DREAM project by Federica
Tommasini & Alessandro Verosimile

Design Document

Deliverable:	DD
Title:	Design Document
Authors:	Federica Tommasini, Alessandro Verosimile
Version:	1.0
Date:	9-January-2021
Download page:	https://github.com/federicatommasini/TommasiniVerosimile.git
Copyright:	Copyright © 2022, Federica Tommasini and Alessandro Verosimile – All rights reserved

Contents

Table of Contents 3

List of Figures 4

List of Tables 4

1 Introduction 6
- 1.1 Purpose 6
- 1.2 Scope 6
- 1.3 Definitions, Acronyms, Abbreviations 7
  - 1.3.1 Definitions 7
  - 1.3.2 Acronyms 7
  - 1.3.3 Abbreviations 8
- 1.4 Revision history 8
- 1.5 Reference documents 8
- 1.6 Document structure 8

2 Architectural Design 10
- 2.1 Overview 10
- 2.2 Component view 11
  - 2.2.1 Class Diagram 11
  - 2.2.2 Component Diagram 12
- 2.3 Deployment view 18
- 2.4 Runtime view 19
- 2.5 Component interfaces 24
- 2.6 Selected architectural styles and patterns 25
- 2.7 Other design decisions 25
  - 2.7.1 Used algorithms 25

3 User Interface Design 29
- 3.1 Farmer Interfaces 29
- 3.2 Policy Maker Interfaces 35

4 Requirements Traceability 40

5 Implementation, Integration and Test Plan 44
- 5.1 Implementation plan 44
- 5.2 Integration and Testing 44
  - 5.2.1 Sequence of components integration 45

6 Effort Spent 50

7 References 51

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

List of Figures

1 High level architecture. 10
2 Class Diagram. 11
3 Component Diagram. 12
4 UML Diagram 15
5 Deployment Diagram. 18
6 1st sequence diagram 19
7 2nd sequence diagram 20
8 3rd sequence diagram 20
9 4th sequence diagram 21
10 5th sequence diagram 21
11 6th sequence diagram 22
12 7th sequence diagram 22
13 8th sequence diagram 23
14 9th sequence diagram 23
15 10th sequence diagram 24
16 Interfaces' dependencies diagram 24
17 Soil condition Algorithm diagram 27
18 Meteo condition Algorithm diagram 28
19 Log in interface (on the left) and Home interface (on the right) for a farmer of Dream. . . 29
20 Meteo forecasts interface (on the left) and Suggestion interface (on the right) for a farmer of Dream. 30
21 Upload suggestion interface (on the left) and Visualize suggestion interface (on the right) for a farmer of Dream. 31
22 Request for help interface (on the left) and Forum interface (on the right) for a farmer of Dream. 32
23 Create topic interface (on the left) and Visualize topic interface (on the right) for a farmer of Dream. 33
24 Profile interface (on the left) and Messages interface (on the right) for a farmer of Dream. 34
25 Log in interface (on the left) and Home interface (on the right) for a policy maker of Dream. 35
26 Environmental information interface (on the left) and Meteo forecasts interface (on the right) for a policy maker of Dream. 36
27 Soil measurements interface (on the left) and Rankings interface (on the right) for a policy maker of Dream. 37
28 Water consumption interface (on the left) and Evaluate farmer interface (on the right) for a policy maker of Dream. 38
29 Messages interface for a policy maker of DREAM. 39
30 Integration of the copy manager service components with the database. 45
31 Integration of the information service and message service components with the database. 45
32 Integration of the Farmer service with the other components. 46
33 Integration of the Help request service. 46
34 Integration of the Suggestion service. 47
35 Integration of the Forum service. 47
36 Integration of the Farmer Login service. 48
37 Integration of the Ranking service. 48
38 Integration of the Evaluation service. 49
39 Integration of the Water consumption service. 49
40 Integration of the Policy maker login service. 49

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

List of Tables

1 Table of definitions 7
2 Table of acronyms 7
3 Table of abbreviations 8
4 Table of versions 8
5 Table requirements' mapping to the software components 43
6 Effort spent by student 1 50
7 Effort spent by student 2 50

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

1 Introduction

1.1 Purpose

The purpose of this document is to describe in a more specific and precise way, with respect to the RASD document, the behaviour and the characteristics of the system. In particular the choices regarding the architecture will be justified and a description of the specific components, needed for the implementation of the system, run-time processes and all other design choices will be analyzed. Furthermore, some additional and more specific user interfaces regarding the system both from the side of the farmer and from the side of the policy maker will be presented. Finally, the implementation and integration plan and the testing plan will be showed in full details. All these elements, exposed in the document, will be used by different stakeholders and by the software engineers that are in charge of the project's implementation and testing, in order to be able to understand the detailed specifications without any ambiguity.

DREAM is a project that aims to improve the productivity of the agricultural system in Telengana, by means of creating a software that makes simpler for the users to access to the information collected by different sources, such as agronomists, Telengana's government, sensors deployed on the territory and the water irrigation system. In this way, farmers can easily optimise their work looking at meteorological forecasts, environmental conditions and personalized suggestions; in addition, they can easily evaluate their production with respect to the others'. Whilst, policy makers, not only can precisely identify the farmers who perform well and the ones that are facing problems and, therefore, need some help, but can also understand if the steering initiatives carried out by agronomists are useful. Furthermore, the communication between the characters becomes more clear and fast.

1.2 Scope

Agriculture plays a fundamental role in the economy of Telengana's state, not only because of the contribution it gives to the wealth of the Indian country, but also because of the large amount of people involved in it. For these reasons, the government undertakes to promote initiatives aimed at improving the quality of production. In order to reach this goal, the DREAM system has been designed to be used by farmers and policy makers with two different modalities. A farmer is a person who owns a land and cultivates it producing a certain amount of food; while a policy maker is a person who is in charge of evaluating the activities of the farmers and managing practices to improve the agricultural system.

The system must allow a farmer to visualize several information useful to plan his work in the most efficient way. These information regard meteorological forecasts and personalized suggestions concerning specific crops to plant or specific fertilizers to use. What a farmer can see is filtered by his location, the type of products that he cultivates or the environment conditions in which he works. In order to create a proper communication network they should also be able to upload specific information about their production, such as the products they cultivate with the respective amount, the fertilizers they use and the practices they put in action.

Another functionality provided by the system is the possibility to request help for a farmer who is in critical conditions. The system should manage such requests dispatching them to other farmers who are able to give them adequate suggestions. In the end, the farmer should be able to confront ideas with other farmers through the means of a discussion forum.

For what concerns policy makers, they can visualize rankings of farmers based on the quantity of their production filtered by the geographical area, meteorological and soil conditions and on the improvements during a certain period of time. Through these rankings, policy makers can identify which farmers are performing well and which aren't, and decide to send them, respectively, some incentives or the help of agronomists. In addition, policy makers can evaluate the farmers' work, relying on the rankings, and assign them a status that represent the quality of their work. Analyzing the improvements that have been made by the farmers through the rankings, policy makers are also able to understand if the mechanism of help carried out by good farmers and agronomists is working as desired. Finally, they can also visualize

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

the farmers' water consumption and, if necessary, ask them to reduce it.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Definition	Description
Status	An evaluation of the quality of a farmer's work established by a policy maker.
Suggestion	Information inserted by a farmer which has a good status and that regard the best practice to use in different situations.
Help request form	A help request form is a section of the application that contains some fields that have to be filled in with textual information concerning a help request.
Discussion forum	A discussion forum is a section of the application in which the farmers can exchange opinions about a particular topic of interest.
Improvement Ranking	Ranking of the farmers that cultivate a specific product based on the percentage increase of the quantity produced (per square meter) in a certain period of time.
Soil ranking	Ranking of the farmers that cultivate a specific product based on the quantity produced (per square meter) and classified by different soil conditions.
Meteo Ranking	Ranking of the farmers that cultivate a specific product based on the quantity produced (per square meter) and classified by different meteorological conditions.
Zone Ranking	Ranking of the farmers that cultivate a specific product based on the quantity produced (per square meter) and classified by different geographical zones.
Conflicting data	The email address inserted during the registration process by a farmer is defined as a conflicting data if there are already others users of the DREAM application that have registered with the same address.
Valid credentials	The term credentials is referred to the email and the password inserted by a farmer during the login process; they are said to be valid if they correspond to the ones inserted by such farmer during the successful registration process.
Valid recognition code	The code inserted by the policy maker during the log in process is said to be valid if correspond to one of the codes that have been previously used to register the policy makers.

Table 1: Table of definitions

1.3.2 Acronyms

Acronym	Description
DD	Design Document
RASD	Requirement Analysis and Specification Document
API	Application Programming Interface
DBMS	Database Management System
HTTP	Hypertext Transfer Protocol
TCP-IP	Transmission control protocol - Internet protocol

Table 2: Table of acronyms

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

1.3.3 Abbreviations

Abbreviation	Description
Rn	n-th functional requirement

Table 3: Table of abbreviations

1.4 Revision history

Version	Date	Description
1.0	09/01/2022	First version

Table 4: Table of versions

1.5 Reference documents

- Specification document: R & DD Assignment A.Y. 2021-2022
- Course slides
- AGRO-METEOROLOGY 20/04/2020
  link: https://www.rvskvv.net/images/II-Year-II-Sem_Agro-Meteorology_a_20.04.2020.pdf
- How Does Rainfall Influence Annual Agricultural Growth Rates in India?
  link: http://irapindia.org/images/irap-Occasional-Paper/Agricultural-Growth-Model-India-Gujarat.pdf

1.6 Document structure

- **Section 1:** In this section, it is described the purpose of the document and, more in general, the purpose of the DREAM project. Furthermore, in the scope section it is analyzed the problem that the system aims to solve, providing a description of the environment in which the system will be used and the goals that it has to reach in order to satisfy the users' needs. In addition, it is reported a list of definitions, acronyms and abbreviations, with their respective description, that will be used throughout the document. Finally, it contains other sub-sections dedicated to the revision history, the reference documents and the document structure.
- **Section 2:** This section deals with the architectural design of the system. It contains: an overview on the most important choices made regarding the architecture, a class diagram, a component diagram and a deployment diagram, each of them paired with the correspondent analysis and description. In addition it contains the runtime view, which includes the updated sequence diagrams with respect to the software components defined in the component diagram. Finally, it contains a sub-section that highlights the utilized interfaces and their dependencies, one that describes the architectural styles and design patterns that have been chosen and one that presents other design decisions.
- **Section 3:** This section contains the graphical user interface mockups for the different views of the system.
- **Section 4:** This section contains a mapping of the requirements, which have been defined in the RASD document, to the software components (described in this document) that will ensure that they are satisfied.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

- **Section 5:** This section presents the implementation, integration and testing plan that it has been decided to adopt for the developing of the system.
- **Section 6:** This section indicates the effort spent by each group member while working on this document.
- **Section 7:** This section contains the references.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

2 Architectural Design

2.1 Overview

The DREAM system involves three principal elements in order to properly provide its functionalities. These elements are a client, a server and a database. The client can be equivalently a mobile application or a web browser and can be used by either a farmer or a policy maker. The server is needed for both the communication among clients and between the client and the data retrieved from the database. The interaction between the clients and the server is managed through a load balancer, which is a device that deals with the distribution of the clients' requests to the multiple servers of the system. Furthermore, the server is used for the interaction with external data sources by means of some APIs. These data will be periodically copied into the internal database of the system that also contains several information inserted by all the users of the system. In order to implement these specifics, it is necessary to use a three - tier architecture. This architecture consists in the division of the system in three different groups of components, that contains the logic to aim specific different purposes:

- **Client tier:** it contains the components responsible for views and controllers that define the user interfaces. All the data needed for the consistency of the user interfaces are obtained through the communication with the application logic tier in order to avoid any unnecessary data allocation on the client side. This choice implies the usage of a thin client: such type of client needs to be connected with the server to provide the system's functionalities.

- **Application logic tier:** it contains all the logic of the application, which is needed to process the data collected through the client tier and retrieved from the data tier. It can also add, delete or modify data in the data tier, with which he communicates using API calls. Finally, he communicates with external services (also in this case with API calls), in order to retrieve relevant information for the system and copy them in the internal database.

- **Data tier:** it corresponds to a database which contains all the data relevant to the system's scope. These data regard information obtained by external services (i.e. information about meteorological forecasts, soil conditions, water consumption and data collected by agronomists), information provided by the users and finally, the association between each user and its credentials. The data tier is consulted by the application tier whenever such information are needed.

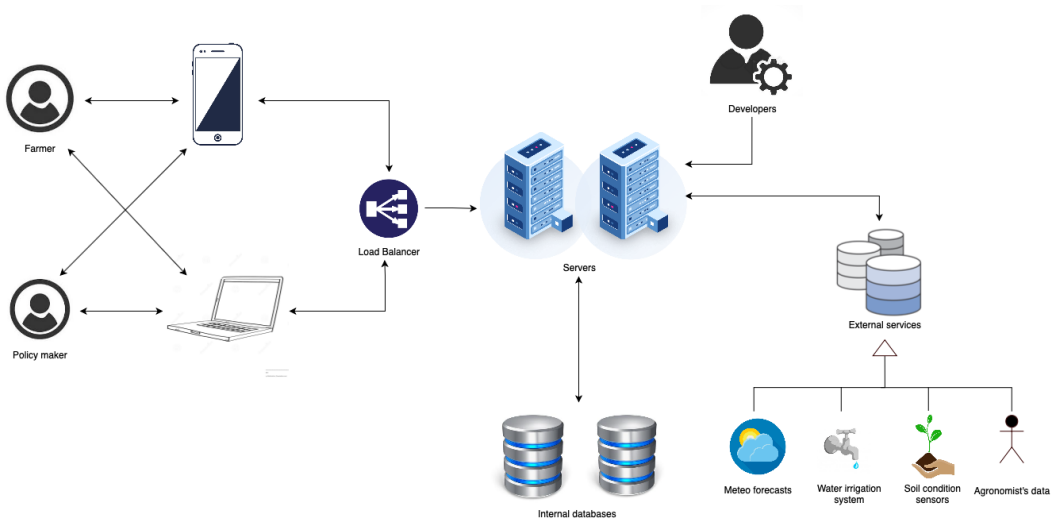


Figure 1: High level architecture.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

In the figure, there are represented two servers and two internal databases, indicating that, respectively, the logic of the application and the data are replicated on several devices in order to face any malfunction that could have led to a loss of information.

2.2 Component view

2.2.1 Class Diagram

In this section, it is reported the class diagram, which has been already analyzed in the RASD document. In this version of the diagram, there have been added the data types for each entity and some attributes that were considered to be important for the implementation of the application, such as the recognition code for the policy maker, email and password for the farmer.

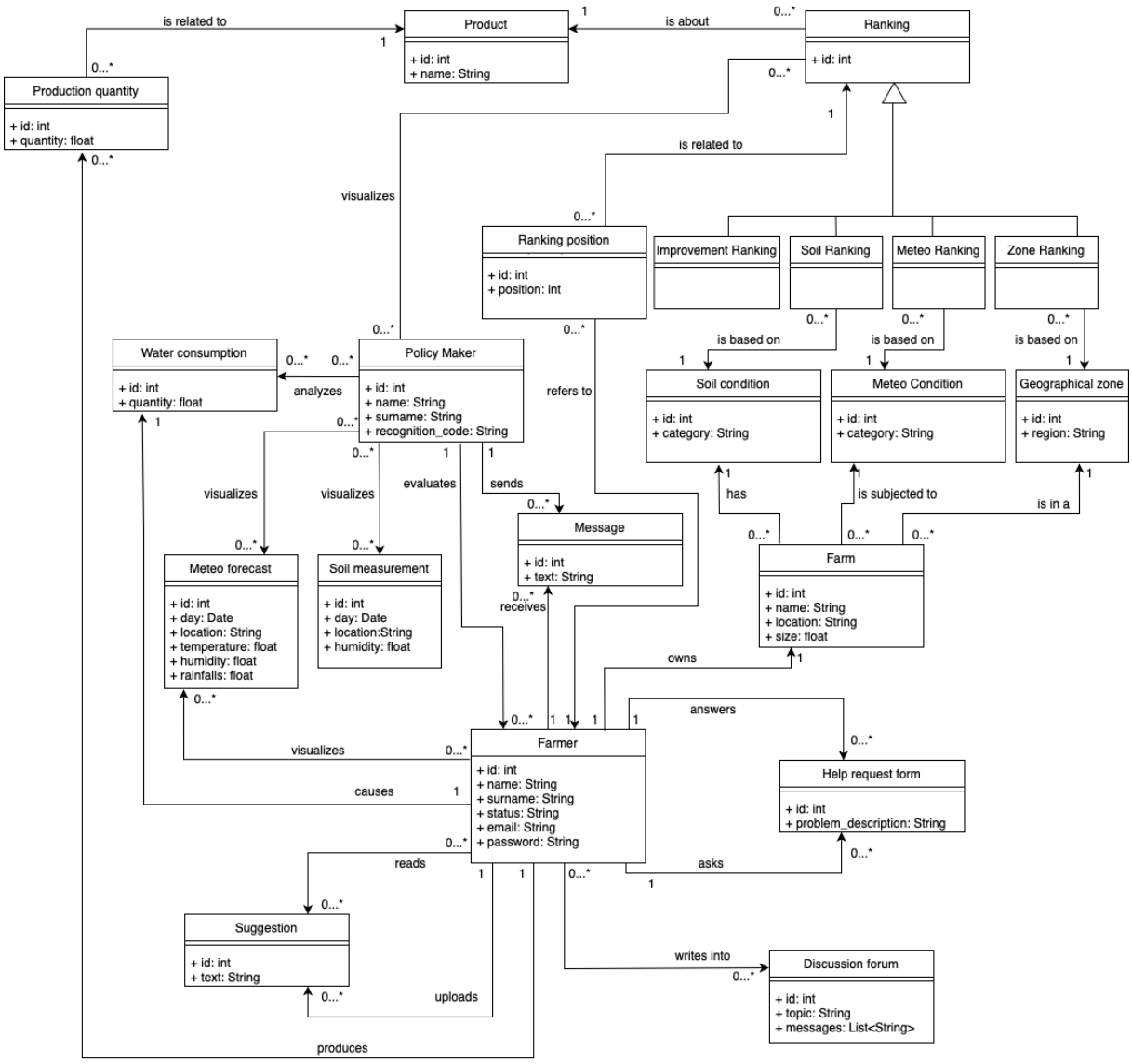


Figure 2: Class Diagram.

The specification of what it is meant with Improvement, Soil, Meteo and Zone rankings, which are all generalized by the concept of Ranking, can be found in the definitions section. As a clarification for the meaning of the entities "Soil condition" and "Meteo condition", their instances correspond to categories that represent, respectively, how much the soil conditions and the meteo conditions have been favourable.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

While, the two entities "Meteo forecast" and "Soil measurement", which have been added to this version of the class diagram, have a different meaning. In fact, they represent daily values for the meteo forecasts and for the soil measurements and they are needed in order to let the users of the application visualize them for a specific day. On the other hand, the categories represented by the entities "Meteo condition" and "Soil condition" are needed in order to generate the rankings and can be computed on the basis of the instances of the other two entities by the means of an algorithm, which will described in detail at a later stage.

Furthermore, the entity "Production quantity" has been added to the diagram, because it is a relevant information not only with respect to the positions of the farmers in the rankings but also as an attribute of the relation between the entities "Farmer" and "Product". These updates have been made also in the RASD document by creating a second version of it.

Finally, the methods that will implement the several functionalities of the application are not represented; however, they will be analyzed in a diagram that will combine the class diagram and the component diagram, which represents all the possible services.

2.2.2 Component Diagram

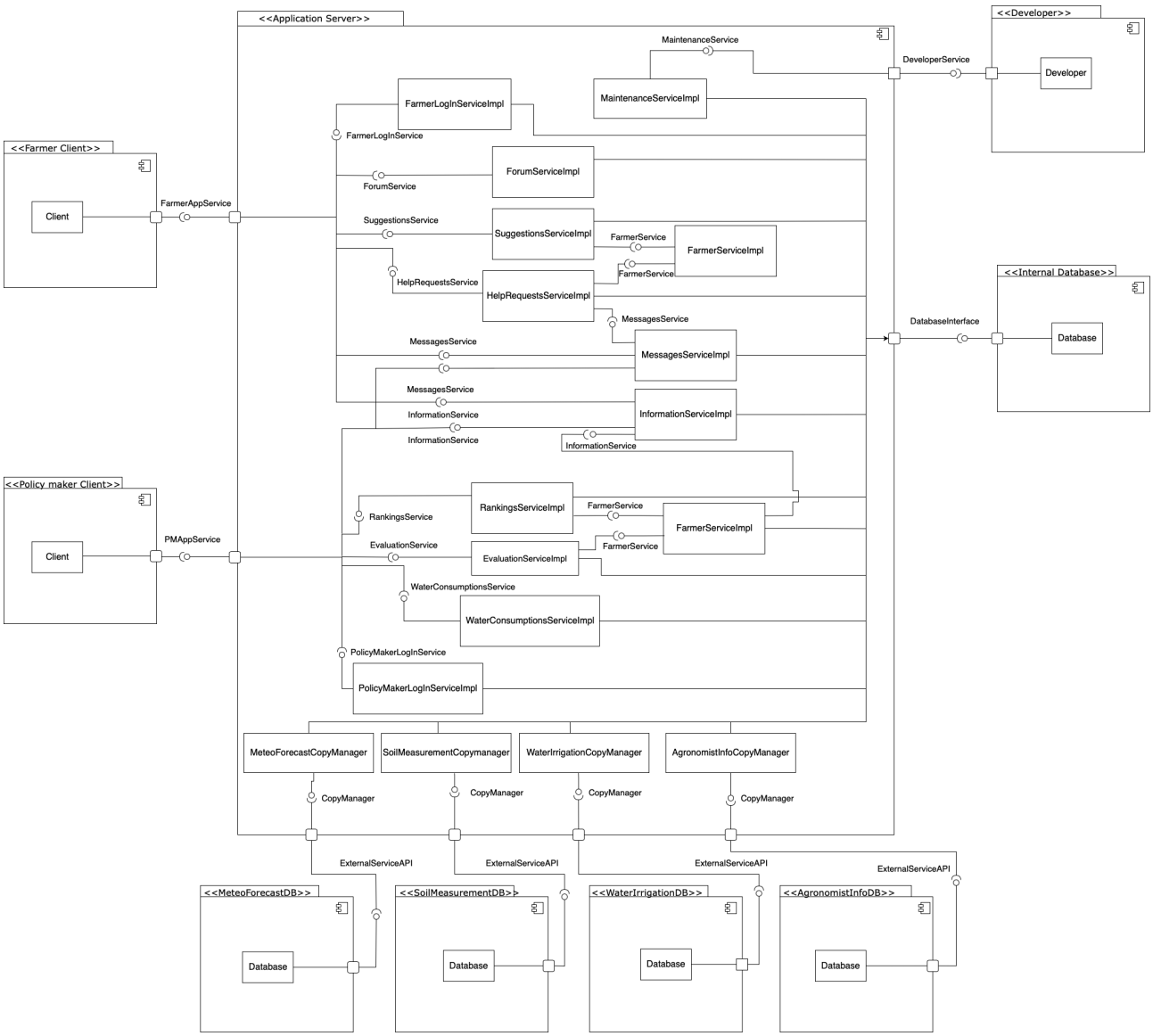


Figure 3: Component Diagram.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

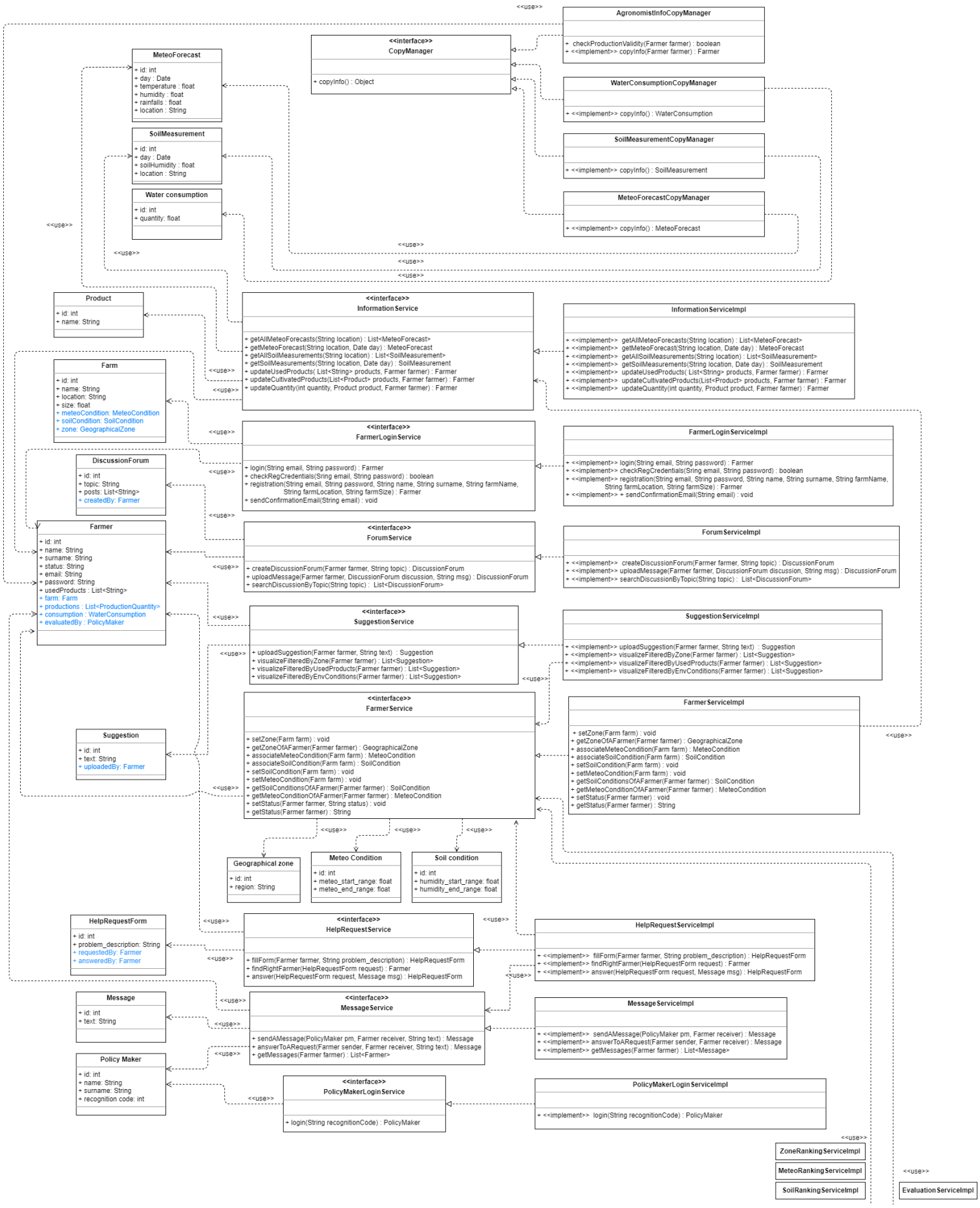
The previous diagram represent the main components of the architecture already described above. The server includes several services, that are needed in order to provide all the functionalities to the users. These services implement more general interfaces, indicating that there could be different ways to implement them. For what concerns the FarmerServiceImpl, it is duplicated in order to provide a more visible diagram; however, for simplicity, only one of the two instances is connected to the InformationServiceImpl. All the logic of the application is contained in the server component, as already mentioned, while the two different types of clients (the one of the policy maker and the one of the farmer) and the database reside on separated software components and interact with the server through specific dedicated interfaces. Finally, the communication of the system with external services, which are represented as external databases that contain data regarding meteo forecasts, soil measurements, water consumption and information collected by agronomists, is managed by the server through the implementations of a specific interface that will be responsible for the access to the external databases and for the copy of their content to the internal database of the application.

To understand better how the two diagrams are related it is provided another more specific UML diagram (which has been divided in two images to make it more visible) that includes the signatures of the main methods of the services specified in the component diagram and which classes of the model are used in each service.

The additional attributes (w.r.t. the ones of the entities in the class diagram) are written in blue and represent the associations between different entities that are necessary for the implementation of the methods in the services.

The interfaces that represent specific services are linked with the classes through some «use» arrows, which mean that such interface makes usage of the respective classes in order to implement its methods. For example, the interface "FarmerLoginService" uses the class Farmer in order to provide a method that checks the credential inserted in the login page. While, if a «use» arrow starts from an implementation of a service, it means that only that specific implementation of the interface uses that specific class. For instance, there are four different implementations of the "RankingService" that manage the four different types of rankings. Each of them uses different classes, such as the "geographical zone" and the "zoneRanking" classes for the service which manage zone rankings. Futhermore, if a «use» arrow links an implementation of a service to the interface of another one, it means that such implementation makes usage of some methods of the other service to provide its functionalities. Finally, the implementations of the services are linked through an arrow to the respective service interface that they implement.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved



Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

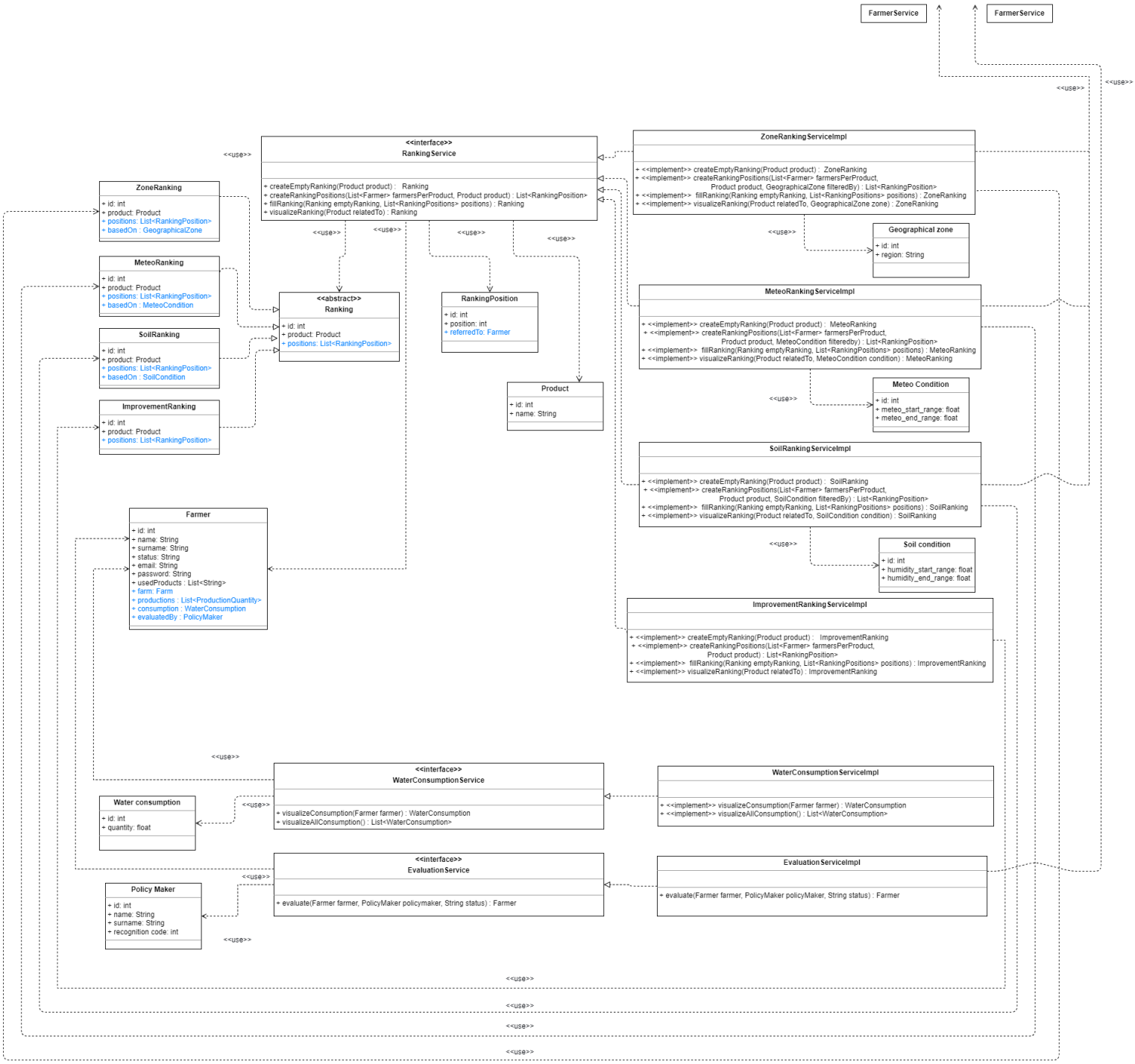


Figure 4: UML Diagram

For each interface, it will now studied in detail the meaning of each implementation with the respective methods.

- **CopyManager interface**: This interface has the scope of copying the data provided by external services into the internal database. To this purpose, the only main method of the interface is "copyInfo" that returns a general Object. Then, for each type of information it has been specified a different implementation of such interface. The "AgronomistInfoCopyManager" is responsible for checking if the information about production inserted by the farmers are consistent with the data collected by agronomists, to this end, it will use the method "checkProductionValidity". Then,

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

where appropriate the service can update the information about production referred to a Farmer through the implementation of the method "copyInfo". For what concerns the other three implementations of the interface, they have only the method "copyInfo", which is used to create an instance of one of the "MeteoForecast", "SoilMeasurement" and "WaterConsumption" classes and then to flush it into the database.

- **InformationService interface**: This interface has the scope of retrieving information of various types in order to let the users visualize them and update only the ones inserted or modified by themselves. In particular: the methods "getAllMeteoForecasts" and "getAllSoilMeasurements" return all the instances of Meteo forecasts and Soil measurements for a specific location; "getMeteoForecast" and "getSoilMeasurements" return the meteo forecast and the soil measurement for a specific location in a specific day; finally the three methods "updateUsedProducts", "updateCultivatedProducts" and "updateQuantity" allow to update respectively the products used by a farmer during his work, the products that he cultivates in his farm and the amount of production for a given product. This interface is implemented by only one service, the "InformationServiceImpl" which implements all its methods.

- **FarmerLoginService**: This interface has the scope of allowing the farmer to log into the system or to register if he is not logged yet. To this purpose, it contains several methods: the "login" method checks the validity of the credentials inserted by a farmer and returns the instance of the farmer if the credentials are valid, in alternative it shows an error message; "checkRegCredentials" is used during the registration to check the validity of the parameters inserted by the farmer; "sendConfirmationEmail" is used at the end of the process of registration in order to send the confirmation e-mail to the farmer; "registration" is used to store the new user in the database at the end of the process of registration. This interface is implemented by only one service, the "FarmerLoginServiceImpl" which implements all its methods.

- **ForumService**: This interface has the scope of managing all the operations regarding the forum. It contains several methods: "createDiscussionForum" allows a farmer to create a new discussion forum about a specified topic; "uploadMessage" allows a farmer to write a post on a specific discussion forum; "searchDiscussionByTopic" allows a farmer to visualize all the discussion forums about a specified topic. This interface is implemented by only one service, the "ForumServiceImpl" which implements all its methods.

- **SuggestionService interface:** This interface has the scope of managing the uploading or the visualization of the suggestions. The method "uploadSuggestion" takes as parameters the farmer who wants to upload a suggestion and the text of the suggestion; then, if the farmer has a good status (checked through the method of the FarmerService), proceeds with the creation of a suggestion. While, the other three methods: "visualizeFilteredByZone", "visualizeFilteredByUsedProducts", "visualizeFilteredByEnvConditions", deal with the visualization of the suggestions filtering them, respectively, by the zone in which a farm related to a farmer is placed, by the products that the farmer uses to cultivate or by the environmental conditions in which he works. The information regarding environmental conditions and geographical zone related to a farm are retrieved through the methods contained in the "FarmerService"; for this reason the "SuggestionService" uses the "FarmerService". This interface is implemented by only one service, the "SuggestionServiceImpl" which implements all its methods.

- **FarmerService**: This interface has the scope of managing all the operations regarding setting and getting of the attributes' values of farmers and their farms. It contains several methods. "setMeteoCondition" sets the value of the "Meteo Condition" of a specific farm (related to a specific farmer) to the right "Meteo Condition" category using the method "associateMeteoCondition", which maps meteo forecasts related to the location of the farm to a specific instance of "Meteo

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

Condition". "setSoilCondition" sets the value of the "Soil Condition" of a specific farm(related to a specific farmer) to the right "Soil Condition" category using the method "associateSoilCondition" which maps soil measurements related to the location of the farmer to a specific instance of "Soil Condition". The information regarding meteo forecasts and soil measurements are obtained through the InformationService. "setZone" sets the value of the Geographical Zone of the farmer through the location of his farm. The first two methods are called periodically by the system in order to provide a coherent association between the farm and his meteorological and soil conditions; whilst, the third method is called only one time by the system to associate the right geographical zone to the farm. "setStatus" sets the value of the status of a farmer. The other four methods are the four getters of the same four attributes that are used by other services in order to provide their functionalities.

- **HelpRequestService**: This interface has the scope of managing the help requests sent by farmers in a critical situation. To this purpose it contains several methods: "fillForm" allows a farmer to describe his problem and send the help request. "findRightFarmer" has the scope of finding a farmer who is in the right conditions to answer to a specific help request; to understand if a farmer is in the right conditions this service use some methods of the "FarmerService" in order to verify if the status of the farmer is good and if the conditions in which he works are similar to the ones of the farmer who uploads the request. "answer" allows a farmer to answer to an help request with a message; to this purpose this service uses the "MessageService" to implement this functionality. This interface is implemented by only one service, the "HelpRequestServiceImpl" which implements all its methods.

- **MessageService**: This interface has the scope of managing the messages between different users. It uses several methods: "sendAMessage" allows a policy maker to send a specified message to a farmer in order to communicate him some information; "answerToARequest" allows a farmer to answer to a request of help sent by another farmer and dispatched to him by the system; "getMessages" allows a farmer to visualize all the messages he received. This interface is implemented by only one service, the "MessageServiceImpl" which implements all its methods.

- **PolicyMakerLoginService**: This interface has the scope of allowing the policy maker to log into the system. To this purpose it contains only one method "login" which allows him to log into the system through a recognition code. It returns the instance of the policy maker if the code is valid, in alternative it shows an error message. This interface is implemented by only one service, the "PolicyMakerLoginServiceImpl" which implements its method.

- **RankingService interface:** This interface is used to manage all the operations that regard the generation or visualization of rankings, it is implemented in four different ways in order to specify the behaviour of its methods with respect to the four different types of rankings. The method "createEmptyRanking" aims to create a ranking and associate it to a specific product, then, through the method "createRankingPositions" it will be created a list of ranking positions ordered accordingly to the quantity produced by the respective farmer; such list will then be associated to the right ranking through the method "fillRanking". In order to associate the right farmers to a ranking, accordingly to specific parameters such as "Meteo Condition", "Soil Condition" and "Geographical Zone", it is necessary to retrieve these information related to a farmer; to this end this service uses the "FarmerService". All these methods are not called by the users but they are periodically called by the system in order to provide up-to-date rankings. Finally, the method "visualizeRanking" it is used to filter the rankings that a policy maker wants to visualize while using the application.

- **WaterConsumptionService interface:** This interface is needed in order to let the policy makers visualize all the farmers' water consumption (through the method "visualizeAllConsumption") or only the consumption related to a specific farmer (through the method "visualizeConsumption").

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

This interface is implemented by only one service, the "WaterConsumptionServiceImpl" which implements all its methods.

- **EvaluationService interface:** This interface has the scope of allowing a policy maker to evaluate a farmer by assigning him a status. To this end, the method "evaluate", which takes as a parameter a farmer, is defined. In order to assign the defined status to the farmer this service uses the FarmerService for the method "setStatus". This interface is implemented by only one service, the "EvaluationServiceImpl" which implements all its methods.

Finally, it can be noticed that the diagram has been divided in two images in order to make it more visible, so the «use» arrows that go from the RankingService's and EvaluationService's implementations to the FarmerService are divided between the two figures.

2.3 Deployment view

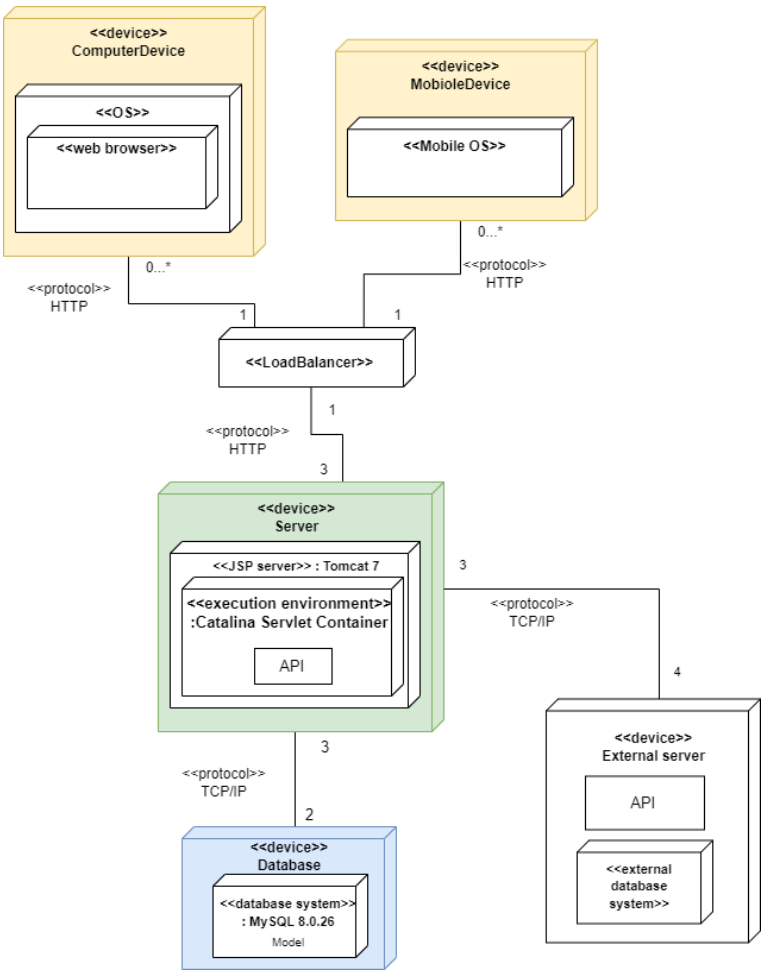


Figure 5: Deployment Diagram.

The above deployment diagram shows the architecture of the system in its distributed nature. The nodes represent hardware or software execution environments and the middleware connecting them. The colors used in the diagram reflect the fact that the system is developed as a three tier architecture (as described in the section 2.1). In fact, the nodes represented in yellow are part of the client tier, the one in green composes the application logic tier and the one in blue the data tier. Furthermore, it is shown a node representing the external services, which reside on external servers and are equipped with APIs

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

that allow the communication with the system. In addition, it is reported the load balancer, which, as already mentioned in the section 2.1, is important to manage the concurrent accesses of several clients to the DREAM application. Finally, it can be observed that the cardinalities of the associations between the nodes are chosen accordingly to the fact that the functionalities of the application logic and data tiers have to be replicated on multiple devices in order to face possible failures and prevent information losses; in particular, it was deemed sufficient to consider three servers and two databases that compose the system, but in future implementations of the DREAM system it could be considered necessary to integrate some additional devices.

2.4 Runtime view

In this section all the sequence diagrams that have been made in the RASD document will be expanded specifying which services of the application server operate to provide such functionalities. The entire communication between the DREAM application and the services is deployed through the services' methods specified in the UML diagram in 2.3. Whilst, for what concerns methods about the communication between user and DREAM application, they are generic and will be provided through specific elements in the user interfaces. In particular, from the side of the client: in order to upload some data, it is used a method upload(some parameters), in order to request a specific view, it is used a method get(specific view), in order to choose some parameters for filtering, it is used a method choose(some parameters) and so on. While, from the side of the application: in order to show a view it is used a method show(specific view), in order to show a success or error message it is used a method show(Message) and so on. In some situations the order of actions of these sequence diagrams differs from the original one in the RASD document. This choice has been made in order to be coherent with the specific methods (explicated only in the previous chapter) that have to be used.

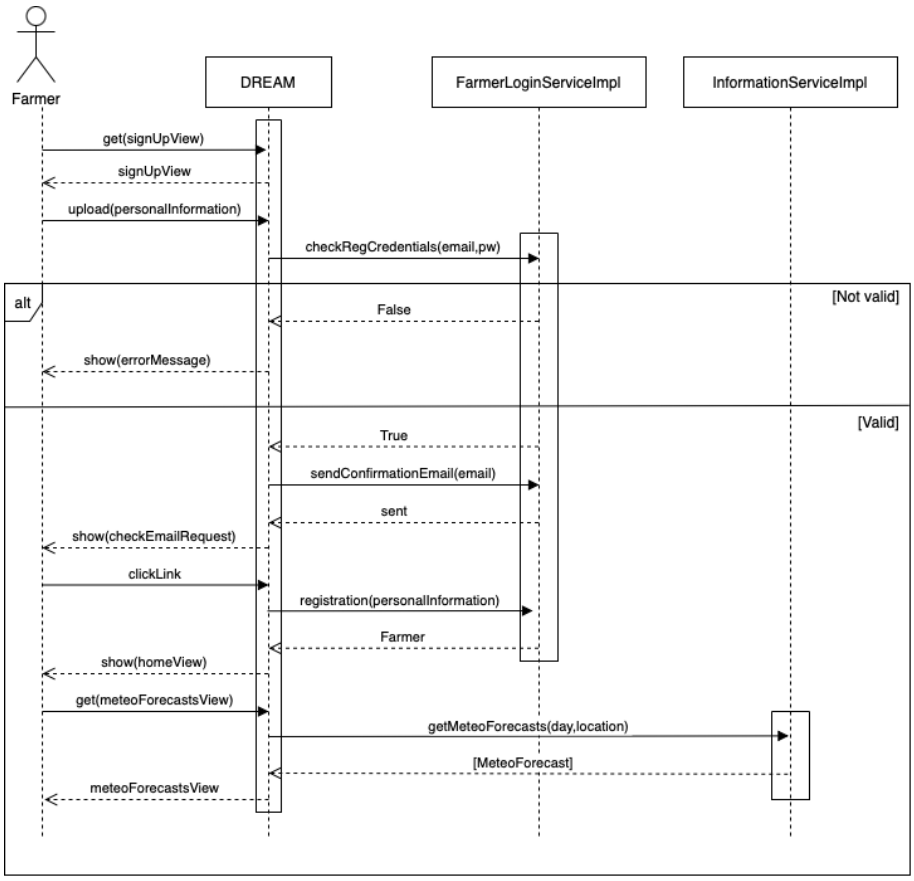


Figure 6: 1st sequence diagram

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

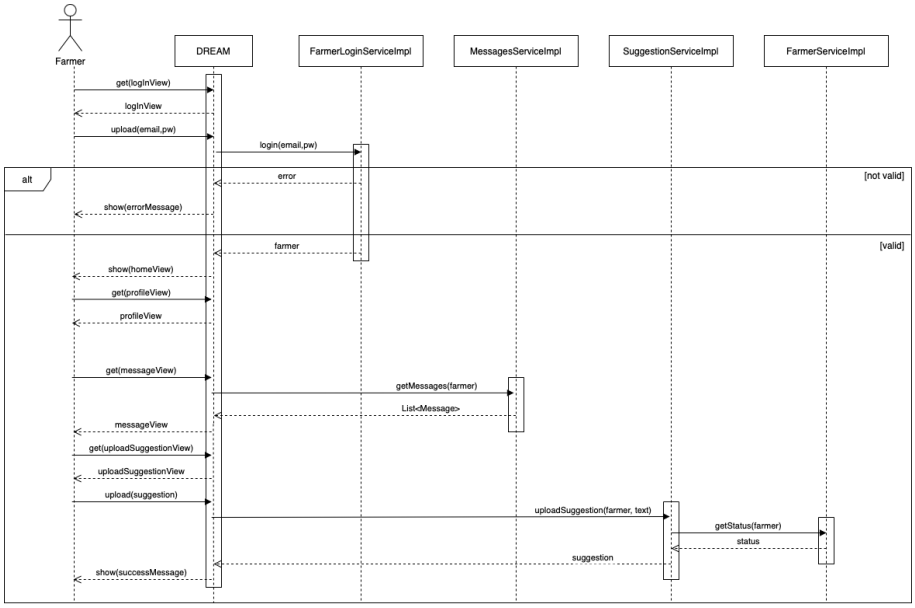


Figure 7: 2nd sequence diagram

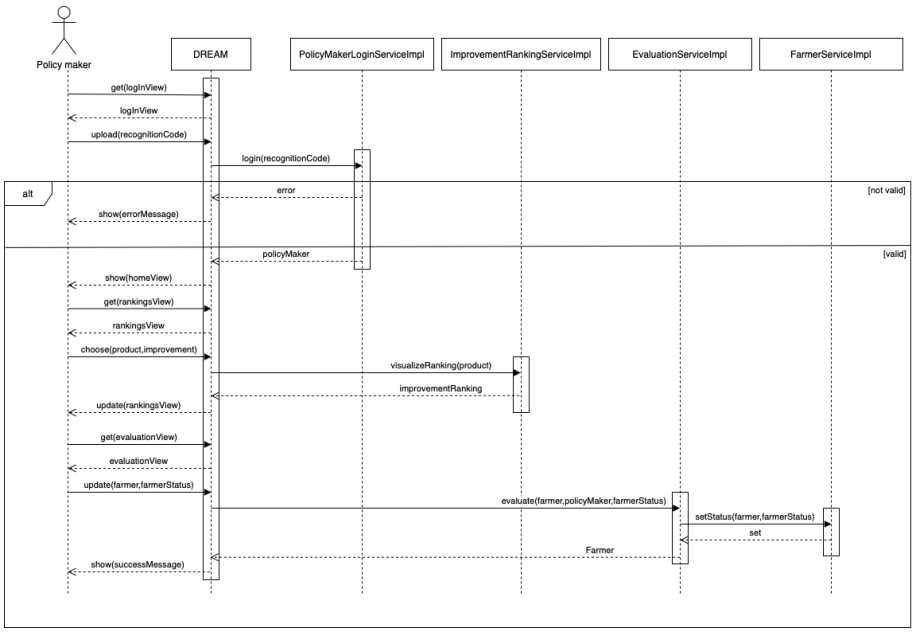


Figure 8: 3rd sequence diagram

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

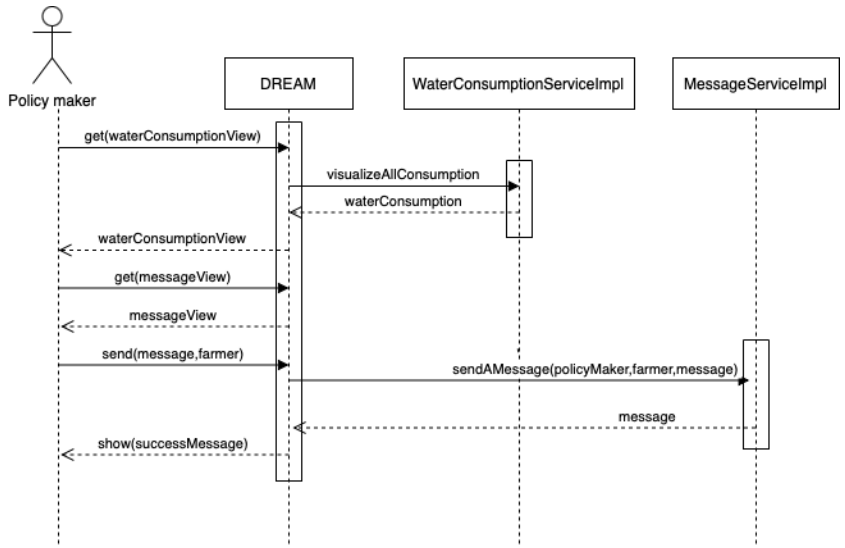


Figure 9: 4th sequence diagram

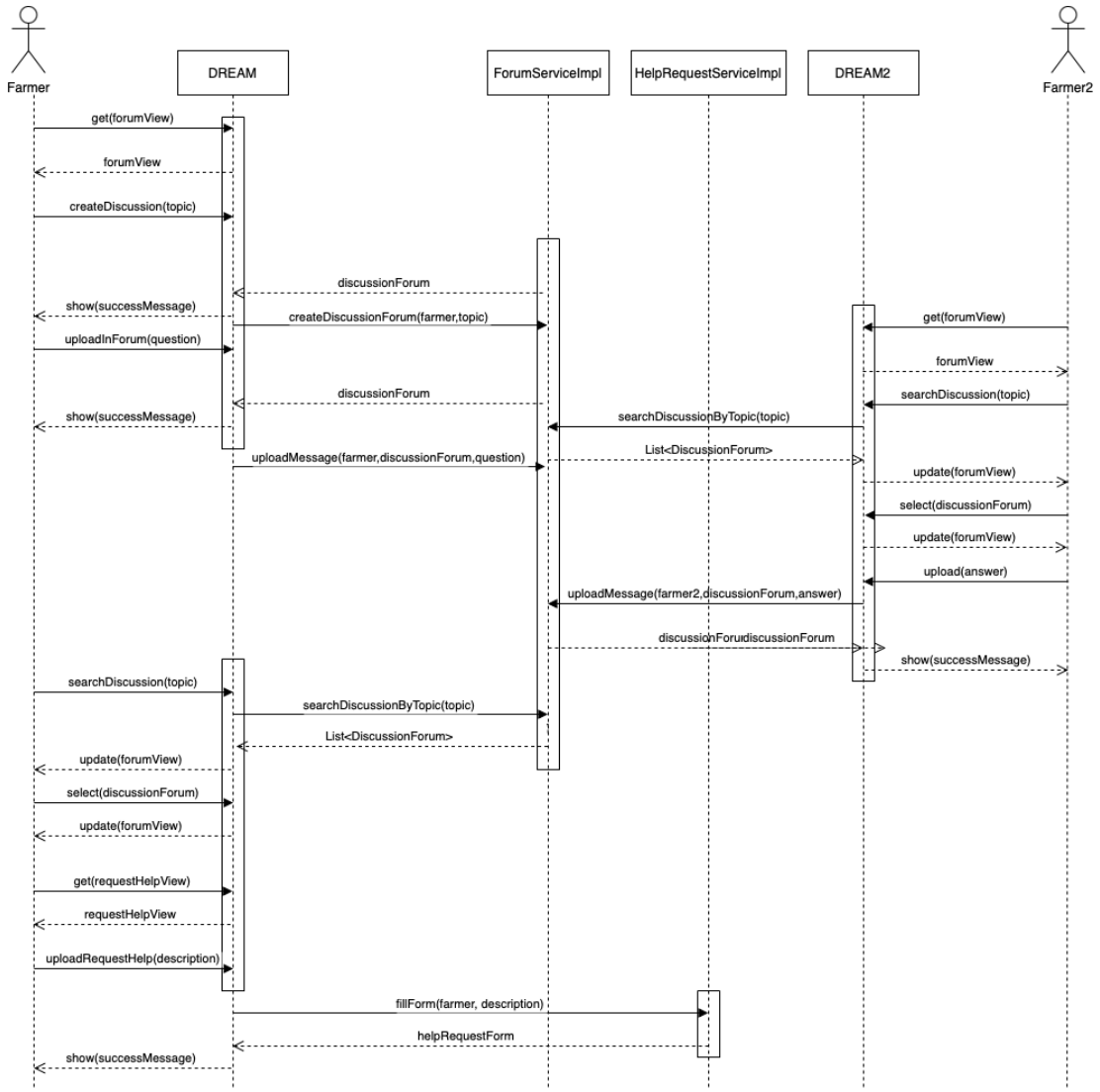


Figure 10: 5th sequence diagram

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

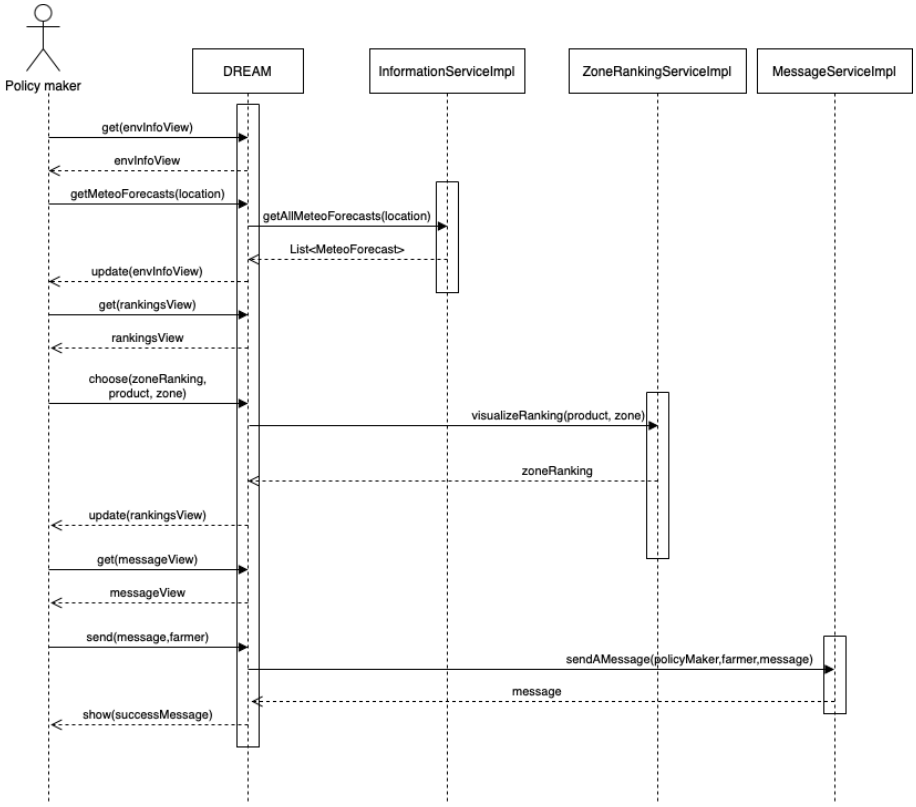


Figure 11: 6th sequence diagram

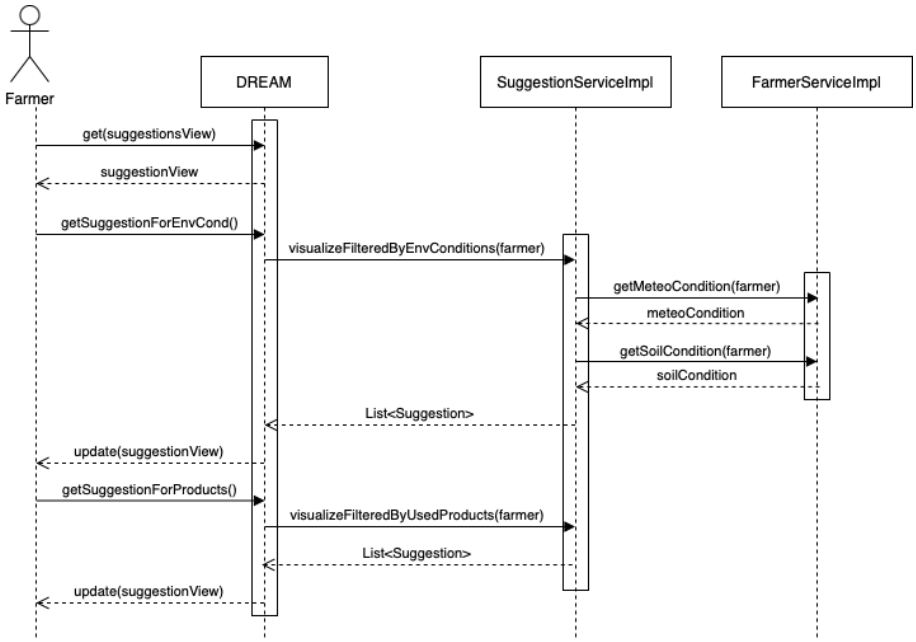


Figure 12: 7th sequence diagram

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

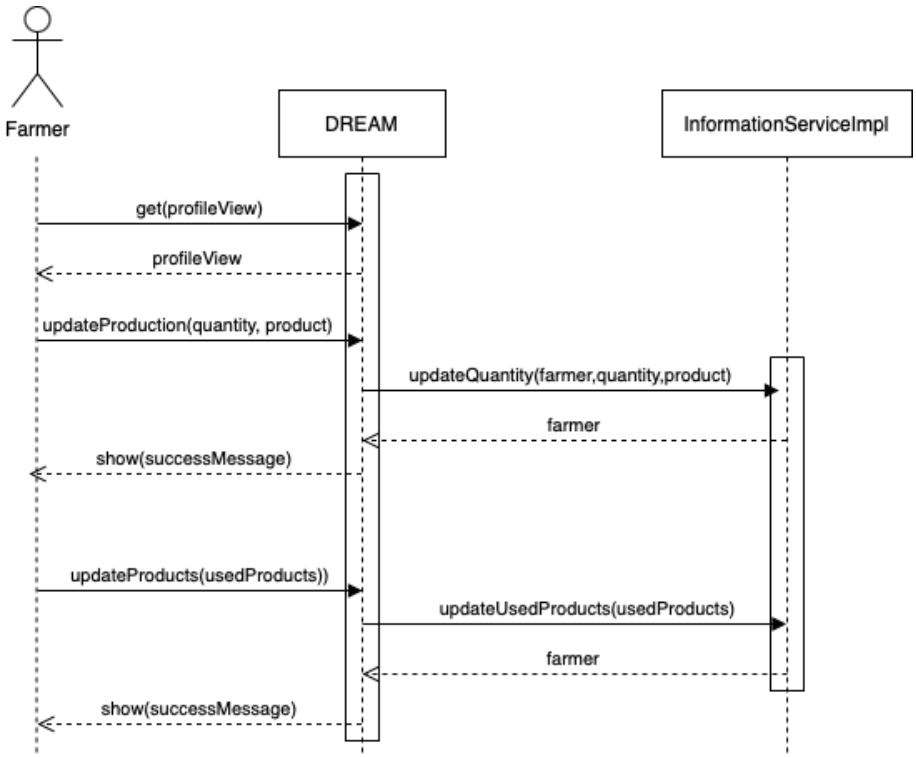


Figure 13: 8th sequence diagram

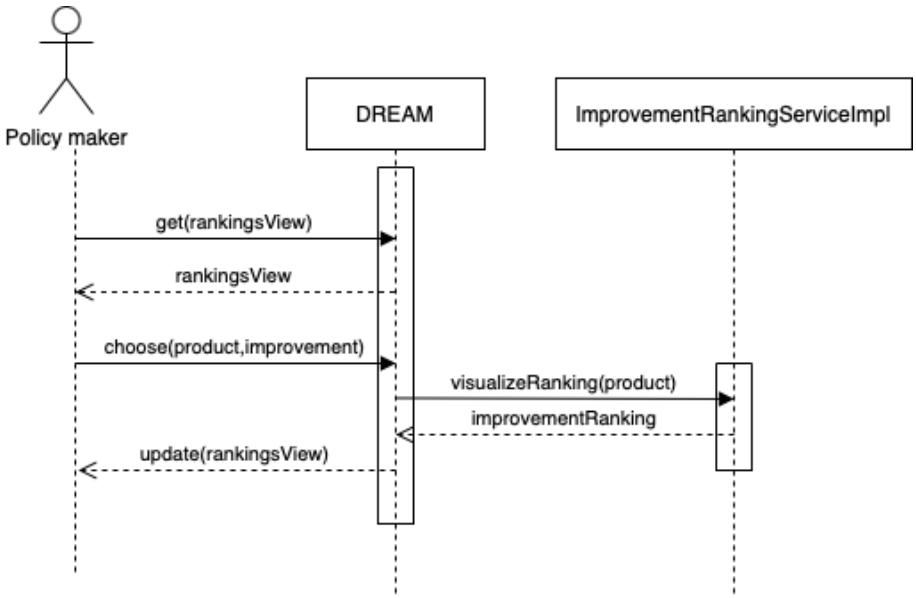


Figure 14: 9th sequence diagram

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

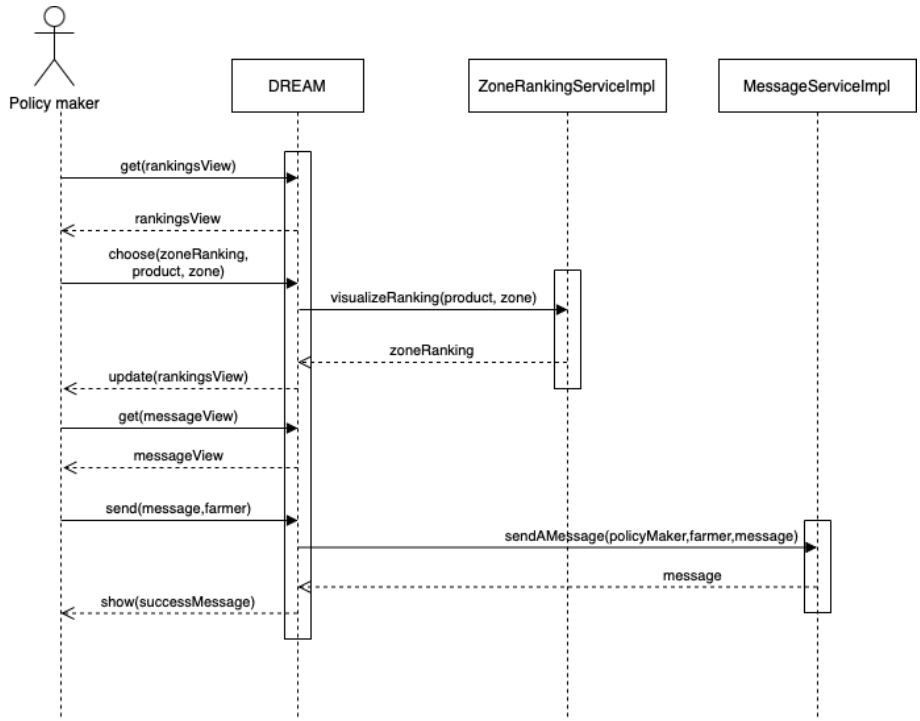


Figure 15: 10th sequence diagram

2.5 Component interfaces

The following diagram represents the dependencies among the interfaces of the system. These dependencies are justified by the usage relations defined in the previous UML diagram and are reported here to make them more clear.

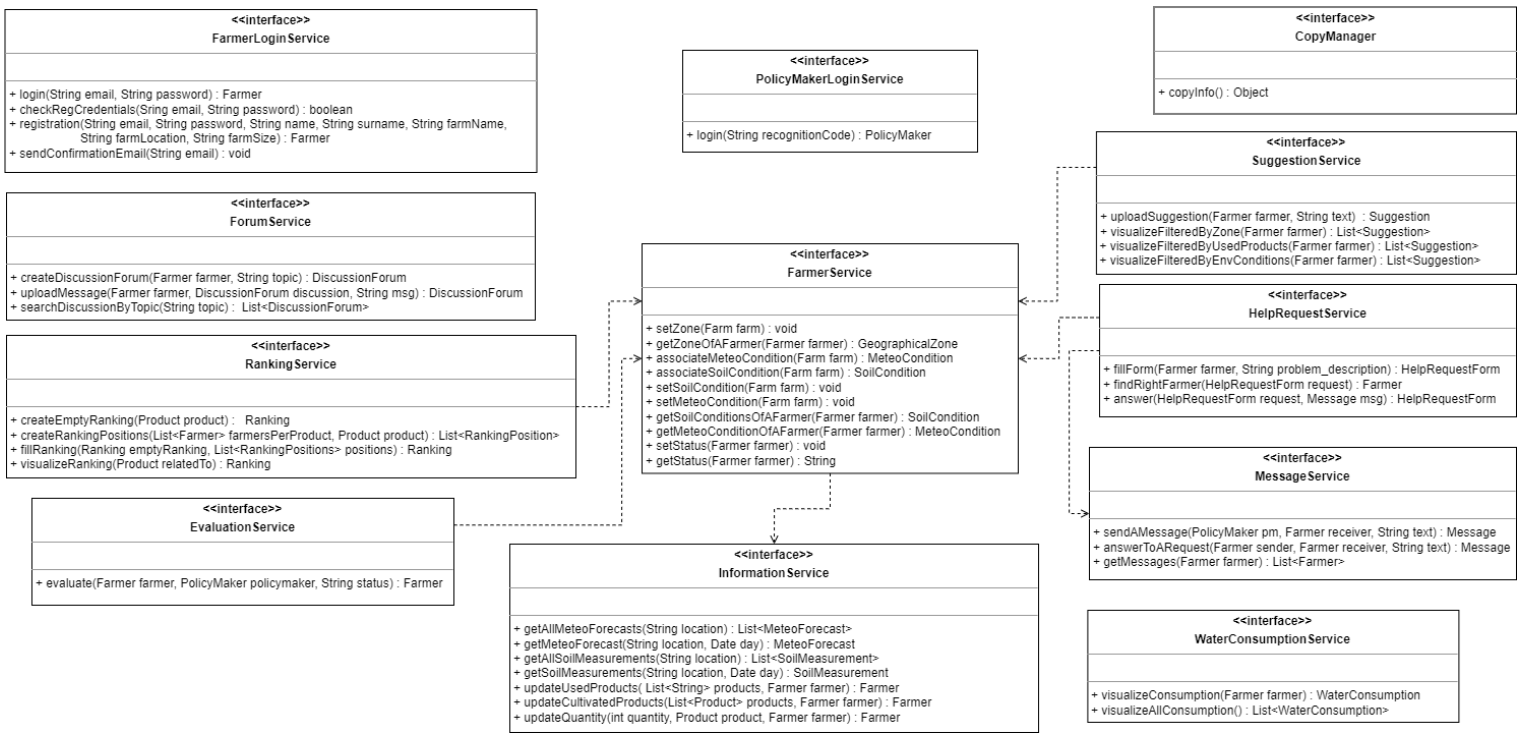


Figure 16: Interfaces' dependencies diagram

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

2.6 Selected architectural styles and patterns

- **Three tier architecture:**
  As already mentioned in the section 2.1, the type of architecture chosen to implement the DREAM system is a three tier architecture. This architecture requires the division of the system into three logical and physical tiers: the presentation tier, the application tier and the data tier. The first has the scope of delivering information to the users of the application and, at the same time, retrieving from them information that have to be processed to provide the functionalities of the system. The second correspond to the part of the system in which reside the main logic of the application; it has the scope of processing all the data retrieved both from the users and the data tier. The application tier is also responsible of managing (i.e. inserting, removing and updating) the data present in the data tier, communicating with it by means of API calls. Finally, the data tier is where all the necessary data are stored. In a three tier architecture, it is not possible for the presentation tier to interact directly with the data tier, in fact, all communication have to go through the application tier. The main benefit of such type of architecture, which is also the reason why it has been chosen for the DREAM system, is that the three tiers can be developed separately and run on different infrastructures. In such way, it will be also easier to update and extend the system whenever it is needed, modifying only the appropriate tier without any undesirable side effect on the other two tiers.

- **Component based development:**
  As it is highlighted in the previous sections, the development of the system is structured in components and sub-components. Therefore, it has been put into practice a component based development, which brings a great number of benefits both in short and long terms. In a system with a great number of functionalities, a component based approach can be useful in order to develop a reusable software system, defining, implementing and composing loosely coupled independent components.

- **Relational DBMS:**
  A DBMS is a software system that allows the storage and management of data in a database. In the development of the DREAM system it has been chosen a relational DBMS, which is a type of management system that stores data in a row-based table structure, in which the data of different tables are related. Furthermore, the DBMS implements some functions that deal with the security, accuracy, integrity and consistency of the data.

- **Model View Controller (MVC):**
  It is a software design pattern that requires the presence of three main logical components: the model, the view and the controller. The model has the scope of managing the data of the database and it corresponds, in the DREAM system, to the classes defined in the class diagram (section 2.2.1). The View deals with the presentation of the information to the users and resides in the presentation tier. The Controller allows the communication between view and model and contains the services (defined in section 2.2.2) that provide all the needed functionalities. The main reason why it has been chosen this pattern is that it leads the software to be easily modifiable and its components widely reusable.

2.7 Other design decisions

2.7.1 Used algorithms

The only relevant algorithm to discuss is the one needed to map the values obtained by meteorological forecasts and soil measurements on categories which describe how much that specific characteristic

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

is good for cultivation. Of course all these factors depend also on the specific products that a farmer produces, on the specific type of soil of his farm and other more specific factors. However, in this algorithm, it will not be possible to consider all these elements because it would need a wide knowledge about the argument and a complex analysis of the data. The approach chosen for the implementation is to consider an average of the best possible values for each type of data and to use them as reference values in order to evaluate the quality of the meteorological conditions and the soil conditions in a specific location. The reference values are obtained through specific researches on several scientific documents.
First of all, it is fundamental to define the categories of "Meteo condition" and "Soil Condition". Both are divided into five categories, that represent the level of goodness of the conditions in that period of time. These categories are labeled as follows:

- very bad
- bad
- normal
- good
- very good

In order to better understand the bases of the analysis, the parameters on which it is based are briefly discussed below. For what concerns "Meteo condition", they are based on the values obtained by the meteorological forecasts, in particular the ones considered are temperature, humidity and rainfalls. For what concerns the "Soil condition", they are based on the values obtained by the soil measurements, in particular the humidity of the soil. The reference values for these parameters are obtained researching their best possible ranges for various type of products and finding average ranges which satisfy almost all of them. In particular, the obtained values are:

- temperature: 26-31 °C
- humidity: 35%-45%
- rainfalls: 85-90 mm
- humidity (of soil): 65%-75%

The algorithm that will be used to calculate the correspondent meteo condition for a location will rely on these values. It is important to underline that this algorithm is not scientific or provided by some third part but it will be implemented by the developers of the project. It may be replaced in the case of a successive discover of a better or more specific one.
Now that all the elements useful for the algorithm are well explicated, it is possible to proceed with the discussion of the logic of the algorithm. For what concerns the "Soil condition" it is needed the analysis of only one parameter, the humidity. If the average humidity of the last month is in the best possible range it will be associated to the "very good" category. If it is out of the best range for a value from 0% (obviously excluded) to 5%, it will be associated to the "good" category. If it is out of the best range for a value from 5% (excluded) to 10%, it will be associated to the "normal" category. If it is out of the best range for a value from 10% (excluded) to 15%, it will be associated to the "bad" category. If it is out of the best range for a value greater than 15, it will be associated to the "very bad" category.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

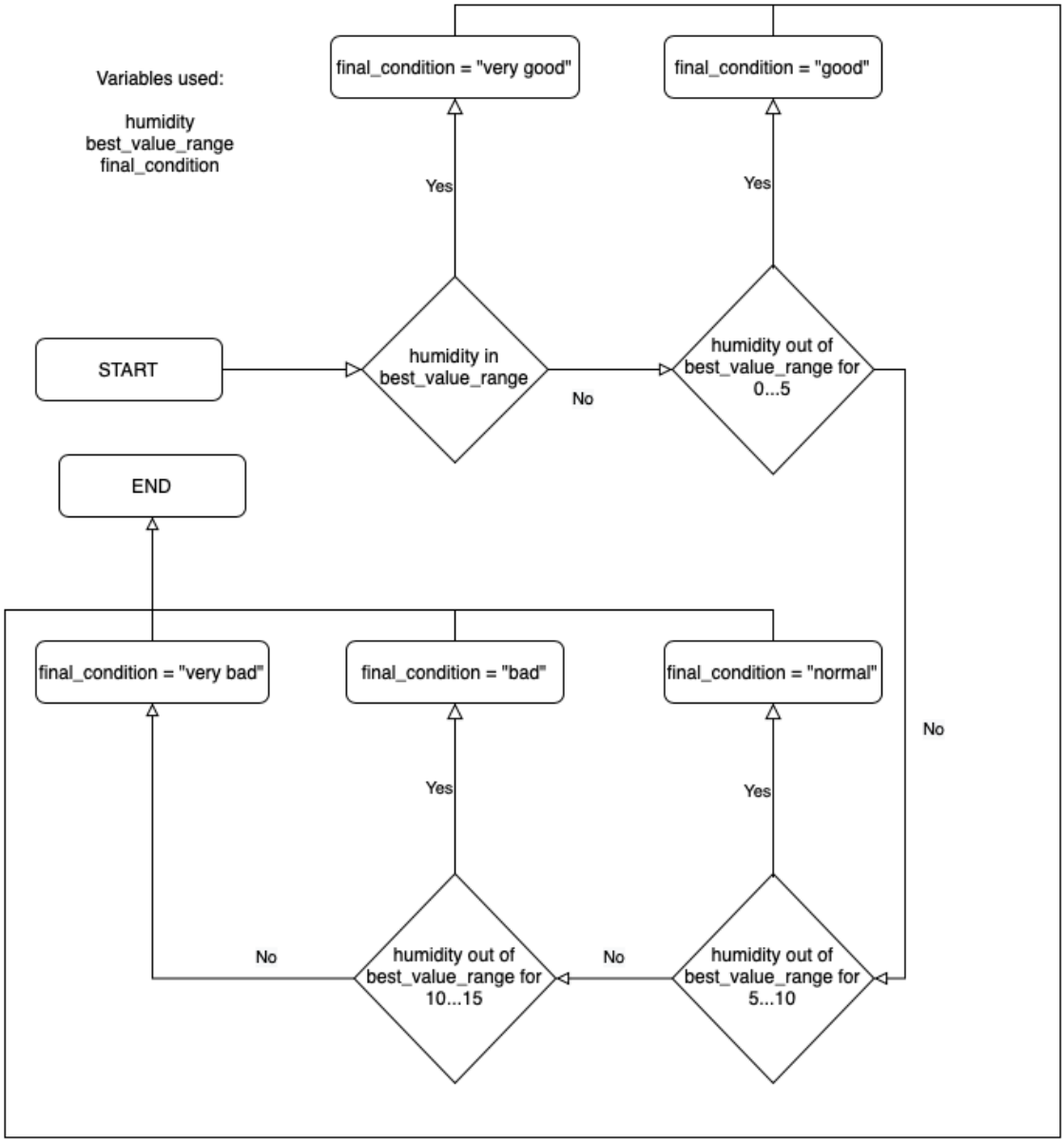


Figure 17: Soil condition Algorithm diagram

For what concerns the "Meteo condition" the analysis is more complex and takes into account three parameters: temperature, humidity and rainfalls. If all the parameters are in the best range it will be associated to the "very good" category. Otherwise, for every parameter, some penalty points will be assigned considering how much the value is out of range. In particular, for what concerns the temperature, if the average temperature of the last month is out of the best range for a value from 0 °C (obviously excluded) to 3 °C, one penalty point will be assigned; if the average temperature is out of the best range for a value from 3 °C (excluded) to 6 °C, two penalty points will be assigned; if the average temperature is out of the best range for a value from 6 °C (excluded) to 9 °C, three penalty points will be assigned; if the average temperature is out of the best range for a value greater than 9 °C, four penalty points will be assigned. Then, for what concerns the humidity, if the average humidity of the last month is out of the best range for a value from 0% (obviously excluded) to 5%, one penalty point will be assigned. If it is out of the best range for a value from 5% (excluded) to 10%, two penalty points will be assigned. If it is out of the best range for a value from 10% (excluded) to 15%, three penalty points will be assigned. If it is out of the best range for a value greater than 15%, four penalty points will be assigned. For what

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

concerns rainfalls, if the quantity of rainfalls of the last month is out of the best range for a value from 0mm (obviously excluded) to 4mm, one penalty point will be assigned; if it is out of the best range for a value from 4mm (excluded) to 8mm, two penalty points will be assigned; if it is out of the best range for a value from 8mm (excluded) to 12mm, three penalty points will be assigned; if it is out of the best range for a value greater than 12mm, four penalty points will be assigned. At the end of this process, if the number of penalty points is smaller or equal than two, it will be associated to the "good" category . If it is between three and five, it will be associated to the "normal" category. If it is between six and eight, it will be associated to the "bad" category. If it is between nine and twelve (the maximum value), it will be associated to the "very bad" category.

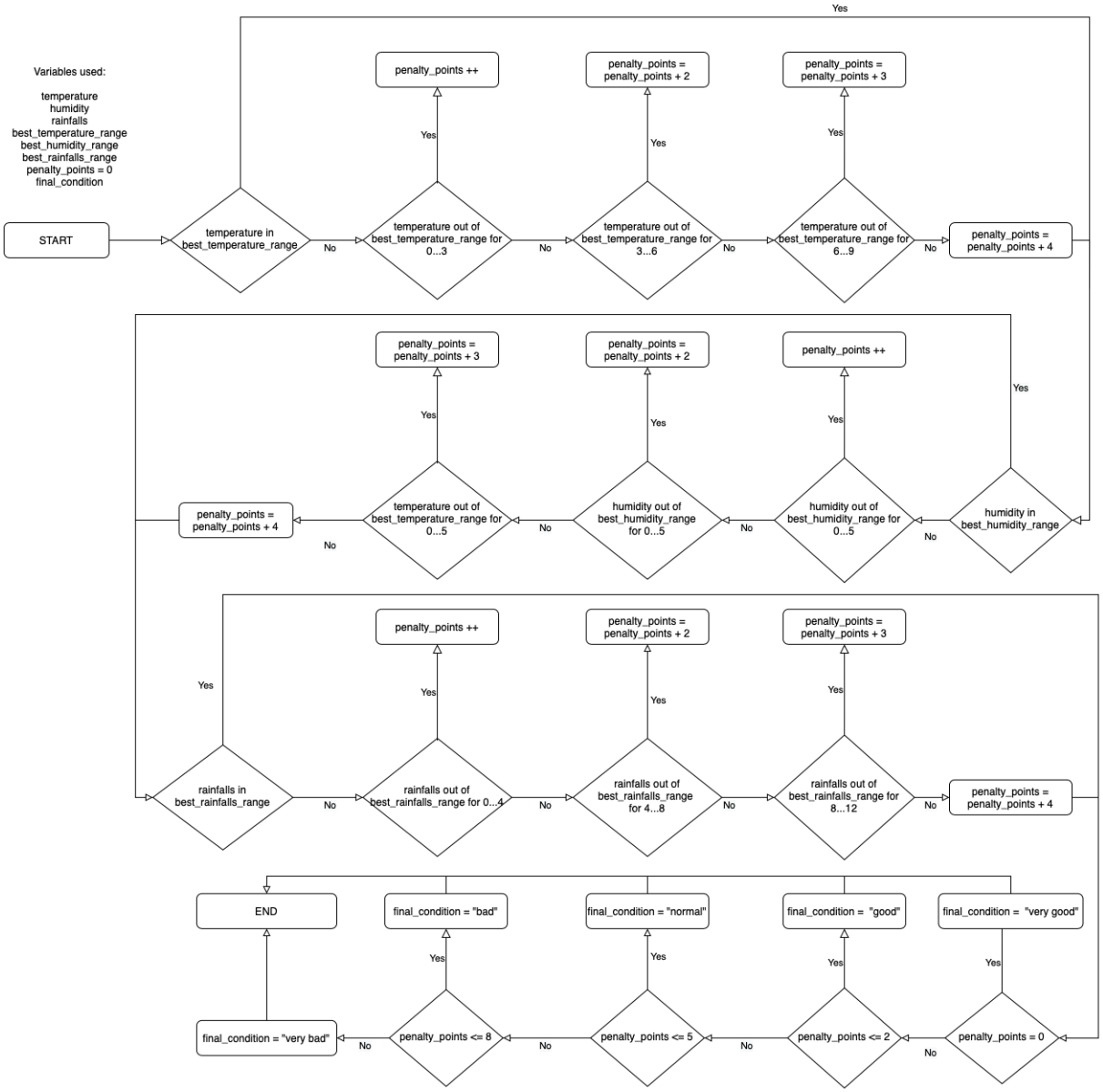


Figure 18: Meteo condition Algorithm diagram

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

3 User Interface Design

In order to provide a clear representation, the links between the various user interfaces will be explicated after each of them. The software is projected both for mobile devices and web devices, but in this document, only the user interfaces related to the mobile application are presented.

3.1 Farmer Interfaces

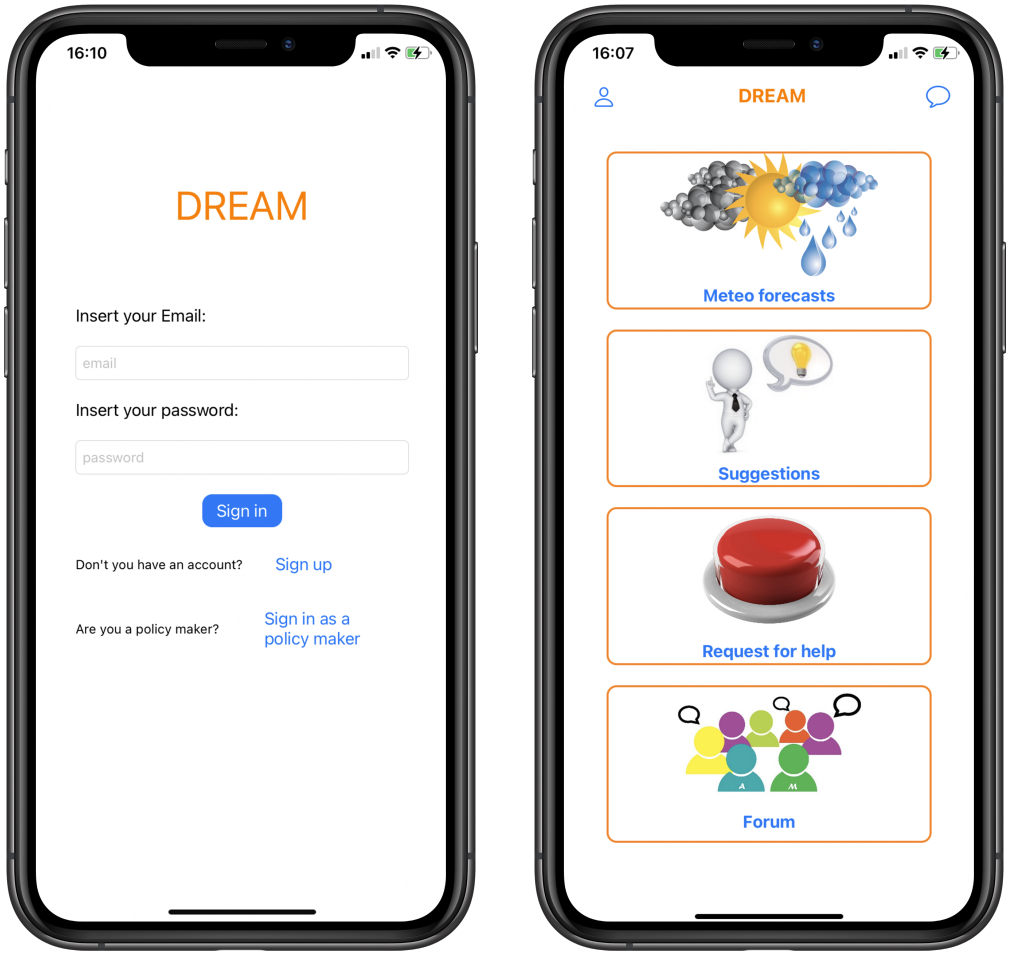


Figure 19: Log in interface (on the left) and Home interface (on the right) for a farmer of Dream.

After inserting his credentials and clicking on the "Sign in" button in the log in interface the farmer moves to the home interface where he visualizes other four buttons representing the main features of the application and two icons at the top for the profile and messages interfaces.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

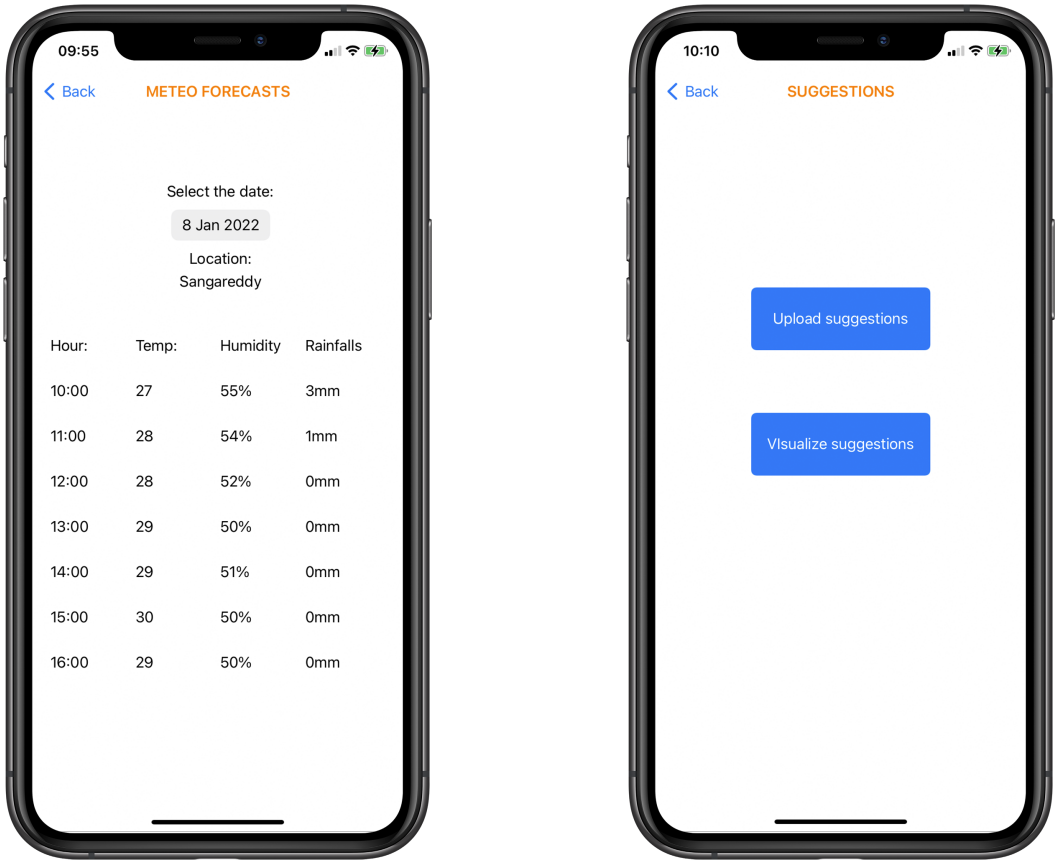


Figure 20: Meteo forecasts interface (on the left) and Suggestion interface (on the right) for a farmer of Dream.

From the home interface, clicking on "Meteo Forecasts" button, the farmer moves to the interface on the left: here he can visualize temperature, humidity and rainfalls values for each time slot of the day he selects related to his location. While, clicking on "Suggestions" button from the home interface, he moves to the interface on the right: here he can choose either to upload or visualize suggestions.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

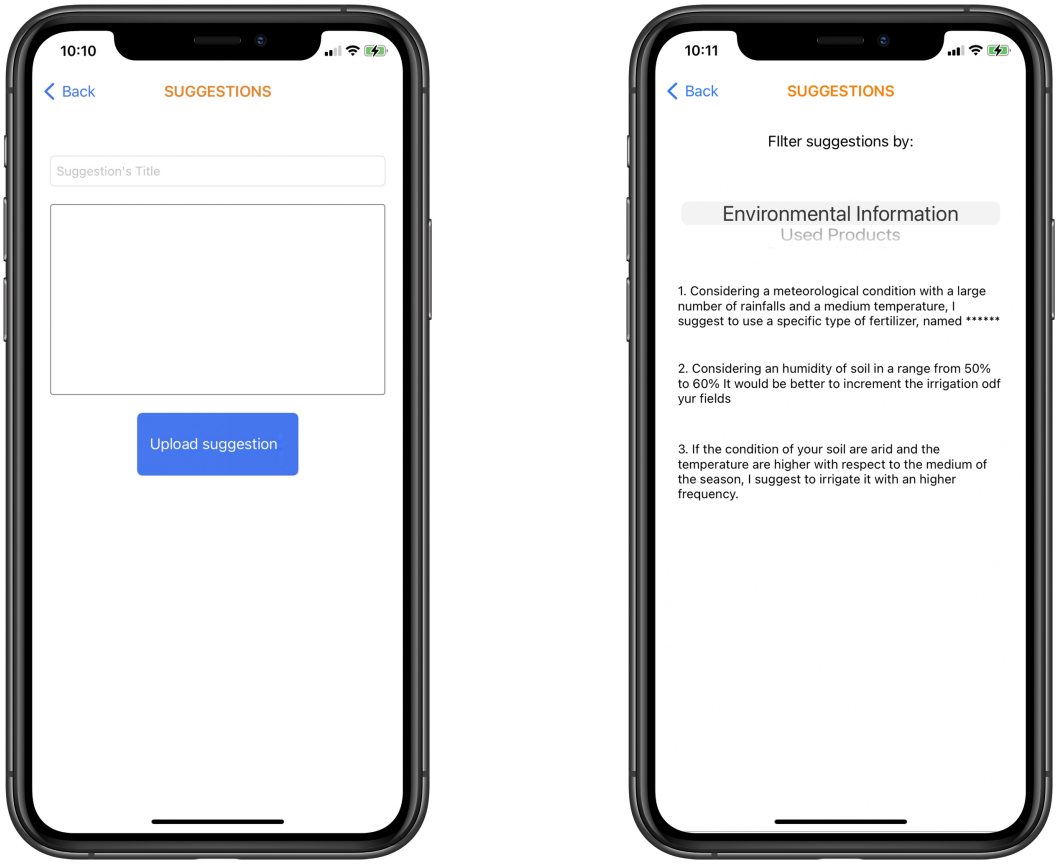


Figure 21: Upload suggestion interface (on the left) and Visualize suggestion interface (on the right) for a farmer of Dream.

From the suggestion interface, clicking on the "Upload suggestions" button, the farmer moves to the interface on the left: here he can write a title with the respective text and then upload the suggestion. While, clicking on the "Visualize suggestion" button, he moves to the interface on the right: here he can visualize the suggestions, previously uploaded by other users, filtering the visualization by one of the available factors.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

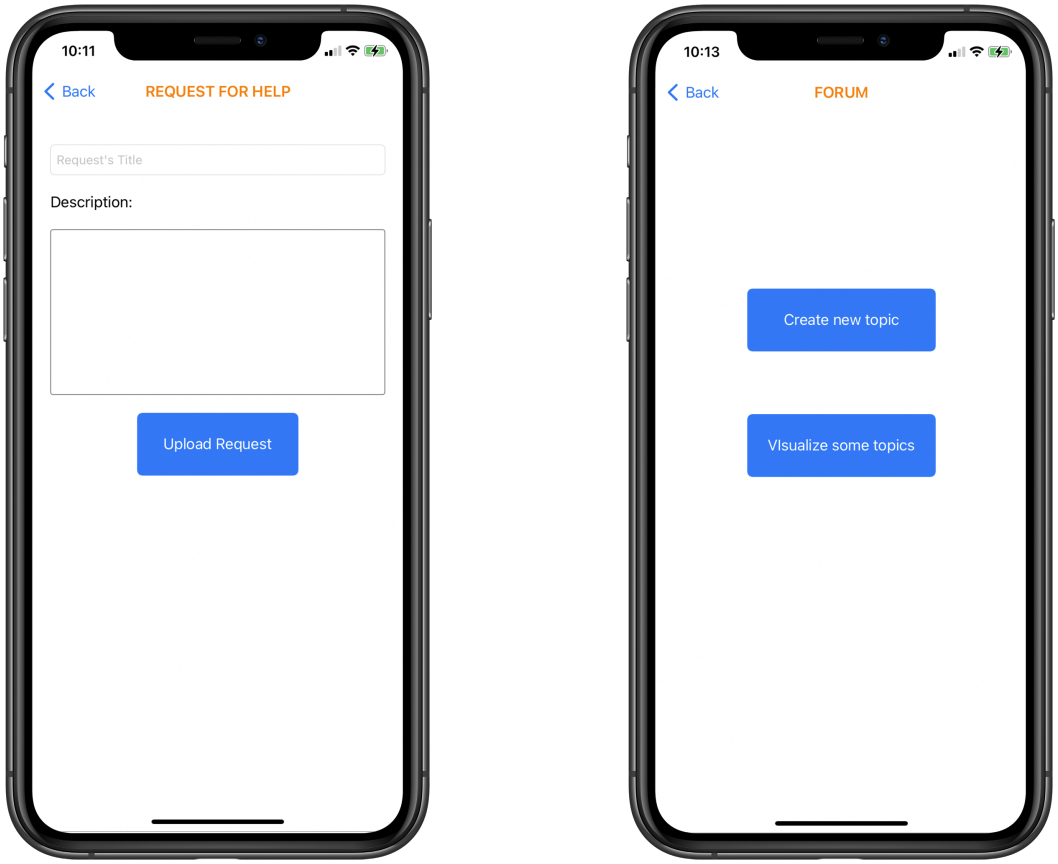


Figure 22: Request for help interface (on the left) and Forum interface (on the right) for a farmer of Dream.

From the home interface, clicking on the "Request for help" button, the farmer moves to the interface on the left: here he can fill in the fields regarding the problem (title and textual description) and send the request. While, clicking on the "Forum" button, he moves to the interface on the right: here he can choose either to create a new discussion or to visualize the discussion forum already present.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

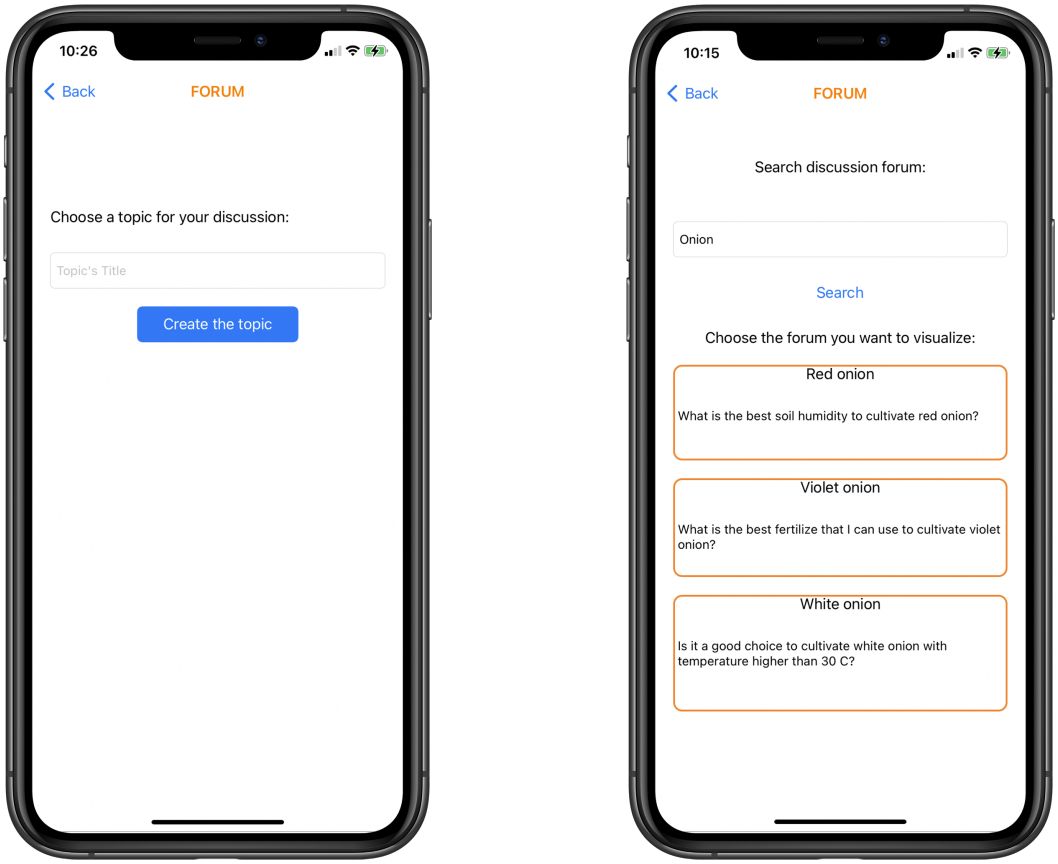


Figure 23: Create topic interface (on the left) and Visualize topic interface (on the right) for a farmer of Dream.

From the forum interface, clicking on the "Create new topic" button, the farmer moves to the interface on the left: here he can choose the topic for the discussion to be created. While, clicking on the "Visualize some topics" button, he moves to the interface on the right: here he can search for a topic and visualize the various discussions about that topic.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

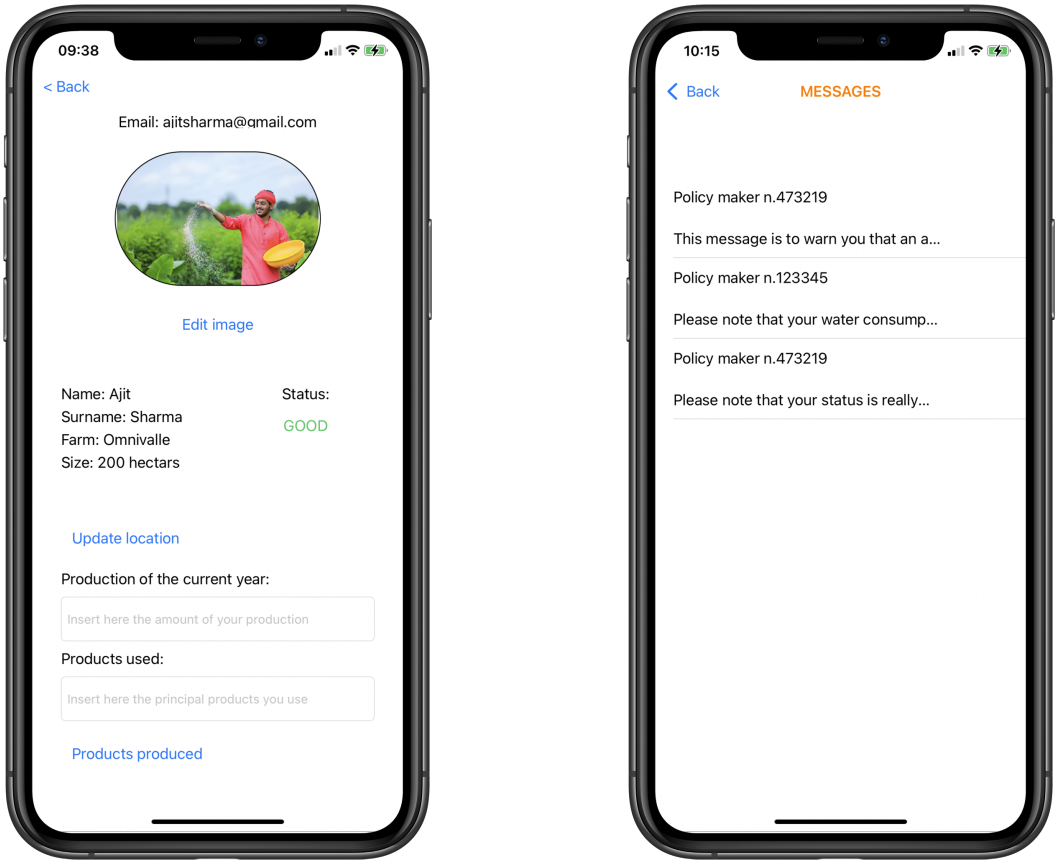


Figure 24: Profile interface (on the left) and Messages interface (on the right) for a farmer of Dream.

From the home interface, clicking on the button at the left top part, the farmer moves to the profile interface: here he can visualize and update his personal information, his status and other information regarding his farm and his production. While, clicking on the button at the right top part, he moves to the messages interface: here he can visualize all the messages addressed to him.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

3.2 Policy Maker Interfaces

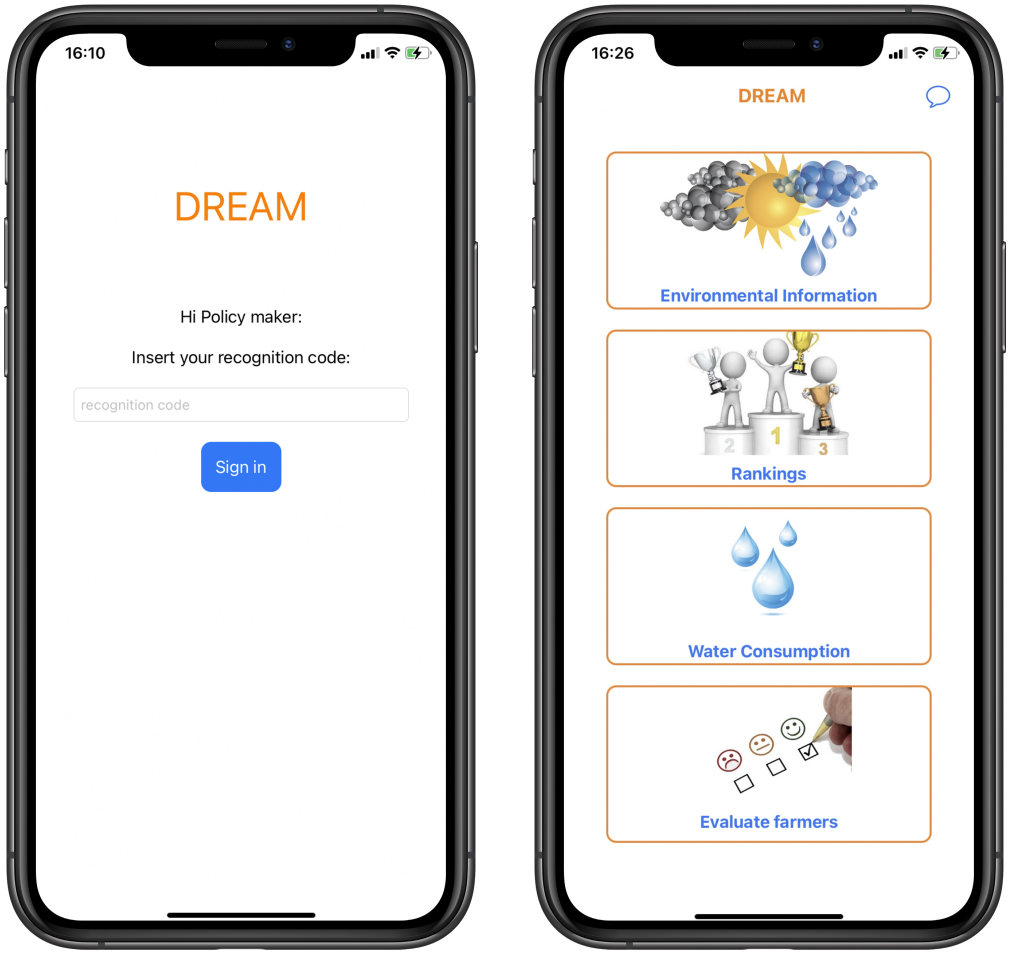


Figure 25: Log in interface (on the left) and Home interface (on the right) for a policy maker of Dream.

After inserting the recognition code and clicking on the "Sign in" button in the log in interface, the policy maker moves to the home interface, where he visualizes other four buttons representing the main features of the application and one icon at the top for the messages interface.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

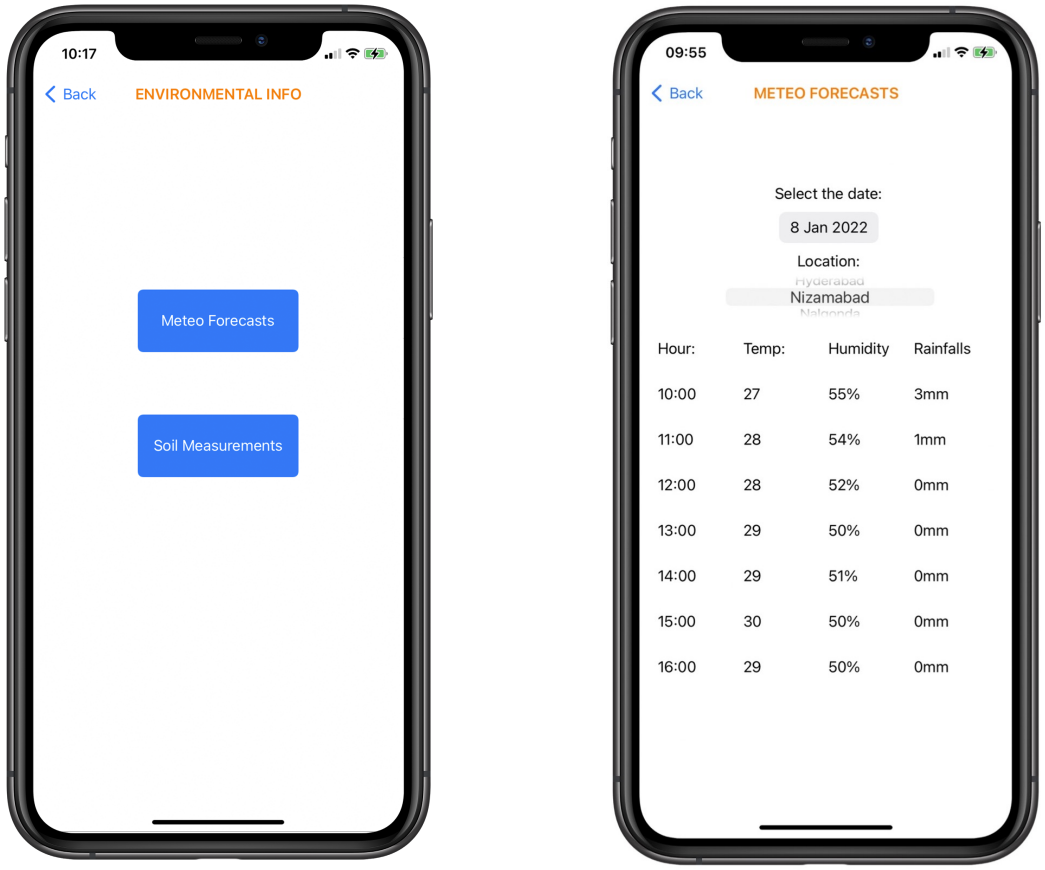


Figure 26: Environmental information interface (on the left) and Meteo forecasts interface (on the right) for a policy maker of Dream.

From the home interface, clicking on the "Environmental information" button, the policy maker moves to the interface on the left: here he can choose to visualize either the meteo forecasts or the soil measurements. Therefore, from this interface, clicking on the "Meteo forecasts" button, he moves to the interface on the right: here he can visualize temperature, humidity and rainfalls values for each time slot of the day he selects related to the location he selects.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

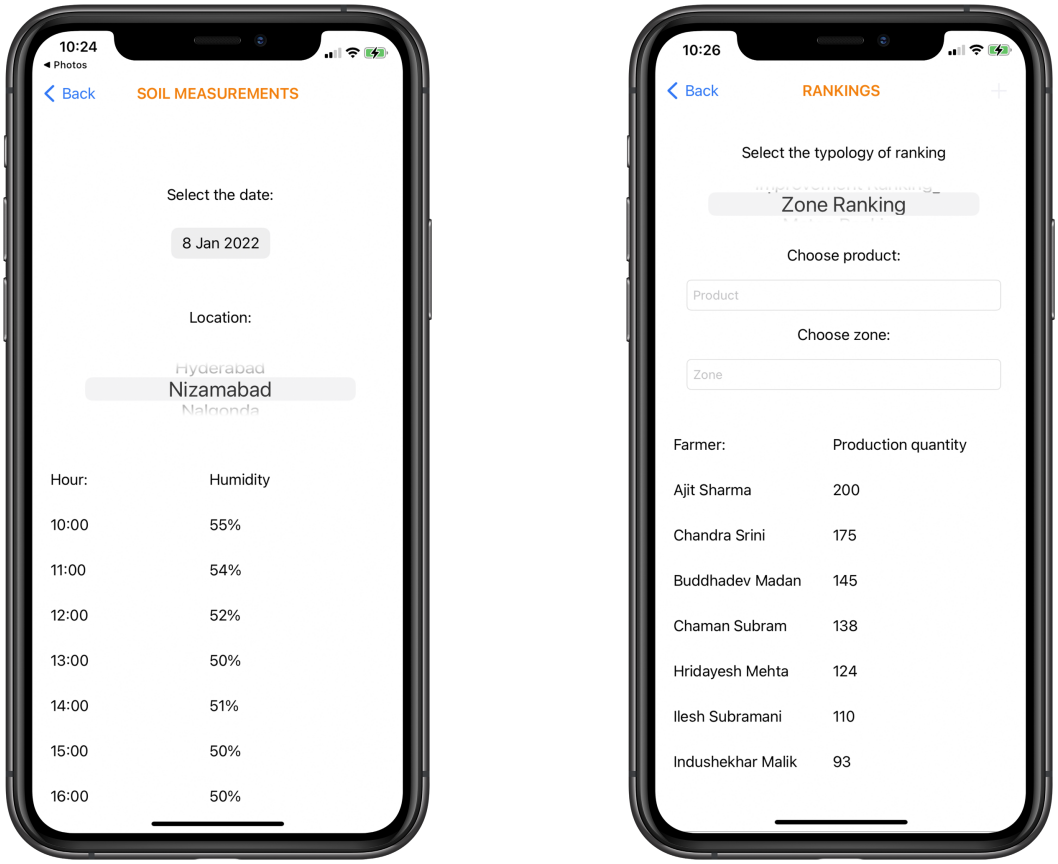


Figure 27: Soil measurements interface (on the left) and Rankings interface (on the right) for a policy maker of Dream.

From the environmental information interface, clicking on the "Soil measurements" button, the policy maker moves to the interface on the left: here he can visualize soil humidity values for each time slot of the day he selects related to the location he selects. While, from the home interface, clicking on the "Rankings" button, he moves to the interface on the right: here he can visualize a ranking selecting the typology and inserting the parameters to find it between all the possible ones.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

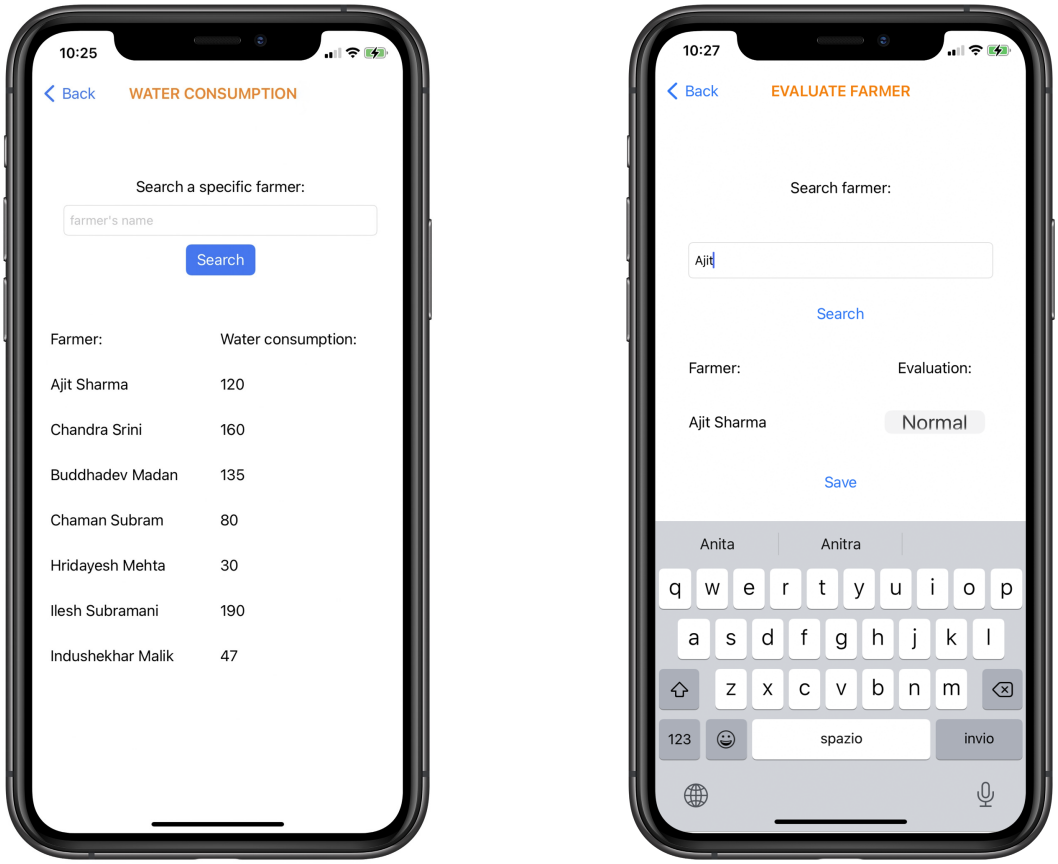


Figure 28: Water consumption interface (on the left) and Evaluate farmer interface (on the right) for a policy maker of Dream.

From the home interface, clicking on the "Water consumption" button, the policy maker moves to the interface on the left: here he can visualize the value correspondent to the water consumption of all the farmers or of a particular farmer that he searches. While, clicking on the "Evaluate farmer" button, he moves to the interface on the right: here he can search for a farmer and assign to him a status selecting it from the available ones.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

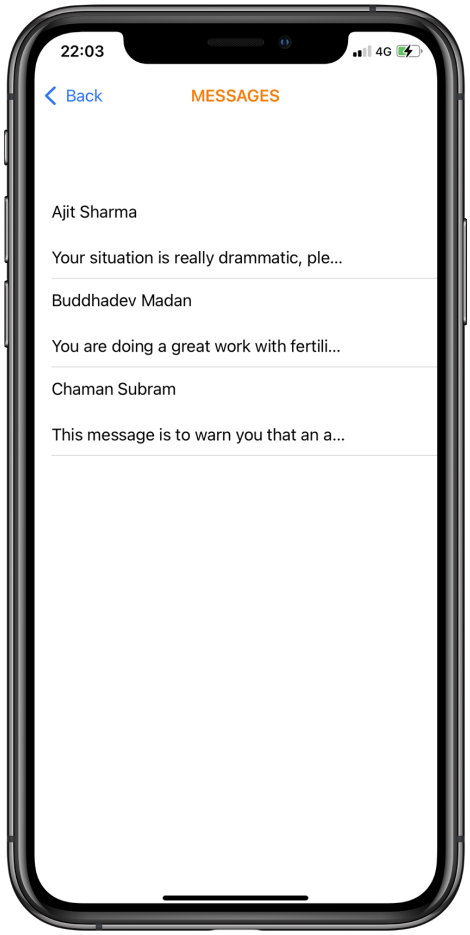


Figure 29: Messages interface for a policy maker of DREAM.

From the home interface, clicking on the button at the right top part, the policy maker moves to the messages interface: here he can visualize all the messages he sent to the farmers.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

4 Requirements Traceability

In this section, it is reported the table of requirements already discussed in the RASD document, with an additional column containing the software components (previously defined in this document) to which each requirements is mapped.

Requirement	Description	Mapped components
R1	The system shall allow an unregistered farmer to create an account in the system.	Farmer Client, FarmerLoginServiceImpl, Internal Database
R2	The system shall allow an unregistered farmer to insert his personal data in the registration page.	Farmer Client, FarmerLoginServiceImpl
R3	The system shall check if the personal data inserted by an unregistered farmer in the registration page are conflicting with respect to other existing accounts.[1]	FarmerLoginServiceImpl, Internal Database
R4	The system shall send an email containing a link to the unregistered farmer once he has inserted his data.	FarmerLoginServiceImpl
R5	The system shall show a confirmation page to the farmer once he clicks on the link contained in the email.	Farmer Client, FarmerLoginServiceImpl, Internal Database
R6	The system shall redirect the farmer, who is now registered, to the homepage.	Farmer Client
R7	The system shall allow an application farmer to log in the system.	Farmer client, FarmerLoginServiceImpl
R8	The system shall check if the credentials inserted by an application farmer in the log-in page are valid.[2]	FarmerLoginServiceImpl, Internal Database
R9	The system shall show to an application farmer only the weather forecasts relevant to his geographical zone.	Farmer Client, InformationServiceImpl, Internal Database
R10	The system shall allow the application farmer to select the parameter by which the visualization of the suggestions will be filtered.	Farmer Client
R11	The system shall show to the application farmer the suggestions about production regarding his geographical area.	Farmer Client, SuggestionServiceImpl, FarmerServiceImpl, Internal Database
R12	The system shall show to the application farmer the suggestions about production filtered by the products used for the cultivation.	Farmer Client, SuggestionServiceImpl, Internal Database
R13	The system shall show to the application farmer the suggestions about production filtered by the environmental conditions in which he works.	Farmer Client, SuggestionServiceImpl, FarmerServiceImpl, Internal Database

[1] The concept of conflicting data is defined in the section 1.3.1.
[2] The concept of valid credentials is defined in the section 1.3.1.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

R14	The system shall allow the application farmer to access to the profile section.	Farmer Client
R15	The system shall show to the application farmer his status in the profile section.	Farmer Client, Internal Database
R16	The system shall allow the application farmer to upload in the profile section the quantity produced during the last year of a certain type of product.	Farmer Client, InformationServiceImpl, Internal Database
R17	The system shall allow the application farmer to upload in the profile section the type of products that he uses to cultivate.	Farmer Client, InformationServiceImpl, Internal Database
R18	The system shall allow to upload a suggestion only to an application farmer who has the status of good farmer.	Farmer Client, SuggestionServiceImpl, FarmerServiceImpl, Internal Database
R19	The system shall allow the application farmer to create a discussion forum.	Farmer Client, ForumServiceImpl, Internal Database
R20	The system shall allow the application farmer to access a discussion forum.	Farmer Client, Internal Database
R21	The system shall allow the application farmer to search for a topic in the discussion forum.	Farmer Client, ForumServiceImpl, Internal Database
R22	The system shall allow the application farmer to write into a discussion forum.	Farmer Client, ForumServiceImpl, Internal Database
R23	The system shall allow the application farmer to fill in a form to request for help.	Farmer Client, HelpRequestServiceImpl, Internal Database
R24	The system shall dispatch the help requests only to application farmers who have a good status.	Farmer Client, HelpRequestServiceImpl, FarmerServiceImpl, Internal Database
R25	The system shall dispatch the help requests only to application farmers who work in similar work conditions.	Farmer Client, HelpRequestServiceImpl, FarmerServiceImpl, MessageServiceImpl, Internal Database
R26	The system shall allow an application farmer to answer to a help request, which has been dispatched to him.	Farmer Client, MessageServiceImpl, HelpRequestServiceImpl, Internal Database
R27	The system shall generate the rankings of the farmers based on the quantity produced of a specific product with respect to the size of their land classified by different geographical areas.[3]	ZoneRankingServiceImpl, FarmerService, Internal Database
R28	The system shall generate the rankings of the farmers based on the quantity produced of a specific product with respect to the size of their land classified by different meteorological conditions.	MeteoRankingServiceImpl, FarmerService, Internal Database

[3]The size of the land is taken into consideration inserting into the rankings the quantity produced per square meter.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

R29	The system shall generate the rankings of the farmers based on the quantity produced of a specific product with respect to the size of their land classified by different soil conditions detected by the sensors.	SoilRankingServiceImpl, FarmerService, Internal Database
R30	The system shall generate the rankings of the farmers based on the percentage increase of the quantity produced for a specific product with respect to the size of their land in a certain period of time.	ImprovementRankingServiceImpl, Internal Database
R31	The system shall allow a policy maker to log in the system through a recognition code.	Policy maker Client, PolicyMakerLoginServiceImpl, Internal Database
R32	The system shall check if the recognition code inserted by a policy maker is valid. [4]	PolicyMakerLoginServiceImpl, Internal Database
R33	The system shall allow the policy maker to visualize the rankings of the farmers based on the quantity produced of a specific product with respect to the size of their land classified by different geographical areas.	Policy maker Client, ZoneRankingServiceImpl, Internal Database
R34	The system shall allow the policy maker to visualize the rankings of the farmers based on the quantity produced of a specific product with respect to the size of their land classified by different meteorological conditions.	Policy maker Client, MeteoRankingServiceImpl, Internal Database
R35	The system shall allow the policy maker to visualize the rankings of the farmers based on the quantity produced of a specific product with respect to the size of their land classified by different soil conditions detected by the sensors.	Policy maker Client, SoilRankingServiceImpl, Internal Database
R36	The system shall allow the policy maker to visualize the rankings of the farmers based on the percentage increase of the quantity produced for a specific product with respect to the size of their land in a certain period of time.	Policy maker Client, ImprovementRankingServiceImpl, Internal Database
R37	The system shall allow a policy maker to assign a status to an application farmer according to a methodology previously established based on the result shown by the rankings.	Policy maker Client, EvaluationServiceImpl, FarmerServiceImpl, Internal Database

[4] The concept of valid recognition code is defined in the section 1.3.1.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

R38	The system shall allow a policy maker to visualize the farmers' water consumption.	Policy maker Client, WaterConsumptionServiceImpl, Internal Database
R39	The system shall allow a policy maker to send a notification to an application farmer who has a bad status to invite him to request for help through the system.	Policy maker Client, MessageServiceImpl, Internal Database
R40	The system shall allow a policy maker to send a notification to an application farmer who as a good status in order to ask him to upload a suggestion in the system.	Policy maker Client, MessageServiceImpl, Internal Database
R41	The system shall allow a policy maker to send a notification to an application farmer who has a bad status in order to tell him that he will receive the visit of an agronomist.	Policy maker Client, MessageServiceImpl, Internal Database
R42	The system shall allow a policy maker to send a notification to an application farmer asking him to reduce his water consumption.	Policy maker Client, MessageServiceImpl, Internal Database
R43	The system shall periodically obtain data from the database regarding measures collected by the sensors.	SoilMeasurementCopyManager, Internal Database
R44	The system shall periodically obtain data from the database regarding measures collected by the water irrigation system.	WaterIrrigationCopyManager, Internal Database
R45	The system shall periodically obtain data from the database regarding meteorological forecasts collected by the government.	MeteoForecastCopyManager, Internal Database
R46	The system shall periodically obtain data from the database regarding information collected by the agronomists.	AgronomistInfoCopyManager, Internal Database

Table 5: Table requirements' mapping to the software components

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

5 Implementation, Integration and Test Plan

5.1 Implementation plan

The main advantage of choosing an architecture structured in sub-components is the possibility of developing them individually and defining specific testing procedures for each part of the system. To this end, it is necessary to previously define the order in which the components will be implemented, following a bottom-up approach. This order will depend on the dependencies among components and sub-components. The implementation order that has been chosen is the following one:

1. Internal Database and derived model of data
2. All the implementations of the Copy manager interface
3. Information and Message services
4. Farmer service
5. All the other services

The first component that will be implemented is the Internal Database with the derived model of data. All the other components and sub-components of the application will interact with the database through the usage of the model of data; therefore its implementation is fundamental for the development of all the system. After the implementation of the internal database it would be possible to start the implementation of the application server. The first fundamental part to implement is the Copy manager; it is not possible to implement it before because it needs the internal database to store the information that retrieves from the external services, however its work is essential for the successive sub-components because it provides all the information from external services that most of the other services use. Then it is possible to start the development of all the services of the application server. The first ones that will be implemented are the Information service and Message service: they don't need any other service to work properly but other services depend on them, so they are essential for continuing the process of developing. After them it will be implemented the Farmer service: it is important to develop it before all the other services because most of them depend on it, but it has to be implemented in a successive phase with respect to the Information service. Now that all the services which create dependencies are developed, it is possible to implement all the remaining services. It is important to notice that some services don't depend on other services, neither there are other services that depend on them: hence the order of their implementation with respect to the other services can change without any problem (for instance, the Farmer Log In service could be implemented also before the Information and Message service). However, for simplicity, they are grouped with all the remaining services. Throughout the development process of the application server, it is also needed the simultaneous implementation of the client applications' modules (which from now on in the diagrams will be labeled as "farmer client module" and "policy maker client module")that allow the testing of the components as they are created. In such way it will be possible to implement an incremental integration and testing.

5.2 Integration and Testing

To integrate and test the several components of the DREAM system it will be used an incremental approach, which means that the integration and testing occur while the components are released, instead of at the end of their implementation. To this end, it will be adopted a bottom-up strategy, which consists in integrating and testing in order the components as they are implemented; however, this approach requires the implementation of specific drivers for each module that has to be integrated and tested. Then, as the other components are added to the system, such drivers will be replaced with the correspondent software components. Previously to the integration of the components, on each of them have to be executed unit tests by the developers, in order to understand whether that portion of the code works

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

correctly. For these unit tests, it will be used the technique of white-box testing, that requires the tester to have a good level of understanding of the code that has to be tested; this can be done on small portions of code, that's why it will not be used this approach for the testing of the integration among components. In addition, unit tests will occur randomly with the generation of numerous test cases, in order to observe the behaviour of the specific component on a large scale. For what concerns the integration testing, in the following section it will be specified in which order the components will be integrated and tested.

5.2.1 Sequence of components integration

The first things to be integrated and tested together will be the model of data associated with the internal database and the implementations of the Copy Manager service, which is needed to retrieve the information from the external services and store them into the internal database. Once integrated, these components will be tested together with a black-box approach and in a random way, inserting input data from the external services and verifying that such data are inserted in the database.

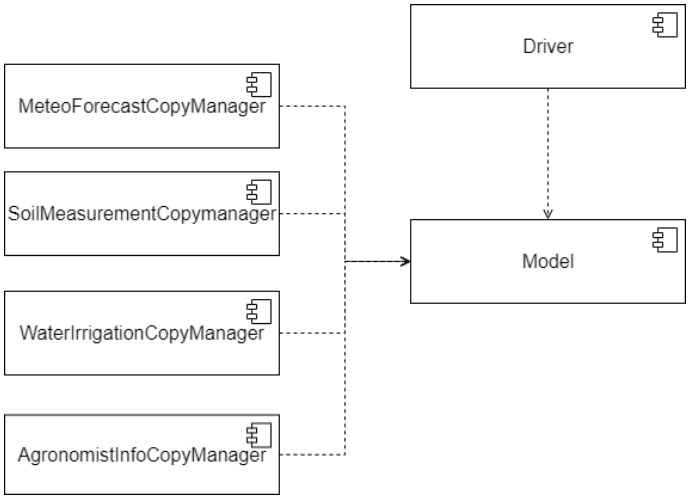


Figure 30: Integration of the copy manager service components with the database.

Then, the information service is integrated with the model of data and its functionalities are tested by means of random testing with a black-box approach, in order to see if its methods are able to retrieve data from the database and update data already present in it. In this way, it will be possible to test the first functionalities that have to be provided to an end-user, namely that of visualizing the meteorological forecasts or the soil measurements and that of updating data into the database. In addition, also the Message service is integrated with the model of data and its simple functionality regarding the creation of a message and its association with the respective sender and receiver is tested. These tests are executed randomly, because at this level it is not necessary to consider a systematic approach, which is usually needed for searching possible bugs derived from the integration, due to the limited number of components involved.

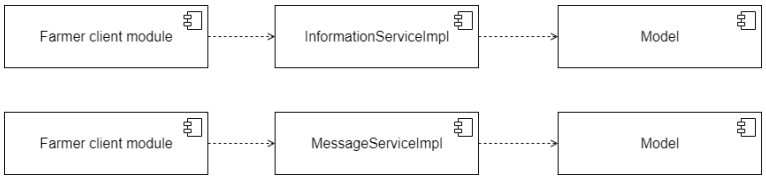


Figure 31: Integration of the information service and message service components with the database.

In order to integrate most of the other services to the system, the functionalities of the Farmer service are needed. For this reason, it will be the next component to be integrated. It can be noticed that it will interact, not only with the model of data, but also with the Information Service, which is used in order

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

to implement some of the methods of the Farmer service. These two services can be tested separately even if there are present already other services integrated to the system (that's why in the figure there are shown only the two of them and the model). To test the integration of these components will be executed a systematic testing, in order to see if the interaction between the two services leads to any malfunction.

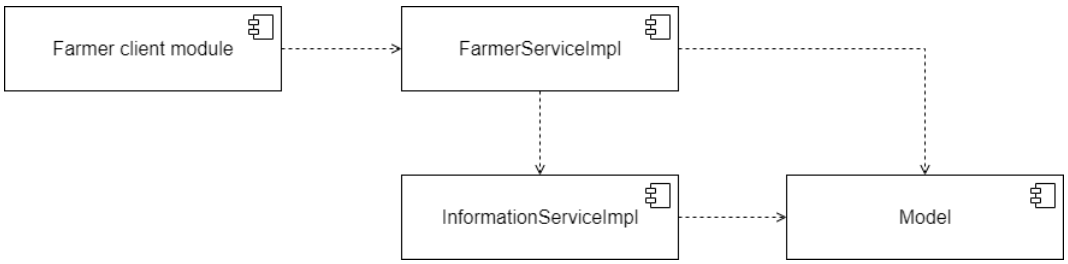


Figure 32: Integration of the Farmer service with the other components.

The order in which integrate the other components is not important, however, there will be integrated first the services that use the ones already discussed. In this way, it will be easier to detect the causes of possible problems in the integration. The Help request service has to be integrated with both the message and the farmer services. In such way it will be possible to test together the four services with the model, testing the functionalities of the help request service, which are: creating a help request starting from information retrieved through a form, individuating the adequate farmer to answer that specific request considering the work conditions of the farmer who sent it, creating a message to answer the request by means of the message service.

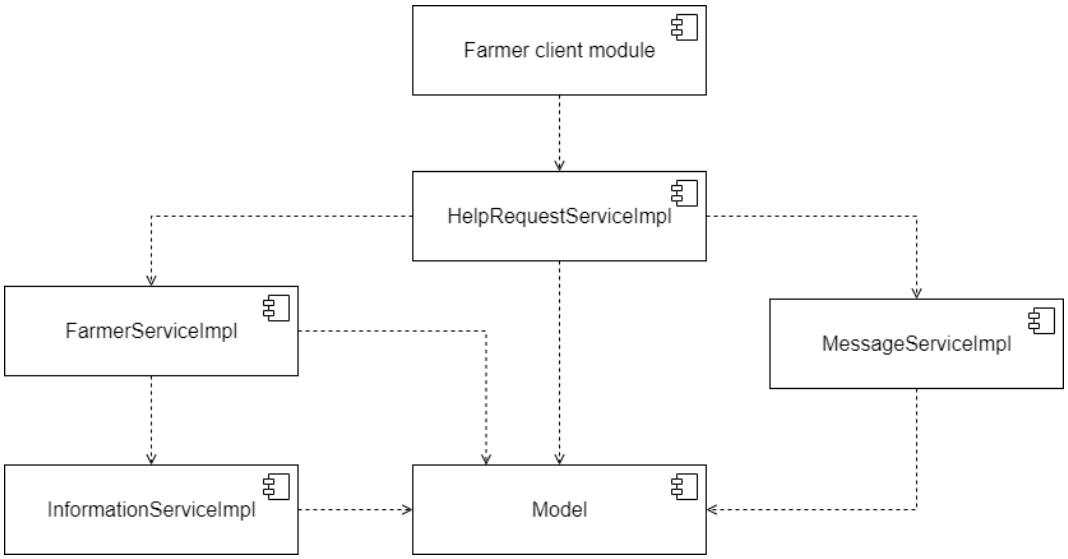


Figure 33: Integration of the Help request service.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

The suggestion service, once integrated, has to be tested together with the farmer and the information services, such tests can be done in a random way.

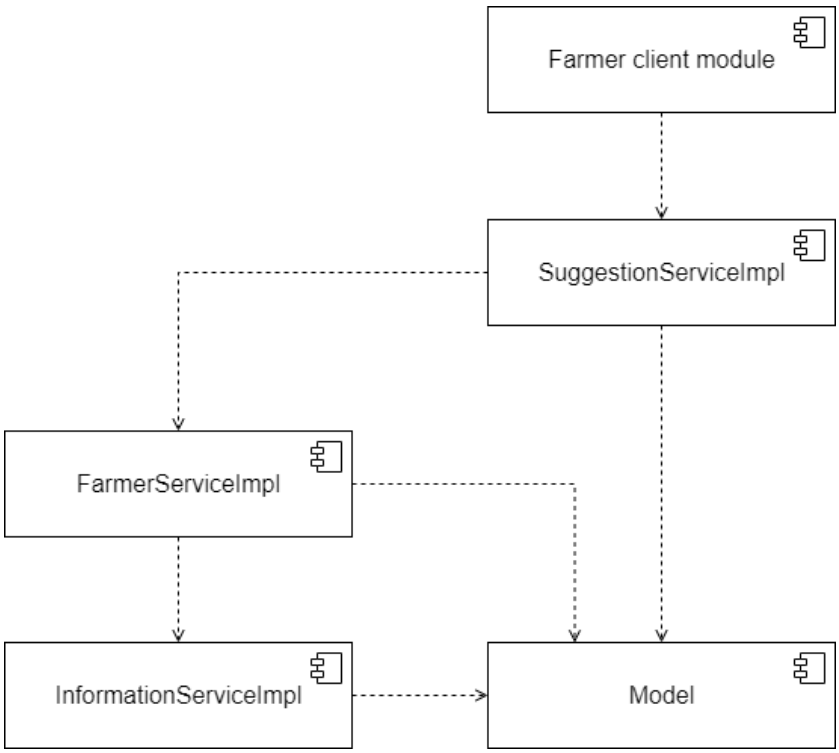


Figure 34: Integration of the Suggestion service.

At this point, we can proceed with the integration of the Forum service, which can be tested only with the model because it doesn't interact with the other services already integrated in the system. Also these tests can be done in a random way.

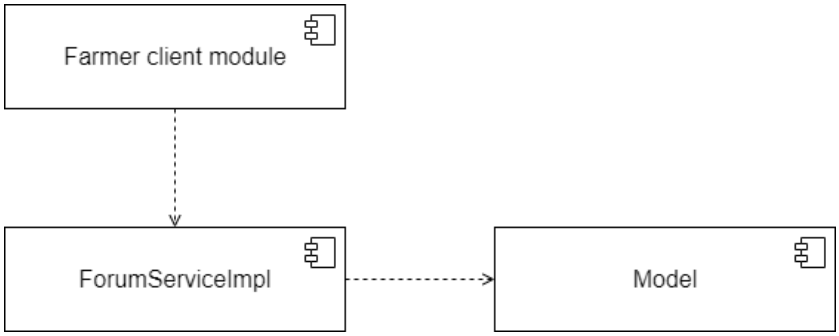


Figure 35: Integration of the Forum service.

The last service that will interact with the farmer client is the login service, which again, can be tested only with the model because it doesn't use other services in order to complete its functions. The tests regarding this service will be of two types, first it will be execute random testing for the registration and login of multiple farmers, in order to see if all works as desired; then, it will be executed systematic testing to see if inserting conflicting data or invalid credentials returns what was expected.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

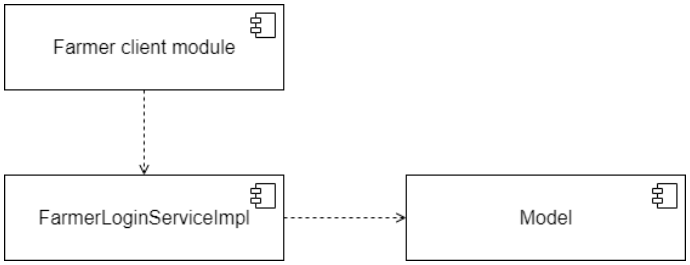


Figure 36: Integration of the Farmer Login service.

Each of the above described tests will require the implementation of specific modules in the farmer client application; once the integration of those services in the application server is completed, it will be possible to put together also all the modules of the user interfaces relevant to the farmer client and test all the functionalities that the system has to provide to a farmer user. This test will be executed with a black-box approach, since it has to verify that all the requirements are satisfied and that the application is ready for the end-user (who knows nothing about how the code of the software components is written) to use it.

The services that are still missing in the integration of the system are the ones related to the policy maker client application. The first to be integrated will be the Ranking service's implementations, which, in order to work properly, have to interact with the farmer service. For this reason, they will be integrated and tested together with the farmer service and the model. The tests will deal with the verification of the service's functionalities, i.e. the generation of the various types of rankings and their visualization. These tests will require a systematic strategy because it is necessary to verify if the integration between the farmer service and the ranking service provides the expected results

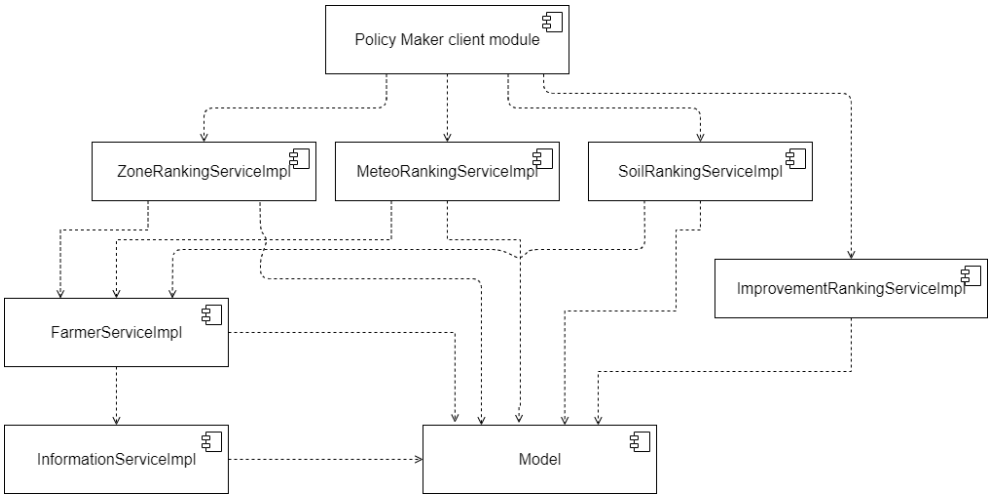


Figure 37: Integration of the Ranking service.

The last service that requires the functionalities of the farmer service is the evaluation service, so it will be the next to be integrated.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

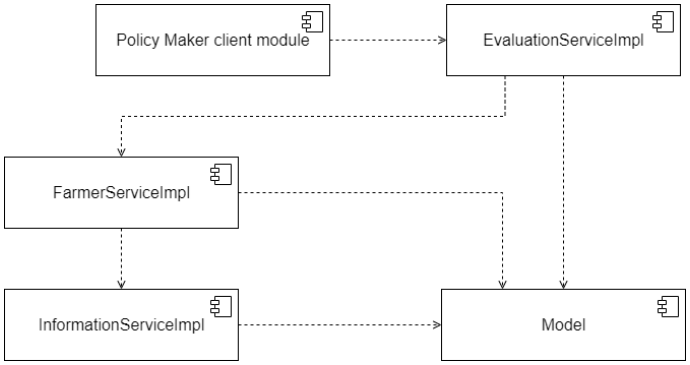


Figure 38: Integration of the Evaluation service.

The water consumption service will be now integrated; it interacts only with the model and the client application, for this reason, the testing can be done considering only these components and it will be executed in a random way because it has the only scope of verifying if the visualization of the water consumption by an end-user works properly.

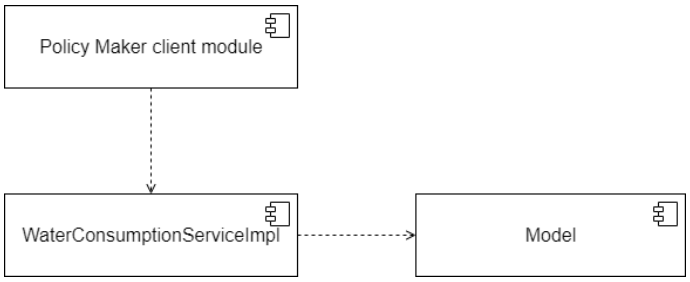


Figure 39: Integration of the Water consumption service.

At the end of the integration process, the policy maker login service can be added and tested. The tests of the functionalities of such service will require the interaction only with the model and the client. The testing strategies used will be of two types: random testing to see if the login works on a large scale and systematic testing to observe the behaviour of the system when the recognition code inserted by the end-user isn't valid.

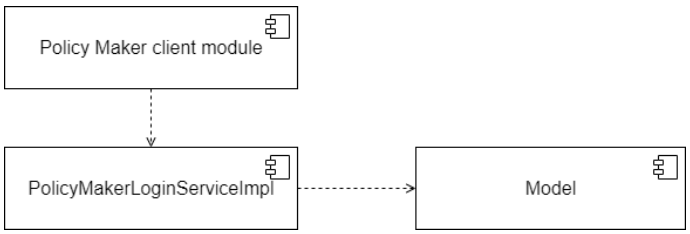


Figure 40: Integration of the Policy maker login service.

At this point, it will be possible to integrate together all the modules of the policy maker client application and test its functionalities. As for the farmer client, also in this case, the tests will follow a black-box approach and will verify that all the features that the system has to present to a policy maker user work accordingly to the requirements.

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

6 Effort Spent

- Student 1

Topic	Hours
Introduction	2:00h
Overview on the architecture	3:00h
Class Diagram	1:30h
Component Diagram	8:00h
Deployment view	1:00h
Runtime view	3:00h
Component Interfaces	2:00h
Architectural styles and patterns	1:00h
Other design decisions	3:30
User interface design	5:00
Requirements traceability	1:30h
Implementation plan	2:30h
Integration and testing plan	2:00h
Document organization	4:00h
Total effort spent	**40:00h**

Table 6: Effort spent by student 1

- Student 2

Topic	Hours
Introduction	2:00h
Overview on the architecture	3:00h
Class Diagram	1:30h
Component Diagram	8:00h
Deployment view	3:00h
Runtime view	1:30h
Component Interfaces	2:30h
Architectural styles and patterns	3:30h
Other design decisions	1:00h
User interface design	2:00h
Requirements traceability	1:30h
Implementation plan	1:30h
Integration and testing plan	5:00
Document organization	4:00h
Total effort spent	**40:00h**

Table 7: Effort spent by student 2

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved

7 References

- All the diagrams have been made with Draw.io
- All the user interfaces have been made with the Storyboard of Xcode

Copyright © 2022, Federica Tommasini, Alessandro Verosimile – All rights reserved