Politecnico di Milano AA 2021-2022

Computer Science and Engineering

Software Engineering 2 Project

DD

Design Document

version 1.0 - 10/12/2021

Blasucci Camilla - 907559

Buono Salvatore – 907445

1. INTRODUCTION

1.1. Purpose

This document aims to describe in detail the DREAM app with a deep look to the architecture used, all the modules of the system with their interfaces, and a runtime view of the main functions of DREAM also through interactions diagrams.
This document also contains an explanation of the implementation and the testing part of the system.

1.2. Scope

DREAM is thought for the different actors of the agronomical field in Telangana that will use it.
Farmers use the app to check their data, read and write comments, ask for help and check the forecast.
Policy Makers use the app to check on the farmers in their area and look at different charts to provide help or give prizes to the farmers.

1.3. Definitions, Acronyms, Abbreviations

Definitions

- **Farmer –** any Telengana farmer
- **Policy Maker –** any Telengana policy maker
- **Agronomist –** any Telengana agronomist
- **Crop –** term used to generalize the crop and the fields of a farmer
- **System –** the entire hardware and software entities that form the service. Also called app, application, DREAM
- **Tier –** a physical (or a set of) machine with its own computational power
- **Object–relational mapping (ORM, O/RM, and O/R mapping tool) –** in computer science is a programming technique for converting data between incompatible type systems using object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language.

Acronyms

- **TSDPS -** Automatic Weather Station
- **DREAM -** Data-dRiven PrEdictive FArMing in Telangana
- **TCP –** Transmission Control Protocol
- **DBMS –** Database management system
- **MVC –** Model View Controller

1.4. Revision History

10th December 2021- version 1.0 started

8th January 2022- version 1.0 finished

1.5. Reference Documents

- Software Engineering 2- year 2021- prof. Di Nitto Elisabetta course slide
- Specification document resources: https://www.tsdps.telangana.gov.in/aws.jsp
- Specification document: "R&DD Assignment A.Y. 2021-2022"
- Design Pattern slide from the Software Engineering 1-year 2020- prof Margara

1.6. Document Structure

Section 1: Introduction

This first section is intended to define the goal of the Design Document, to give an overview of the DREAM app main functionalities and explain the different vocabulary used in this document.

Section 2: Architectural Design

The second section is the core of the document where the different design chosen to implement the app are fully explained.

Starting with a high level description, the different paragraphs describe in depth the architectural choices to fulfil the functional and non-functional requirements.

This also contains the UML sequence diagrams that give an inside look of the communication between the different components.

Section 3: User Interface Design

This section contains a larger collection of mockups to have a deeper look at the design of the different interfaces.

Section 4: Requirements Traceability

This section is intended to explain the connection between the logical components of the design documents and the fulfilment of the requirements presented in the RASD.

Section 5: Implementation, Integration and Test Plan

This section is addressed to the implementation process of the app. Here there is the description of the procedures for implementing and testing the different components.

2. ARCHITECTURAL DESIGN

2.1. Overview

The DREAM system is a distributed application based on the client-server paradigm, with two different types of clients:

- The first client type is an RIA (Rich Internet Application) web app. Since it completely depends on the server, is a thin client: it only presents a presentation layer, it does not contain any logic.
- The second type is a mobile application. This is a thick client, since it needs a database inside to be more independent from the server.

The server used is a fat server, it contains all the data and the logic needed to make DREAM work properly.
DREAM is based on a three layers architecture; this is obtained also by the physical separation of the three layers on different tiers:

- Rendering layer: the layer that administer the interactions with the users and so the presentation logic.
- Logic layer: it manages the logic behind the functions provided by DREAM
- Data layer: this layer manages the data storage

DREAM must be accessible for all users through web and mobile app.
The architecture of the application also contains some replicated web server, which perform as a middleware between user's browser and application servers.
Furthermore, the core of the software of mobile application is installed on user's device, which interfaces with the business layer's APIs which send and receive all the information.
On the other hand, the application servers are considered stateless, which means that they do not maintain any information about the state of the user. The service is perfectly capable of operating by using only the information available in the request payload, or by acquiring the necessary information from a database.
Finally, the application servers' interface with the DBMS APIs, which are divided in Stateful API for the communication between user and database and Event-Driven API for notification service.
For accessing the data, they will use an ORM programming technique to interface
with the DBMS exploiting the advantages of the object-oriented paradigm.

2.2. Component View

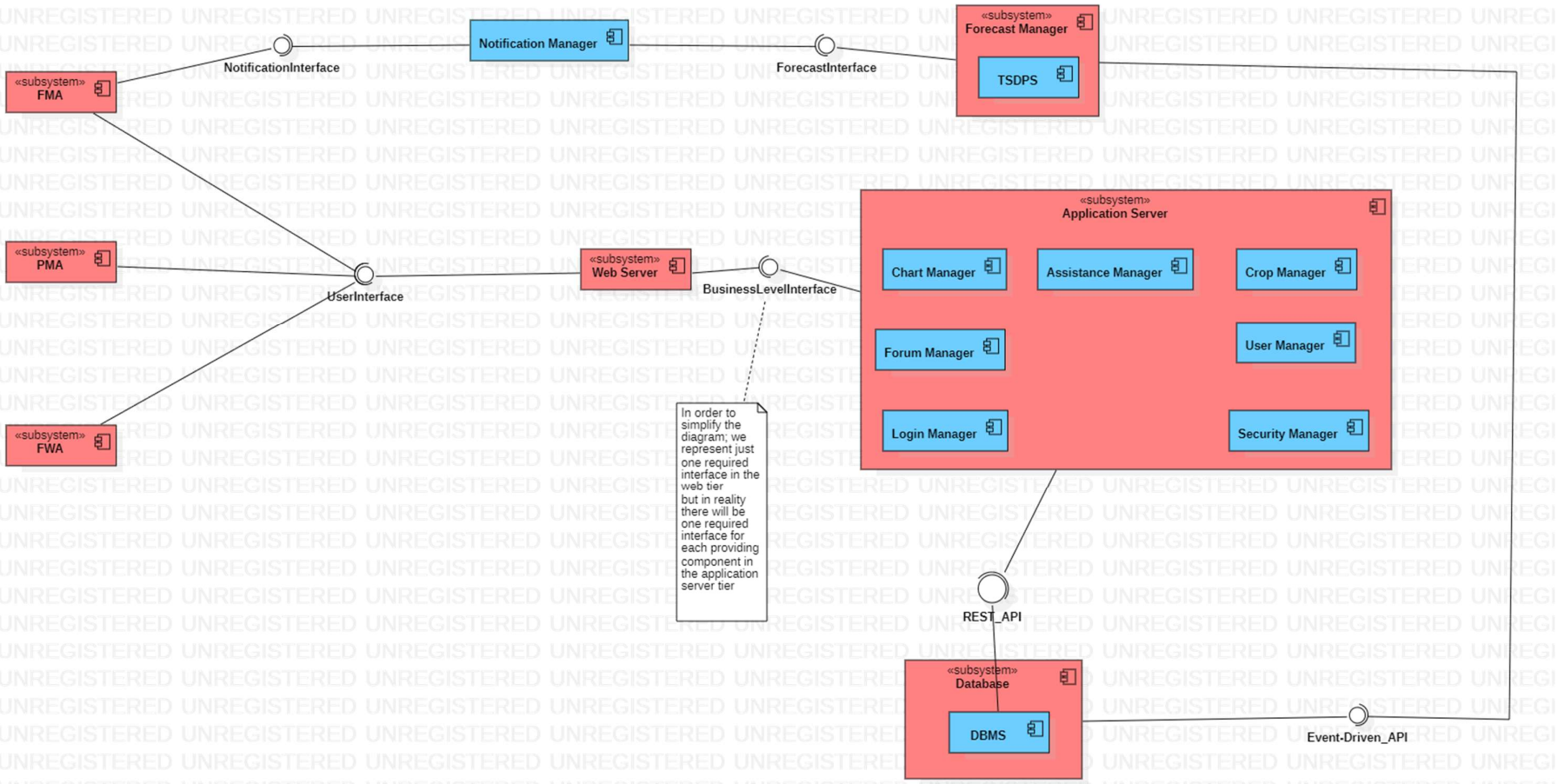


Figure 1: UML Component Diagram

In figure 4 we can see a more detailed diagram representing the three layers.
The job of the web server is to direct the browser requests to the application server and send back its responses.
Here are summarized the most important feature of the subsystems provided.

- FMA

  It is the subsystem in charge of the communication between the farmer and DREAM system. It allows farmer to register into the system, visualize his personal data and all relevant information about his crops. It is built as a mobile application; it needs to access to the mobile notification system.

- FWA

  It is the subsystem which has the same functionalities of FMA, but it is built as a web application and designed for the most common browser web.

- PMA

  It is the subsystem in charge of the communication between policy maker and DREAM system. It allows them to access to all farmers' profile, see global chart and send prize and contact farmers every time they want. It is also built as a mobile application.

- Web Server

  As highlighted before, it acts as a middleware between user browser and application server. It is interested in connecting user with the system.

There are also different modules that handle the various functions of the app:

- Login manager

  This module handles the log in or sign in operation through a LoginInterface. Once the client has logged in, the module shows only the other types of interfaces relying on the type of log in: a client logged as a farmer will exploit the FMAUserInterface, a policy maker a PMAUserInterface.

- Chart Manager

  This module handles the chart requests, collects the data and sends back the right updated chart. It also works with the crop manager to update the data to have always the most correct chart visualized.

- Forum Manager

  This module's job is to add the user's comments on the forum, answer the requests of visualization of the forum and update it each time there is a change. It includes discussion creation, discussion update, comment check.

- Crop Manager

  Similarly to chart manager, it collects the data inserted by the users and update the profiles. It handles the verification of the data inserted and provide to notify the user in case of forgotten data insertion.

- Assistance Manager

  In case of forgiven password, it ensures the recovery procedure sending an email and when the farmer reset the password, it updates user credentials in the database.

- Security Manager

  It ensures that every credential stored in the database is encrypted to external users and ensures that each communication between users and DREAM system is based on TLS *(Transport Layer Security)* protocol.

2.2.1. FMA/PMA

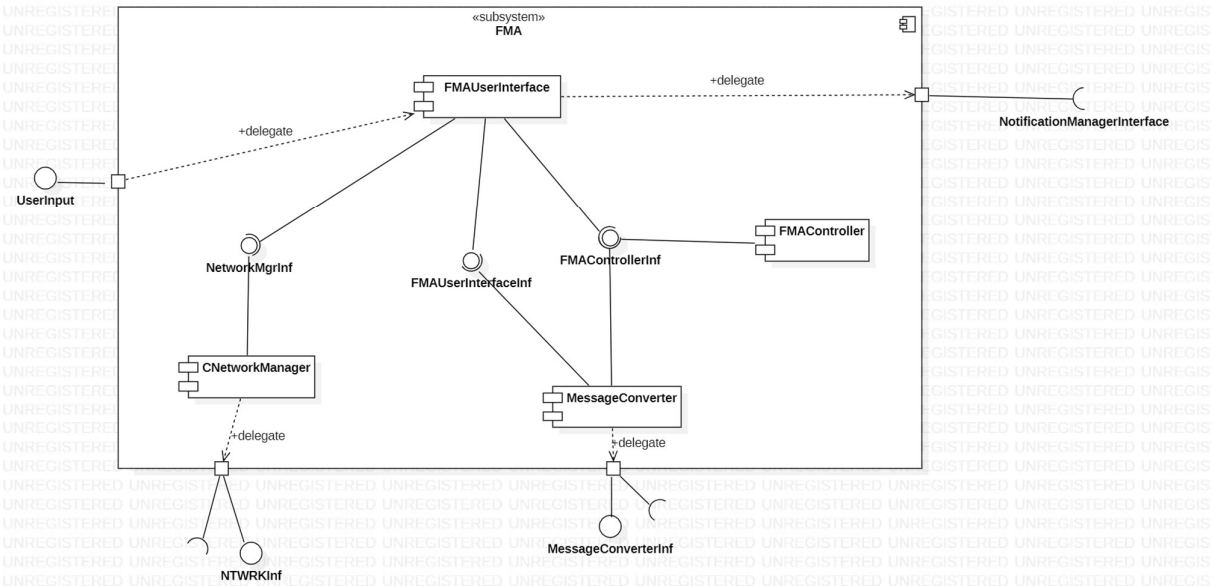


Figure 2: UML component diagram - FMA

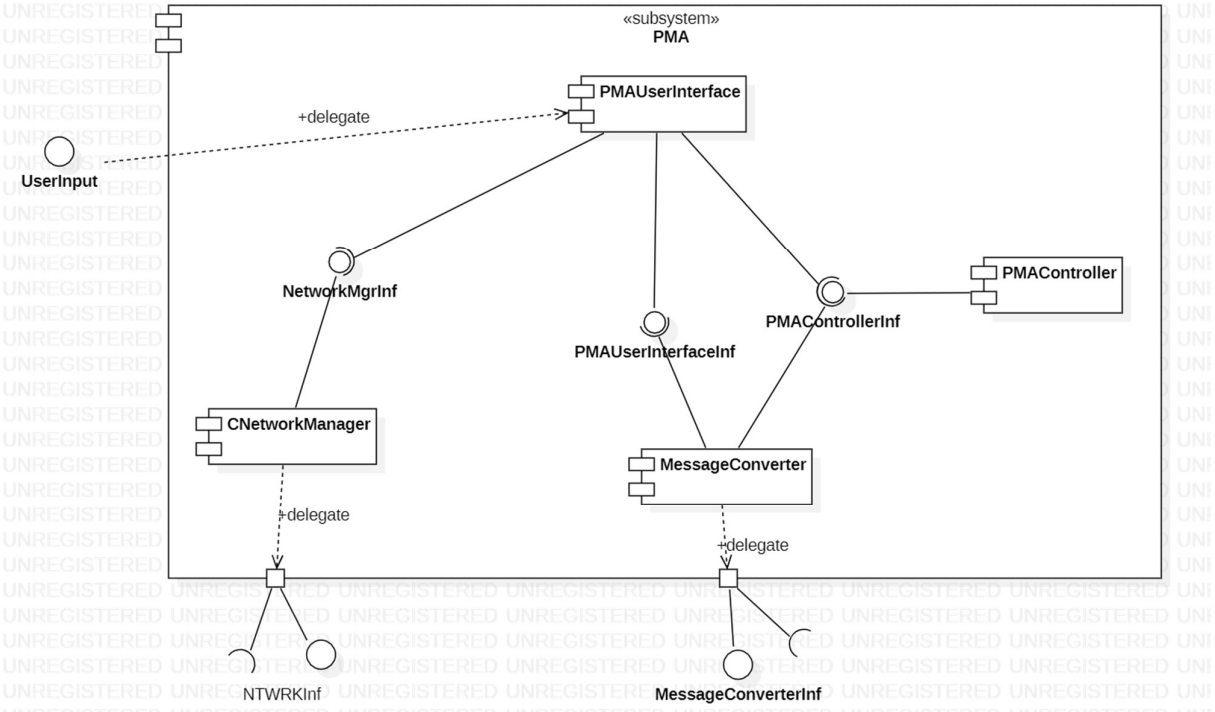


Figure 3: UML component diagram - PMA

- *FMA/PMA UserInterface:* it is in charge of showing to the farmers and policy makers messages coming from the system and enable farmers and policy maker to insert proper information when needed. FMA notifications from TSDPS are shown in the UI through NotificationManagerInterface.

- *CNetworkManager:* it provides the high-level functions to send and receive message on the network and it manages the low-level network concerns. It is able to notify the view when a message comes.

- *MessageConverter:* it is in charge of formatting commands into XML messages to be sent to or received from the network.

- *FMA/PMA Controller:* it is in charge of receiving commands from the UI and perform all operation needed to fulfil the action (like message conversion, checking the applicability of a specific command), possibly using the connected component.

2.2.2. Web Server

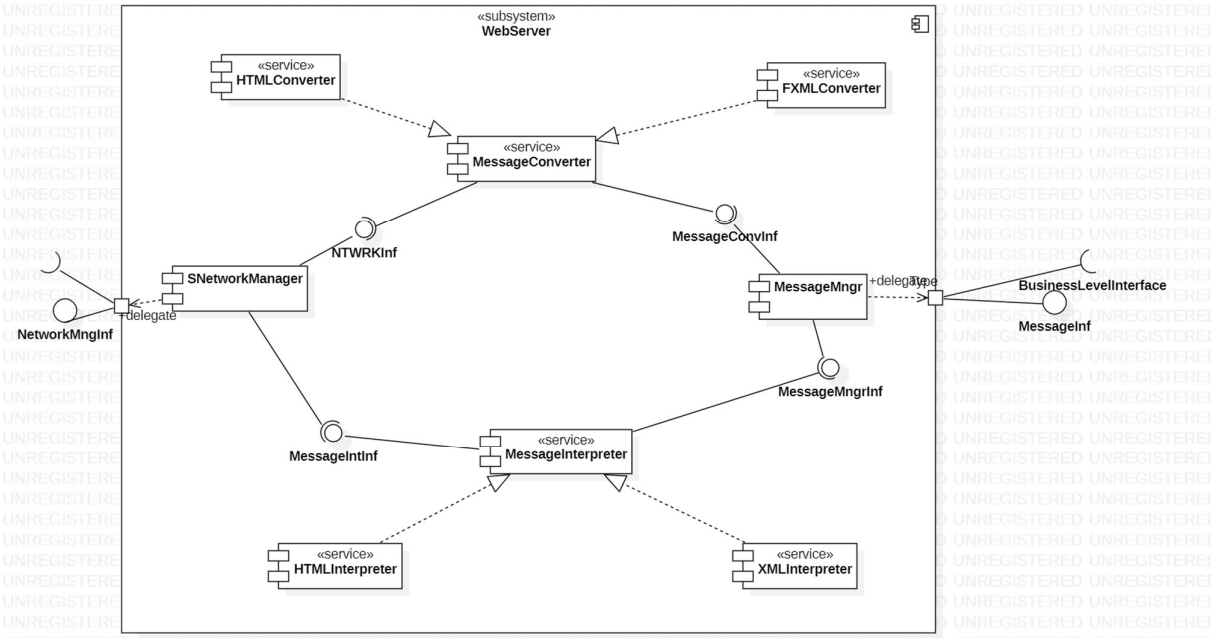


Figure 4: UML component diagram – Web Server

- *MessageConverter:* it is the service able to translate the information or commands from Application Server into a specific format, that can be sent over the network and interpreted by applications. There are two different implementations of this component:
    - *HTMLConverter:* it generates an HTML page with all relevant information related to the event for web clients.
    - *FXMLConverter:* it generates FXML code with all relevant information related to the event for mobile clients.
- *MessageInterpreter:* it has the opposite role of the MessageConverter service, because it aims to translate messages from the network into commands to be executed by the MessageMngr. Symmetrically, according to the message received, we have two different implementation of this component:

- *HTMLInterpreter:* it converts HTML information (typically commands passed by means of POST or GET) into a command.

- *XMLInterpreter:* it converts an XML code into a command.

- *MessageMngr:* it receives commands from the MessageIntepreter and executes them by invoking methods of the ApplicationServer and, in case, sends the result to the MessageConverter by means of an event. It can be also directly invoked by the ApplicationServer in case the client has to be notified of an event (eg. The farmer has received a prize from a policy maker).

- *SNetworkManager:* it provides the high-level functions to send and receive message over the network and it manages the low-level concerns. It is also able to notify the view when a message comes. It is also in charge of the secure communication (it manages, for instance, cryptography).

2.2.3. Application Server

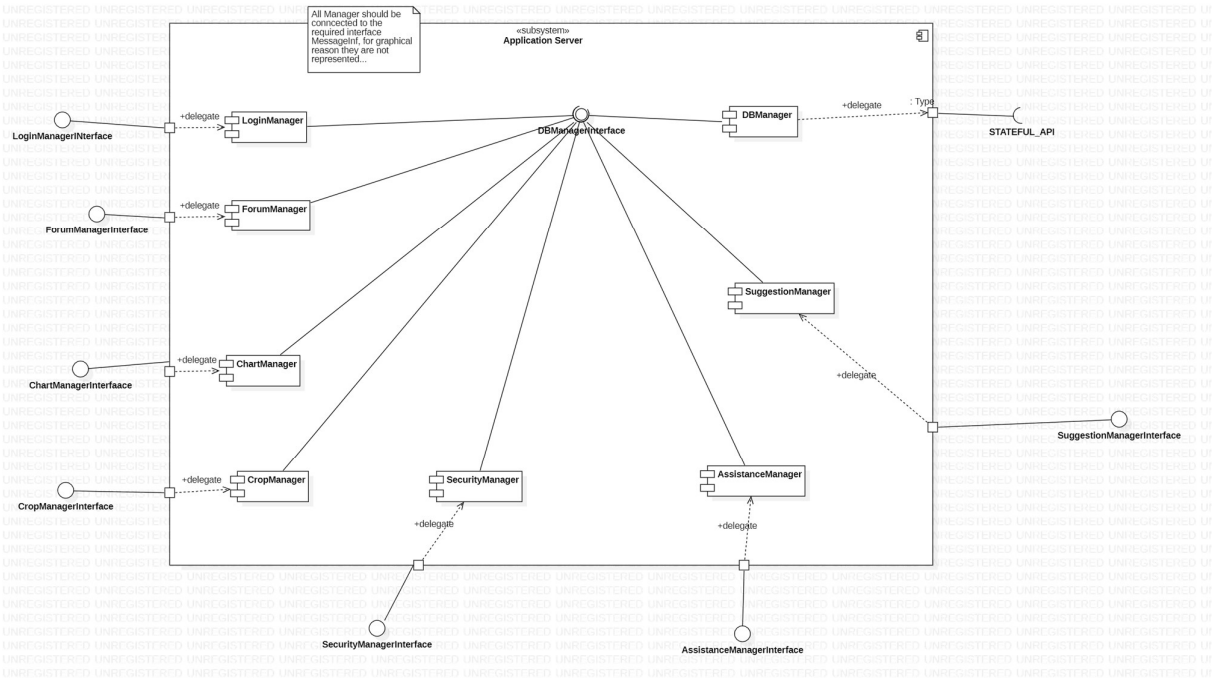


Figure 5: UML component diagram – Application Server

- *LoginManager:* it is the component devoted to all operations related to the management of client personal information, for example registration, login, change of the password, logout. It is also in charge of verifying if the data provided by the users at registration are valid, whether the credentials are correct when logging in by querying the database.

- *ForumManager:* it is the component in charge of the management of contents in the forum like the publication and deletion of comments and posts. It is also in charge of filtering based on affected content and real-time updating of new post of other farmers.

- *ChartManager:* it is the component which consent the visualization and the update of farmer chart based on parameters like harvest and performance during meteorological adverse event. The component is usually used by commands invoked by policy maker, because it provides interfaces which consent the assignment of a prize to a specific farmer.

- *CropManager:* it is the component in charge of all operation related to insertions of data about each farmer crop. It provides different forms where farmers can insert information about daily crop, it updates history and provides interfaces which enable farmers to see the growth trend of their harvests.

- *SecurityManager:* it is the component devoted to the security of transmission of messages between the network and the database. It relies on TLS protocol, and it prevent any SQL injection from external users.

- *DBManager:* it is the component in charge of the interaction between the system and the DBMS. In particular it manages the connection and it is able to formulates query to be executed against the database, starting from the information required by other components. It provides other components with proper interfaces for querying the database and it handles the persistence of data with proper programmatic representation of the tables (e.g. JPA).

- *AssistanceManager:* it is the component in charge of the operation of assistance in case of problems.

- *SuggestionManager: it is the component in charge of the suggestion to the farmer based on his location and his crops*.

2.3. Deployment View

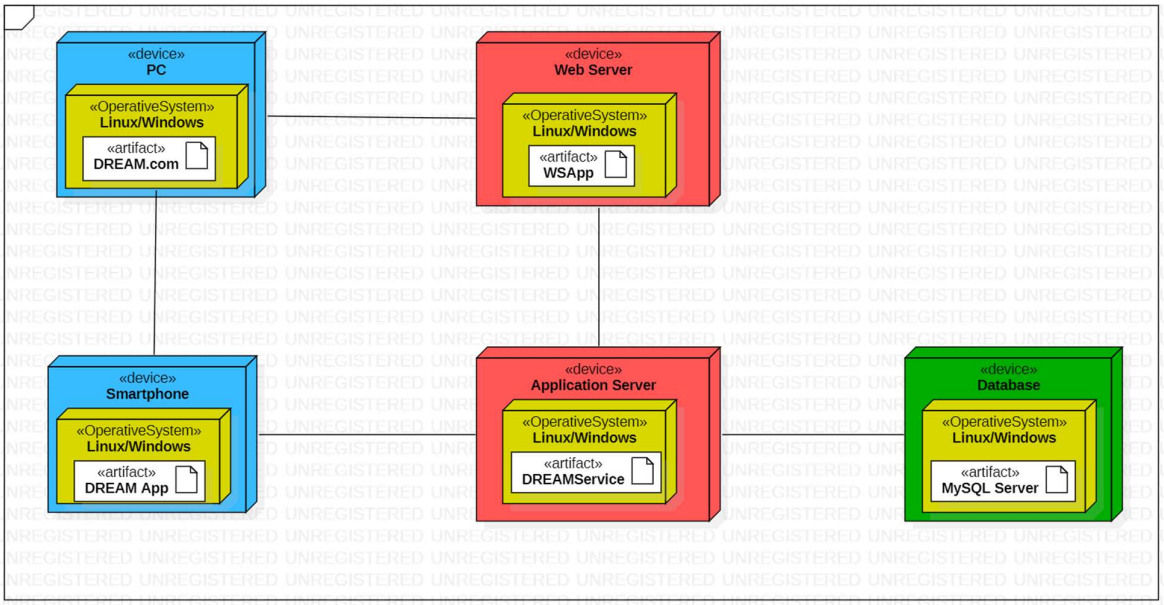


Figure 6: Deployment Diagram

The deployment diagram above shows the components for a correct system behaviour, each of them has its own Operating System:

- Tier 1: This tier is the machine used by the client, it can be both the mobile app downloaded by an app store, or a computer on which the software runs through a web browser.
- Tier 2: This tier contains the web servers, that give the client side an HTML to build the page and, as said before, take the user requests and route them to the application servers.
- Tier 3: This tier includes the application servers, where the whole application layer can be found. It communicates with the data tier through the DBMS gateway and communicates with the client tier thanks to APIs.
- Tier 4: Here there are the DBMS servers, they have the role to store data and maintain them in an efficient way in the database.

2.4. Runtime View

The component diagram gives just a static representation of the components, with their dependencies and their interfaces; in order to better understand how those components work, in this section are exhibited some UML Sequence Diagrams showing the dynamical interaction between components.
Most actions represented are directly inspired from some of the use cases (see RASD), but here the level of abstraction chosen is lower.

2.4.1 Registration

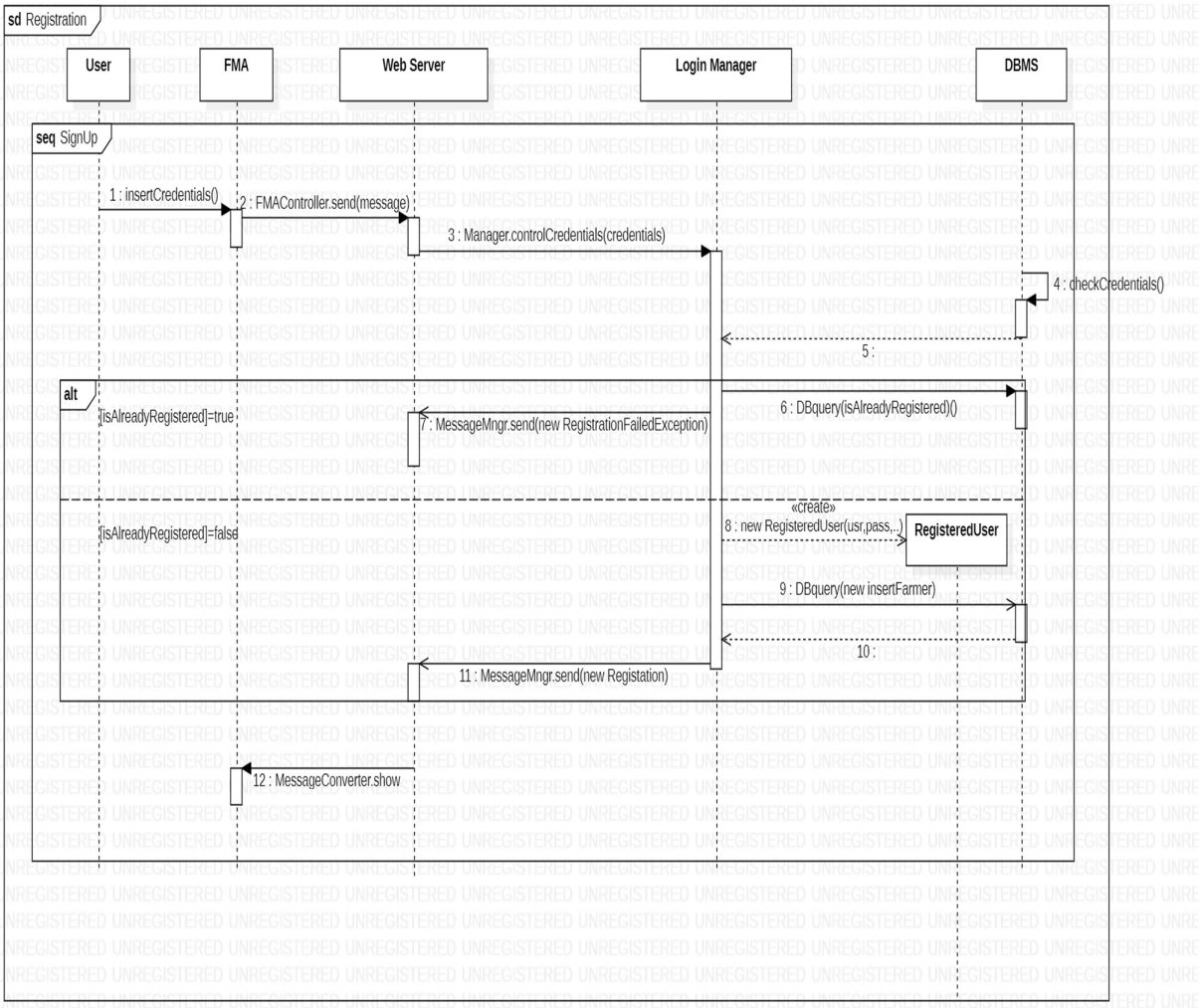


Figure 7: UML Sequence Diagram – Registration

2.4.2 Login

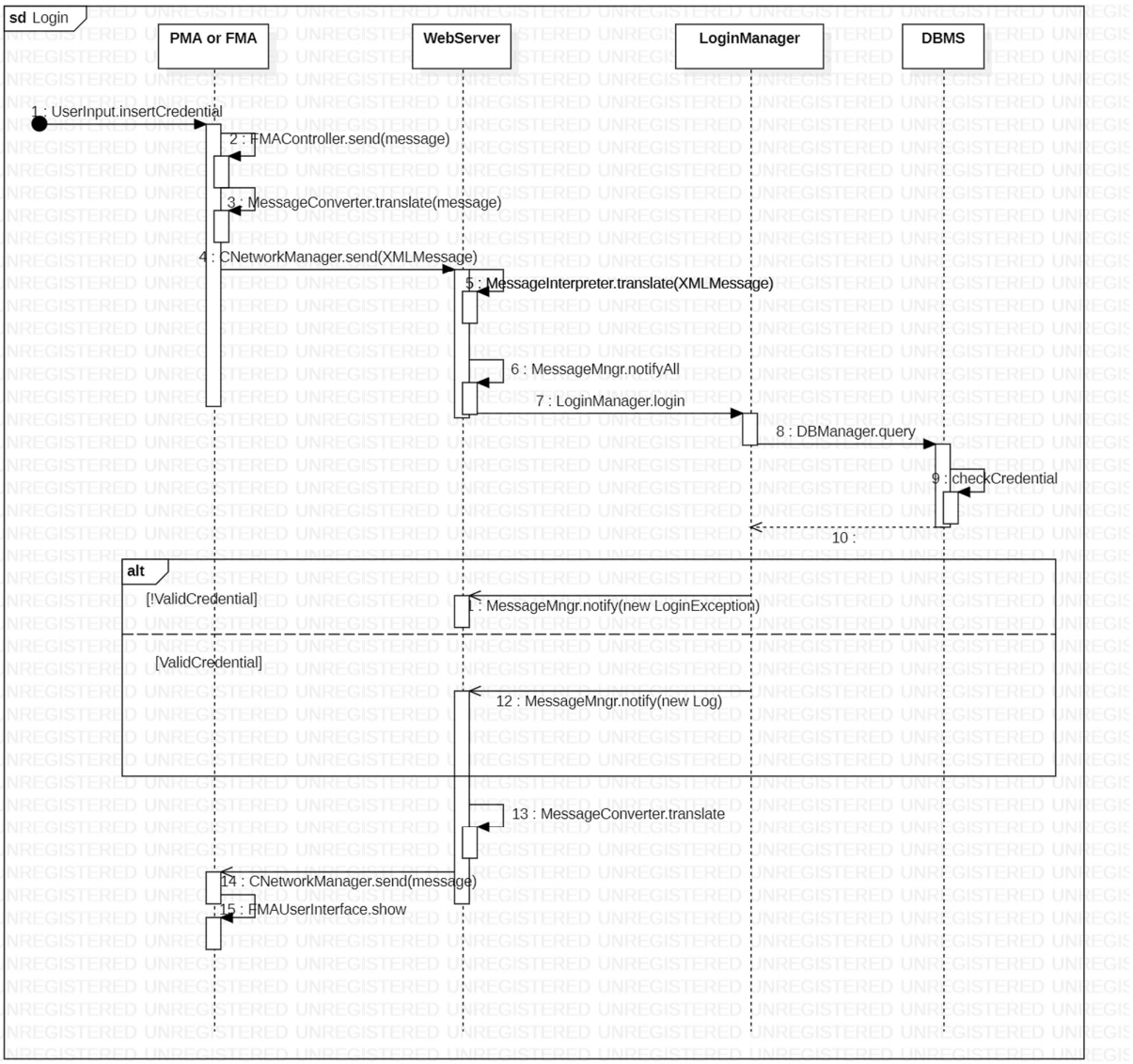


Figure 8: UML Sequence Diagram – Login

2.4.3 Data Insertion

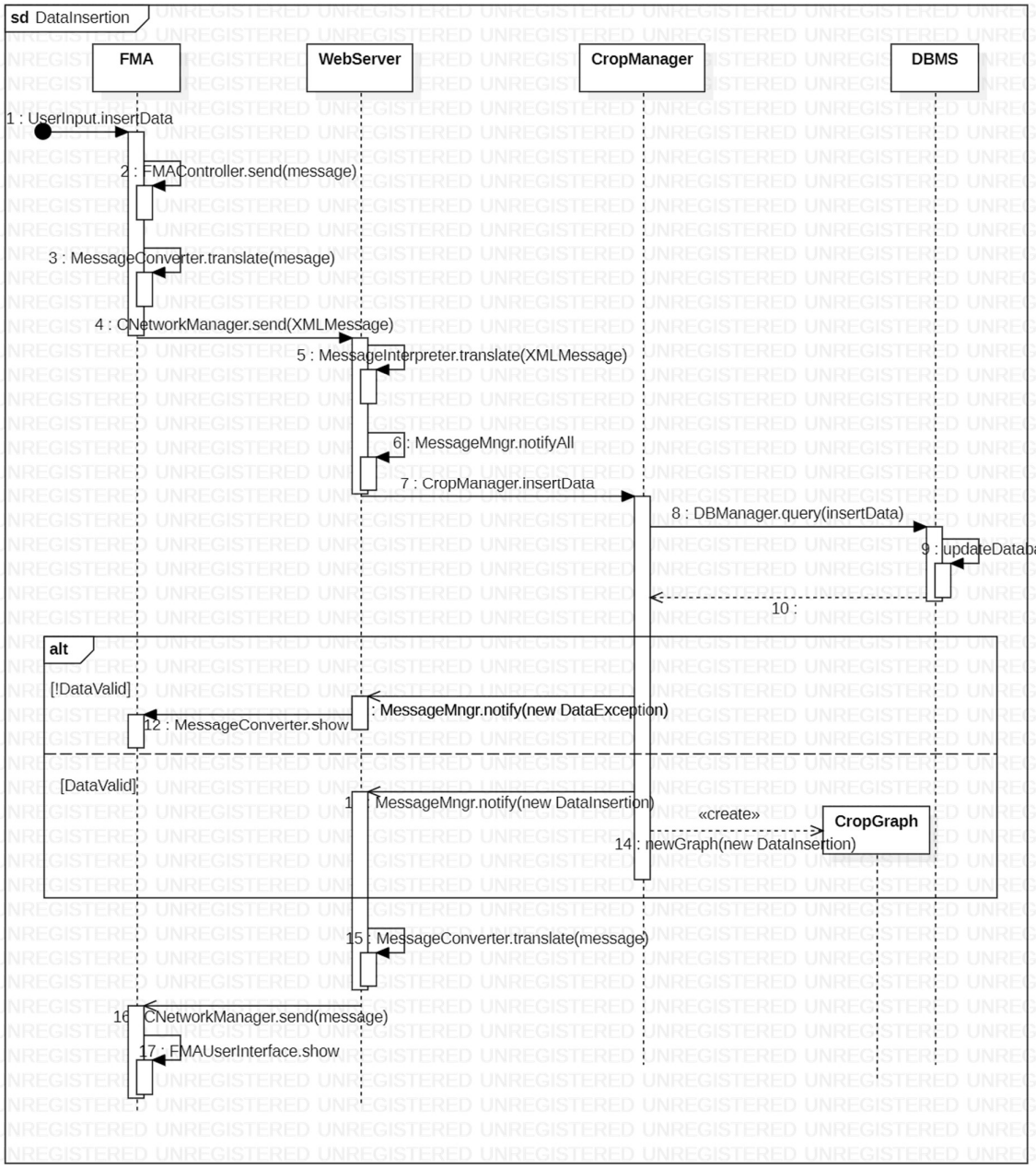


Figure 9: UML Sequence Diagram – Data Insertion

2.4.4 New discussion or new comment in the forum

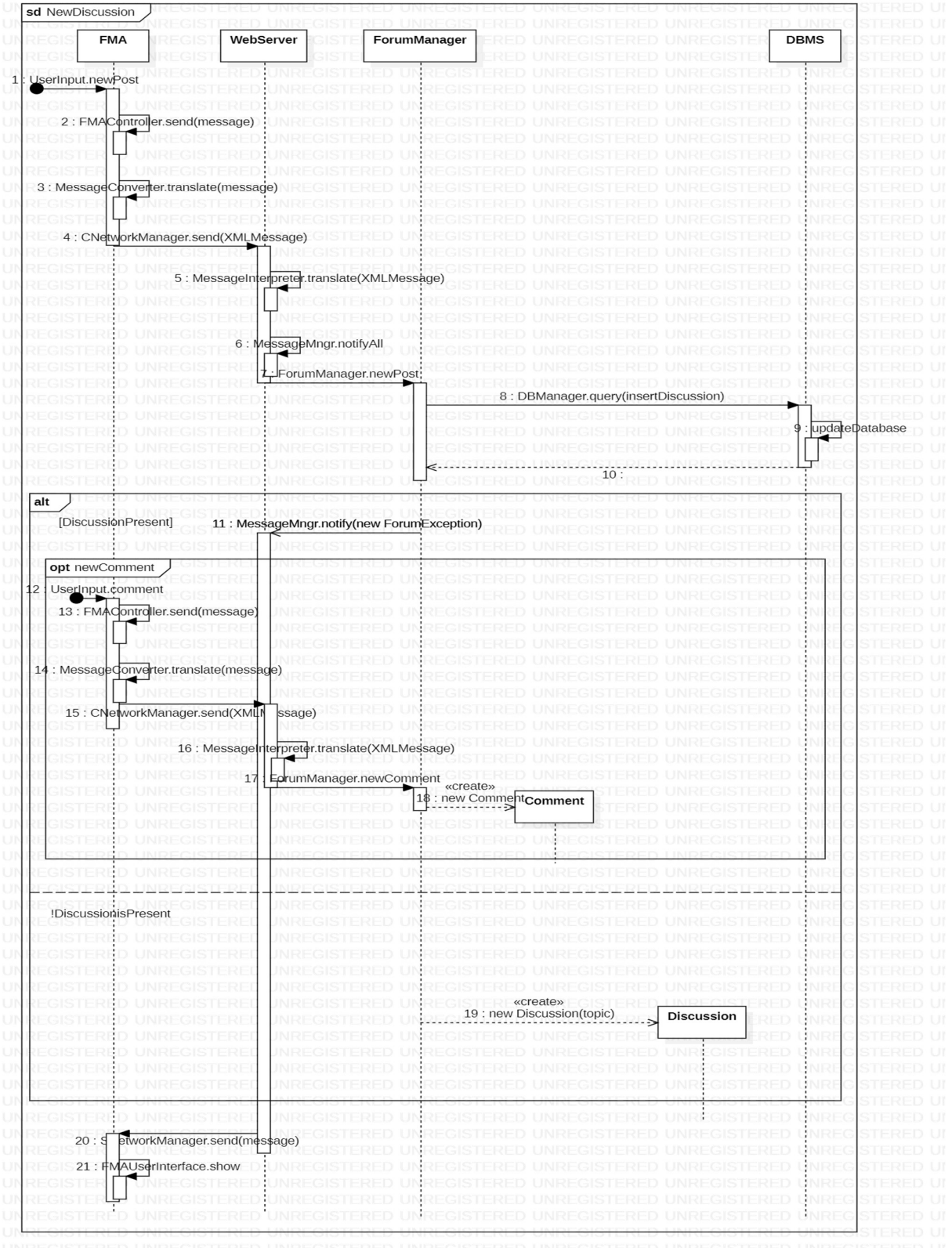


Figure 10: UML Sequence Diagram – Forum

2.4.5 Prize recognition to a farmer

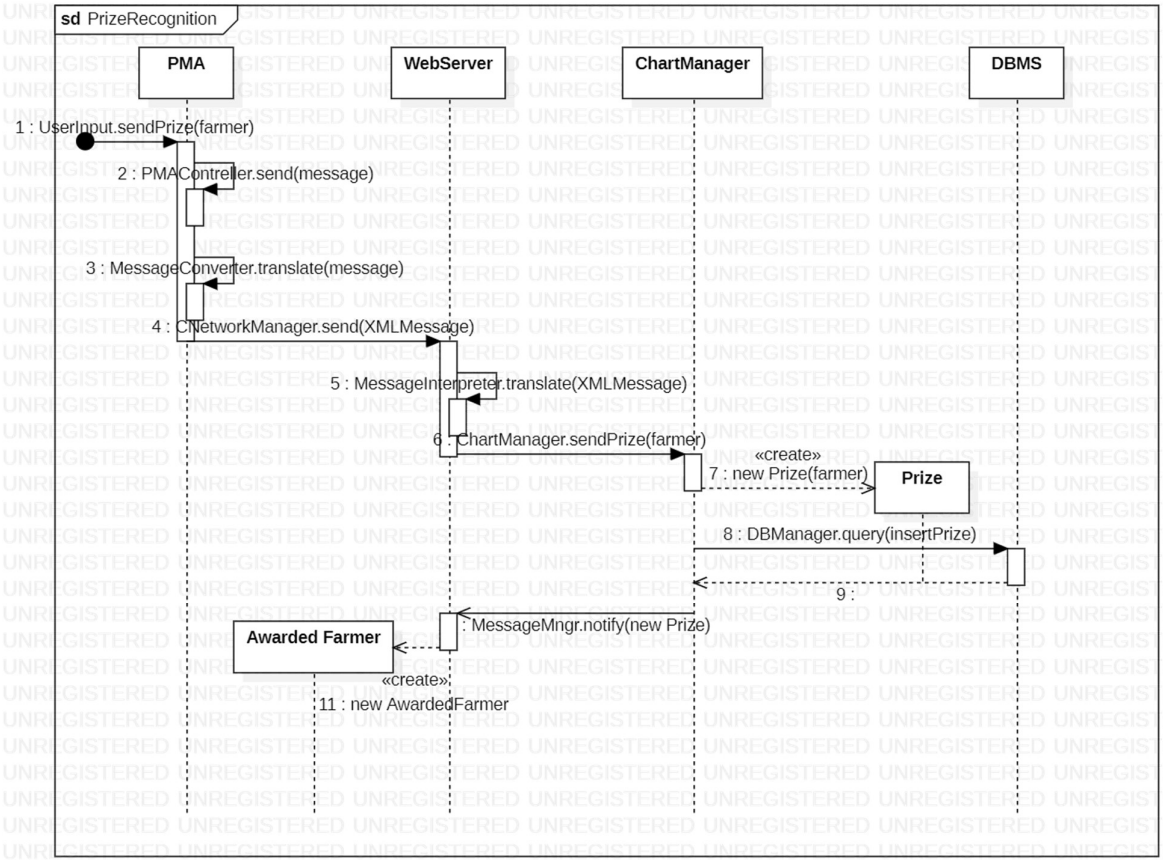


Figure 11: UML Sequence Diagram – Prize Recognition

2.4.6 Suggestion to the Farmer

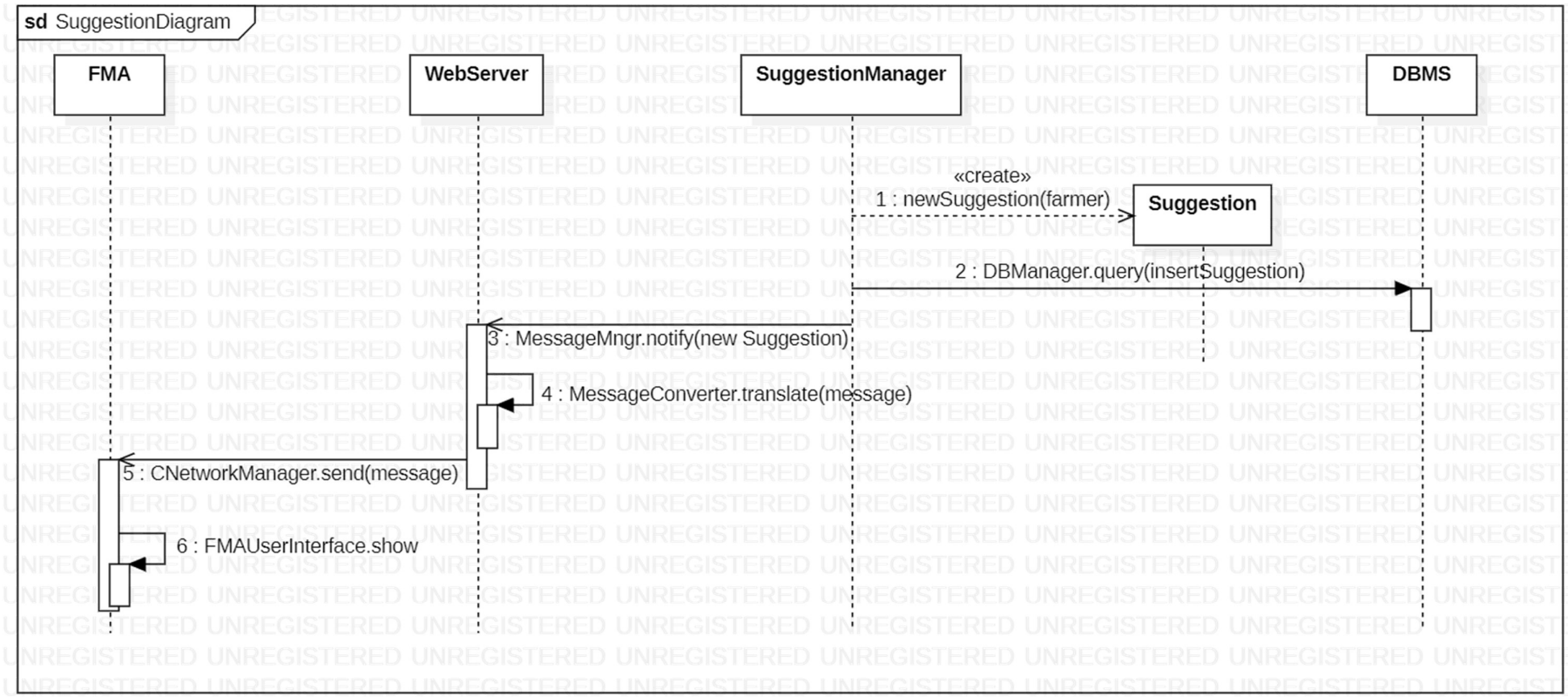


Figure 12: UML Sequence Diagram – Suggestion to a farmer

2.5 Component interfaces

Application Server

Component	Interface	Methods
LoginManager	LoginManagerInterface	login (email: String, password: String): RegisteredUser register(name:String, lastname: String,email: String, password: String, role: Role,location: Location): RegisteredUser forgotPassword(email: String)
ForumManager	ForumManagerInterface	newDiscussion (user: RegisteredFarmer, topic: Topic): Discussion newComment(user: RegisteredFarmer, topic: Topic,discussion: Discussion, text: String): Comment deleteComment(comment: Comment) deleteDiscussion(discussion: Discussion) selectTopic(user: RegisteredFarmer, topic: Topic): Discussion[] selectDiscussion(user: RegisteredFarmer, topic: Topic, discussion: Discussion): Discussion
ChartManager	ChartManagerInterface	selectChartTime(user: RegisteredPolicyMaker,timeframe: TimeFrame): FarmerChart selectFarmer(user: RegisteredPolicyMaker, chart: FarmersChart): PrizeNotification updateChart(users: RegisteredFarmer[]):FarmersChart
CropManager	CropManagerInterface	selectCropGraph(user: RegisteredFarmer, crop:Crop,time:TimeFrame):CropGraph updateGraph(user: RegisteredFarmer, crop: Crop, cropHarvest:Int)
SecurityManager	SecurityManagerInterface	encrypt(message: Message) decrypt(message: Message)
AssistanceManager	AssistanceManagerInf	recovery(user : RegisterUser) sendHelp(farmer: RegisteredFarmer)
DBManager	DBManagerInterface	query(query:Query):Object()
SuggestionManager	SuggMngrInf	giveSuggestion(farmer:RegisteredFarmer) : Suggestion

Web Server

Component	Interface	Methods
SNetworkManager	MNGInterface NTWRInterface	submit(message: Message) send(message: Message)
MessageConverter	MessageConInterface	convert(event: Event): Message
MessageManager	MessageManagerInterface	send(message: Message) notify(event: Event, user: RegisteredUser): Message notifyAll(event: Event): message
MessageInterpreter	MessageInterpreterInterface	interpretMessage(message: Message): Command

FMA

Component	Interface	Methods
FMAUserInterface	FMAUserInterfaceInf NotificationManagerInterface	show(message: XMLMessage)
FMAController	FMAControllerInf	sendCommand(command: Command)
MessageConverter	MessageConverterInf	convert(command: Command): Message
CNetworkManager	NetworkInf NetworkMGRInf	notify(event: XMLEvent, user: RegisteredUser): Message send(message: XMLMessage)

PMA

Component	Interface	Methods
FMAUserInterface	FMAUserInterfaceInf	show(message: XMLMessage)
FMAController	FMAControllerInf	sendCommand(command: Command)
MessageConverter	MessageConverterInf	convert(command: Command): Message
CNetworkManager	NetworkInf NetworkMGRInf	notify(event: XMLEvent, user: RegisteredUser): Message send(message: XMLMessage)

2.6 Selected architectural styles and patterns

2.6.1 Four Tiers

The decision to use a four-tiered architecture comes from different reasons; first an application divided on different tiers allows a maximization of the performance, and on the other hand a minimization of the costs, thanks to a scaling approach used only in components that are more problematic.
Different tiers give the chance to divide the load on different machines, so that an overload on the system is less probable.
Finally, each module is defined not depending by the others, so the overall system is flexible, each part can be improved without touching the others.

2.6.2 Stateful Principle

Since a user can do more than one action in a session, the system has to memorize the state of the user, instead of creating a new session from zero for each action.
That is why the system will be a stateful application.

2.6.3 Model-View-Controller

MVC is a software designed pattern used to develop user interfaces dividing the logic into three elements. The three elements are:

- *Model*: it represents enterprise data and the business rules that govern access to and updates of this data. Often the model serves as a software approximation to a real-world process, so simple real-world modelling techniques apply when defining the model.

- *View*: renders the contents of a model. It accesses enterprise data through the model and specifies how that data should be presented. It is the view's responsibility to maintain consistency in its presentation when the model changes. This can be achieved by using a push model, where the view registers itself with the model for change notifications, or a pull model, where the view is responsible for calling the model when it needs to retrieve the most current data. The controller must never intervene directly on the graphic elements.

- *Controller*: translates interactions with the view into actions to be performed by the model.

One advantage of the division is that the model is centralized and the other components interact with it, but through the different interfaces.

Another advantage of the MVC is the possibility to develop in parallel the different components of the application, making the implementation faster.

The MVC pattern helps to break up the business layer from frontend code into separate components. This way, it's much easier to manage and make changes to either side without them interfering with each other.

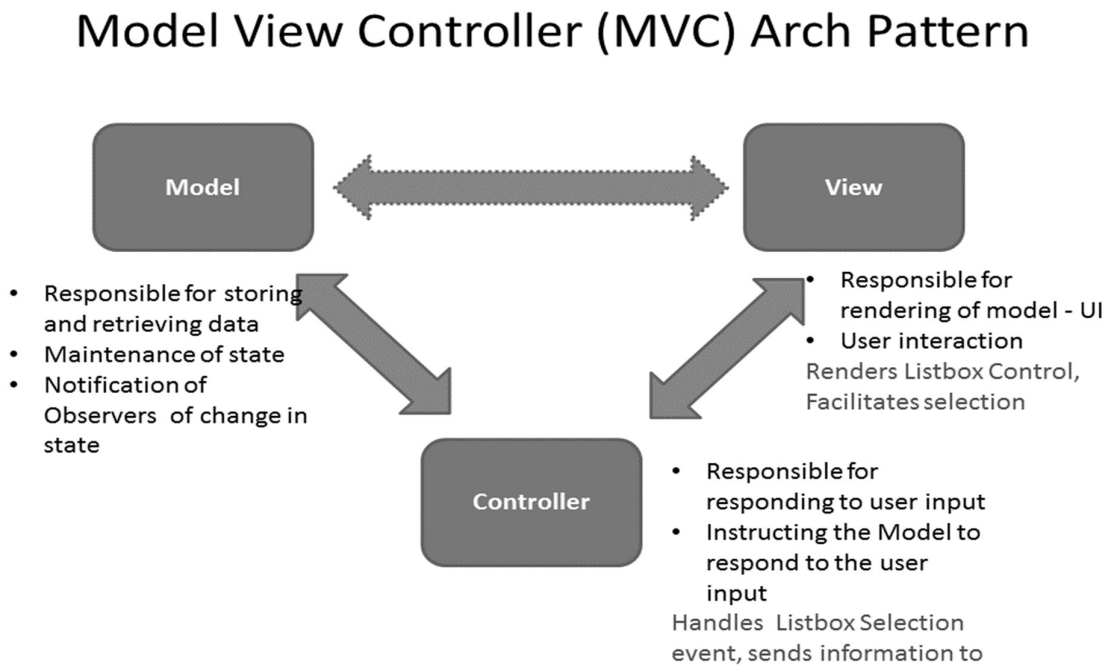


Figure 13: Model View Controller Pattern

2.7 Other design decision

2.7.1 Client And Server with Thin Client

The clients (web and app) are thin clients.

The thin client choice is made to assure a stable connection and to make it possible to the app to work also on devices even without a high computational power, since it will be a light application. This is made by leaving all the business logic on the server.

Due to this decision, it could be interesting to see some cloud computing to support the deployment phase.

The model used is a client-server, since the server is a centralized system with a computational power that needs to be a lot higher than the clients.

To make the application stabler, it is introduced a parallelization of the server on different tiers.

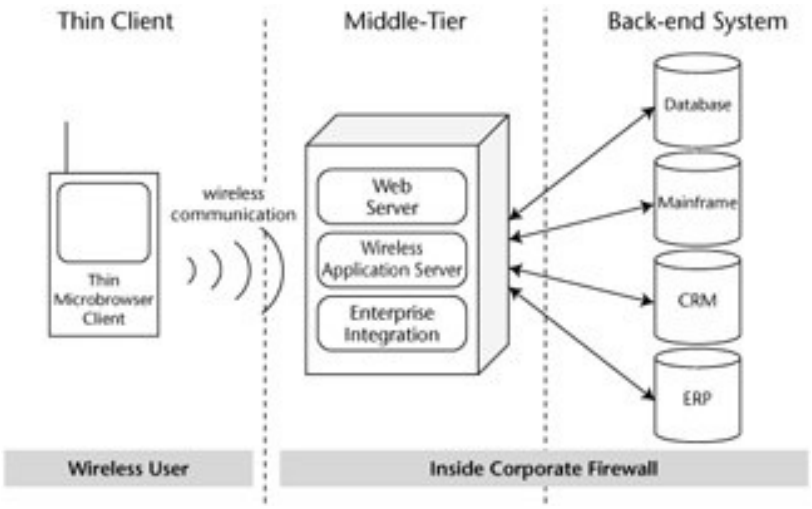


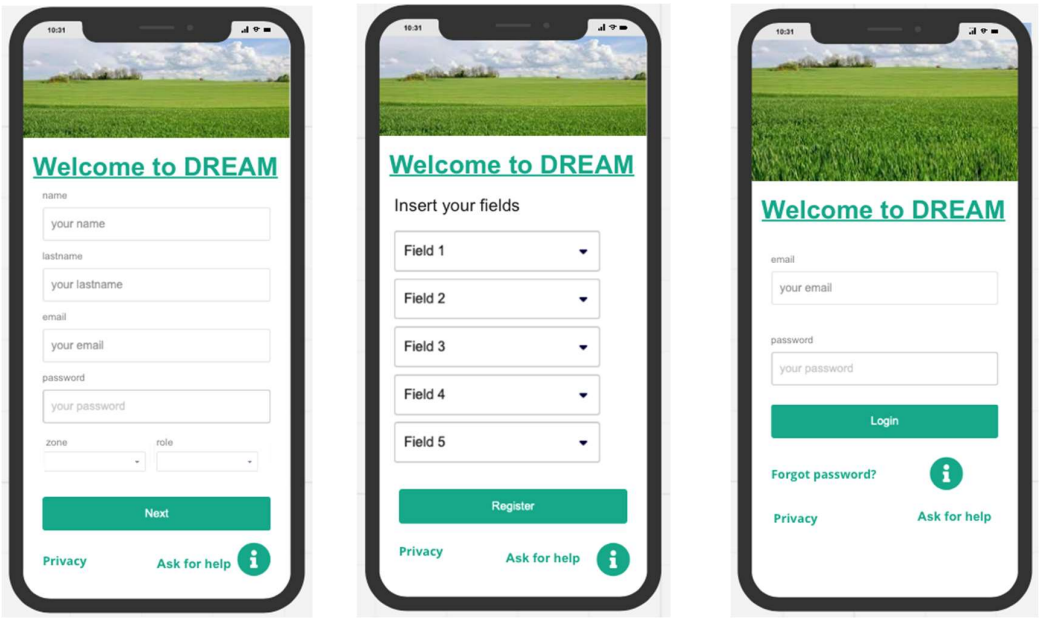
Figure 14: Thin Client Architecture

3. USER INTERFACE DESIGN

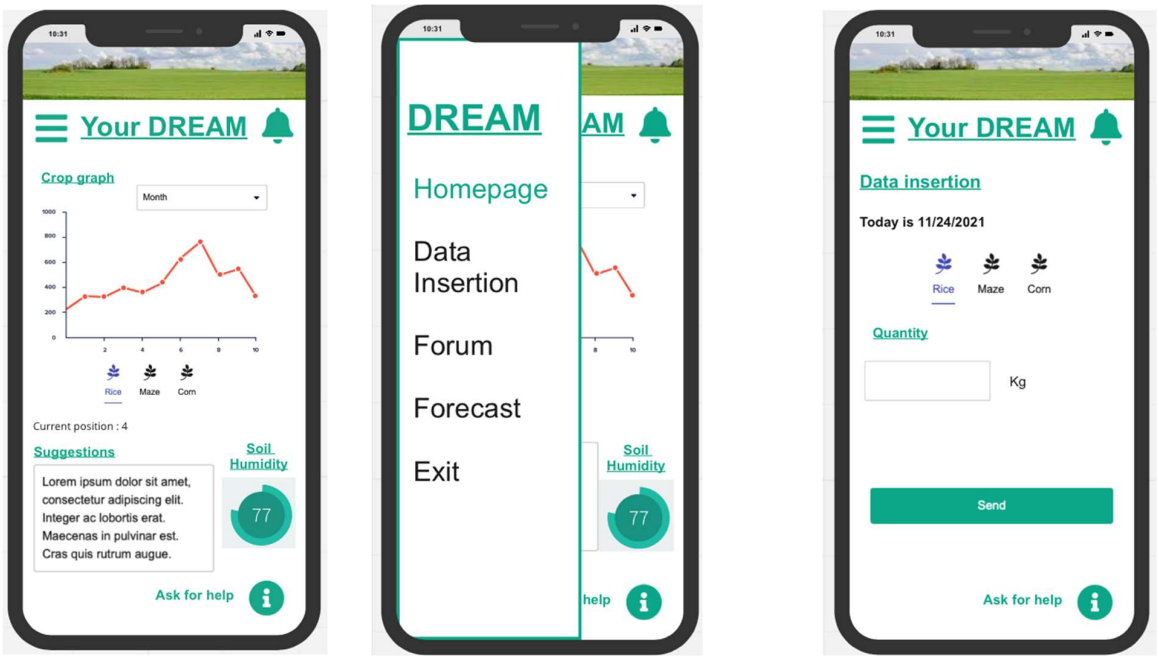
3.1 User interface mock-ups

3.1.1 FMA - Mobile App

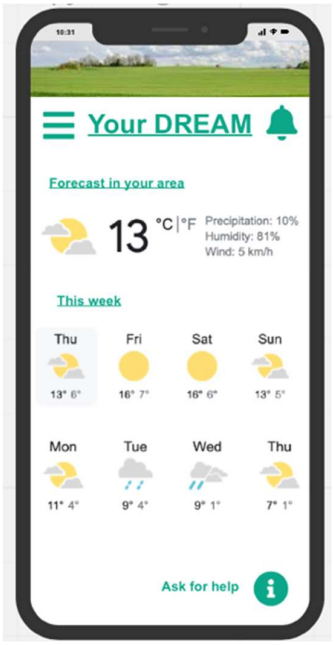
3.1.1.1 Registration and Login

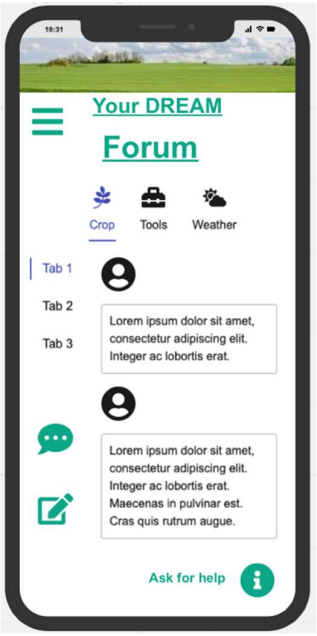


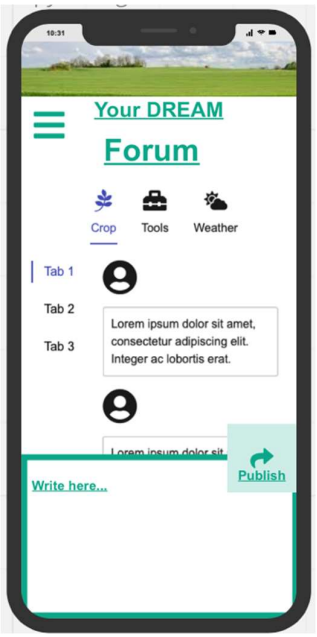
3.1.1.2 Farmer's homepage & menu / Data Insertion



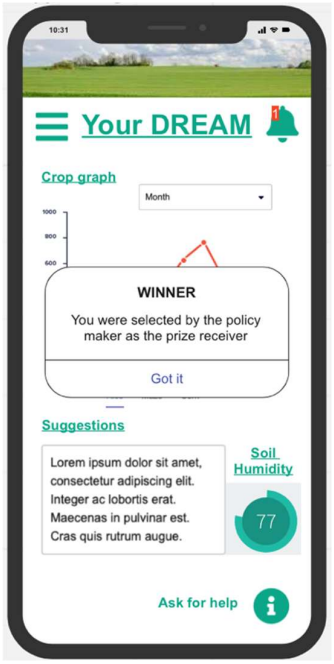
3.1.1.3 Forecast & Forum Page

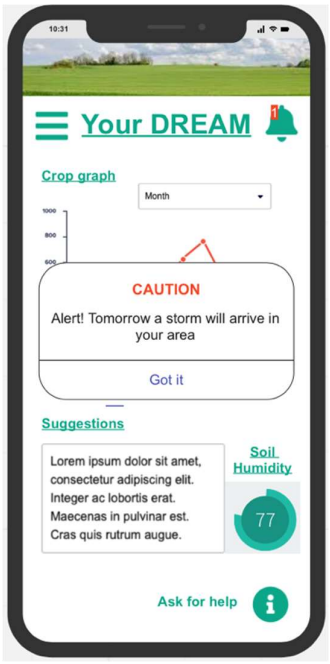






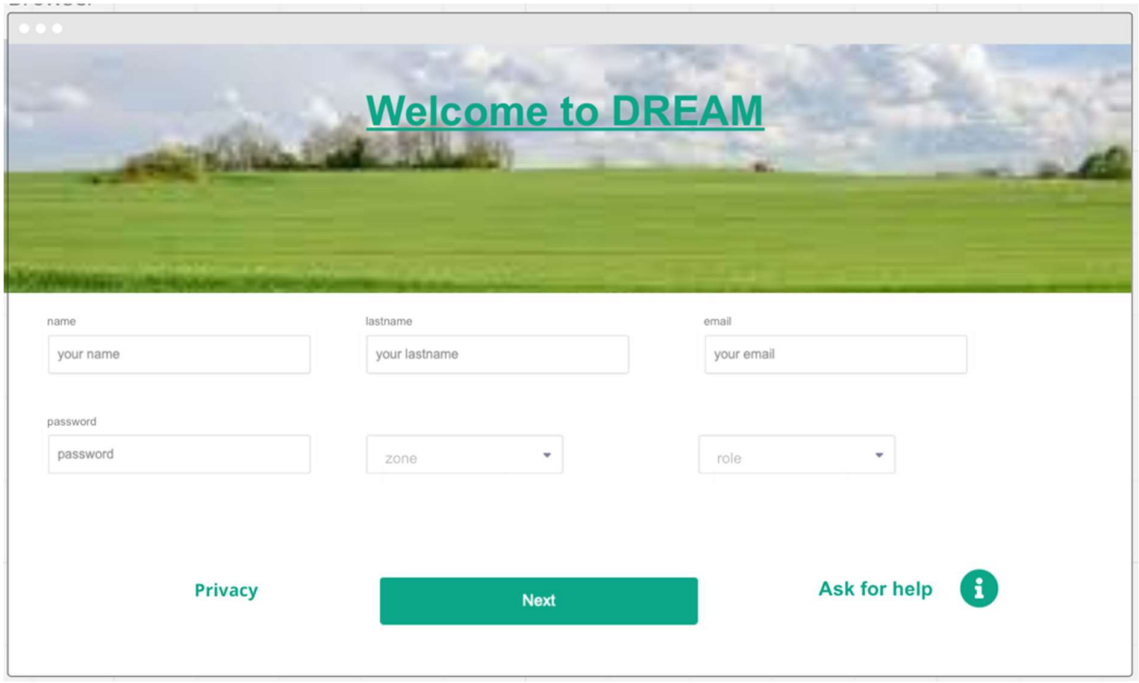
3.1.1.4 Notification of victory

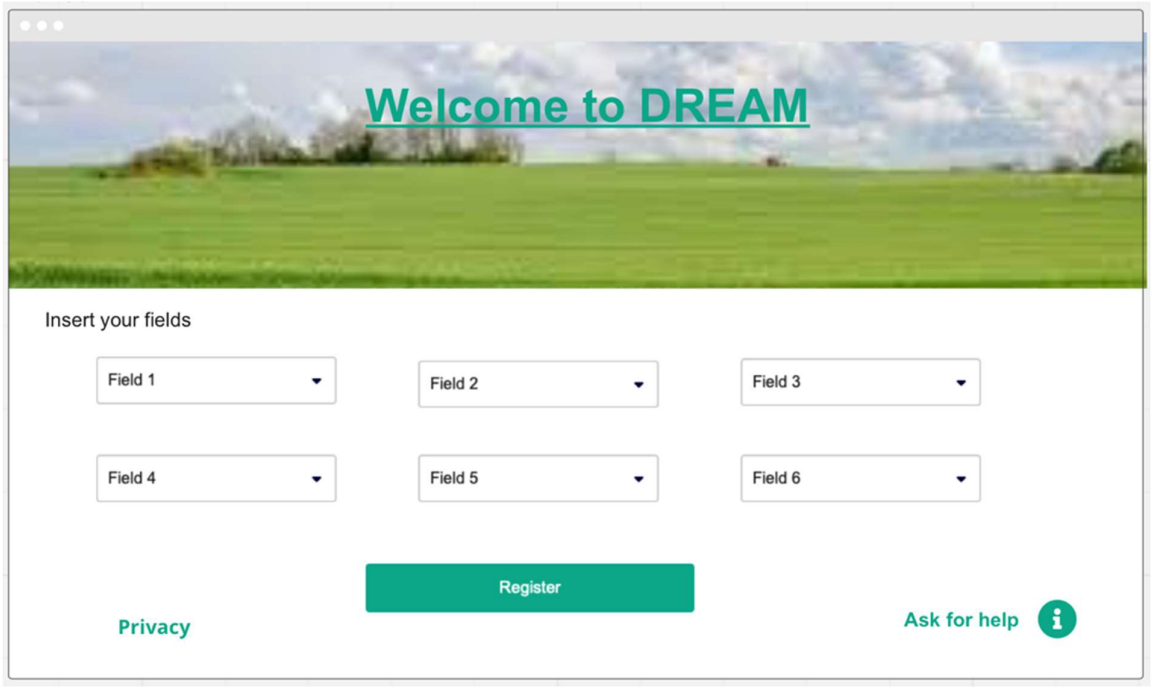


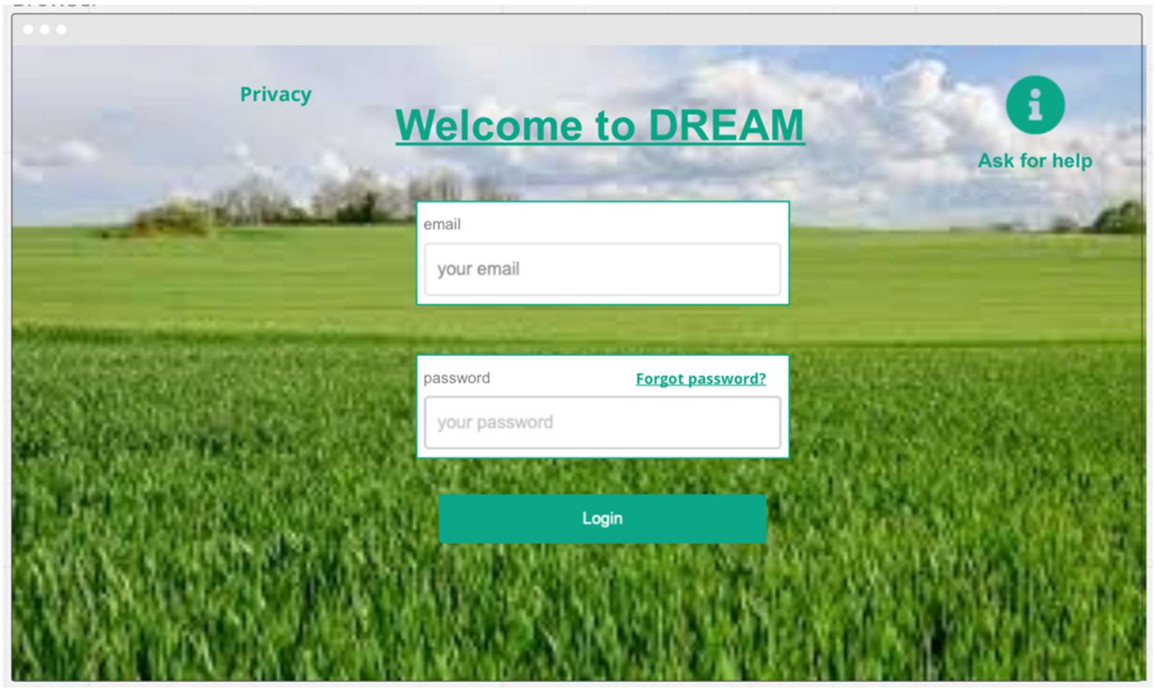


3.1.2 FMA - Web App

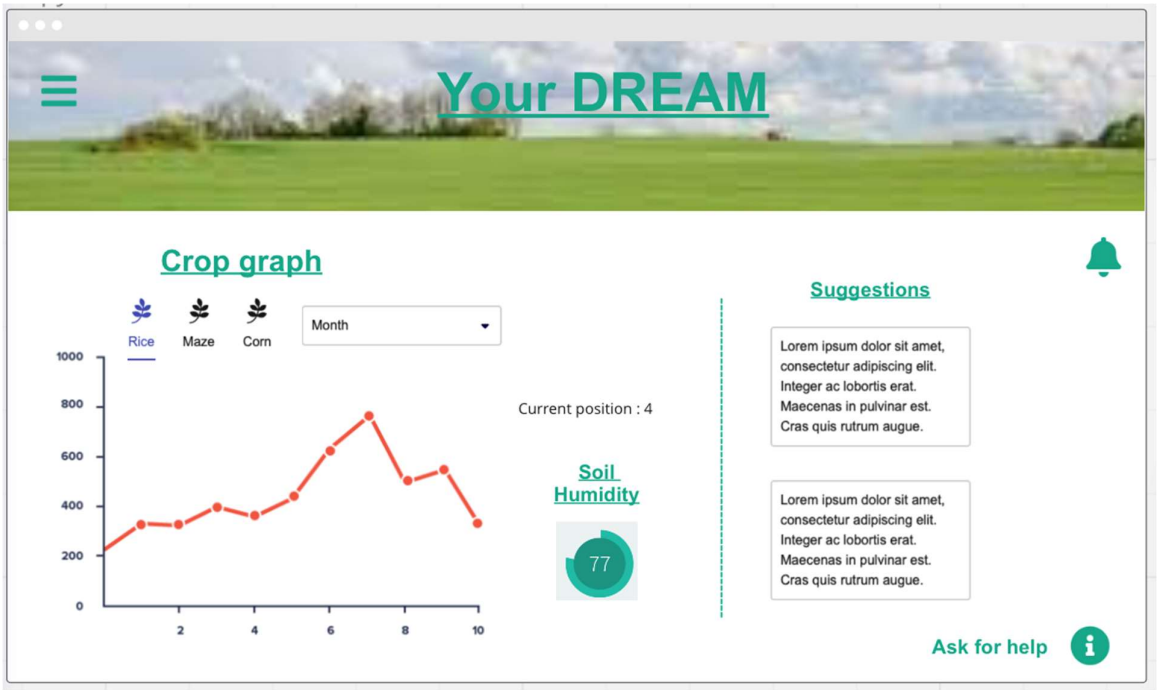
3.1.2.1 Registration and Login





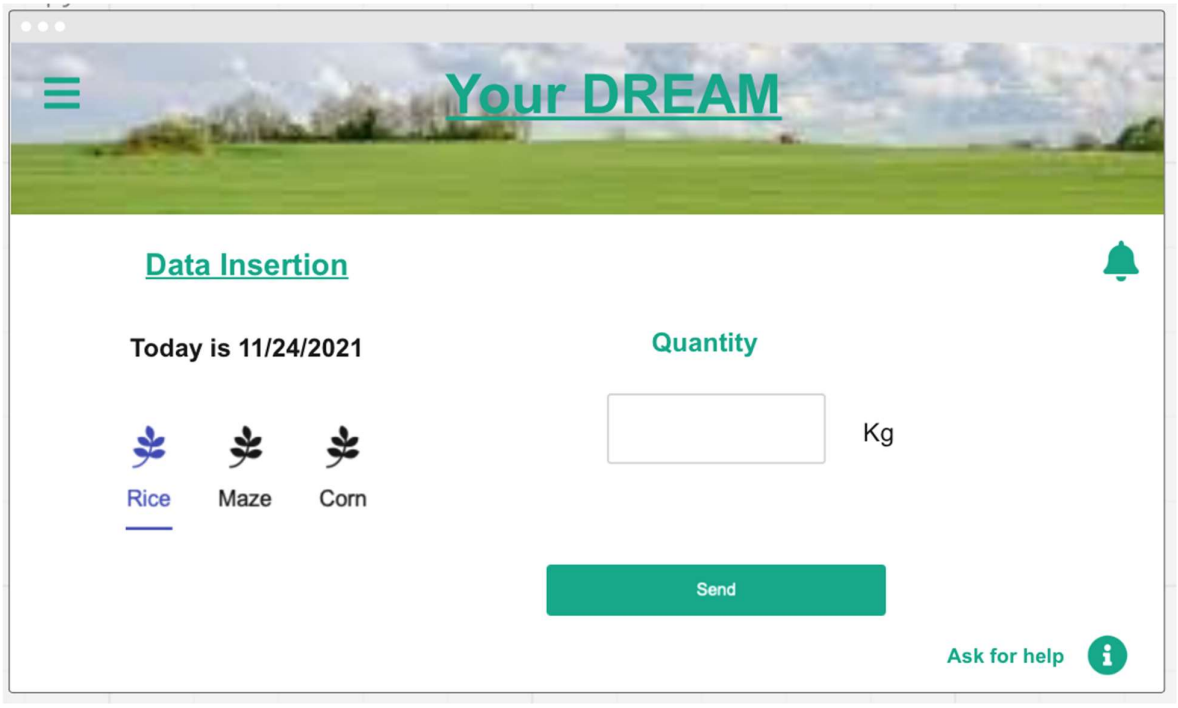


3.1.2.2. Farmer home page

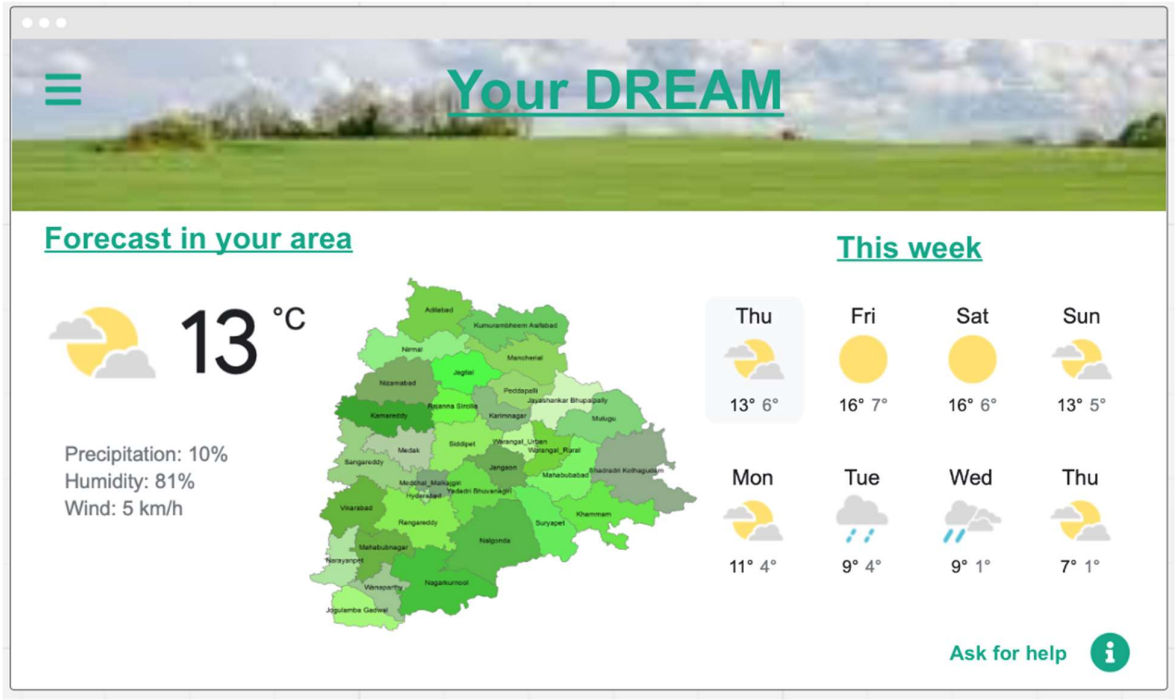


3.1.2.3. Menu

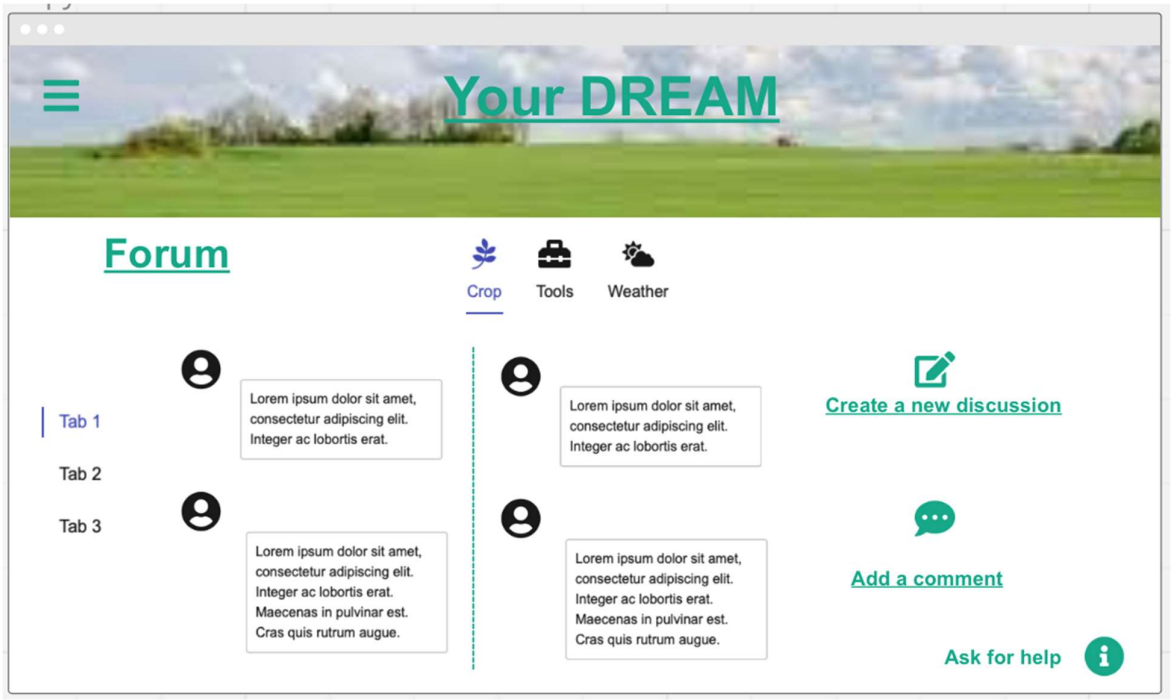
3.1.2.4. Data Insertion

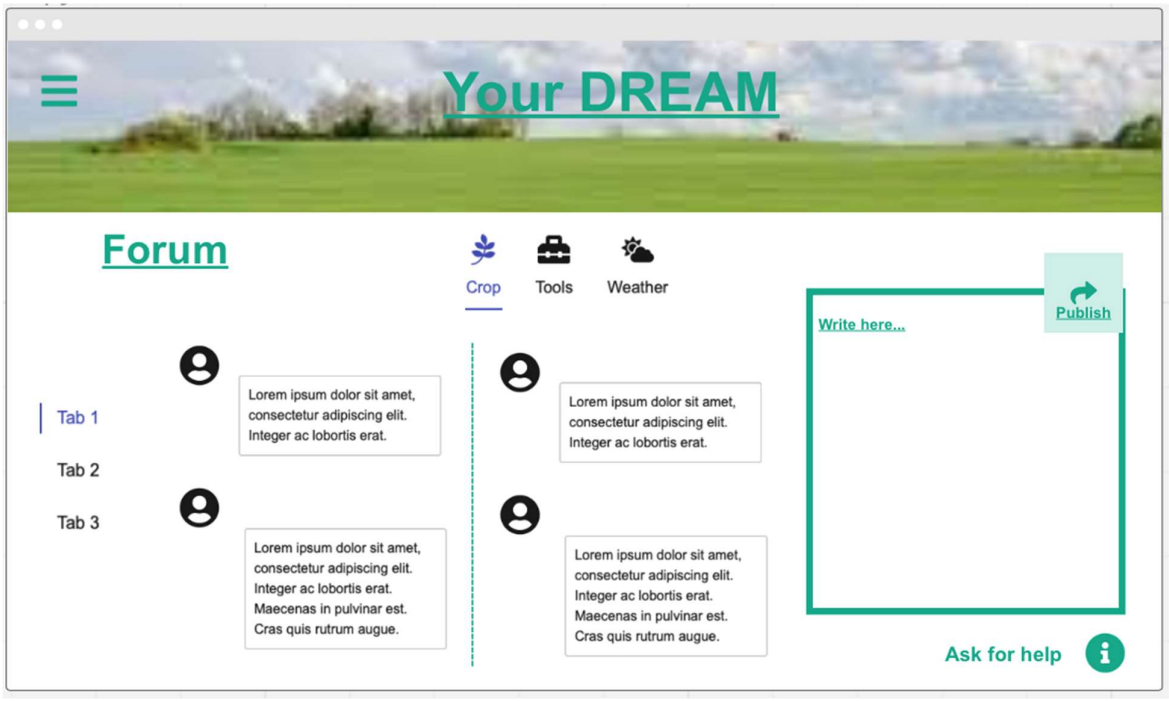


3.1.2.5. Forecast

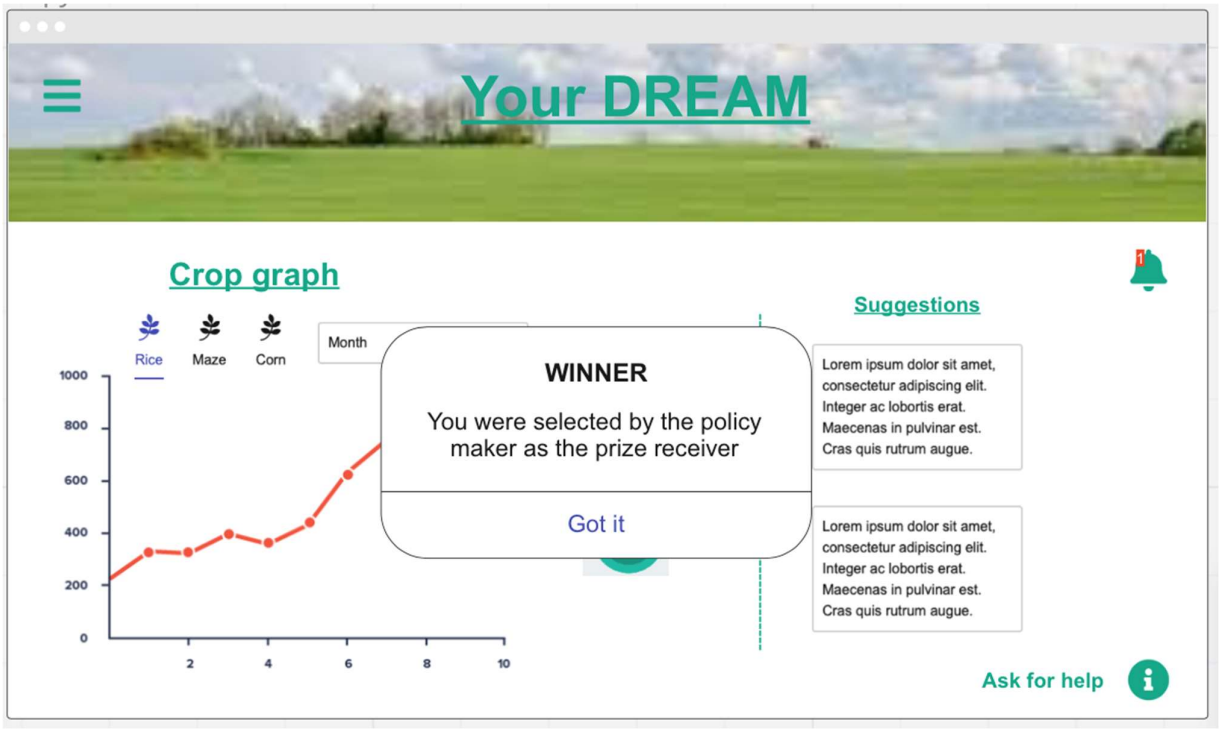


3.1.2.6. Forum reading and writing





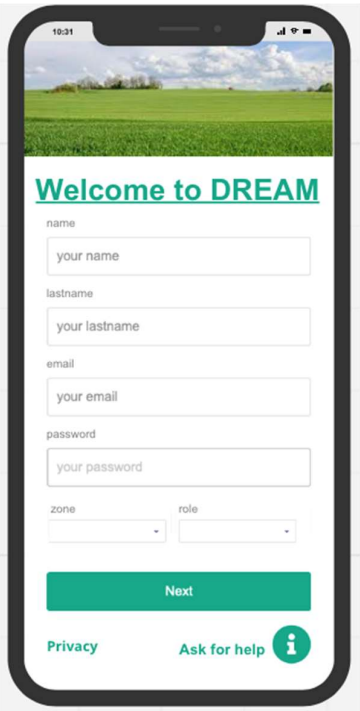
3.1.2.7. Notification of victory

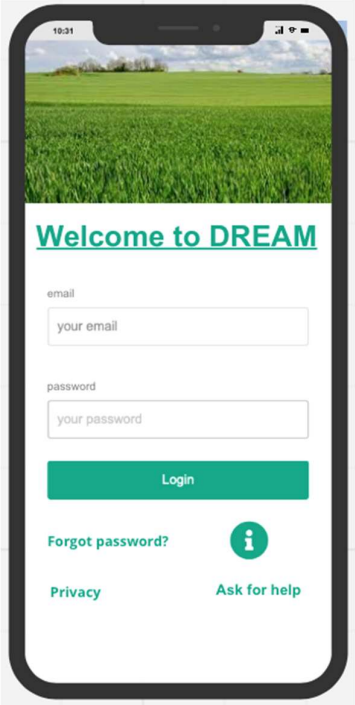


3.1.2.8. Notification of bad weather

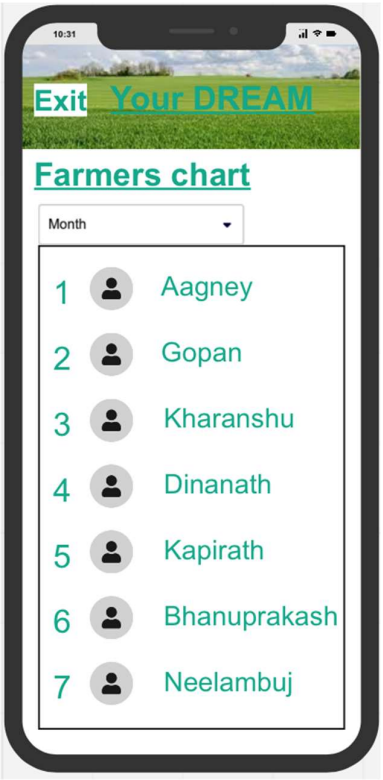
3.1.3. PMA - Mobile App

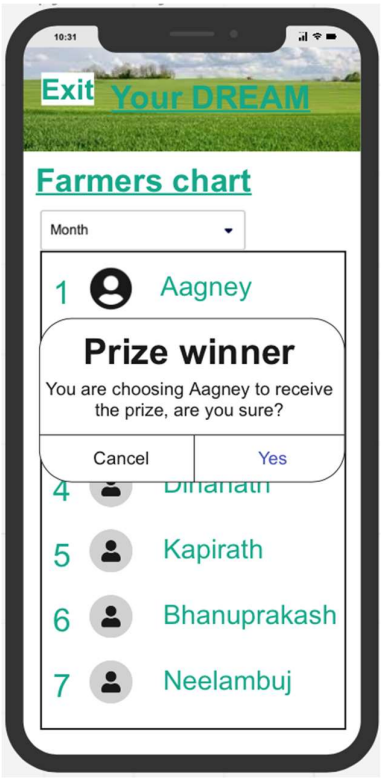
3.1.3.1. Login and registration





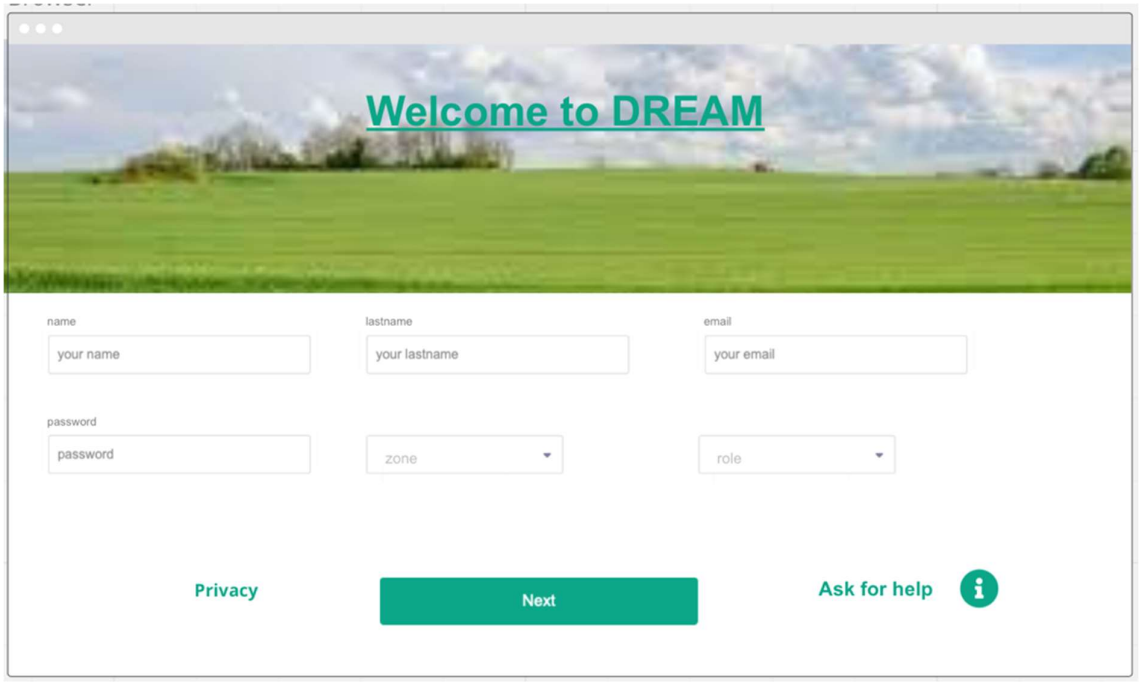
3.1.3.2. Farmers chart & Winner choice

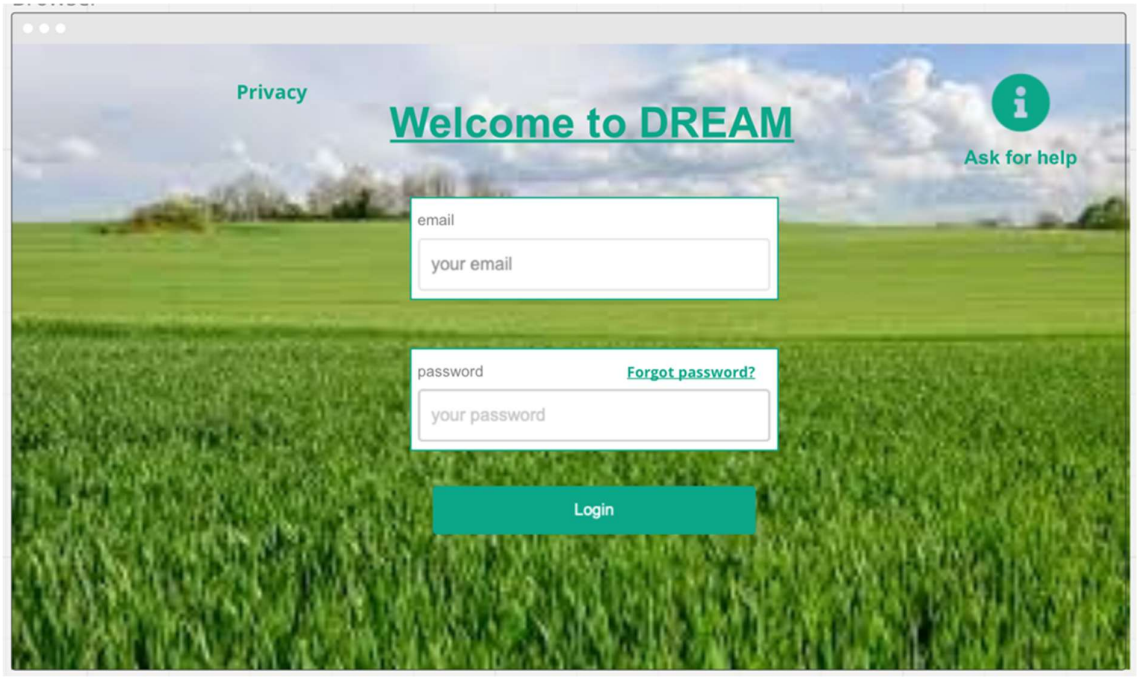




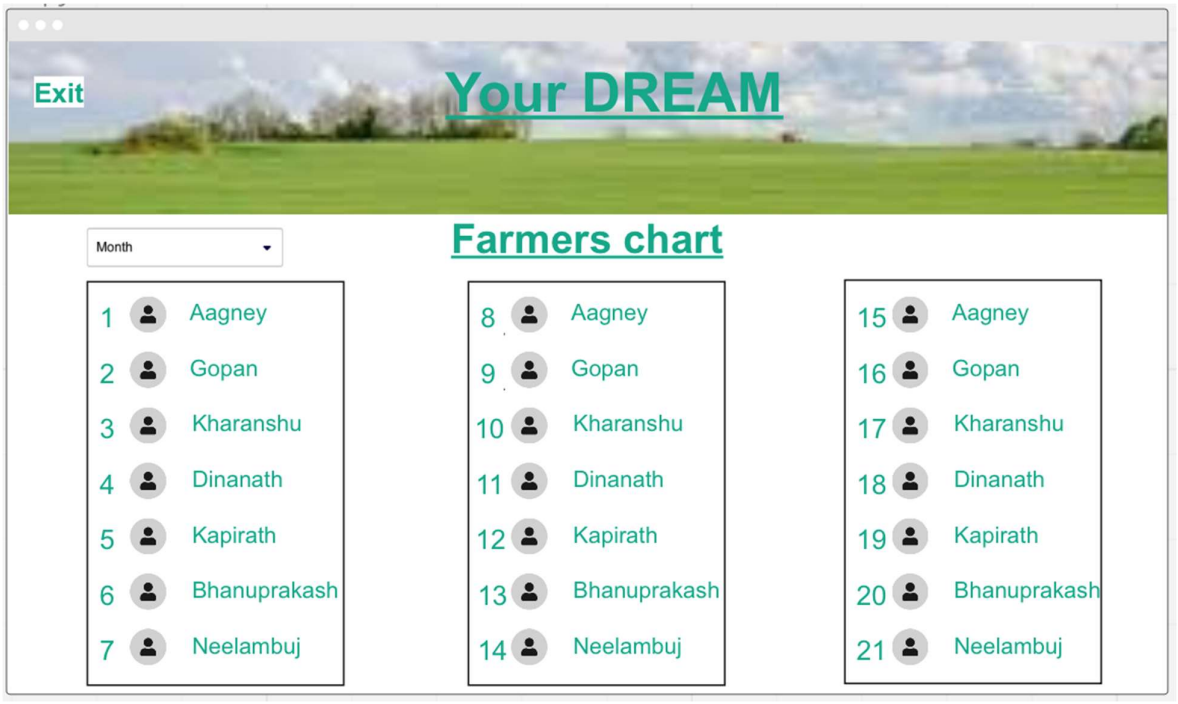
3.1.4. PMA - Web App

3.1.4.1. Login and registration

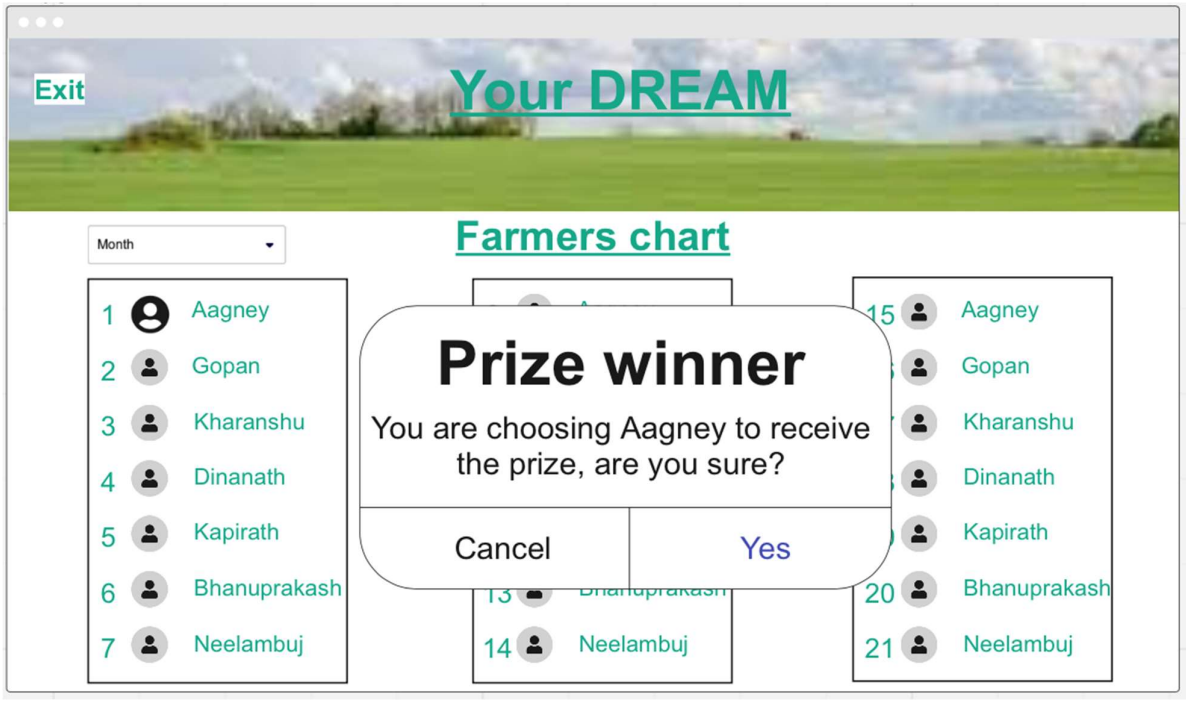




3.1.4.2. Farmers chart



3.1.4.3. Winner choice



4. REQUIREMENTS TRACEABILITY

4.1. Functional Requirement

Requirement / Component	Login Manager	Forum Manager	Chart Manager	Crop Manager	Security Manager	Assistance Manager	DB Manager	Suggestion Manager	SNetworkManager	Message Converter	Message Manager	Message Interpreter	PMA/FMAUser Interface	PMA/FMAController	Message Converter	CNetworkManager
R1.1	X				X		X		X	X	X	X	X	X	X	X
R1.2	X				X		X		X	X	X	X	X	X	X	X
R1.3					X				X				X			
R1.4	X				X		X		X						X	X
R1.5					X	X			X							X
R1.6					X			X	X							X
R2.1					X				X				X			X
R2.2					X				X				X			X
R3.1		X			X		X		X							X
R3.2		X			X		X		X	X	X	X		X	X	X
R3.3		X			X		X		X							X
R4.1			X		X		X		X	X	X	X		X	X	X
R4.2			X		X				X				X			X
R4.3			X		X		X		X	X	X	X		X	X	X
R5.1					X				X				X			X
R5.2					X				X				X			X
R5.3					X				X		X		X			X
R6.1					X				X		X		X			X
R6.2					X				X				X			X
R7.1				X	X		X		X	X		X	X	X	X	X
R7.2				X	X		X		X							X
R8.1					X	X			X	X	X	X	X			X
R8.2					X	X			X	X	X	X	X	X	X	X

4.2. Non-functional requirement

Requirement / Component	Login Manager	Forum Manager	Chart Manager	Crop Man-ager	Security Manager	Assistance Manager	Suggestion Manager	DB Man-ager	SNetwork-Manager	Message Converter	Message Manager	Message Interpreter	PMA/FMA User Inter-face	PMA/FMA Controller	Message Converter	CNetwork-Manager
Reliability	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Availability						X		X	X	X	X	X	X	X	X	X
Security	X	X	X	X	X	X	X	X					X			X
Maintainability	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X
Portability													X			

5. IMPLEMENTATION, INTEGRATION AND TEST PLAN

The DREAM software will be formed by:

- client (web and mobile)
- web server
- app server
- external services (forecast website, sensors website)
- database

The logic that will be used to implement them is going to be a bottom up one; this to start with easier things to develop and avoid any complication in the implementation of greater structures.
The levels of difficulty for the implementation are shown in the following scheme, where can be also found the level of importance of the function for the users.

5.1. Implementation and Testing precedencies

Functionality	Importance for customer	Difficulty
Sign Up & Login	High	Low
Data Insertion	High	Medium
Chart	High	High
Forum	Medium	Medium
Forecast	Medium	Low
Notification	Low	High
Suggestion	Low	Medium

5.2. Functionalities

The first component to be implemented is going to be the database, since all the functions are based on it.

Then the implementation will move on in order of importance and usefulness.

- **SignUp and Login**:

IMPLEMENTATION	TESTING
This is the first step for all the users, obviously the signUp part will be implemented first and all the data will be saved in DB. The main components needed are the LoginManager and DBManager.	The testing part will focus on how data are saved in the database, checking the congruency for the login and it also will be tested the function of password recovery. For the signUp, it will be checked if the system can recognize if a user is already registered and the correct saving of the data.

- **Data Insertion:**

IMPLEMENTATION	TESTING
First of all, the home page of the farmer will be implemented, to have the fields where he can insert the daily data and choose the right crop, then the data analysis and the update in the business layer. The main components needed are the CropManager and DBManager.	It is based on the DB: congruency between the data inserted and the data saved, the right update of the data and finally the update of the graph and chart. This last point will be tested when the graphs and the charts will be implemented. It is also tested the view part where it is presented the latest version of graphs and charts.

- **Chart:**

IMPLEMENTATION	TESTING
In this part the implementation is related to the grouping of farmers' and policy maker's data. It also presents the difficulty of the continuous update needed to always have the right chart order. The main components needed are the ChartManager and DBManager.	It is tested the right assignation of points to the farmers and their presence in the right chart based on the location (DB). It is very important getting congruent data from the database so the testing will also comprehend the chart manager. Last but not least the effectiveness of the update to the latest version.

- **Forum:**

IMPLEMENTATION	TESTING
It is based on view module which gives the farmer the possibility to create discussions and comments, and it is based on the forum manager and the saving in the database. The main components needed are the ForumManager and DBManager.	Also in this case it is used to test the congruency of the data inserted and data saved in the database, and it is also related to the work of the forum manager that has to present the right scenario based on the farmer choice. Here like in the other components, it is important to test the update of the forum and also deletion of the comment and discussion.

- **Forecast**

IMPLEMENTATION	TESTING
It needs to implement the connection with TSDPS provider, and the view part for the farmer.	It is tested the connection with provider and the congruency data showed on the farmer side and the ones on the website.

- **Notification**

IMPLEMENTATION	TESTING
They are managed by the NotificationManager. The implementation is related to the different connection with: the ForecastManager, ChartManager and the AssistanceManager.	It is tested with a demo version of the App, where all the other components works, and it can be tested the notifications are sent in the right moment, to the right receiver and with the right message.

- **Suggestion to the farmer**

IMPLEMENTATION	TESTING
It is managed by the SuggestionManager and it is implemented through the location and the crop of the farmer.	This is also tested in the demo version, to see if the suggestion are actually pertinent with the farmer.

6. EFFORT SPENT

Salvatore Buono 25h

Camilla Blasucci 25h

7. REFERENCES

7.1. Used Tools

1. Microsoft Word (https://www.microsoft.com/it-it/microsoft-365/word) to write this document.
2. Star UML (http://staruml.io/) for use case diagram, class diagram, sequence diagram, activity
3. Miro Wireframe (https://miro.com/) for detailed mock-ups

7.2 Reference Documents

- Software Engineering 2- year 2021- prof. Di Nitto Elisabetta course slide
- Specification document resources: https://www.tsdps.telangana.gov.in/aws.jsp
- Specification document: "R&DD Assignment A.Y. 2021-2022"
- Design Pattern slide from the Software Engineering 1-year 2020- prof Margara
- https://en.wikipedia.org/wiki/Object–relational_mapping