides the link to confirm a registration

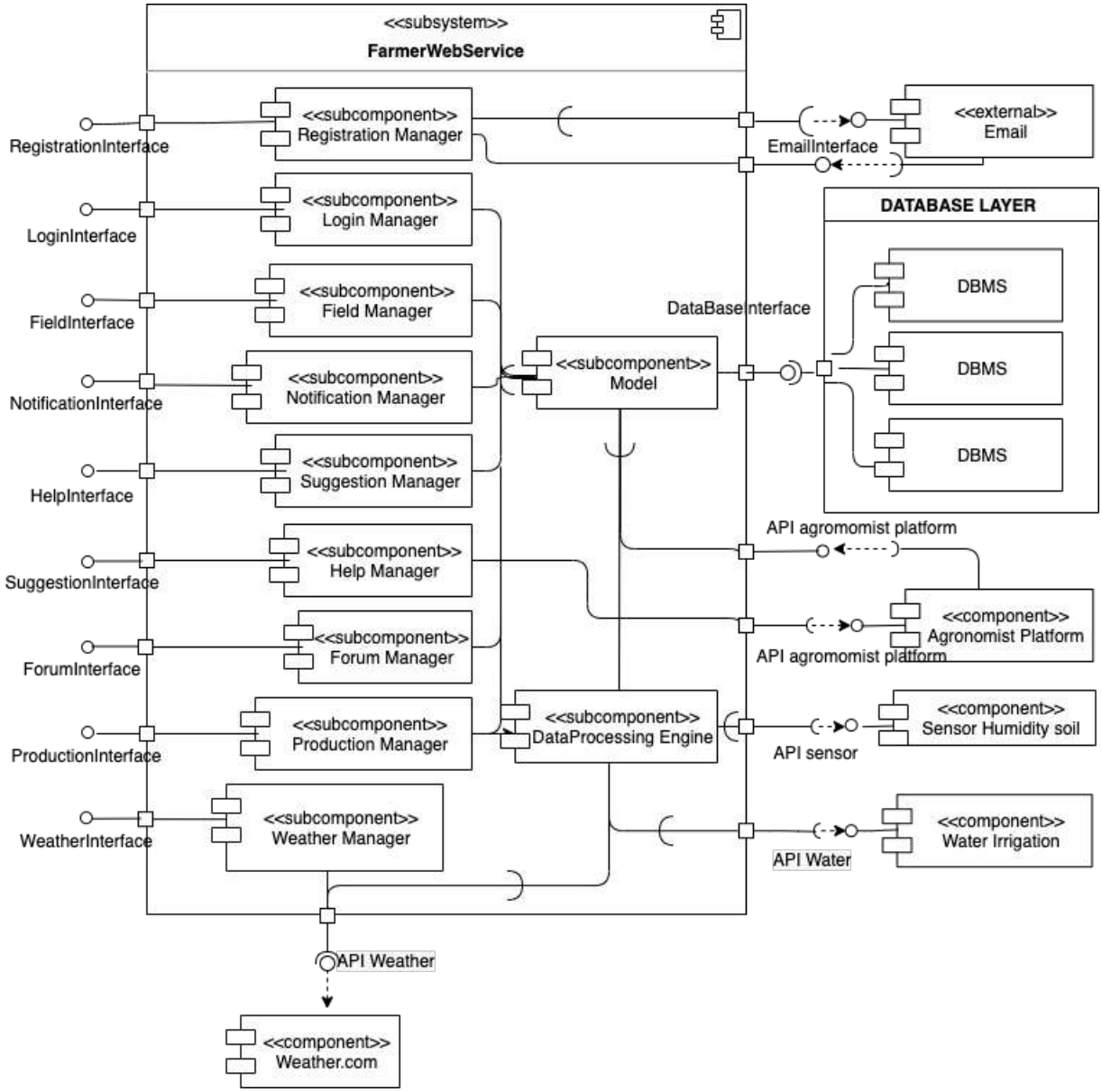

Figure 6: Farmer Component Diagram

There are several subcomponent in the PolicyMakerWebService:

- **Registration Manager**: responsible of the registration of a Policy Maker
- **Authentication** : responsible of the authentication of a policy Maker
- **Field Manager**: responsible of display all the field owned by a single farmer and the information of them.
- **Production Manager**: responsible of display all the productions of a single field. It is linked to the Field Manager because the policy maker first chooses the field of his interest then visualizes the productions belonging to the farmer
- **Ranking Manager**: responsible of display the ranking of a single production. Moreover it provides the functionalities to send an agronomist to help a farmer who are performing bad or to send a notification to a farmer who will receive an incentive.

- **Incentive Manger** : displays all the incentive already sent to the farmer.

- **Agronomist Report Manager**: displays the Report done by an agronomist, in particular shows the problem faced by the agronomist and the score before and after his intervention.

- **Model**: has the same role of the FarmerWebService.

In policyMaker interfaces there are two kind of external component, the email that provides the information to the Policy Maker to enter in the system and the Agronomist Platform. In particular the Agronomist Platform receives request from the Ranking Manager when the Policy Maker wants to help a farmer, and sends the reports to the Model.

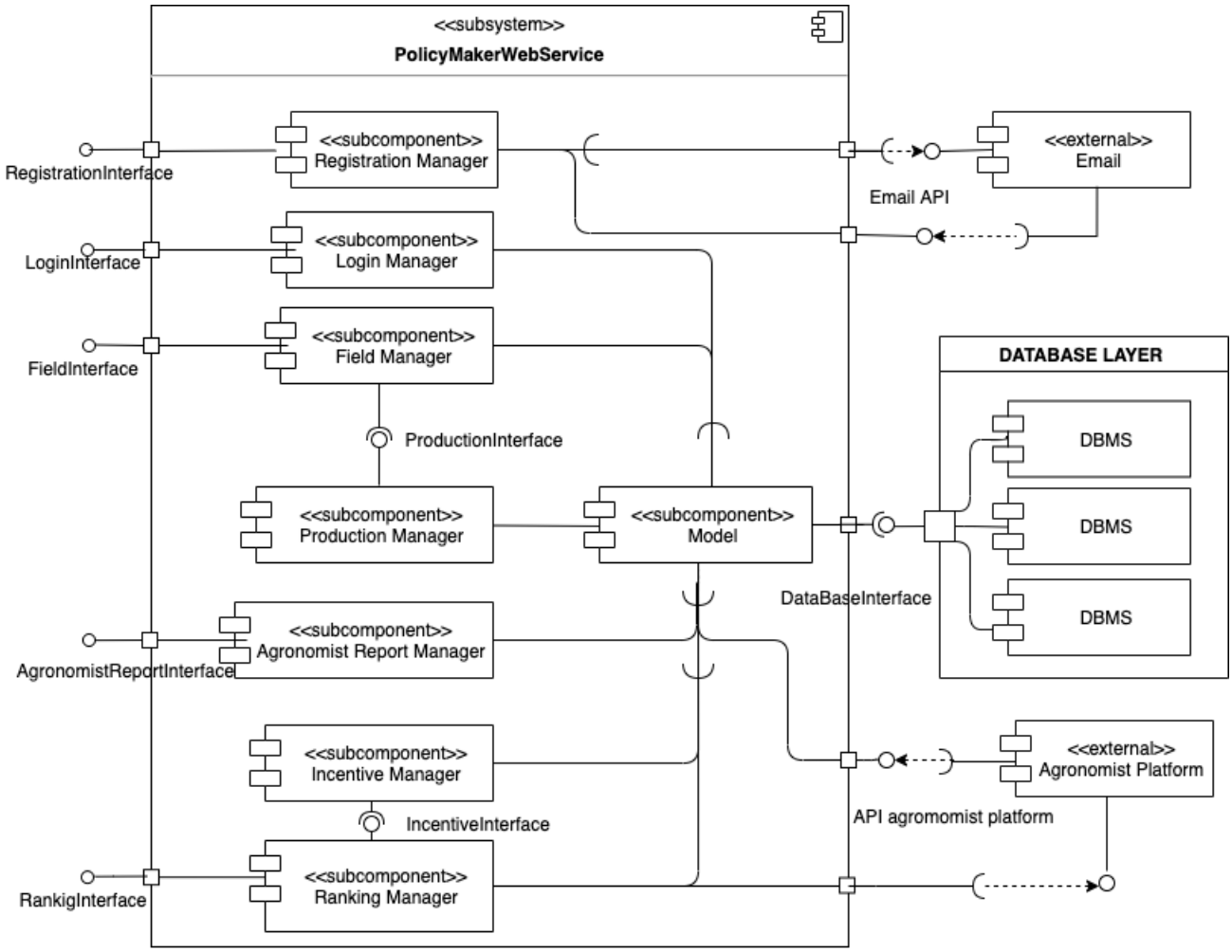

Figure 7: Policy Maker Component Diagram

This diagram below describes better the interaction between the manager component and the classes.

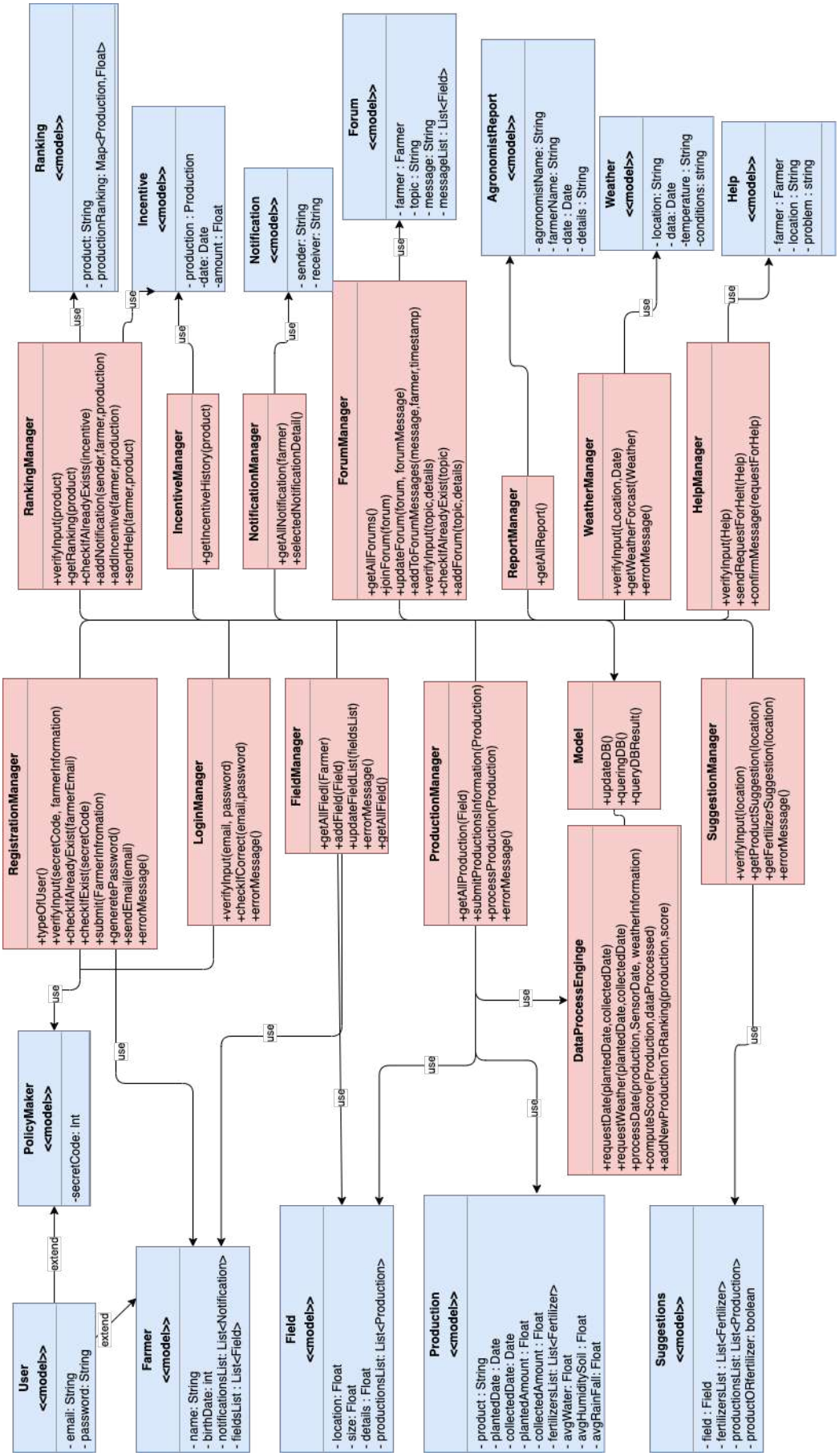

Figure 8: Classes and component

There are some things that must be point out, in order to describe how the component works.

- Only the interfaces offered by the application are represented, the Sensor Humidity soil, Water irrigation, weather.com, the agronomist platform and the email are considered outside of the application and they are represented in 6 and 7
- Each interface is connected to the model that is responsible for acting as a communicator between the interfaces and the database. In fact it takes care to connect with the database and make requests through the queries coming from the other interfaces.
- The Registration Manager controls the registration of the user. The method generatePassword() generate the password of the policy maker if the secret code inserted from the policy maker is right. The method sendEmail(email) sends an email in order to confirm the registration in case of the registration of a Farmer, whereas it sends the generated password in case of the registration of a PolicyMaker. Pay attention that the secret code exist already in the database, so the methods checksIfExists(secretCode) checks if exists already the secret code in the data base. The method checkIfAlreadyExist(farmerEmail) checks if exists already a farmer with the same email, if not it creates a new Farmer with all the attributes describes in class diagram 4
- LoginManager checks if the user inserts the right information with the method checkIfCorrect. if it is correct create an object of type farmer or an object of type policy maker.
- The Fieldmanager creates a new Field trough addField(Field) but update also the list of the field presented in Farmer object. In this way the field manager can show the fields to the farmer without having to query the database, and thus be faster.
- The ProductionManager creates a new Production and shows the production to the farmer and the policy maker. As the FieldManager it updates the productionList in order to not query the database easch time the farmer ask to show the production.
- RankingManager displays the ranking of each product, the method addNotification allows to policy maker to send an agronomist to help him, and the addIncentive allows to policy maker to send a notification that say that he will receive an incentive.
- DataProccesEngine process the data coming from the sensors and calculate the ranking asynchronously to the insertion of a production in the database. Through the method computeScore it calculates the score and then inserts it in the database-
- helpManager deals with the farmer's request to the agronomist trough the method sendRequestForHelp(Help)
- WeatherManager gets the weather forecast
- NotificationManager gets all the notification saved in the model and shows the details trought the method selectedNotificationDetail
- ForumManager deals with the forum, in particular the method getAllTheForum takes the all forum in presens in the database, the method addForum first checks if already exist in the forumList and then adds in the database trough the model.

2.4 Deployment view

The figure above describes the execution architecture of the system and represents the deployment of software. In general, it shows the hardware components of the system "nodes" and the "artifact" that are the software components. In particular the system is divided in three tiers, in the Tier 1 the presentation

layer is deployed. Trough the Personal Computer the User can connect to the website and enters in DREAM. We have chosen only the WebSite because in Telangana Farmer could not have an Android or iOS system or even an internet access. Both PolicyMaker and farmer connects to the system trough a website. In the Tier 2 is deployed the application logic. The Server implements all the business logic, communicate with the database trough HTTP and presentation layer trough TC/API. It communicates also with the external device like Sensor, Weather.com or AgronomistPlatfor. It must handle all the request and provide appropriate answer to the external device, the web and the database.The software chosen is the Servlet Container in order to support Multithreaded and to support Security. In the Tier 3 the data access is deployed. The database collects all the data of the system, in particular there are three DBMs in order to duplicate the data and to increases both scalability properties and faults tolerance. The database is relational in order to link information from different table trough uses of foreign keys and can handle the big quantify of data.

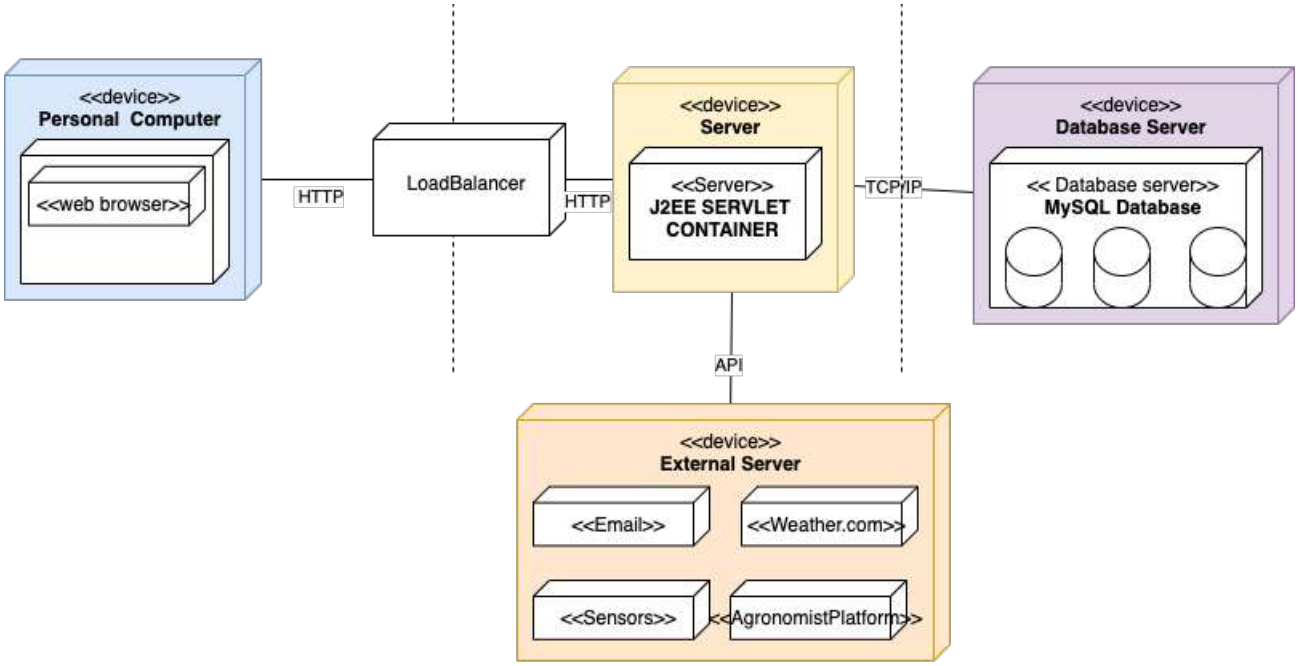

Figure 9: Deployment View

2.5 Runtime view

In this subsection are represented and described more in detail all the use cases and sequence diagrams in the RASD (Chapter 3: Specific Requirements).

Also in all these runtime views the three-tier division described in the previous subsections is highlighted.

1) **Farmer registration**

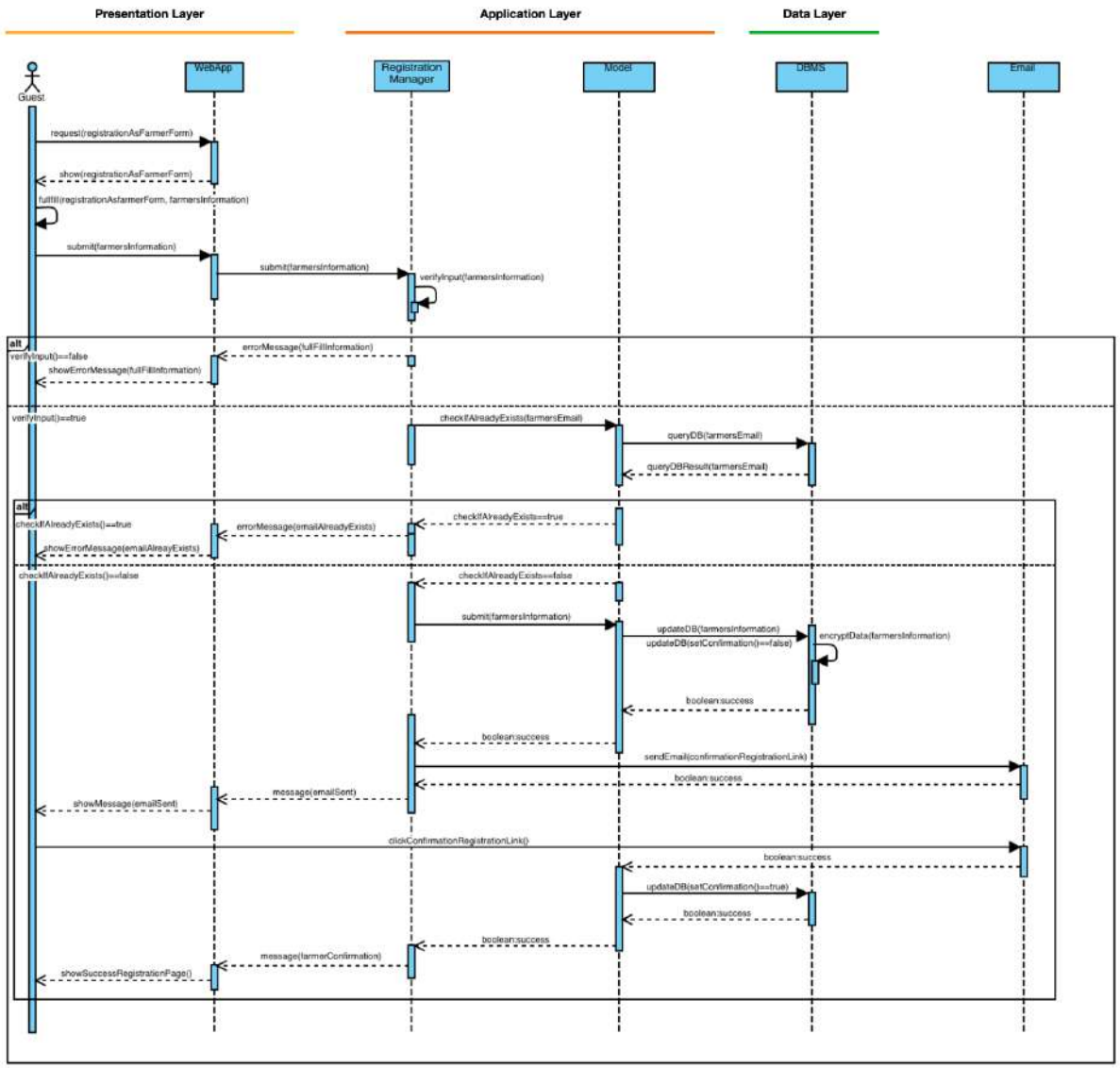

Figure 10: Runtime view: Farmer registration

In this diagram the Farmer registration is analyzed. We consider that the Guest (the actors' name have already been specified in the RASD) is on the FirstPage of Dream, where he can click on Register As Farmer. The system shows the form with all the information he/she has to provide (farmer's information). He fills the form and submit the information, which is managed by the Authentication Manager. VerifyInput() is needed to check that all the form has been filled. If it returns false, the system shows and error message to the Guest. Otherwise it has to check that the email is unique. In order to do this the authentication manager, thanks to the Model, can interact with the DBMS, where are stored all the email of the Farmers and checks if the email the Guest inserted is already used. If the query returns one item, it means that the email is already in the database, so checksAlreadyExists() returns true, and the system shows an error message to the Guest. Otherwise, if the query returns zero items, checksIfAlreadyExists() returns false, and the Authentication manager with the Model can update the database adding the new farmer's information, which will be encrypted by the DBMS. Moreover, when the Database is update, the new farmer's attribute "Confirmation" is set to false. Once the operation has been successful, the Authentication Manager can send the email with the link to confirm the registration. The system shows a message to the Guest, in order to show him/her the the email has been sent. The Guest

can checks the Email and clicks on the confirmation link. Thus the Model can update the DBMS setting the Confirmation attribute to true and the SuccessRegistrationPage is shown to the Guest. From now on the the Guest is registered and is allowed to log into the system.

2) Farmer login

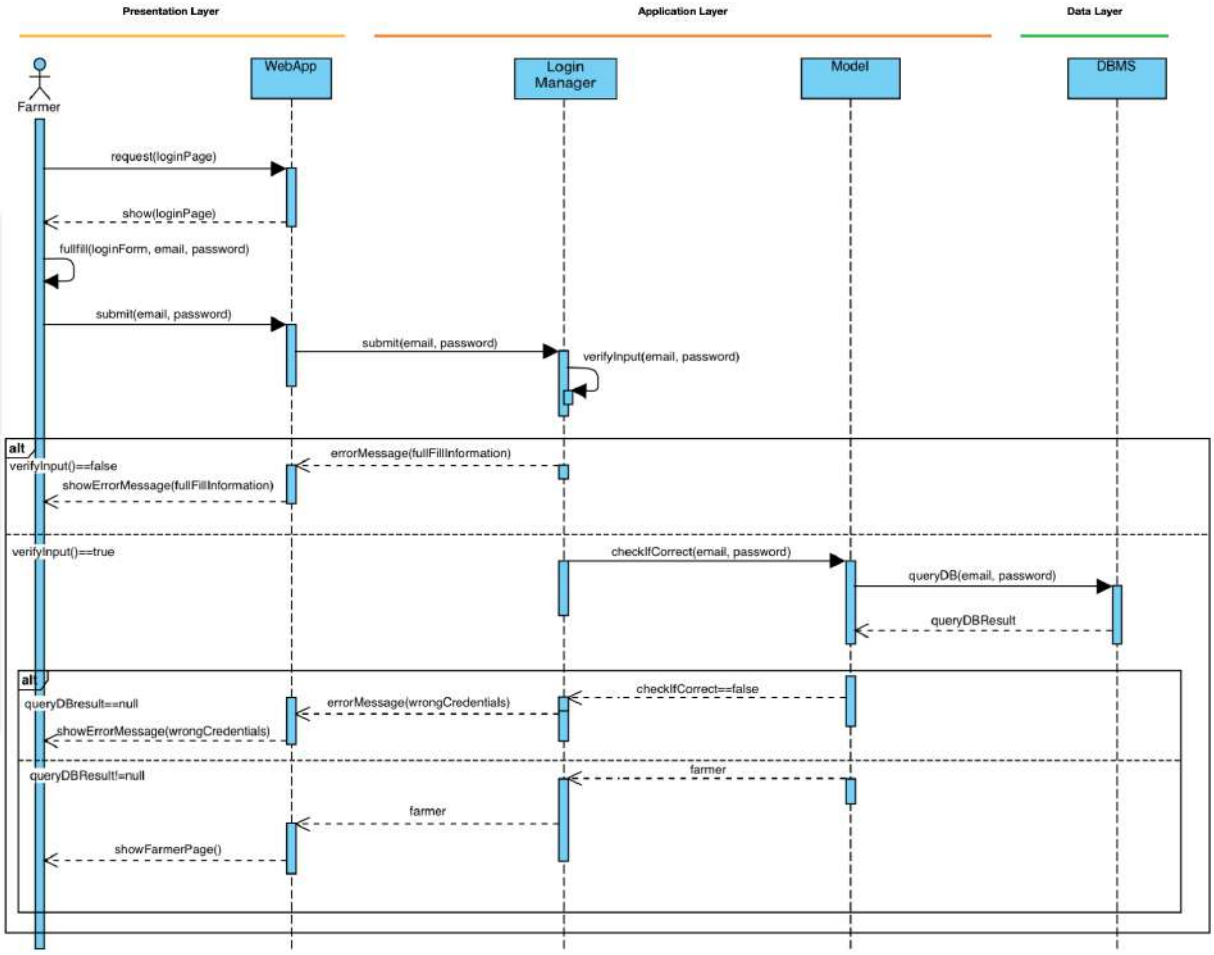

Figure 11: Runtime view: Farmer login

The Farmer is on the First Page of Dream, where he/she can clicks the button Login. The system shows the loginPage, with the loginForm. The Farmer provides his/her email and password, and as the case of the registration, the Authentication Manager checks if the User fills all the information needed. If not, the system shows an error message, else, the manager and the model send a query to the database, in order to check if there is a farmer registered with these email and password. If the query returns zero elements, checkIfCorrect() is set to false and the system shows another error message, otherwise, if it returns one element, it is sent to the Authentication Manager, and an object of type farmer is allocated. In this way, whenever attributes of that farmer (e.g list of its fields, or list its productions) need to be extracted, they are provided to the WebApp directly from the Application Layer, without the need to ask the database for them (unless its attributes change, e.g. if a field, or a production is added, in which case it will be necessary to synchronise again with the database). Thus checkIfCorrect() is set to true and the system displays the FarmerPage.

3) Farmer checks weather forecasts

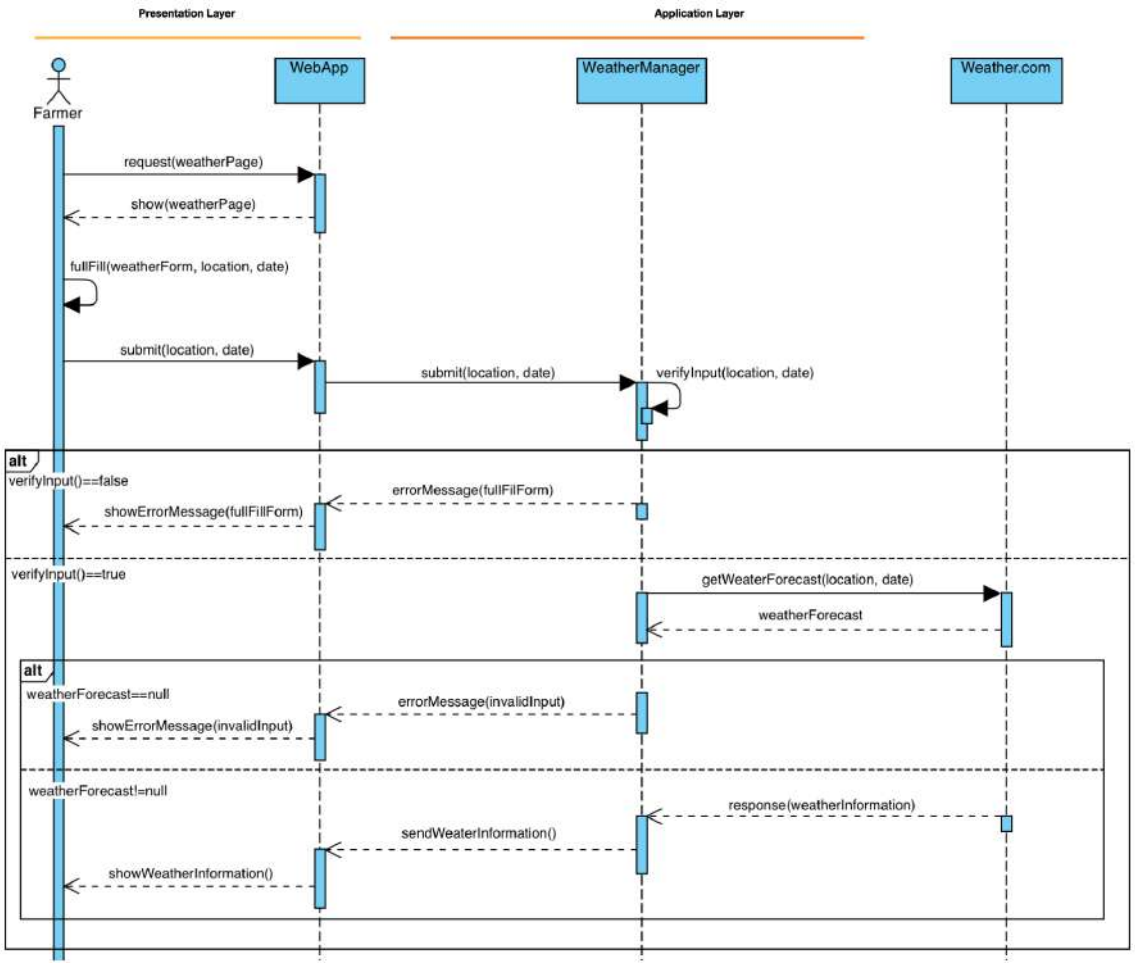

Figure 12: Runtime view: Farmer checks weather forecasts

The Farmer is on the Farmer Page, where he/she can clicks the button Weather. The system shows the weatherPage, with the weatherForm. The Farmer provides the location and the date for which he/she wants to check the weather forecasts. The Weather Manager checks if the User inserted either the location and the date. If not, the system shows an error message, else, the Weather Manager uses the weather API to interact with Weather.com and fetch the weather forecasts for that location and date. If the API request returns zero element, the system shows another error message, otherwise, the Weather Manager sends the data provided by Weather.com to the WebApp, which shows them to the Farmer.

4) Farmer checks his/her fields

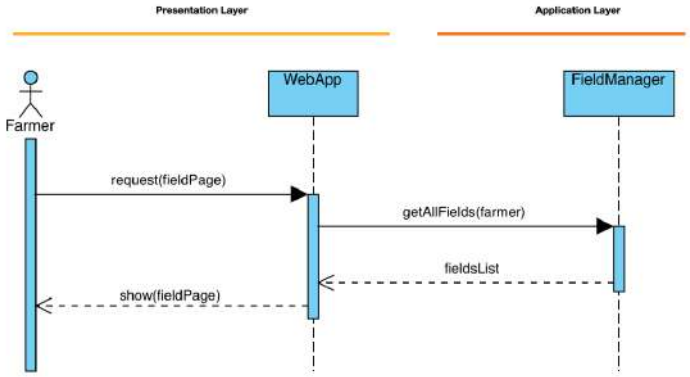

Figure 13: Runtime view: Farmer checks his/her fields

The Farmer is on the Farmer Page, where he/she can clicks the button Field. The WebApp takes from the FieldManager the list of fields of that Farmer and shows him/her the FieldPage with a table with all the fields he/she already inserted in the system

5) Farmer adds a field

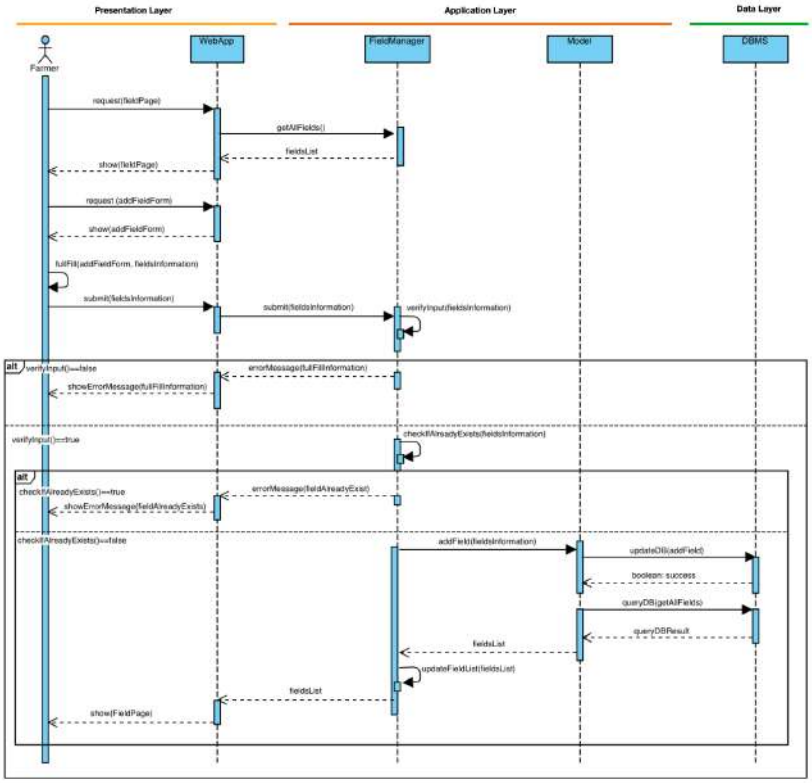

Figure 14: Runtime view: Farmer adds a field

The Farmer is on the Farmer Page and performs the same actions as described in the previous point. Then it clicks on the button Add and the WebApp shows the AddFieldForm. The Farmer fills the form providing the field's information. The WebApp submit it to the Field Manager, which checks if the form has been filled in completely. If not, the WebApp shows an error message to the farmer, else the manager checks if the field the farmer wants to add is already in his/her fields list. If it already exists, the web app shows another error message, else the Field Manager and

the Model send a query to update the database with the new field. Then another query is needed to obtain the new field list from the database. The Field Manager replace the old field list of the farmer with the new one and allow the WebApp to show again the FieldPage and its fields table updated with the new one just added.

6) Farmer checks his/her productions

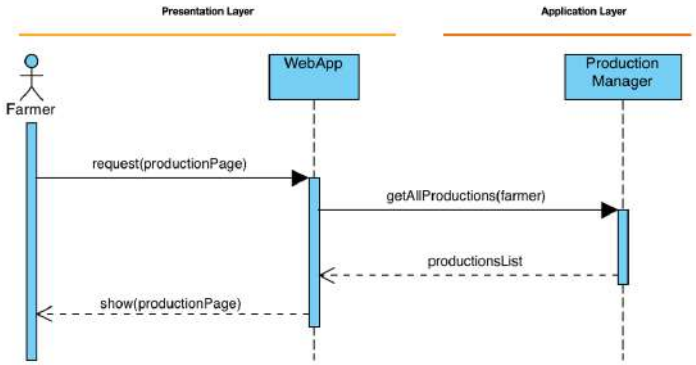

Figure 15: Runtime view: Farmer checks his/her productions

The Farmer is on the Farmer Page, where he/she can clicks the button Production. The WebApp takes from the ProductionManager the list of productions of that Farmer and shows him/her the ProductionPage with a table with all the productions he/she already inserted in the system

7) Farmer adds a production

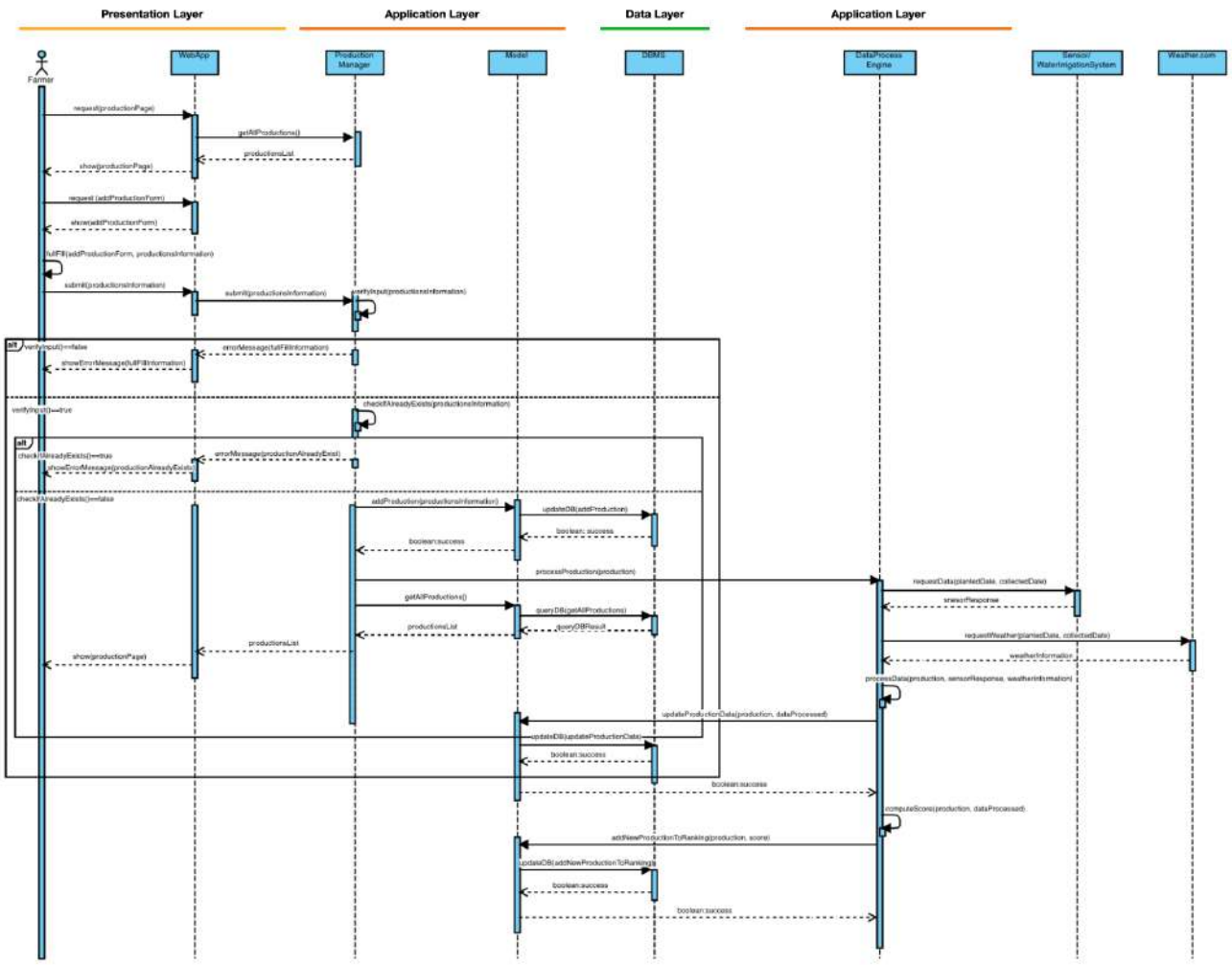

Figure 16: Runtime view: Farmer adds a production

The Farmer is on the Farmer Page and performs the same actions as described in the previous point. Then it clicks on the button Add and the WebApp shows the AddProductionForm. The Farmer fills the form providing the field's information. The WebApp submit it to the Field Manager, which checks if the form has been filled in completely. If not, the WebApp shows an error message to the farmer, else the manager checks if the field the farmer wants to add is already in his/her fields list. If it already exists, the web app shows another error message, else the Field Manager and the Model send a query to update the database with the new production. Then another query is needed to obtain the new field list from the database. The Field Manager replace the old field list of the farmer with the new one and allow the WebApp to show again the ProductionPage and its productions table updated with the new one just added.

However, there are other steps that need to be described, which are necessary to calculate the score of the new production, so that it can be included in the corresponding Ranking. After the production has been added to the database, the Production Manager sends it to the Data Process Engine. When it receive the new production, it sends an API request to the water irrigation system and to the humidity soil sensors (in the run time view they have been represented as a single entity to make them easier to read, but in reality they are divided and with two different APIs). The Data Process Engine asks for the data collected between the plantedDate and the collectedDate of that production. The same request is also sent to Weather.com. Data obtained by the sensors, the water irrigation system and Weather.com are processed, in order to calculate the average and sent to the

Model. It performs a query to update the DB with these data. Then, the Data Process Engine thanks to the data processed can calculate the score of that production. Again, it sends the score to the Model, which performs a query in order to add the new production, with its score to the corresponding Ranking.

8) Farmer asks for suggestion about a product

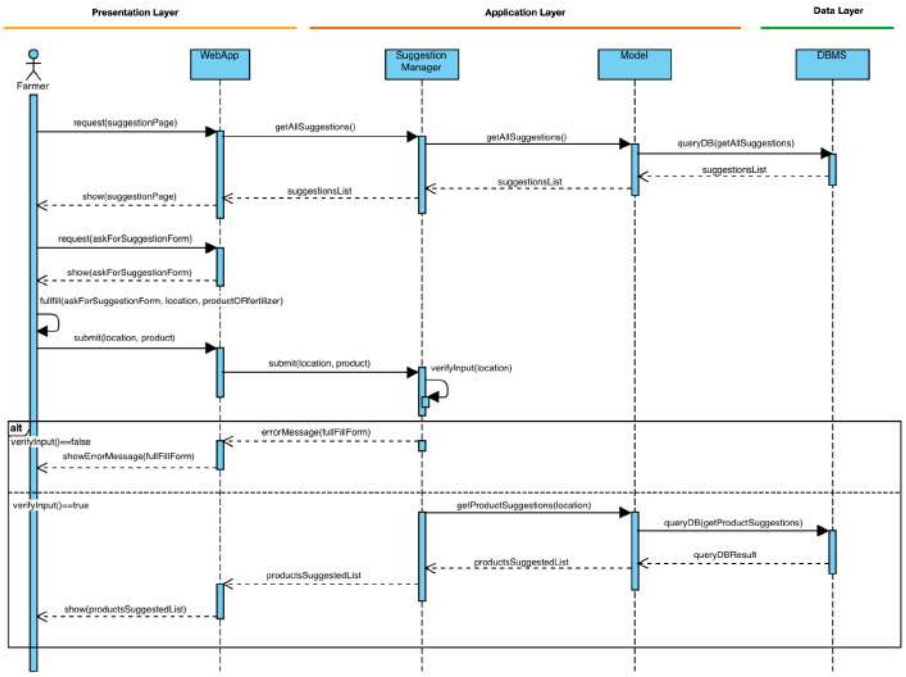

Figure 17: Runtime view: Farmer asks for suggestion about a product

The Farmer is on the Farmer Page, where he/she can clicks the button Suggestion. The WebApp shows the SuggestionPage with the askForSuggestionForm. The Farmer fills the form providing the field location for which he/she needs suggestion and select product (he/she can chooses between product or fertilizer). The WebApp submits the information to the Suggestion Manager, which verifies that the Farmer provided all the input needed. If verifyInput() return false, the WebApp shows an error message, else the Suggestion Manager notifies the Model, which performs a query in the database in order to get the suggested product for that field location. The Suggestion Manager receives the result of the query from the Model and send the productSuggestedList to the WebApp which shows it to the Farmer.

9) Farmer asks for suggestion about a fertilizer

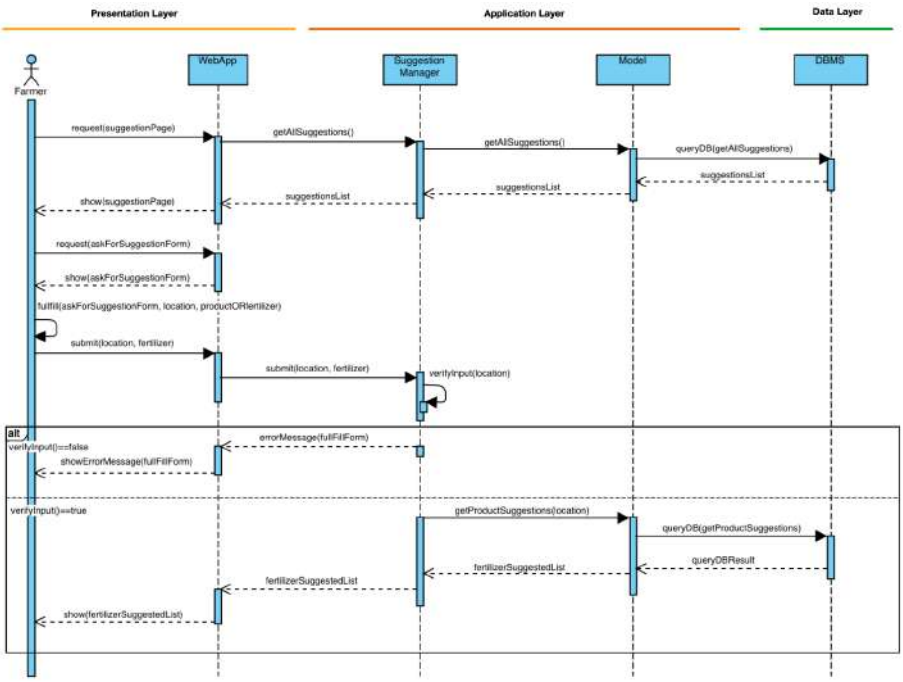

Figure 18: Runtime view: Farmer asks for suggestion about a fertilizer

The process shown by this runtime view is similar to the previous one. The only difference is that the Farmer select fertilizer instead of product in the askForSuggestionForm.

10) Farmer requests for help

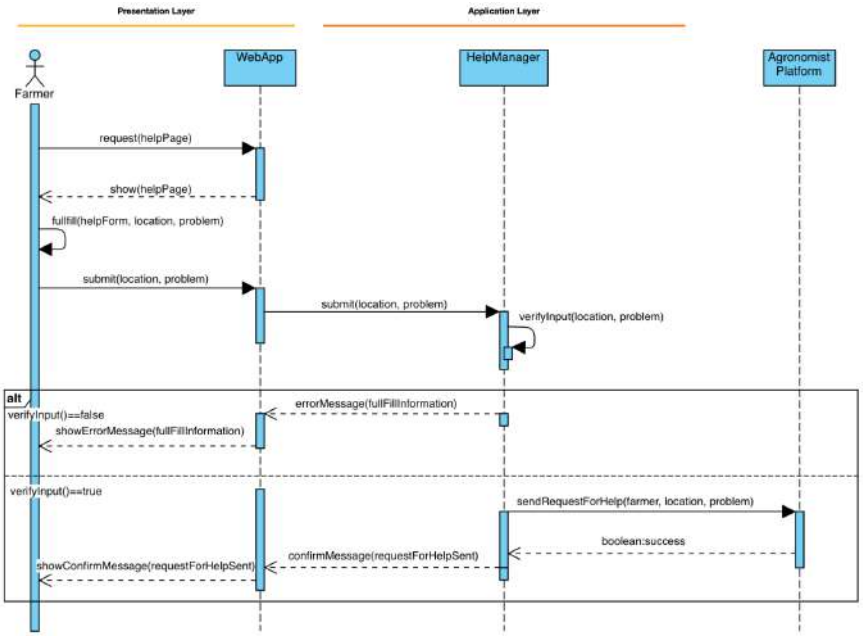

Figure 19: Runtime view: Farmer requests for help

The Farmer is on the Farmer Page, where he/she can clicks the button Help. The WebApp shows the HelpPage with the HelpForm. The Farmer fills the form providing the field location for which he/she needs help and the description of the problem he/she faced. The WebApp submits the information to the Help Manager, which verifies that the Farmer provided all the input needed. If

verifyInput() return false, the WebApp shows an error message, else the Suggestion Manager uses the AgronomistsPlatform API to send the request for help of the farmer (with the location of the field and the problem description) to the AgronomistsPlatform. If the operation is successful, the WebApp shows the message that the help request has been sent.

11) Farmer joins a forum

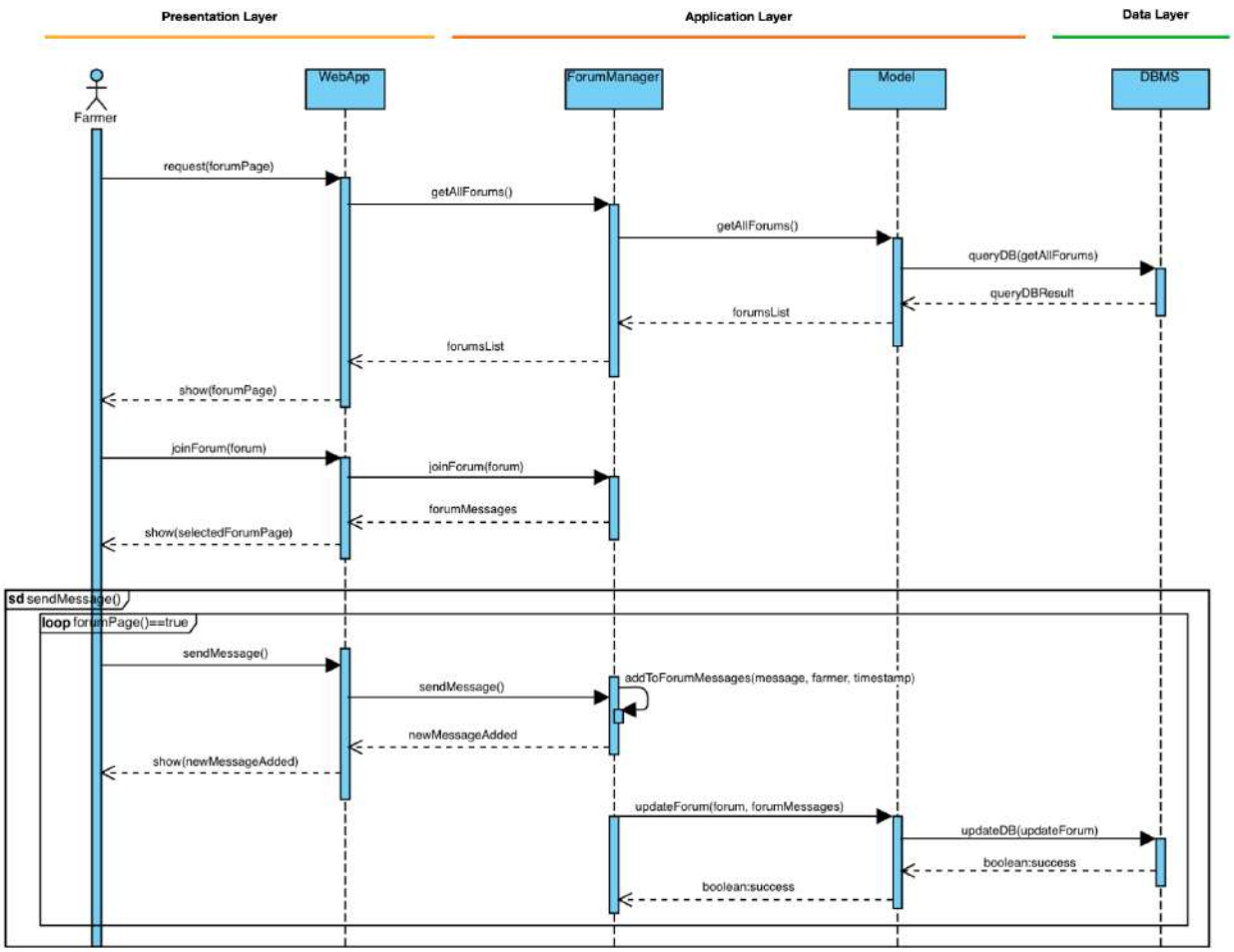

Figure 20: Runtime view: Farmer joins a forum

The Farmer is on the Farmer Page, where he/she can clicks the button Forum. The WebApp submits the request to get all the forums to the Forum Manager, which forwards it to the Model. The Model performs a query on the database in order to get all the forums stored (already created by other farmers). The Model sends the query results to the Forum Manager, which in turns sends the forum lists to the WebApp. The WebApp displays the ForumPage with the forum table, whose rows contain the topic of each forums obtained from the database. The Farmer can click on one of the row of the table, and the WebApp will show the SelectedForumPage.

In this run time view is also shown how a Farmer can send a message in a Forum. Once he/she is in the SelectedForumPage, he can write what he/she wants. The WebApp submit the message to the ForumManager, which will add it to the messages attribute of that forum, in order to allow the WebApp to display it. Then it send it to the Model, which will update the databse adding the new message, in order to maintain synchronisation with other farmers who decide to join that forum from another device.

12) Farmer creates a forum

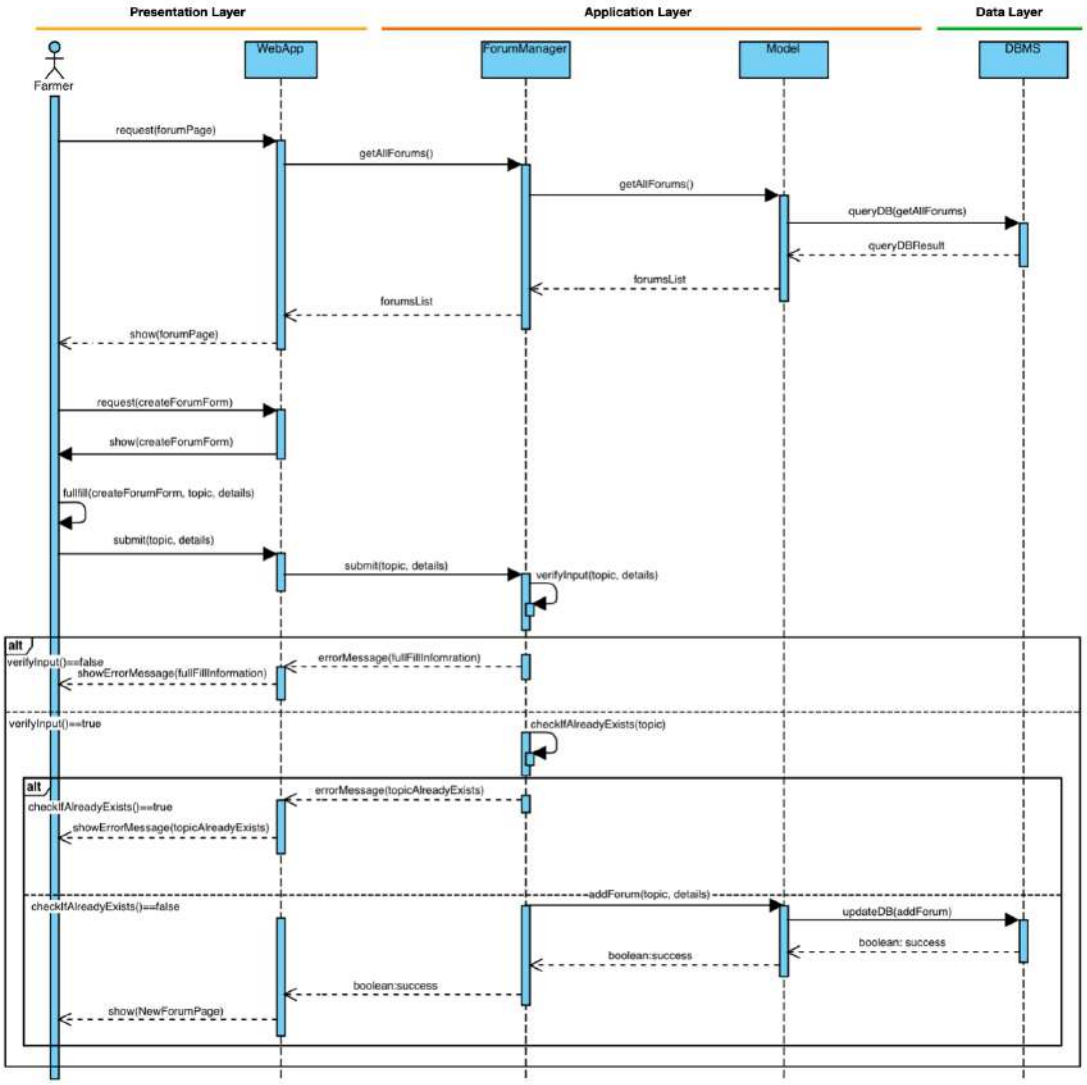

Figure 21: Runtime view: Farmer creates a forum

The Farmer is on the Farmer Page and performs the same actions as described in the previous point. Then it clicks on the button Create and the WebApp shows the createForumForm. The Farmer fills the form providing the topic and a brief description of it. The WebApp submit it to the Forum Manager, which checks if the form has been filled in completely. If not, the WebApp shows an error message to the Farmer, else the manager checks if the forum the farmer wants to add has been already created by another farmer. If it already exists, the WebApp shows another error message, else the Forum Manager and the Model send a query to update the database with the new forum. The Field Manager notifies the WebApp that the operation has been successfully completed and allows it to show the newForumPage to the Farmer.

13) Farmer checks notifications

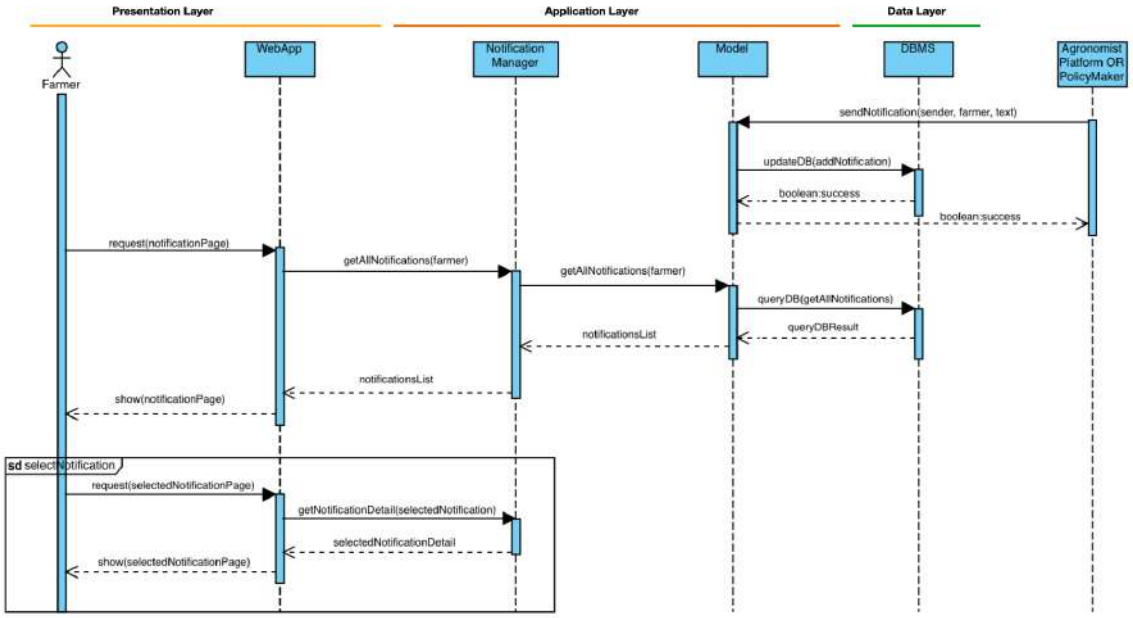

Figure 22: Runtime view: Farmer checks notifications

At the beginning of the runtime view it is shown a Farmer receives a notification. As we explained in the RASD, he/she can receive only notification from Policy Maker or from the AgronomistsPlatform. They are actually two separate entities, which are represented as one to make them easier to read. In both case, the notification is sent to the Model, in order to store it in the Database.

When a Farmer wants to check his/her notifications, he/she clicks on the Notification button in the Farmer Page. The WebApp asks for the notifications to the Notification Manager which forward the request to the Model. It performs a query on the database, and Notification Manager obtains the notificationsList. It sends it to the WebApp which can now displays the NotificationPage.

It is also shown what happens if the Farmer clicks on one of the notifications shown. The WebApp get the selectedNotificationsDetail from the Notification Manager and shows it to the Farmer.

14) PolicyMaker registration

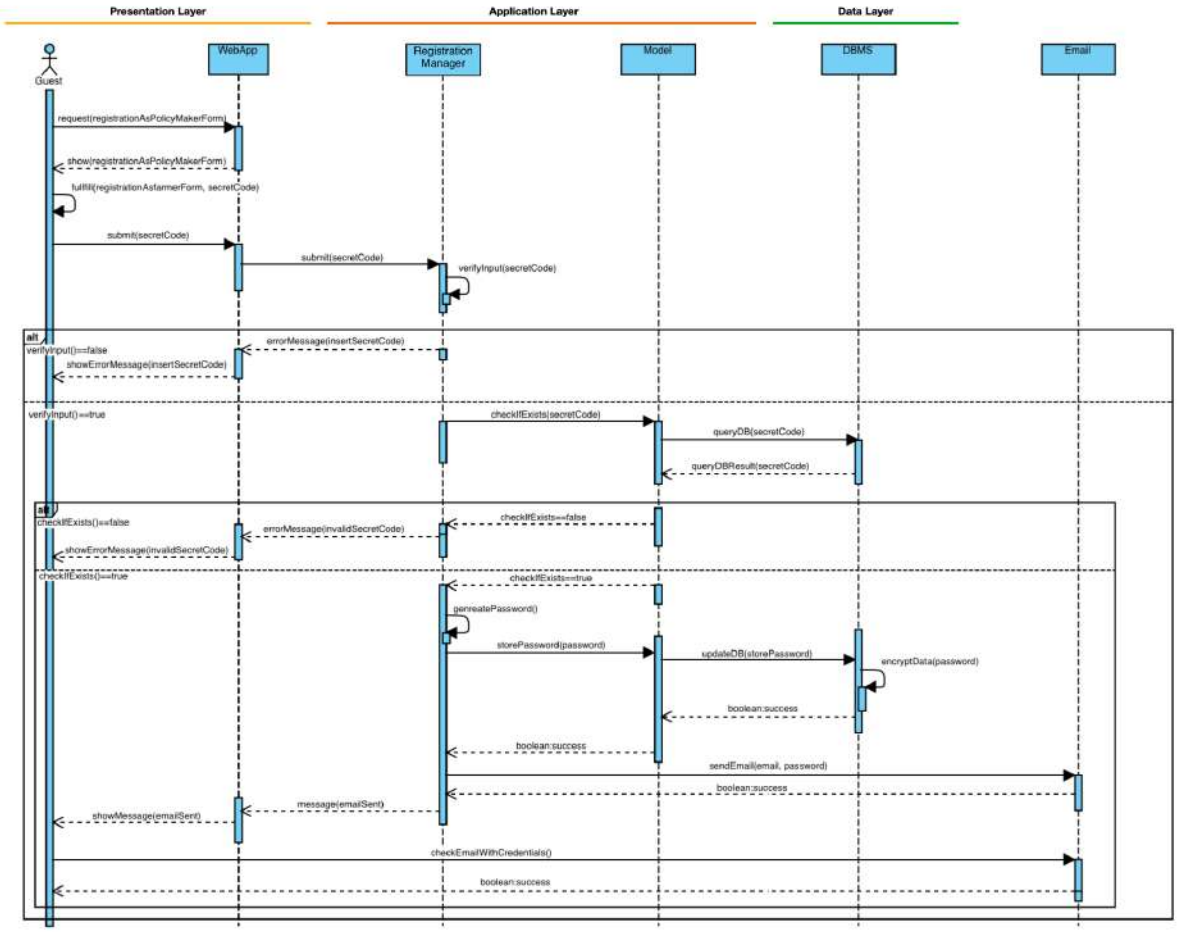

Figure 23: Runtime view: PolicyMaker registration

In this diagram the PolicyMaker registration's is analyzed. We consider that the Guest is on the FirstPage of Dream, where he/she can click on Register As PolicyMaker. The system shows the form with the SecretCode he/she has to provide. He fills the form and submit the Secret Code, which is managed by the Authentication Manager. VerifyInput() is needed to check that a code has been provided. If it returns false, the system shows and error message to the Guest. Otherwise it has to check that the SecretCode exists in the database. In order to do this the Authentication Manager, thanks to the Model, can interact with the DBMS, where are stored all the email of the Policy Makers and the corresponding Secret Codes. The Model checks if the Secret Code the Guest inserted is in the database. If the query returns zero items, it means that the code is not in the database, so checksAlreadyExists() returns false, and the system shows an error message to the Guest. Otherwise, if the query returns one item, checksIfAlreadyExists() returns true, and the Authentication manager can generate a secure password for him/her. It sends it to the Model which can update the Database inserting the password, which will be encrypted by the DBMS. Once the operation has been successful, the Authentication Manager can send the email with the credentials to log into the system to the Policy Maker. The WebApp shows a message to the Guest, in order to notify him/her the the email has been sent. The Guest can checks the Email to see his/her login credentials. From now on the the Guest is registered and is allowed to log into the system.

15) PolicyMaker login

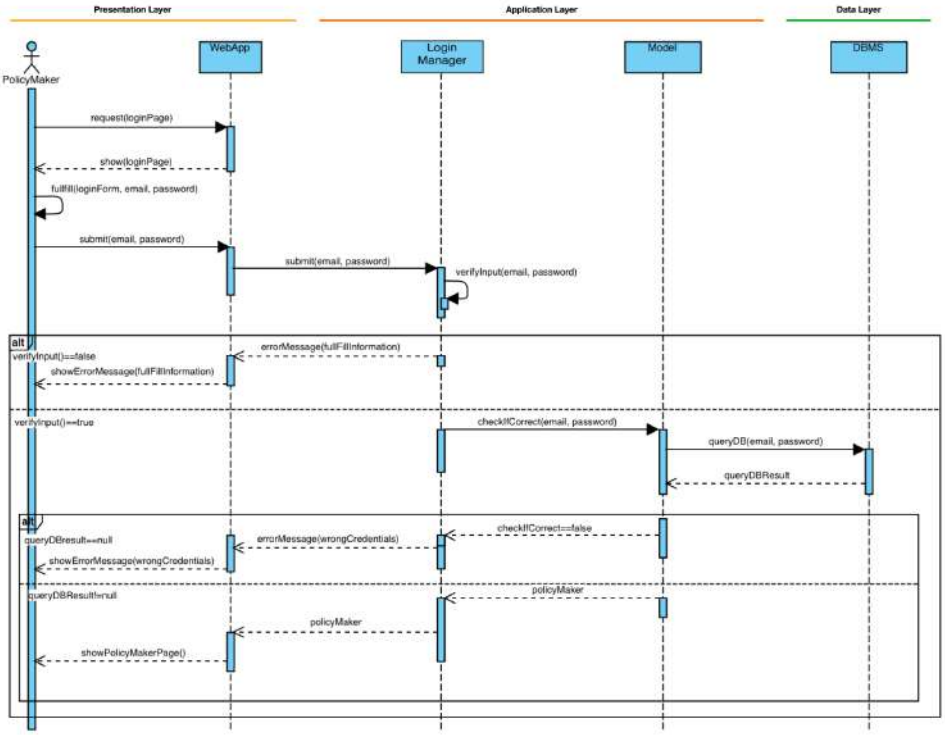

Figure 24: Runtime view: PolicyMaker login

The PolicyMaker login is the same as that of the Farmer. The only difference is that in the end the WebApp shows the PolicyMakerPage (instead of the FarmerPage).

16) PolicyMaker checks productions

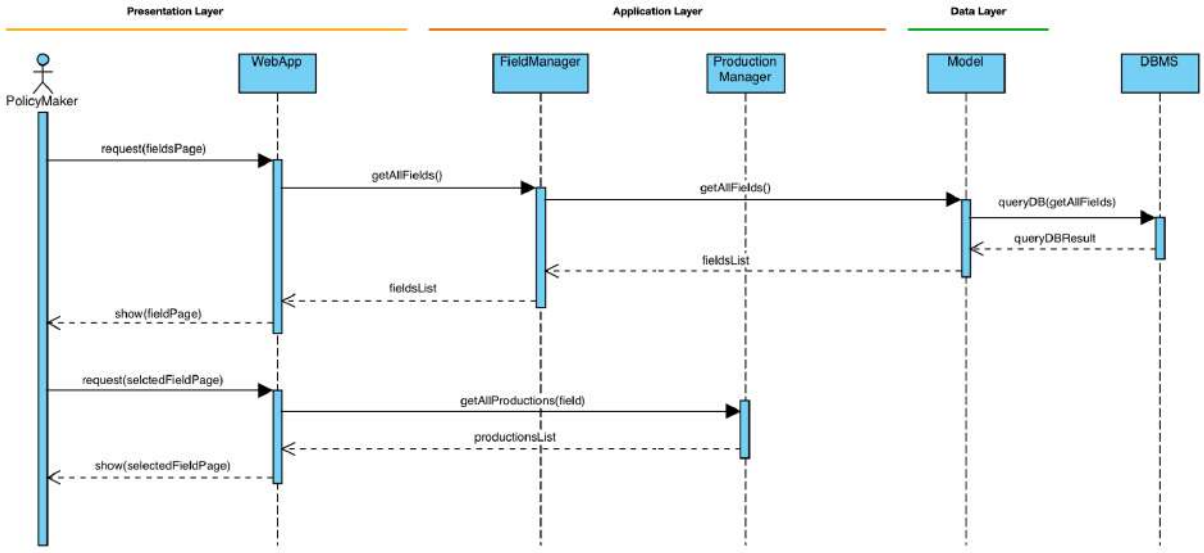

Figure 25: Runtime view: PolicyMaker checks productions

The PolicyMaker is in the PolicyMakerPage and he/she clicks the button Fields. The WebApp asks for the fields to the FieldManager, which forwards the request to the Model. It performs a query on the database and get all the fields inserted by the farmers. So the FieldManager has a fieldList, which is a list of object of type fields. The WebApp receive the list and show the FieldsPage,

with a table which contains all the fields, to the PolicyMaker. When he/she clicks on one of the table rows, the WebApp get from the FieldManager the list of productions of the field contained in that row (because each object of type Field has a list of productions as attribute), and shows the selectedFieldPage to the PolicyMaker, with a table which contains all the productions and the productions' information of that field.

17) PolicyMaker sees best Farmers (and worst Farmers)

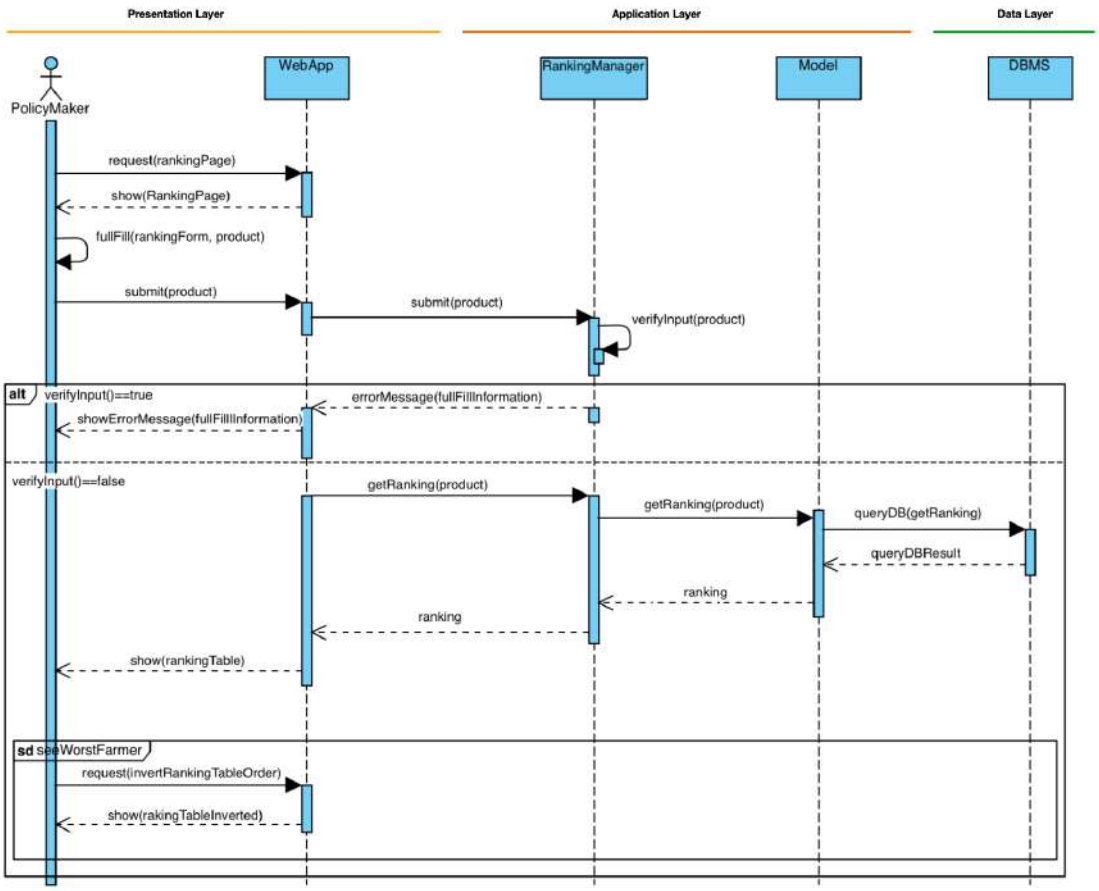

Figure 26: Runtime view: PolicyMaker sees best Farmers (and Worst Farmers)

The PolicyMaker is in the PolicyMakerPage and he/she clicks the button Rankings. The WebApp shows the RankingsPage with the rankingForm, by which the Policymaker can select the product for what he/she want to see the ranking. He/she selects a product, the WebApp submits it to the RankingManager which checks it with verifyInput(). If it returns false the WebApp shows an error message, else the RankingManager asks for the rankings to the Model, which performs a query on the DBMS and receives the ranking for the product selected by the PolicyMaker. The RankingManager receives the ranking and allow the WebApp to show it. In this way the PolicyMaker can see who are the best farmers for the production of the product he/she chose, because the ranking in shown in descending order (from the best farmer to the worst).

For the sake of simplicity, and due to the initial steps are the same, we have also integrated into this view the one describing "PolicyMaker sees worst Farmers". In fact, after the system shows the ranking, the PolicyMaker can click the button Order to invert the ranking order and see who are the worst farmers for the production of the product he/she has chosen.

18) PolicyMaker sends notifications for incentives

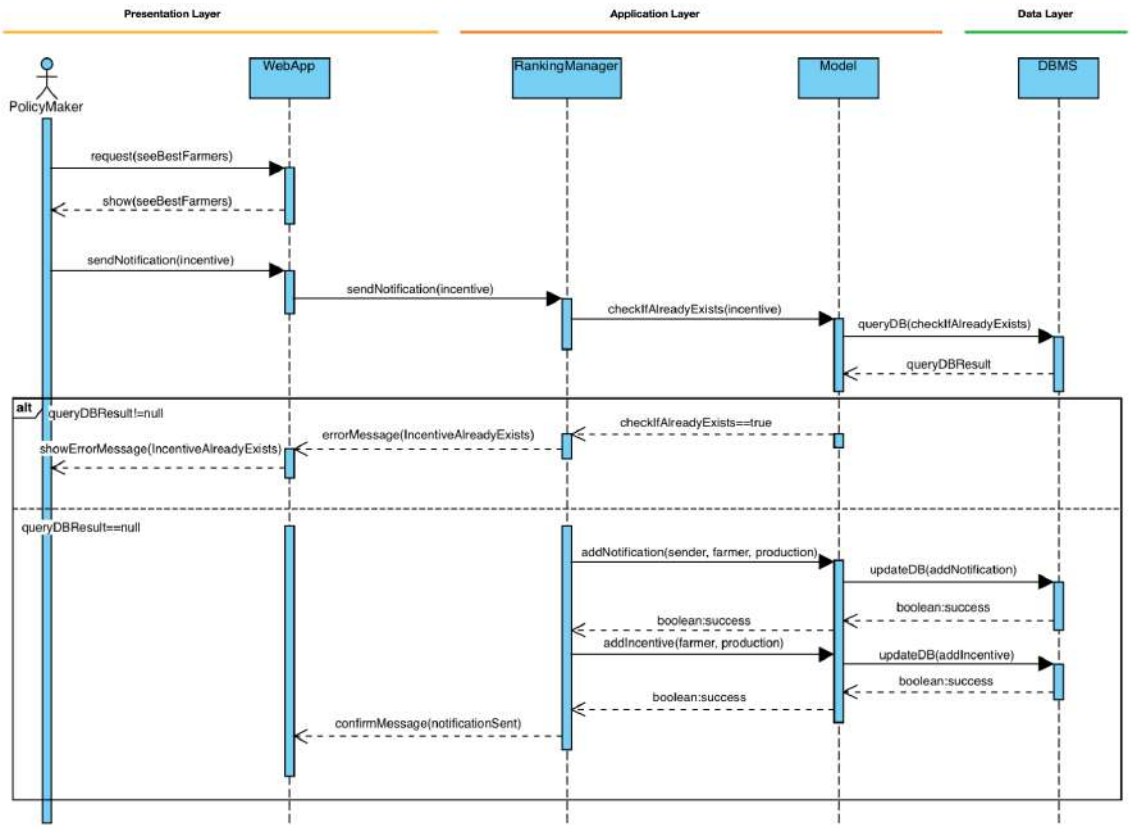

Figure 27: Runtime view: PolicyMaker sends notifications for incentives

The initial steps are those of the runtime view “PolicyMaker sees best Farmers”, which for ease of reading we will not quote, but are summarised in this diagram with “request(BestFarmers)” and “show(BestFarmers)”. The PolicyMaker can click the button Send Incentive near the Farmer to which he/she wants to send the Notification. The Ranking Manager, at first checks if that farmer has already received an incentive for the production, through the Model, which performs a query on the database. If the query returns one element, checkIfAlreadyExists() is set to true, and the WebApp displays an Error Message, otherwise a notification is sent to the Farmer: the Ranking Manager call the method sendNotification(), which interacts with the Model which performs a query in order to update the database and store the new notification, so that the Farmer will see it in his/her NotificationsPage. The PolicyMaker will see, through the WebApp the confirmMessage that the notification has been sent.

19) PolicyMaker sees incentives history

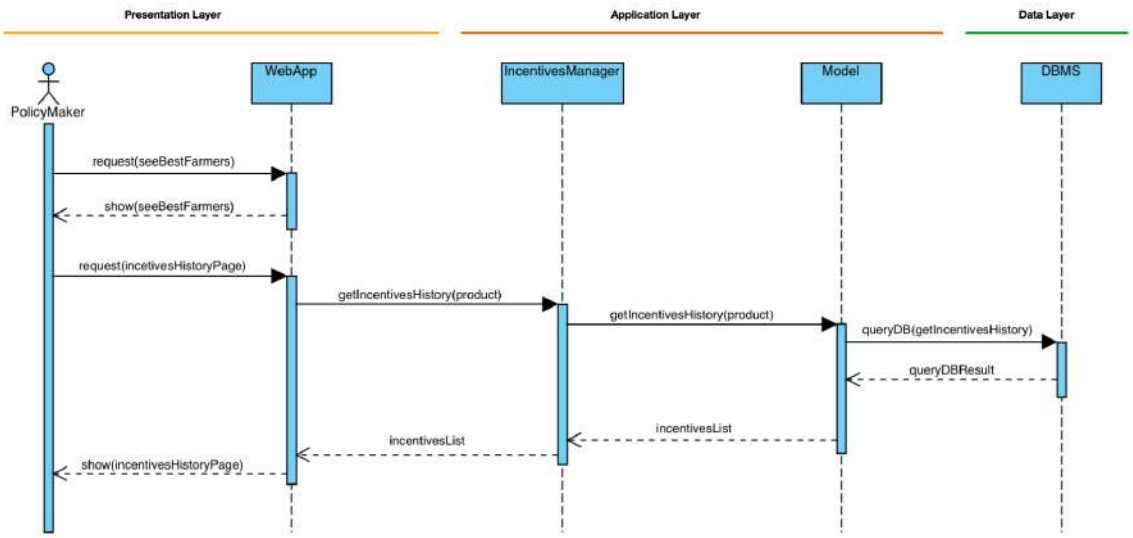

Figure 28: Runtime view: PolicyMaker sees incentives history

The initial steps are those of the runtime view "PolicyMaker sees best Farmers", which for ease of reading we will not quote, but are summarised in this diagram with "request(BestFarmers)" and "show(BestFarmers)". He clicks the button Incentives History. The IncentiveManager asks for the incentives to the Model, which performs a query on the DBMS and receives the list of incentives sent to the farmers for the production of product selected by the PolicyMaker in the RankingPage. The IncentiveManager receives the ranking and allow the WebApp to show the IncentivesPage, with the table containing the list of incentives fetched from the database.

20) PolicyMaker sends requests for help to the Agronomist

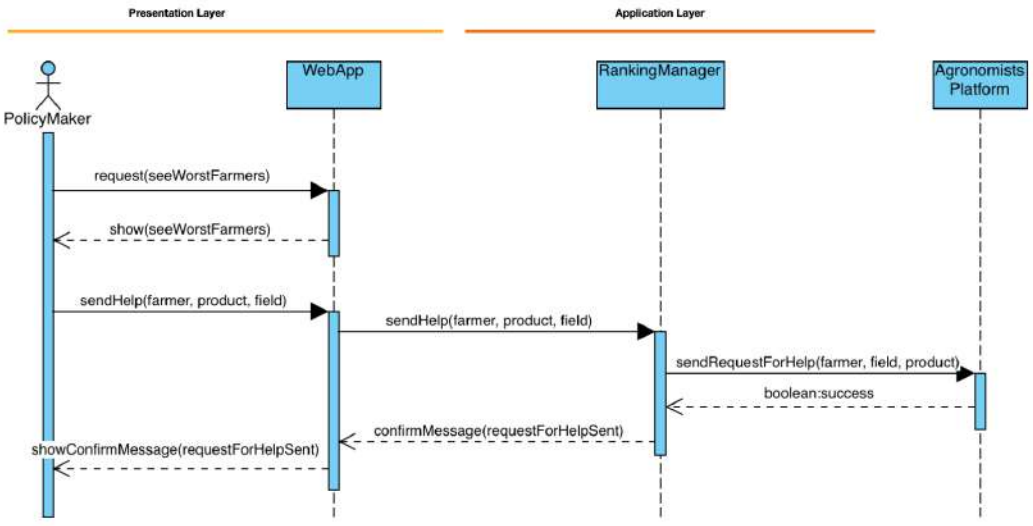

Figure 29: Runtime view: PolicyMaker sends requests for help to the Agronomist

The initial steps are those of the runtime view "PolicyMaker sees worst Farmers", which for ease of reading we will not quote, but are summarised in this diagram with "request(WorstFarmers)" and "show(WorstFarmers)". So the PolicyMaker can click the button Send Help near the Farmer to which he/she wants to send the help. The RankingManager, using the API, sends the request for help to the AgronomistPlatofrm, including the farmer who need help, the field and the produc-

tion in which he/she faced problems. If the request is sent successfully, the WebApp displays a confirmation message.

21) PolicyMaker checks agronomists' reports

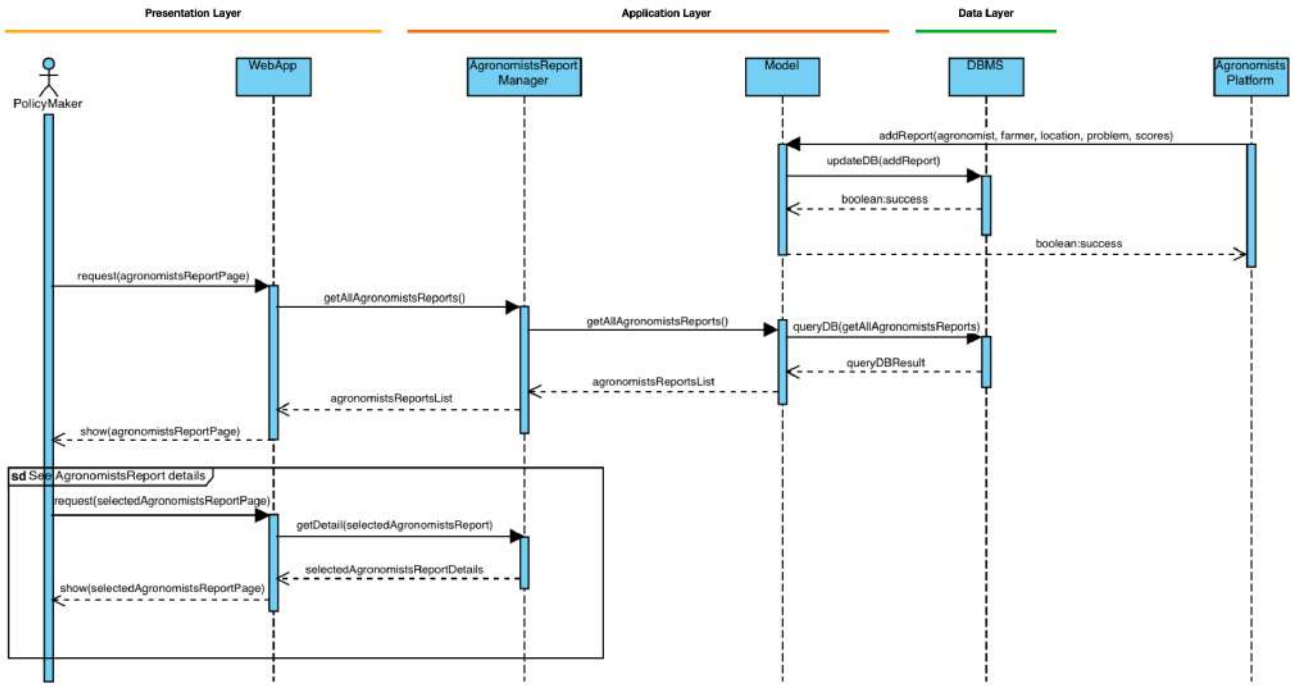

Figure 30: Runtime view: PolicyMaker checks agronomists' reports

The beginning of the runtime view shows how the model receives the reports sent by the AgronomistsPlatform and stores them in the database.

The PolicyMaker is in the PolicyMakerPage and he/she clicks the button AgronomistsReport. The AgronomistsReportManager asks for the reports to the Model, which performs a query on the DBMS and receives the list of agronomistsReport. AgronomistsReportManager receives the list and allow the WebApp to show the AgronomistsReportPage with a table which contains all the elements of the agronomistsReportList.

If the PolicyMaker wants to see more details about a report, he/she can click on one of the rows of the table and the WebApp will show she selectedAgronomistsReportPage, in which there are: the name of the farmer helped, his/her field, his/her score before and after the intervention of the agronomist and other details describing the intervention.

2.6 Component interfaces

This diagram represent in more details the difference interfaces that the system has and how they interacts thanks to the arrows that represents dependency. It is linked to the 8 diagram and a better explanation of methods is presented there. In general, methods written in the Component Interfaces diagrams are not to be considered exactly as the method that the developers will be uses, could be some changes.

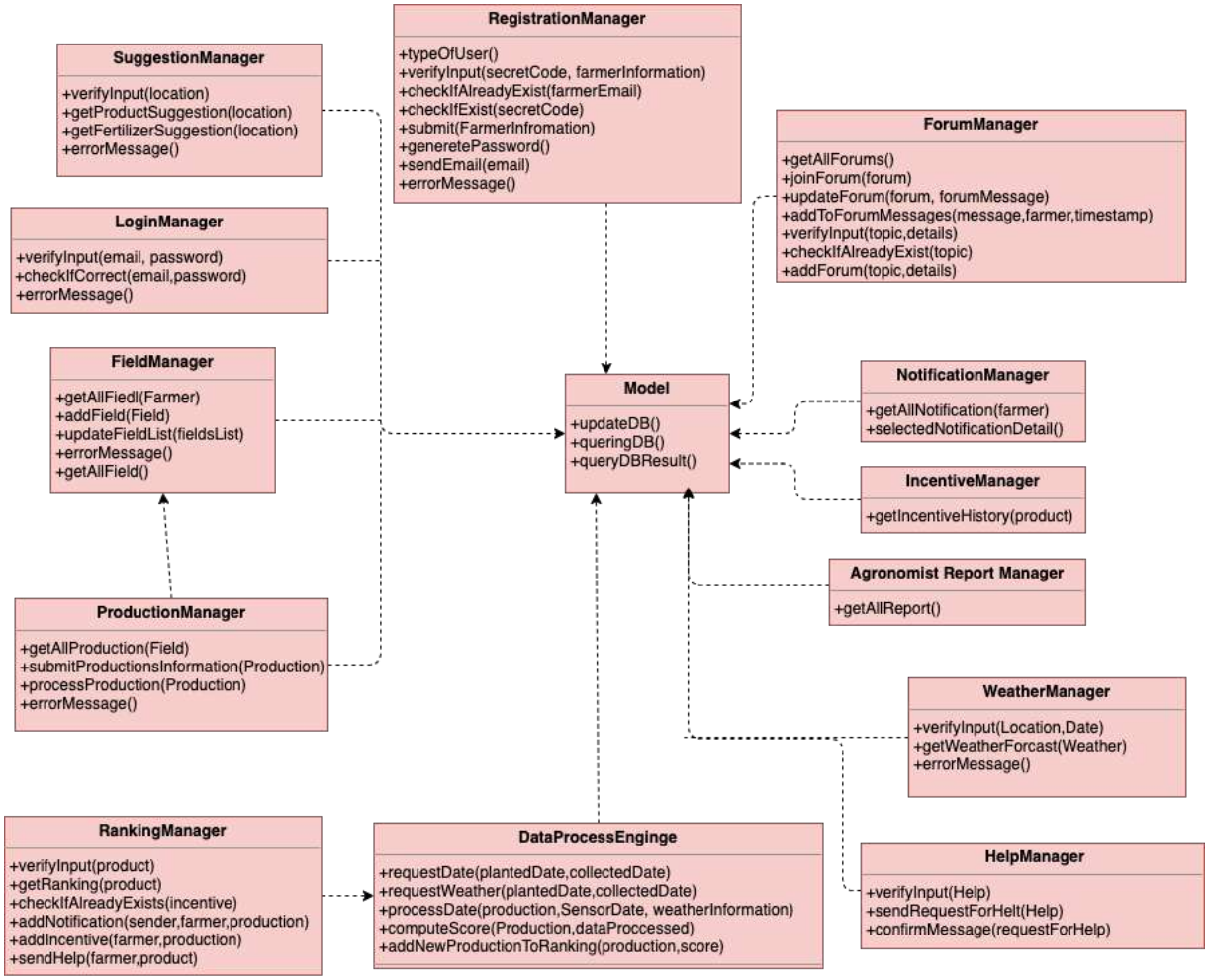

Figure 31: Component Interfaces

2.7 Selected architectural styles and patterns

- **Three-tier architecture**

  The three-tier architecture has been already described in the subsection 2.1, and it refers to the division of the system in three parts: Presentation Layer, Application Layer and Data Layer.

- **MVC - Model View Controller**

  MVC is one of the most used design patter, which provides for the division of the code In blocks with different functionalities: Model, View and Controller. So we have:

  – Model: contains the data access methods.

  – View: takes care of displaying data to the user and manages the interaction between the user and the underlying infrastructure.

  – Controller: receives user commands through the View and reacts by performing operations that may affect the Model and generally lead to a change of state of the View.

  We decided to use this pattern in order to guarantee the reusability of code, flexibility and also to guarantee a parallel development.

- **Client-Server**

The Client-Server paradigm is the one that better fits DREAM. Client-server architecture is an architecture in which many clients request and receive service from the server. Client computers provide an interface to allow a computer user to request services of the server and to display the results the server returns. Servers wait for requests to arrive from clients, manage and then respond to them.

The client is represented by the user devices, which are computer connected to the Internet, either for farmers or for policy makers. The server is represented by the Application Layer and by the Data Layer.

In particular the chosen style is the thin client, because all the computations are done server side, the only goal of the client is to display something to the user and to catch interactions.

2.8 Other design decisions

2.8.1 Score Algorithm

DREAM must have an algorithm that calculates for each production a score to be shown in the ranking. First of all each time a Farmer adds a new Production, the productionManager asks to the DataProccessingEngine to connect with the external components. Trough an API it connects with the Sensor Humidity Soil, the Water Irrigation System and Weather.com. The DataProccessingEngine requests the data from the planted date to collected date's Production in order to have only the useful data. The methods presented in DataProccessingEngine calculates the avg of sensor data, water data and weather data and then it inserts the avg in the algorithm to calculate the score. All these operations are done asynchronously in the system so that it is not slowed down. In this way the farmer can see all the productions added without having to wait for the score to be calculated. In fact the farmer doesn't see the score of each production, only the policy maker needs to see it in the Ranking.

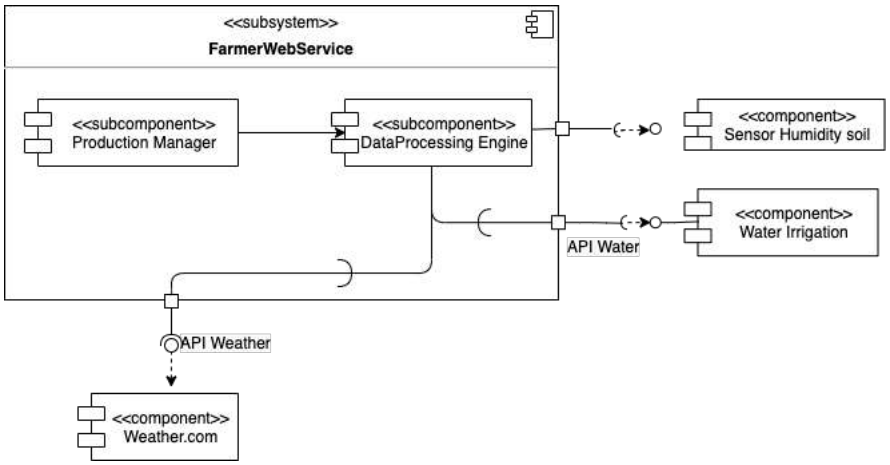

Figure 32: Algorithm

The function which computes the score is called ComputeScore(), which takes in account different arguments. These arguments are have been chosen on the basis of some sections written in the article "Agricultural Productivity Indicators Measurement Guide" by Patrick Diskin. So we have:

- amountCollected: this is the quantity of product collected, which is in kilos.
- amountPlanted: this is the quantity of product planted, which is in kilos.
- avgWaterUsed: this is the average quantity of water used by the Farmer, so it is calculated with the data collected by the water irrigation system between the plantedDate and the collectedDate of the production, it is in litres

- avgRainfall: this is the average amount of rainfall, during the period between the plantedDate and the collectedDate of the production, it is in litres
- avgHumidtySoil: this the average of the humidity soil level detected during the period between the plantedDate and the collectedDate of the production. It is in litres for m^3
- size: the size of the field to which the production belongs, it is in m^2

ComputeScore() computes an operation of this kind:

$$score = \frac{amountCollected - \alpha\ amountPlanted - \beta\ avgWaterUsed - \Delta\ avgRainfall - \varepsilon\ avgHumidtySoil}{size}$$

Thus, the function considers the amountCollected from which it subtracts different factors. All factors that are subtracted are factors that help the farmer. This means that if amountCollected is high and the less these factors are involved in production, the higher the score will be because the farmer has been able to produce well even under unfavourable conditions. On the other hand, if these factors are higher, the score decreases because it means that the farmer has been advantaged.

At first the lossProduct, because the more product that is lost, the lower the score will be.

Then the avgWaterUsed, because the more irrigation water the farmer uses, the more the score will drop.

avgRainfall is subtracted the greater the amount of rain, the more the farmer is helped, therefore the lower his score will be. We also need to consider that the rain helps the farmer up to a certain threshold. So there is a “if” statement in the function that checks the value of avgRainfall. in fact if avgRainfall is greater than the threshold, it means that it puts the farmer in difficulty because it damages production and it is changed in this way: avgRainfall = (-avgRainfall). As a result, in the formula, subtraction becomes addition, thus increasing the score. This reasoning was done in the light of what was learned from reading the article “Excessive rainfall leads to maize yield loss of a comparable magnitude to extreme drought in the United States”.

As far as avgHumidtySoil is concerned, the argument is the same as for avgRainfall: it is subtracted because a damp soil certainly helps production more than a dry soil, but as long as the moisture level is below a certain threshold. If it exceeds this, then we have avgHumidtySoil = (-avgHumidtySoil).

As you can see in the formula there are different coefficients in front of the above mentioned factors. They have two main purposes: the first is to keep the formula consistent in terms of order of magnitude, and the second is to have different weights according to the importance of the factor in the final score calculation.

This weighted sum is then divided by the size of the field, as everything must be proportionate to this factor in order not to have disparities between the farmers, who have fields of different sizes.

3 User Interfaces Design

The user interfaces are part of the Presentation Layer, so they allow the User to inset data and to show him/her the contents that are elaborated by the Application and data Layer, in order to respect the MVC pattern, explained in the previous section. The FirstPage of DREAM is already described in the RASD, but we report it here for completeness.

Figure 33: FirstPage

From the FirstPage an User can decide between different options: Register As Farmer, Register As PolicyMaker or LogIn. If he/she clicks Register As Farmer the WebApp will display the RegisterAs-FarmerPage.

Figure 34: RegisterAsFarmerPage

If he/she clicks Register As PolicyMaker the WebApp will display the RegisterAsPolicyMakerPage.

Figure 35: RegisterAsPolicyMakerPage

If he/she clicks Register As PolicyMaker the WebApp will display the LoginPage.

Figure 36: LoginPage

When the User provides his/her email and password and log into DREAM, the WebApp show the FarmerPage or the PolicyMakerPage depending on whether he/she is a Farmer or a PolicyMaker. The FarmerPage of DREAM is already described in the RASD, but we report it here for completeness. In the FarmerPage there are different buttons through which the Farmer can access the various pages of DREAM from which he can use its functions.

Figure 37: FarmerPage

If the farmer clicks the Weather button the WebApp displays the WeatherPage, with the weatherForm. If the Farmer fills it, DREAM will provide the weather forecasts.

Figure 38: WeatherPage

If the farmer clicks the Field button the WebApp displays the FieldPage, with the fields table, where there are all the fields he/she inserted in the system.

Figure 39: FieldPage

Clicking the button AddField he/she can add a new field, filling the form in the AddFieldPage.

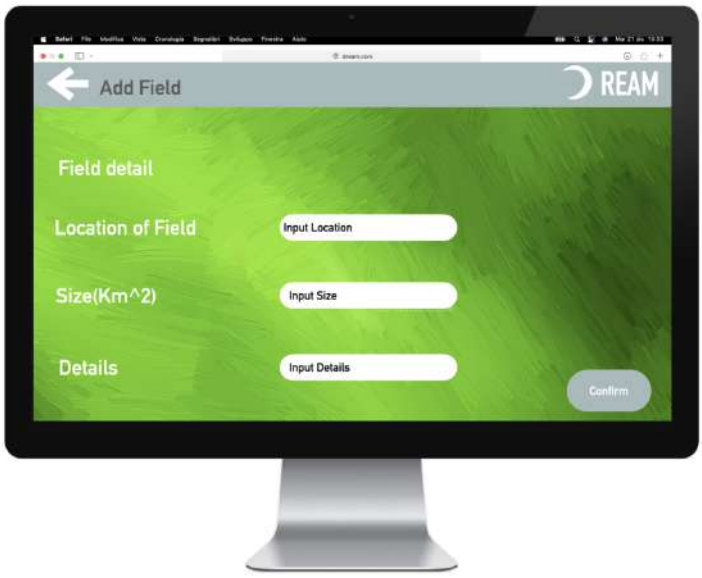

Figure 40: AddFieldPage

If the farmer clicks the Production button the WebApp displays the ProductionPage, with the fields table, where there are all the fields he/she inserted in the system.

Figure 41: ProductionPage

Clicking the button ProductionField he/she can add a new production, filling the form in the AddProductionPage.

Figure 42: AddProductionPage

When the farmer clicks the Suggestions button the WebApp displays the SuggestionPage, with the suggestions table, where there are all the suggestions he/she has already received by the system.

Figure 43: SuggestionPage

Clicking the button AskSuggestion he/she can ask for a new suggestion for a product or a fertilizer, filling the form in the askForSuggestionForm.

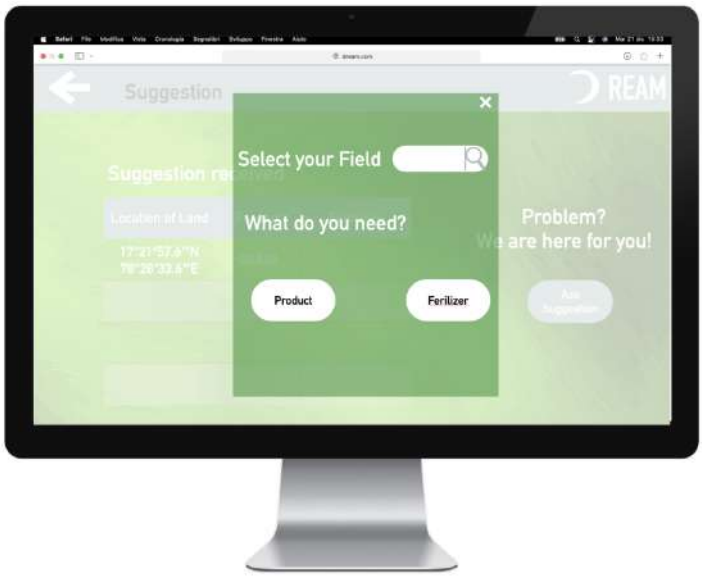

Figure 44: askForSuggestionForm

If the farmer clicks the Forum button the WebApp displays the ForumPage, with the forum table, where there are all the forums created by other farmers.If he/she clicks the button Join he/she will join one of these forums, in order to read other farmers' messages or to write something.

Figure 45: ForumPage

Clicking the button NewForum he/she can create a new forum, filling the newForumForm

Figure 46: newForumForm

If the farmer clicks the RequestForHelp button the WebApp displays the RequestForHelpPage, with the RequestForHelpForm. The Farmer can fills it in order to send a requet for help to the AgronomistsPlatform.

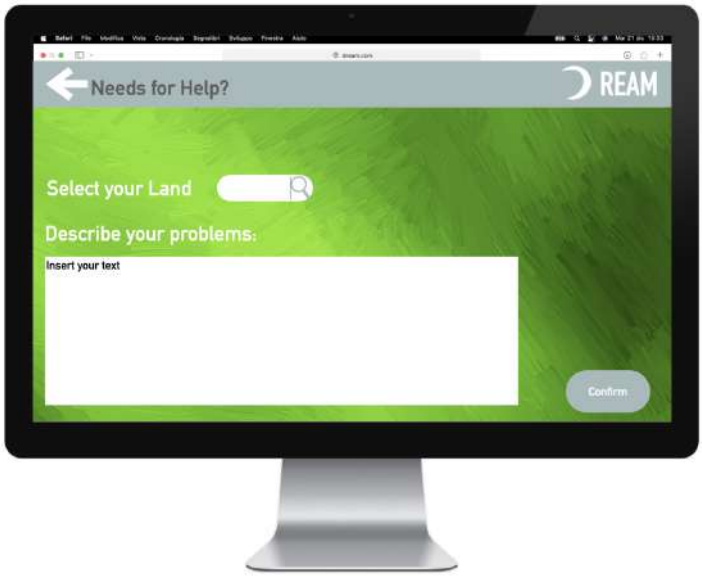

Figure 47: RequestForHelpPage

If the farmer clicks the Notifications button the WebApp displays the NotificationsPage, with the notifications table, where there are all the notifications received by the farmer. If he/she clicks one of the rows of the table the WebApp will show more detail about the selectedNotification.

Figure 48: NotificationsPage

The PolicyMakerPage of DREAM is already described in the RASD, but we report it here for completeness. In the PolicyMakerPage there are different buttons through which the Farmer can access the various pages of DREAM from which he can use its functions.

Figure 49: PolicyMakerPage

If the PolicyMaker clicks the Fields button the WebApp displays the FieldsPage, with the fields table, where there are all the fields the farmers inserted in the system.

Figure 50: FieldsPage

Clicking one of the row of the fields table, the WebApp will display the ProductionsPage, with the productions table, with all the productions of the selected Field.

Figure 51: ProductionsPage

If the PolicyMaker clicks the Ranking button the WebApp displays the RankingPage, with the rankingForm, thanks to which he/she can select the product for which he/she wants to see the ranking. So the WebApp will displays the ranking tables, with the productions and their score. He/She can clicks the Send Incenitve button or the Send Help one in the ranking table to send notifications of incentives or requests for help to the AgronomistsPlatform.

Figure 52: RankingPage

If he/she clicks the IncentivesHistory button in the RankingPage, the WebApp will show the IncentivesHistoryPage, with the IncentivesHistory table, which contains all the already given incentives for the production of the product selected in the rankingForm.

Figure 53: IncentivesHistoryPage

If the PolicyMaker clicks the AgronomistsReport button the WebApp displays the AgronomistsReportPage, with the AgronomistsReport table, where there are all the agronomists' reports received by the AgronomistsPlatfrom. Clicking one of the row of the AgronomistsReport table, the WebApp will display the SelectedAgronomistsReportPage, which contains more details about the SelectedAgronomistsReport.

Figure 54: AgronomistsReportPage

4 Requirements Traceability

Requirement	Description	Component
R1	The system shall allow a Guest to register	Registration Manager
R2	The system shall not accept an already Farmer email	Registration Manager
R3	The system shall be allow the Farmer to insert his/her information (Farmer's information)	Registration Manager
R6	The system shall send an email to the Farmer after he/she fills the form to confirm his/her registration	Registration Manager
R7	The system shall store the data provided by the Farmer	Model
R8	The system shall allow the Farmer to login to Dream by entering his/her credentials	Login Manager
R9	The system shall check if the password and the email are correct	Login Manager
R10	The system shall display the FarmerPage with the different services it offers to the Farmer	Login Manager
R11	The system shall be able to provide weather forecasts for a certain area and period	Weather Manager
R12	The system shall allow the Farmer to insert his/her field's information	Field Manager
R13	The system shall store the data about Farmer's fields	Model
R14	The system shall allow the Farmer to insert information of his/her Productions (Production's information)	Production Manager
R15	The system shall store the Production's Information	Model
R16	The system shall allow the Farmer to see his/her Fields and the Fields' Information	Field Manager
R17	The system shall allow the Farmer to see his/her Productions and the Productions' Information	Production Manager
R18	The system shall allow the Farmer to ask for suggestions about a product for a certain Field	Suggestion Manager
R19	The system shall allow the Farmer to ask for suggestions about a fertilizer for a certain Field	Suggestion Manager
R20	The system shall provide suggestions about a product for a certain field	Suggestion Manager
R21	The system shall provide suggestions about a fertilizer for a certain field	Suggestion Manager
R22	The system shall allow the Farmer to ask for help for a certain Field to the agronomist	Help Manager
R23	The system shall allow the Farmer to check Notifications	Notifications Manager

R24	The system shall allow the Farmer to create a forum based on a certain topic and description	Forum Manager
R25	The system shall allow the Farmer to join a forum created by another Farmer	Forum Manager
R26	The system shall allow the Farmer to send messages in a forum	Forum Manager
R27	The system shall allow the Farmer to read other Farmers' messages in a forum	Forum Manager
R29	The system shall allow the policy maker to insert his/her secret code	Registration Manager
R30	The system shall generates the login credentials of the policy maker	Registration Manager
R31	The system shall store the login credentials of the policy maker	Model
R32	The system shall send an email to the policy maker after he/she inserts the secret code with the credentials to login	Registration Manager
R33	The system shall allow the policy maker to login by inserting his/her credentials	Login Manager
R34	The system shall display the PolicyMaker-Page with the different services it offers to the policy maker	Login Manager
R35	The system shall allow the policky maker to check the Farmers and the fields they owns	Field Manager
R36	The system shall allow the policy maker to see Productions' Information for each field	Production Manager
R37	The system shall allow the policy maker to see the weather condition during the period of a certain Production	Data Process Engine
R38	The system shall allow the policy maker to see the amount of water used by a Farmer for a certain Production	Data Process Engine
R39	The system shall allow the policy maker to see the average humidity soil level of a field during the period of a certain Production	Data Process Engine
R40	The system shall allow the policy maker to see the Ranking of the best Farmers for a certain product	Ranking Manager
R41	The system shall allow the policy maker to see the Ranking of the worst Farmers for a certain product	Ranking Manager
R42	The system shall allow the policy maker to send notification that a Farmer will receive an incentive	Ranking Manager
R43	The system shall store the history of the given incentives	Incentives Manager

R45	The system shall not allow the policy maker to send notifications of incentives to Farmers who have already received one for the Production of that product	Ranking Manager
R46	The system shall allow the policy maker to send messages to the agronomists in order to help the worst Farmers	Ranking Manager
R47	The system shall allow the policy maker to check the report of the agronomist who helped Farmer who requested for help	Report Manager

5 Implementation, Integration and Test Plan

5.1 Implementation Plan

The system is divides in various subsystems:

- The presentation Layer : Webapp
- The deployment Layer : FarmerWebService and PolicyMakerWebService
- The Data Base Layer

The implementation of DREAM will follow this division, but also the integration of the various subparts and the testing will be done module by module, following a bottom-up strategy, paying close attention to the order in which these modules are implemented.

First of all we decide the DataBase Server and we implement it with all the entities and the relations. In particular, a relational DataBases is used. Then we need to implement all those classes that make up the entities in the db, so Farmer, Policymaker, Field, Production etc, which will be part of a package called “model”. Only then we can implement all the Managers that make up the FarmerWebService 6 and the PolicyMakerWebService 7, because they use the classes created in the package "model". The last part of the implementation is focused on the WebApplication, in order to create an interface for the User simple and as intuitive as possible.

It is important to notice that the external component need to be tasted and implemented when their related model are implemented and tested.

5.2 Integration Plan

The integration of all the different components is described and divided in different phases, which follow the order described below.

- **Integration with the DBMS** As you can see fig.55 the interaction with the database is allowed and handled by the Model. The model is in charge of making requests to the database and receiving responses from it.

Figure 55: Model DB

- **Integration with external services** This phase is related to the integration with the external services used by DREAM, which takes place thanks to the API. As we have explained, there are several external services that dream uses. We list them below:
  - Weather.com: used in order to fetch the weather forecasts. It interacts with the WeatherManager in order to allow the farmer to see the weather forecasts, and by the DataProcessEngine which collects and processes data about the weather during a certain production in order to show them to the policymaker or to use them in the algorithm which compute the score of that production.

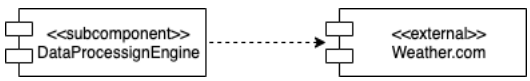

Figure 56: Weather.com integration with Data Processing Engine

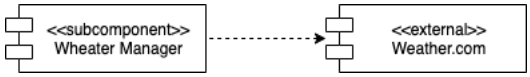

Figure 57: Weather.com integration with Weather Manager

– AgronomistPlatform: it is used by the agronomist to receive and respond to the request for help sent by the farmers or by the policy makers for the farmers. It interacts with the Help-Manager, when a farmer sends a request for help, and with the RankingManager when a policy maker sends one. Moreover it can interact with the Model(that interacts with the DB) in order to send Notifications to the farmer in reply to their request.

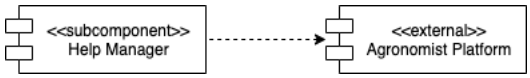

Figure 58: Help Manager integration with Agronomist Platform

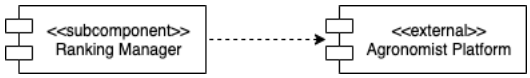

Figure 59: Ranking Manager integration with Agronomist Platform

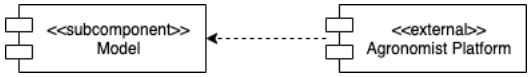

Figure 60: Model integration with Agronomist Platform

– WaterIrrigationSystem: it is integrated with the DataProcessEngine to collect data about the amount of water used by the farmer in a specific Production.

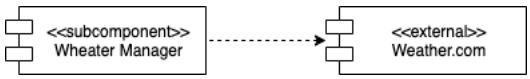

Figure 61: Model integration with Agronomist Platform

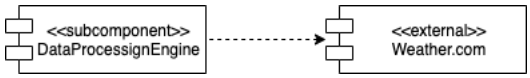

Figure 62: Data Processing Engine integration with Weather.com

– Humidity soil sensors: it is integrated with the DataProcessEngine to collect data about the humidity soil level in a specific Production

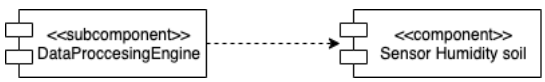

Figure 63: Data Processing Engine integration with Sensor

– Email: the RegistrationManager use the API in order to send email during the registration process.

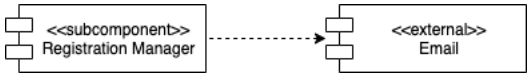

Figure 64: Registration Manager integration with Email

- **Integration of the components of the Application Layer**

  Each Manager is integrated with the Model, because they need to send or receive data from the database. The Model works like a proxy between the several Manager and the Data Base

Figure 65: Registration integration Model

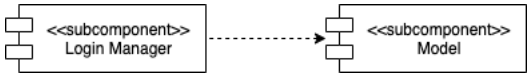

Figure 66: Login integration Model

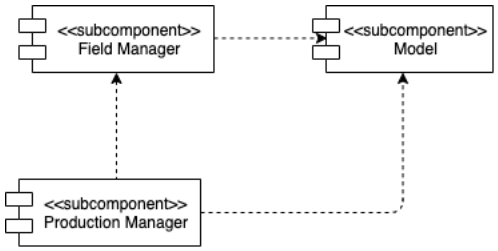

Figure 67: Field integration Model

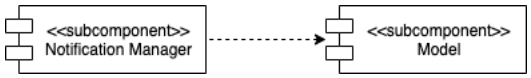

Figure 68: Notification integration Model

Figure 69: Suggestion integration Model

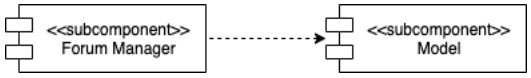

Figure 70: Forum integration Model

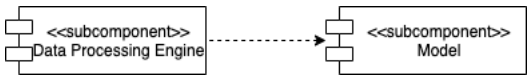

Figure 71: Data Processing Engine integration Model

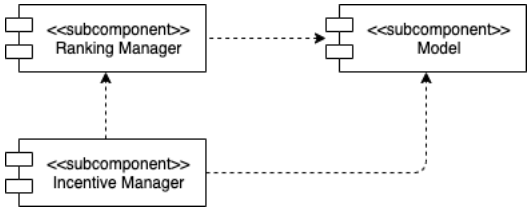

Figure 72: Ranking Manager and Incentive Manager implementation Model

- **Integration of Presentation and Application Layer** This is the final phase of the integration, with the highest level components. It should be done last,or after of the application server parts have been developed and integrated. This last integration allow the system to show what it processes in the Application and Data Layer to the Users.

5.3 Testing Plan

Considering the structure of the system, and the dependence of the various components on each other, the testing strategy chosen is the bottom up. It is not necessary to wait all the development of the component but we can start the integration and it's testing during the implementation. Starting from the low level of the hierarchy, we test every single component, then, step by step the integration of macro-components.

It should be noted that each integration in the same level of hierarchy, defined by the groups of integration in the previous subsection, is independent and there is no specific order to complete them. For example, we can integrate and test the AgrnomistsPlatofrm with the HelpManager or with the RankingManager at first, and then Weather.com with the Wetahermanager , or vice versa: this will not be a problem. Instead if we test and integrate the DataProcessEngine with the Model before having done the integration obove the external service will create problems. In this way, the integration process and its testing are more flexible.

Regression testing will be implemented as well, in the same time we add new model, in order to allow changing if it is necessary. Enfact the regression testing allows to ensure that previously developed and tested software still performs after a change.

6 Effort Spent

Member Group :Alessandro Pindozzi

Task	Hours
Purpose, scope, definitions	1
Component view	4
Class view	2
Runtime view	11
Component interfaces	0:30
Selected architectural style and patterns	1
User interfaces	1:30
Algorithm	2:30
Requirements traceability	1
Implementation, integration and testing	4
Document Organization	4

Member Group :Diletta Quarticelli

Task	Hours
Purpose, scope, definitions	1
Component view	11
Class view	3
Runtime view	4
Component interfaces	1
Selected architectural style and patterns	0:30
User interfaces	2:30
Algorithm	1
Implementation, integration and testing	3
Requirements traceability	1
Document Organization	4

