POLITECNICO
DI MILANO

Version 1.0.0

Design Document

Authors:
Marco Zanghieri - 10588478
Gabriele De Santis - 10820992
Michele Terziani - 10617090

January 2022

Contents

1 Introduction

A Purpose

Agriculture plays a pivotal role in India's economy. The majority of the rural households depends on it, and many of them are below poverty. Moreover, the climate change is one of the major threat and make it harder the cultivation of crops, resulti in losses in the farm income. Apart from this, the COVID-19 pandem increased the the vulnerability of marginalized communities and small holder farmers.
DREAM - Data-dRiven PrEdictive FArMing is an application project, thought and developed to make it easy communication and help among the farmers and with the government.

In the Design Document we provide more technical and detailed information about the software discussed in the RASD document, providing an overall guidance to the architecture of the system and to the interaction among all the components.
Here we explain the design decision made in the system, going from a general overview of the high level architecture to the specific component, the design pattern adopted and the interfaces along with a presentation of some graphical user mockups.

B Scope

The DREAM application is though to make it easy communication and help among the farmers and with the government.
Each Farmer, by accessing the application through an industrial tablet, can insert in the system all the data concerning their production (type of crops, water used, fertilizer used, etc.). If they face any kind problem, they can easily contact the Agronomist by sending them a ticket message to ask for suggestion or to ask for a visit in the land. Moreover, they can visualize the weather forecast in a specific page and interact in a discussion forum which is composed only by farmers. Each farmer belongs to a land and each land has an Agronomist.
The Agronomist is a governmental actor responsible of a certain area in which there are multiple lands. He receives message tickets from the Farmers through which he has to help them. He can also receives reports from the Policy Makers which can be useful to identify the Farmers in need of help. Since he has a good knowledge of his area, he's able to prevent threaths due to bad weather condition. In order to do that, he can visualize data concerning weather forecasts. Finally, the Agronomist have a calendar style view in order to manage the daily plans for the visits. Indeed, the Agronomist have to visit each farmer in his land at least twice per year. The Agronomist, in the calendar, can manage the daily plans by creating a visit event. Each visit of a certain day have to be created at least one day before the beginning of the visit. When a day is finished, the Agronomist have to submit a report in which he writes whether everything went correctly or if there were any problems.
Finally, the Policy Maker is another governmental actor responsible of identifying well performing and bad performing farmers. He has a complete view of all the farmers with all the data they inserted in the past days. He can inspects all the

farmers crop data and compare them with others kind of data, for example the amount of water given or the wheather, to understand if the good or bad performance is due to something in particular. Once the Policy Maker have identified the good or bad performing farmers, he can write a report to send to the Agronomist in onder to award or help the farmer. Optionally, the Policy Maker can insert the Farmers he wants to check more often in a list.

C Definitions, Acronyms, Abbreviations

C.1 Acronyms

D Revision history

Date	Modification
01/11	First Version

2 Architecture

A Overview: High-level components and their interaction

The DREAM platform, once implemented, will be a powerful tool for Telengana. In this chapter we explain the choices made in term of components that will be developed, deployed and tested. Since the whole system offers different functionalities based on the type of user, a brief overview is presented to better understand the diagrams in this section. Every user has its own component and approaches the DREAM platform using different interfaces. While Farmer will mostly interact with the system by using tickets and messages managers, Agronomist and Policy Makers can exploit other services. They can fetch useful data of Farmers stored in the database, schedule visits using an external calendar service and report any initiative they perform. Policy Maker will also have a general overview of the entire region. All three users have access to external services and therefore to weather forecast, water consumption and soil composition. Messages and notifications managers will be implemented with a common interface that can be used by every other components, ensuring an high level of reusability and reducing the development efforts. Further details of this topic are presented in the next sections.

A.1 High-level components

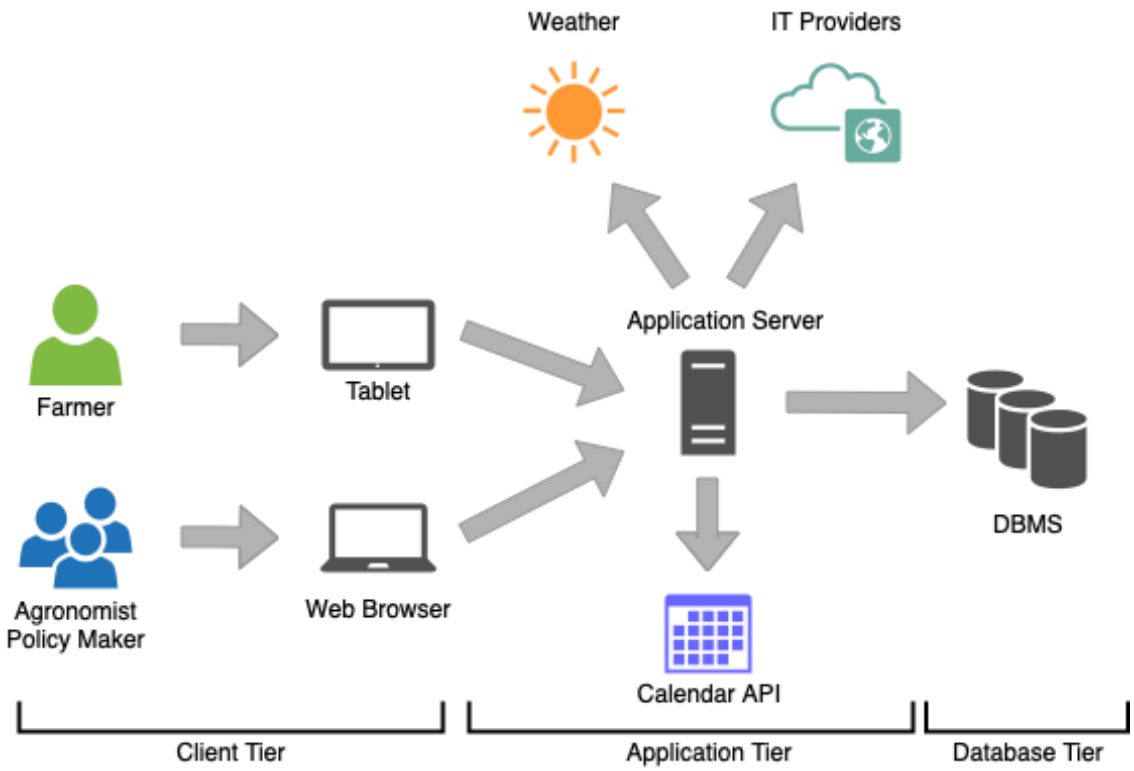
The best suitable structure for this platform is the 3-tier Architecture:

- **Client Tier**
  it contains the GUI provided to all users and therefore handle the interaction with the system. Functionalities are different based on the user type (Farmer, Agronomist, Policy Makers) but not depends on the device (Tablet, Web Browser, Smartphone)

- **Application Tier**
  it's the core part of the DREAM platform, contains all the logic part and includes components to interact both with users and storage system along with external services.

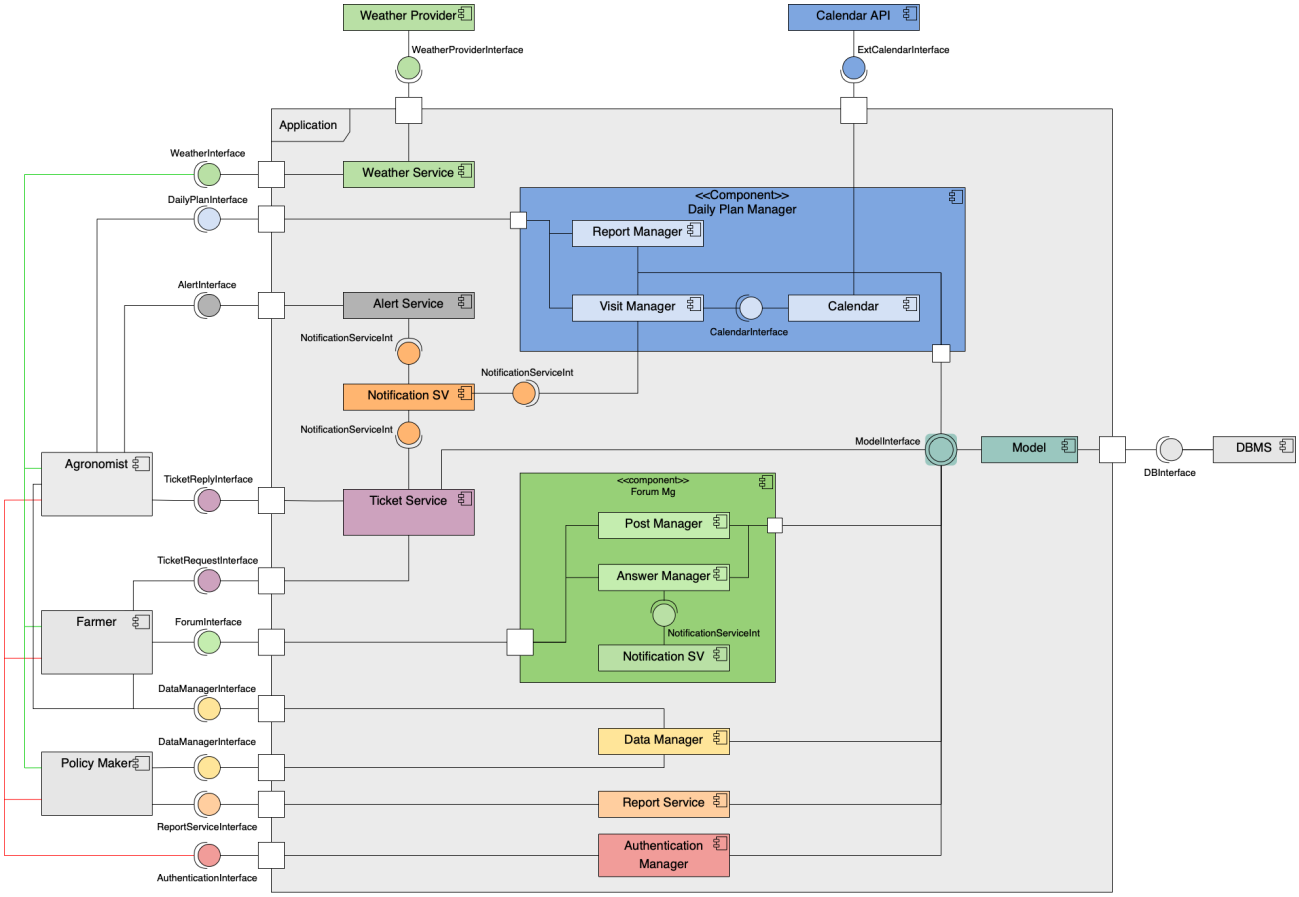
- **Database Tier**
  serve as the storage of the entire system, ensure persistence and availability of data and present a single point of access to interact with.

Following this structure, every component will be implemented in a different environment depending on the tier to which they belong. Devices and services of each tier along with users are represented in this diagram below.



B Component View

The following is a brief description of the component view. It mostly focus on the application tier and how each component of it relates to the clients and to the database of the DREAM platform.



- **Authentication Manager**
  This component identifies all the users by looking for the personal code inserted in the DBMS. The same interface is provided to all the components in the Client Tier

- **Alert Service**
  This component is used by agronomist to send alert to Farmers in the land. It makes use of the Notification Service interface in order to dispatch the alert to other devices

- **Notification Service**
  This component is fundamental to ensure communication across users. It handles all the notification that are generated in other services in the application such as the Alert Service, as described above, the Ticket Service and the Visit Manager subcomponent in the Daily Plan section.

- **Ticket Service**
  This item provides two interfaces meant for two different user. The Agronomist use one interface to retrieve all the request and eventually to send a reply while Farmers use the other one to create a new ticket (visit or help request) and to fetch all the related answers.

- **Weather**
  Agronomists and Farmers that interact with the DREAM platform are inter-

ested in weather condition. This service is provided by the Weather component that directly communicates with the external service provided by the Telengana Government (RASD document, chapter 1, section B)

- **Daily Plan Manager**
  With this component, an Agronomist can manage his daily plan. It consists in three subcomponents: the Visit Manager allow to organize the visits by exploiting the Calendar component interface. The Calendar service makes use of an external Calendar API. The Report Manager instead allows to add a report of a Daily Plan. The Visit and Report component also use the DBMS interface to store and update all the data inserted by the Agronomist

- **Forum Manager**
  This component, made of three other subcomponents, is responsible for the Farmers' forum. The Post Manager allow the creation of a new discussion while the Answer Manager instead allow to participate to a discussion. As shown in the diagram, the Answer Manager uses a Notification Service, which is the same one used by other components.

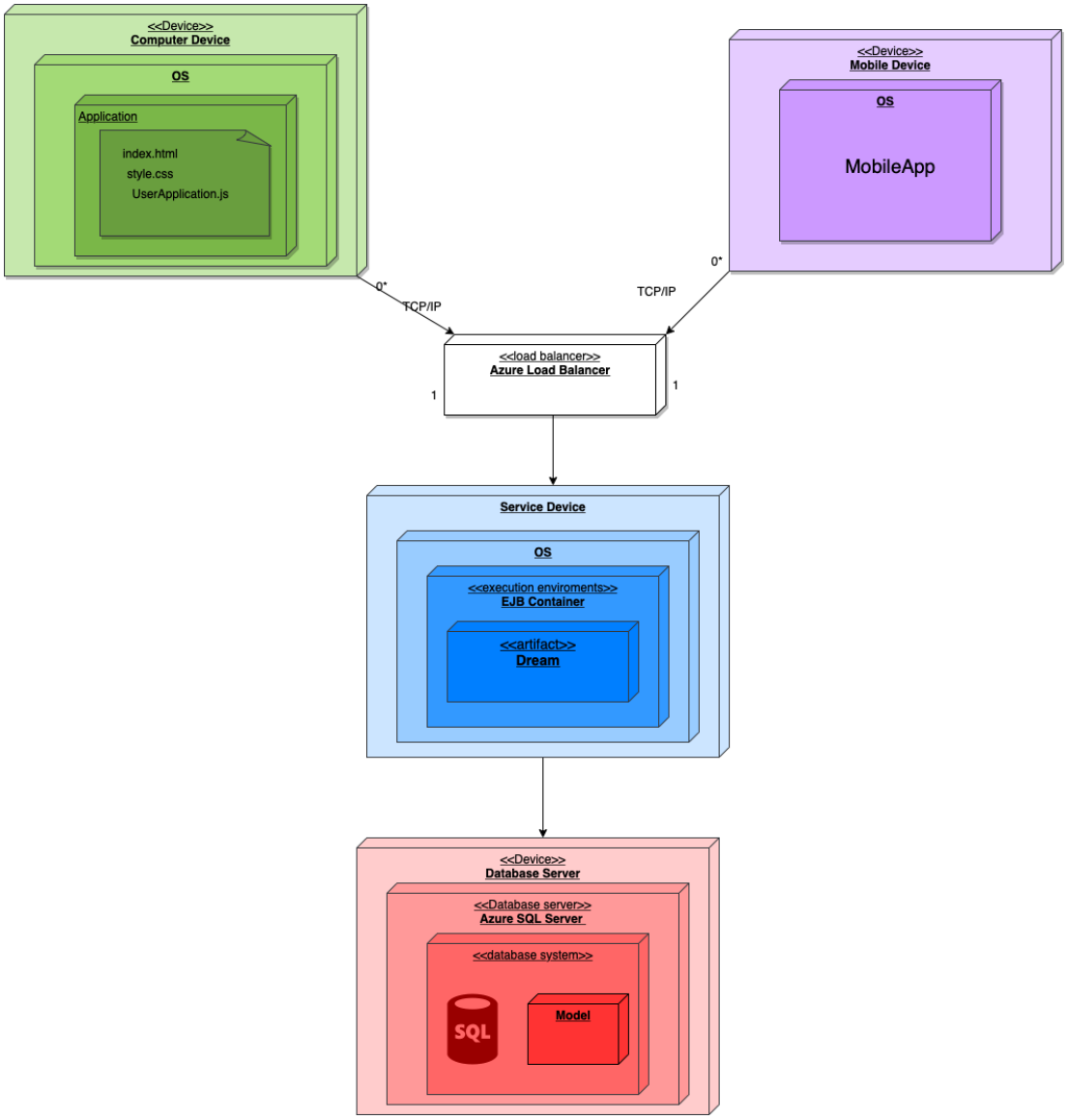
- **Data Manager**
  With this component, a Farmer can upload data about his productions while a Policy Maker can retrieve such data for evaluation. In both cases this component acts as an intermediary in the communication with the Database tier.

- **Report Service**
  This component is used by Policy Maker to add and update reports and works as the Data Manager described above.

- **Model**
  This component is the single point of access between the Database Tier and the Application Tier and presents an interface to every component that interacts with it to make query and insertion to the DBMS.

- **Client Components**
  Every users is represented with his own component in order to distinguish the division between the interfaces and the type of user. Those component are not part of the Application Tier but only represent the interaction between the clients and the system.

C Deployment View



- **Computer Device**
  This is the application which can be used by the Policy Maker and the Agronomist. The computer application is available both for Windows and iOS.
  It has a TCP/IP connection with the Application Server and communicates with it through HTTPS.

- **Mobile Device**
  This is the application which is used by the Farmer but also the Policy Maker and the Agronomist, if they need to. The mobile application is available for Android and iOS.
  The connection, as for the Computer Device, works through HTTPS.

- **Load Balancer** It is needed to distribute the incoming network traffic and act as a single point of contact for clients in order to route the requests to the

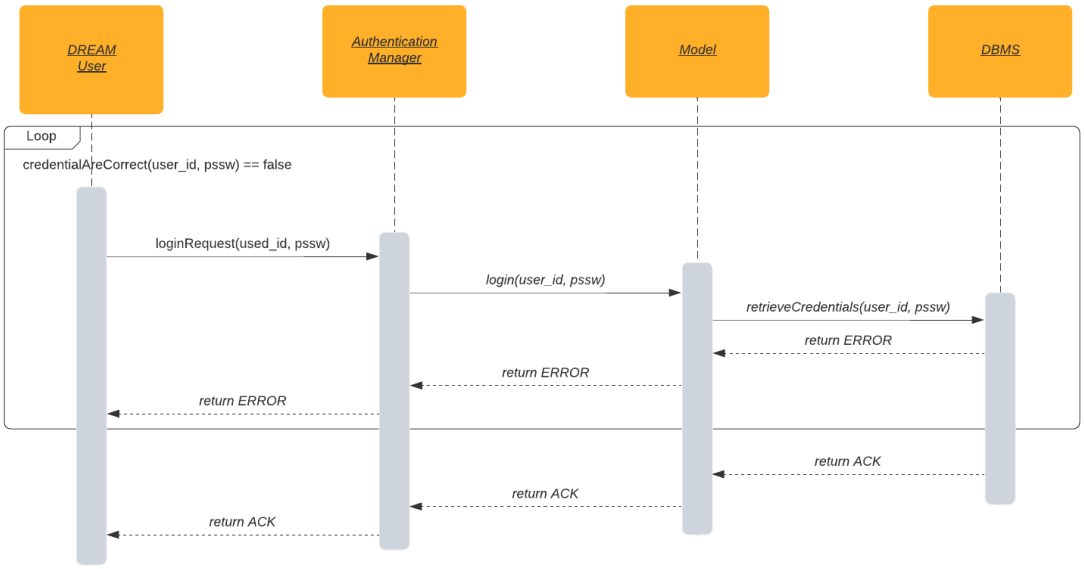
Application Server.

- **Application Server:** it is the business logic of the application and uses the Enterprise Java Beans to ensure the functionalities of the application. It communicates with the Database Server through the TCPS protocol.
- **Database Server:** it is needed to store the data and retrieve them when necessary.
  The Azure SQL Server is a relational cloud database developed by Microsoft and provided as a service.

D Runtime view

D.1 User

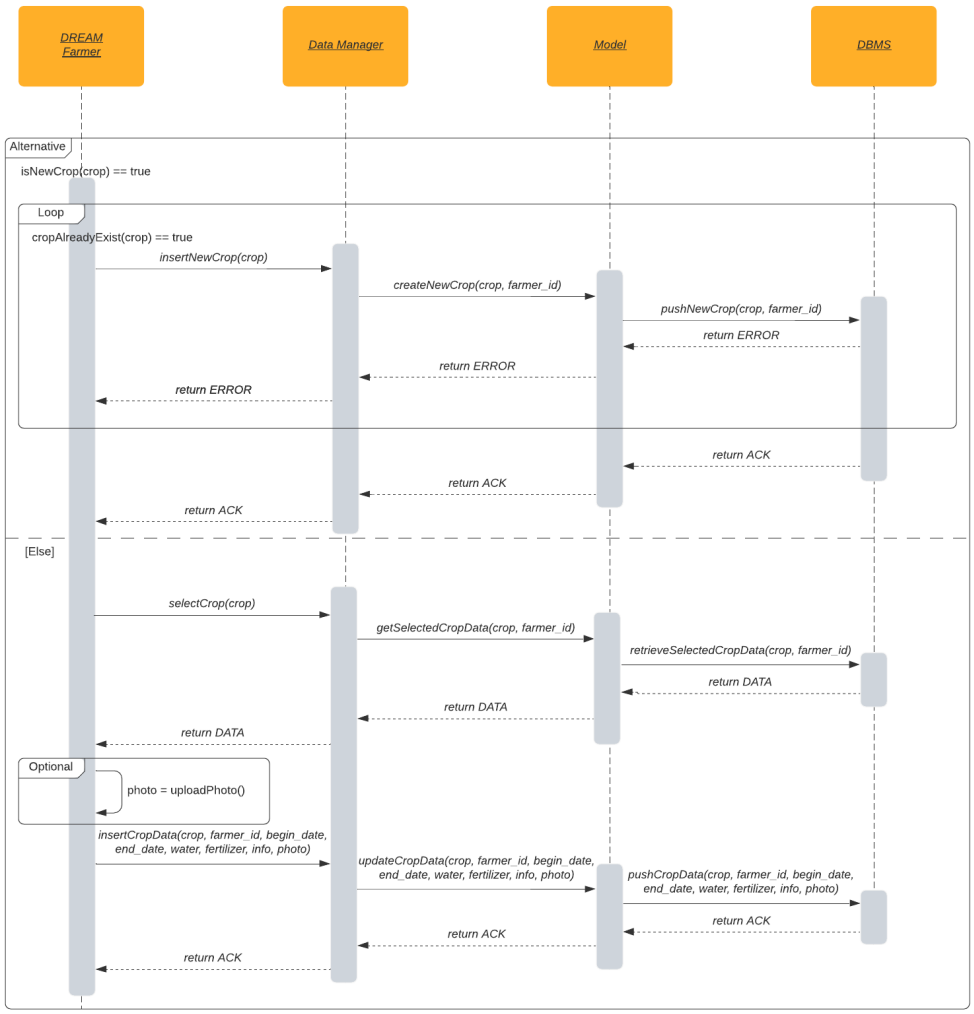
- **Login**



In this sequence diagram we analyze the Login of a User.
The User enters its information inside the Login page. Each of the User has a predefined id. Once the credential have been inserted, the loginRequest is forwarded to the Authentication Manager on the Server side. Then, it forwards the credentials to the Model which retrieve them from the DBMS. If the credentials doesn't match, an Error is returned, while if they does, a confirmation ACK is returned and the User can access its page.

D.2 Farmer

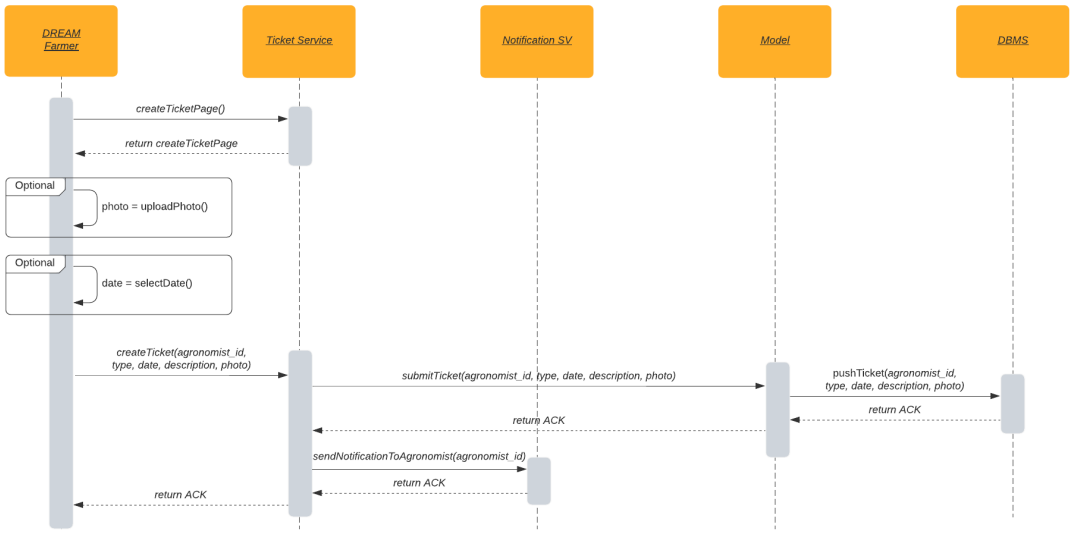
- **Insert data about crop**



In this sequence diagram we analyze the case in which a Farmer want to inserts data about its crop.
First of all, we check whether the crop inserted by the Farmer is new. In that case, a new Crop is created and it is stored in the Database. If the Crop already exists in the Database, an Error is returned; otherwise, a confirmation ACK is returned.
If the Farmer want to modify an already existing crop, first of all he selects it to visualize all the information already inserted in the past. The Data Manager, through the getSelectedCrop function, retrieves the past data from the DBMS. Once it is done, the Farmer can modify the information he prefers,

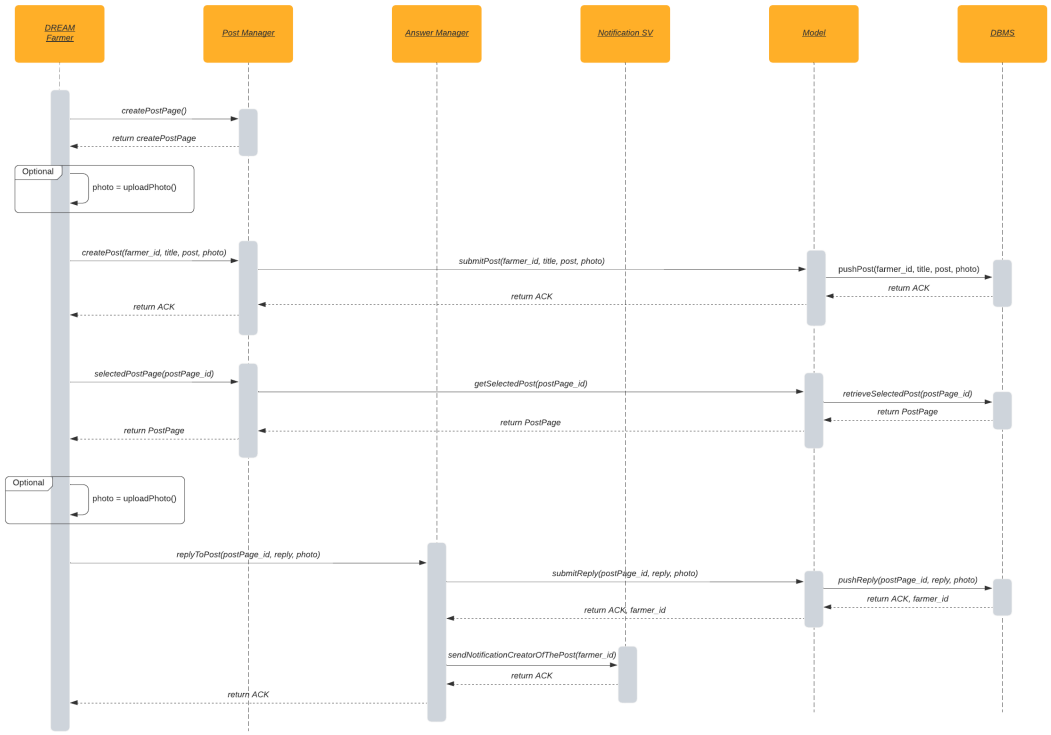
such as: the begin or end date of the crop, the amount of water given, the fertilizer used and some additional information. Optionally, he can also upload a photo. Once the information have been stored in the Database, an ACK is returned.

- **Send a Ticket**



In this sequence diagram we analyze the case in which a Farmer want to send a ticket to his Agronomist.
First of all, the Farmer clicks on the TicketPage which is retrieved from the Ticket Service. After that, he can optionally upload a photo. If he has chosen a "request a visit" ticket, he has to select a date and write some additional information; otherwise, if he has chosen a "help ticket", he can just write the additional information. After that, he can call the createTicket function which triggers the submitTicket of the Ticket Service. The ticket is saved in the DBMS thanks to the Model entity. When the TicketService receives the confirmation ACK from the DBMS, it calls the method sendNotificationToAgronomist, which take care of notifying the Agronomist of the ticket sent.

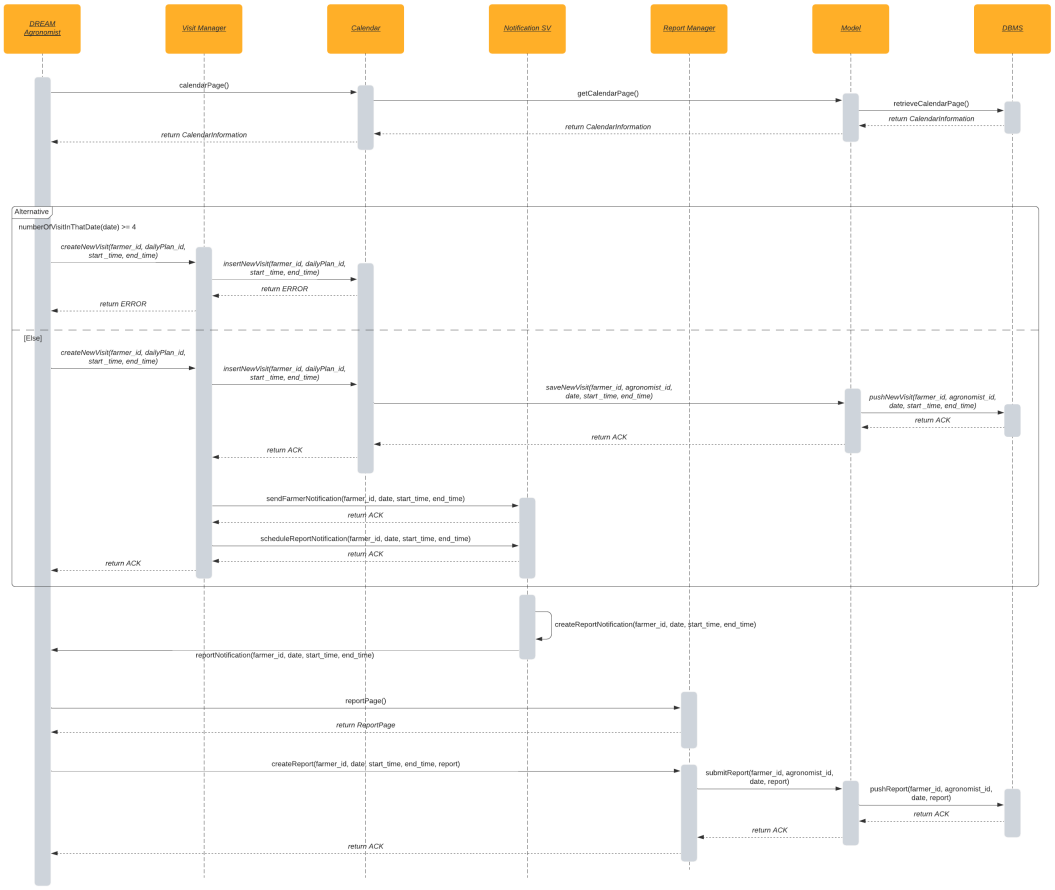
- **Interact in the Forum**



In this sequence diagram we analyze the case in which a Farmer want to interact in the Forum by creating posts, visualizing already existent posts and replying to them.
If the Farmer want to create a post, first of all he calls the method createPostPage, which retrieves the page to create a post from the Post Manager. In that page, he can write in the form everything he want. Optionally, he can also upload a photo. After that, he call the function createPost which triggers the submitPost function inside the Post Manager. The post is created and saved inside the DBMS.
The Farmer can also select a post to visualize it. For the sake of brevity, we suppose all the title's post are already saved in the front end cache of the browser, but of course they should be retrieved from the DBMS. Furthermore, initially we only retrieve the title's post and not all the posts information.
When the Farmer selects a post, the Post Manager call the getSelectedPost function which retrieves it from the DBMS and load it on the front end. The Farmer, can write an answer to that post and optionally upload a photo. After that, by calling the function replyToPost, the function submitReply of the Answer Manager is triggered. The answer is saved in the DBMS. When the confirmation ACK is returned to the answer manager, the sendNotificationCreatorOfThePost is called to inform the Notification Service to notify the creator of the post that a reply has been sent.

D.3 Agronomist

- **Manage Daily Plan**

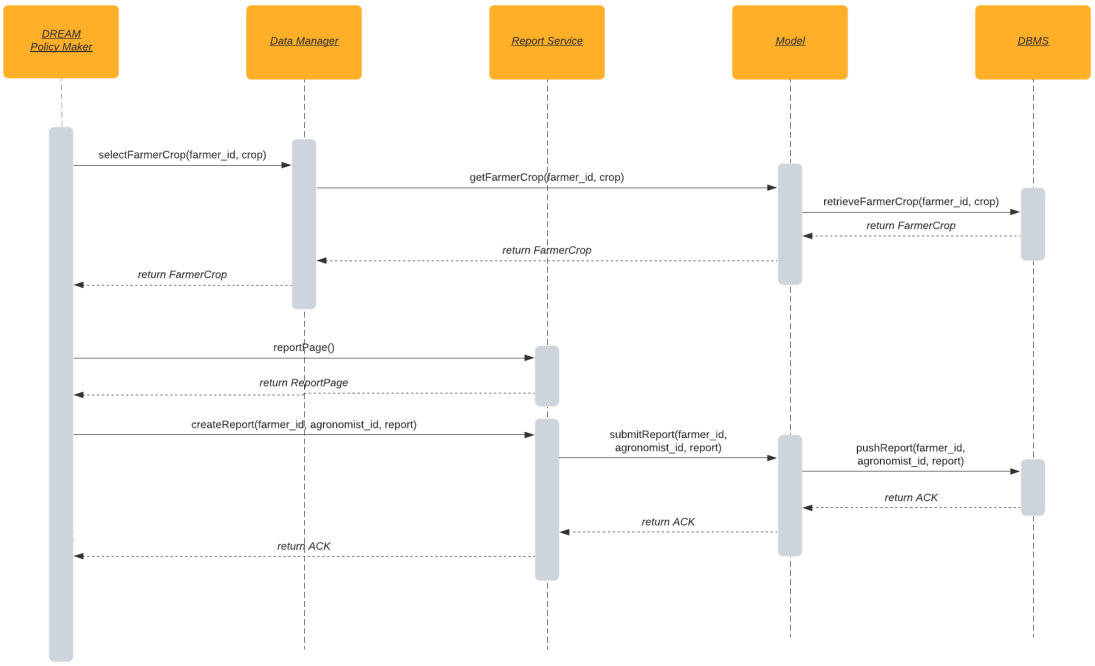


In this sequence diagram we analyze the case in which an Agronomist wants to interact in the Calendar to manage the Daily Plans. We remind that a Daily Plan is considered as a day of the Calendar and each Daily Plan can contain at most 4 visits.
First of all, all the information of the Calendar page have to be retrieved from the DBMS. It is done thanks to the method calendarPage which trigger getCalendarPage which retrieves all the CalendarInformations from the DBMS.
After that, the Farmer can create a visit; in order to do that, he has to choose a Daily Plan, which is modeled thanks to the dailyPlan_id, the start_time and the end_time of the visit. If the Daily Plan already contains 4 visits (and we know that thanks to the CalendarInformation we rerieved before), an Error is returned. Otherwise, the Calendar entity calls the saveNewVisit function which saves in the database the Visit planned. After that, when the confirmation ACK arrives to the Visit Manager, a notification to the Farmer of the visit is sent and a notification for the Agronomist's Report of the visit

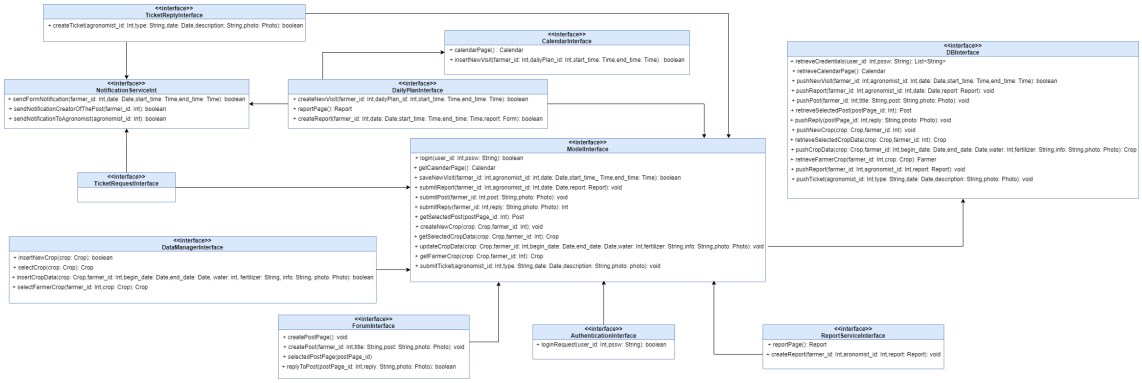
is scheduled. When the visit is completed (and we know that thanks to the end_time), the Report notification is created and sent to the Agronomist. He get the ReportPage thanks to the reportPage method and subsequently writes the Report. Thanks to the Report Manager, the Report is saved in the DBMS.

D.4 Policy Maker

- **Report a Farmer**



In this sequence diagram we analyze the case in which a Policy Maker wants to create a report of a Farmer.
First of all, the Policy Maker has to select a Land, a Farmer and the Crop of the Farmer to analyze. For the sake of brevity, we modeled just the selection of the Crop of the Farmer. When the Crop is selected, the Data manager retrieves all the information about it from the Database. The Policy Maker can analyze all the Crops and all the Farmers he wants to. Once he has identified a Farmer to Report, he clicks on the Report button which trigger the reportPage function to retrieve the page in which the Report has to be written. After that, he creates the Report and, once it is ready, it is sent through the createReport function. Finally, the Report Service send the Report to the DBMS through the submitReport function. When the submit is done, a confirmation ACK is sent back.



E Component Interfaces

F Selected architectural style and patterns

F.1 Architectural Style

Since the number of QoS is influenced by the choice of architectural style, a three-tier client-server architecture has been used to develop the entire system. This choice promotes a high decoupling of the system, increasing the reusability, scalability and flexibility. The system does not embody a pure client-server paradigm. This is clear considering the inclination towards a client-centered model, because the modules don't interact only through the data base, but also between the actors, since each component is in charge to deal with a set of data which belong to its role. Actually, it's neither a pure client-centered model because the data is centralized and accessed frequently by other components, which can modify data. Pay attention to the fact that we're using stateless components, they completely rely on external persistent storage.

F.2 Communication Protocols

The communication protocol used to exchange messages is HTTPS, which means we're using http over SSL. This is also endorsed by the crucial role that data plays in the system. In this way it is possible to reduce the coupling among client and server components. Besides, SSL is used to guarantee the security and the reliability of the connection. Regarding the format in which the data are transmitted, JSON is used for its performances. It is less verbose and so more readable, since it is designed specifically for data interchange, unlike xml which can provide a lot more than data interchange. Besides, it is faster than xml by definition and allows a much faster parsing due to its lower complexity. When dealing with Google calendar API, the communication protocol used is RESTful: Google Calendar is indeed used as an external service and this protocol clearly suits this situation.

F.3 Database

As written in the Deployment view, the database is SQL-like. Such decision is justified by the fact that we will have lots of requests from the database coming from different actors, so to cope in a proper way with this concurrency context we need

to use the Transaction system typical of an SQL DBMS. Since our system needs to handle dynamically the tickets and then save them in the database, different components such the Ticket Service component or the Forum Manager, has to interact many times with the database. The calls to the database are costly for our application, so in order to bring them to the minimum, a set of triggers are designed on the database to notify the other Services involved every time a Create or Delete operation is executed. Thereby, the calls to the database are optimized, since they are done only when an event occurs and update the other tables affected by it.

3 User Interface Design

A Login

(a) Login page on a laptop and a smartphone

(b) Login page on the tablet

B Farmer

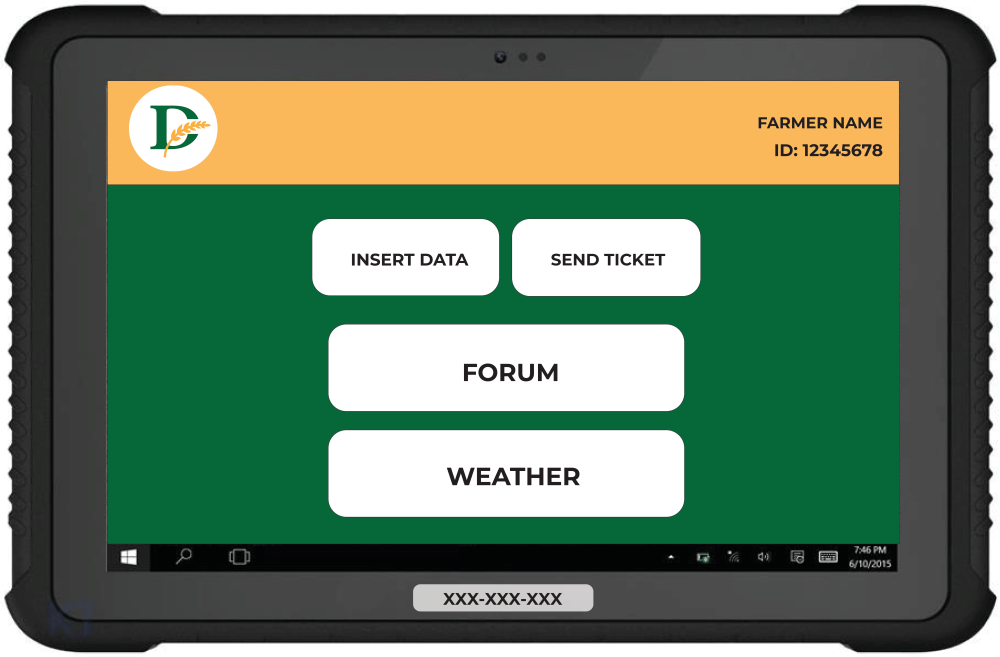


Figure 2: Homepage

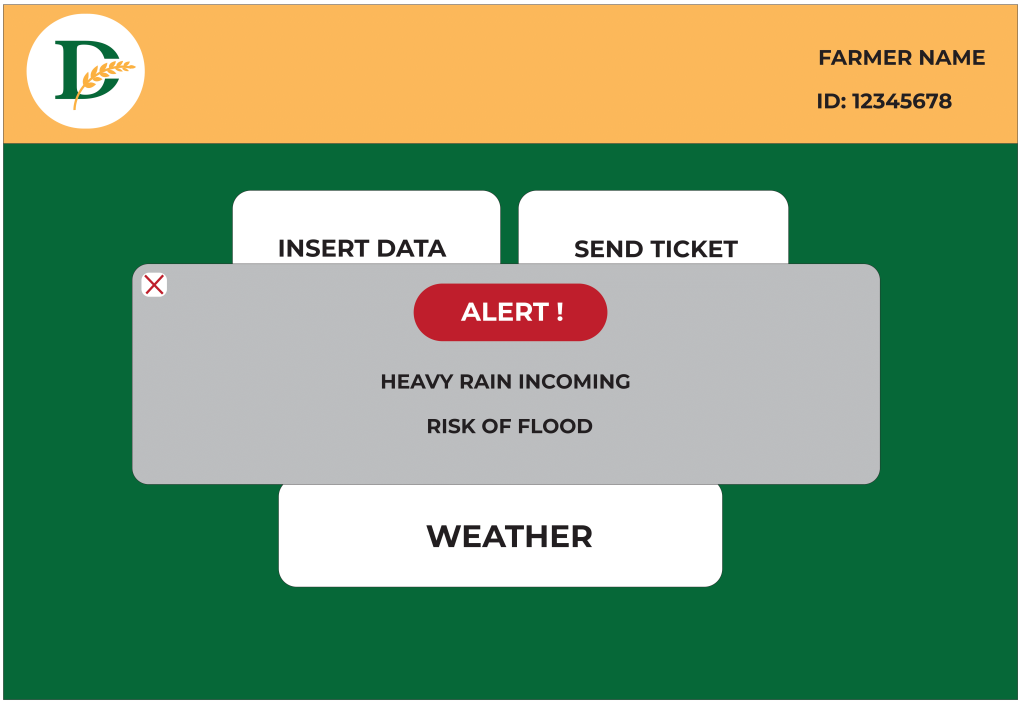


Figure 3: Notification alert

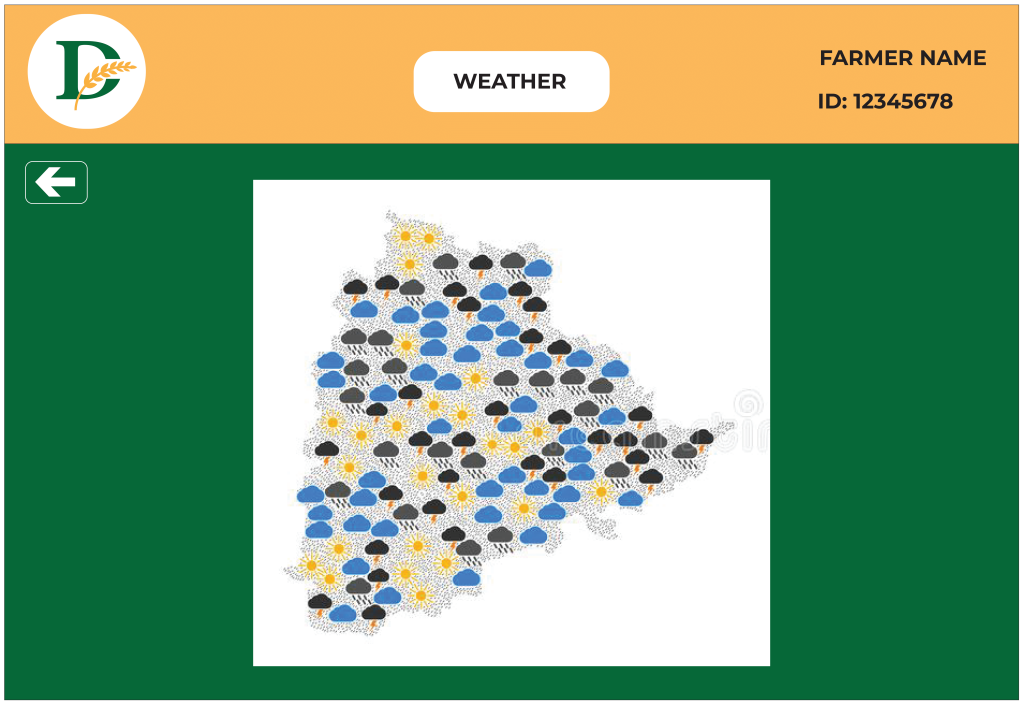


Figure 4: Weather map of the region

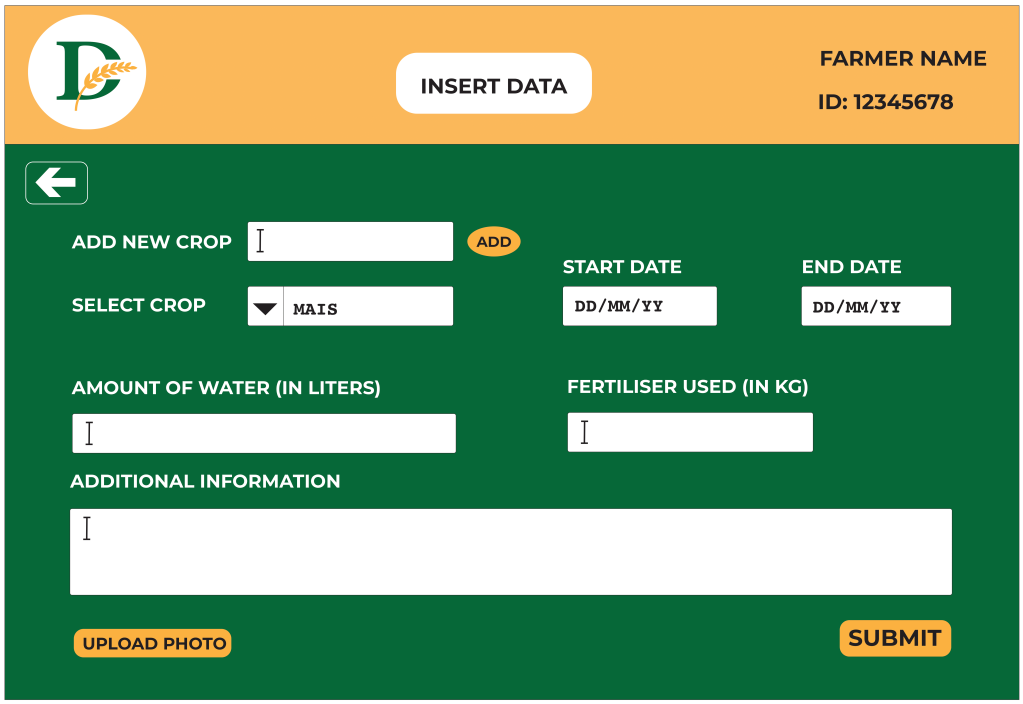


Figure 5: Production data form

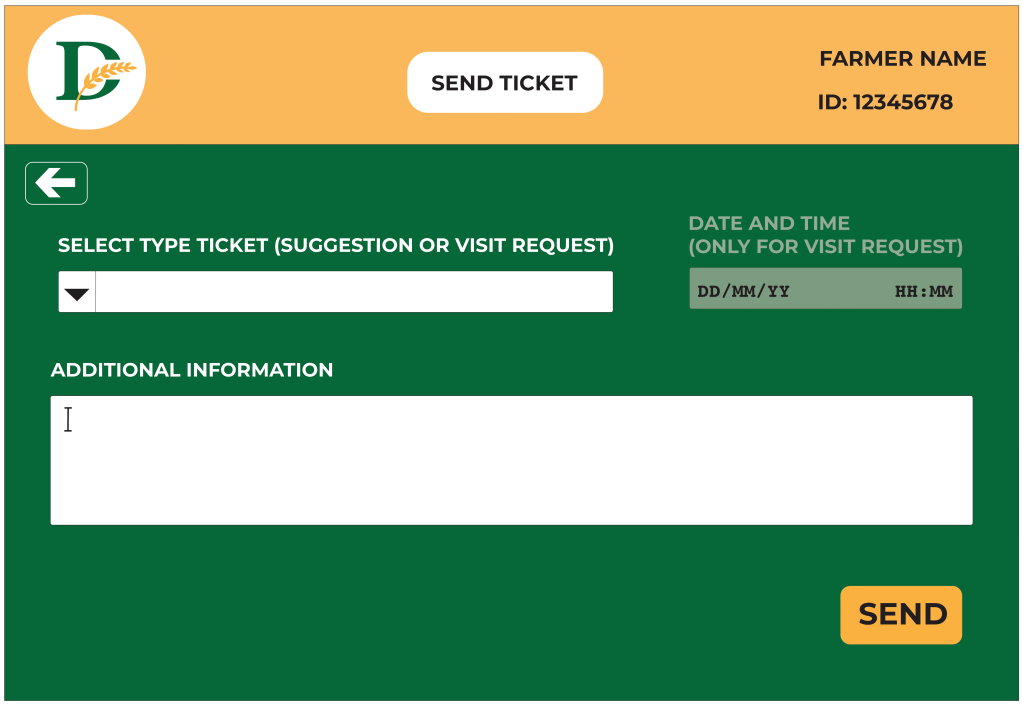


Figure 6: Ticket form

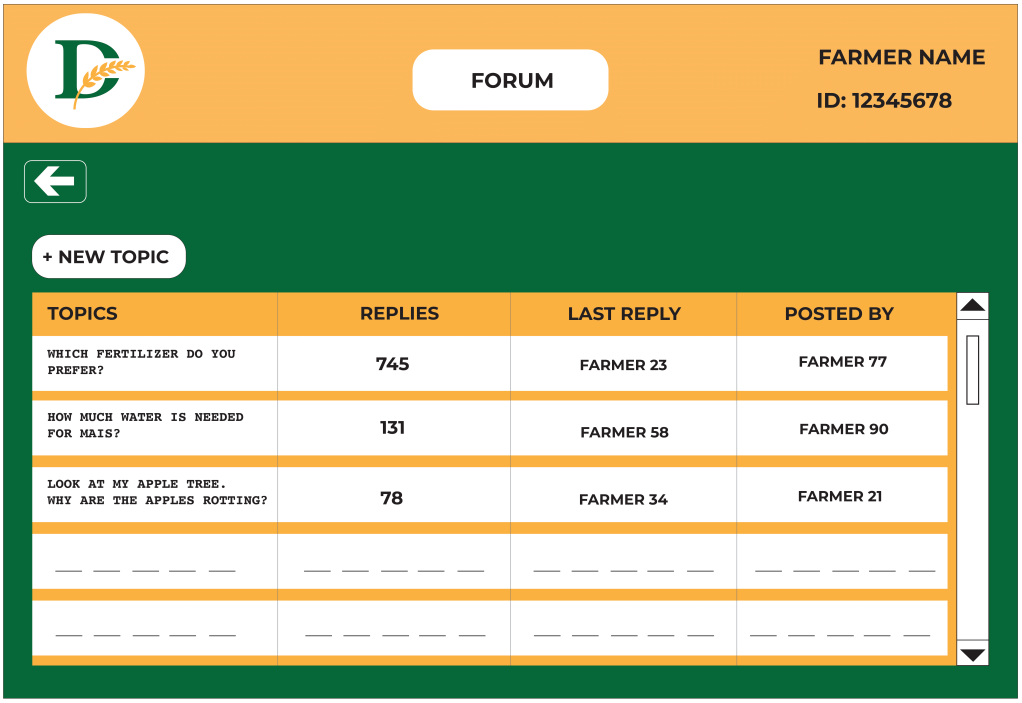


Figure 7: Forum page

C Agronomist

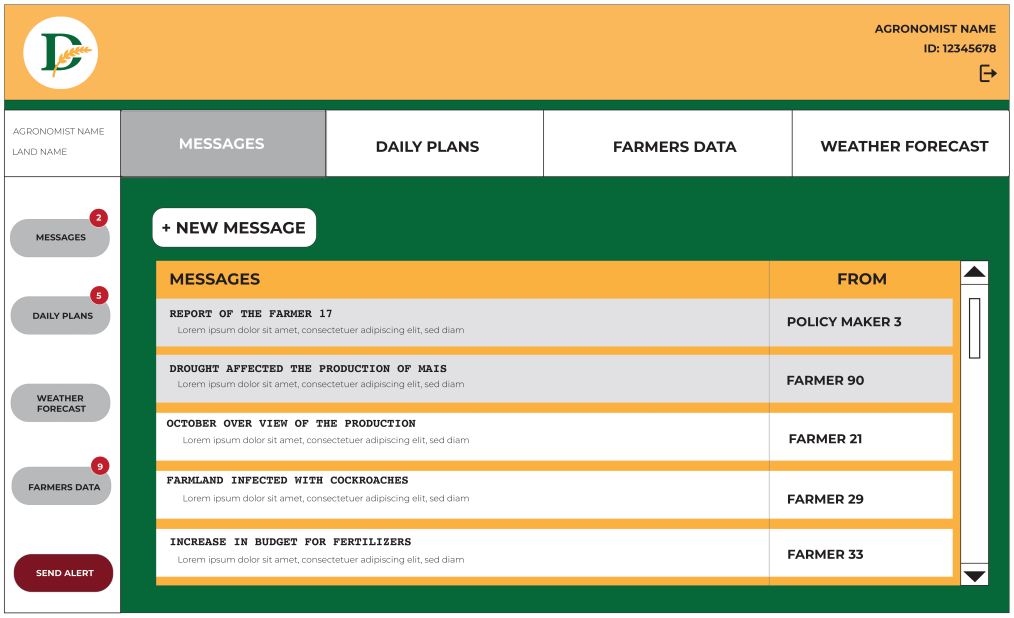


Figure 8: Ticket Page

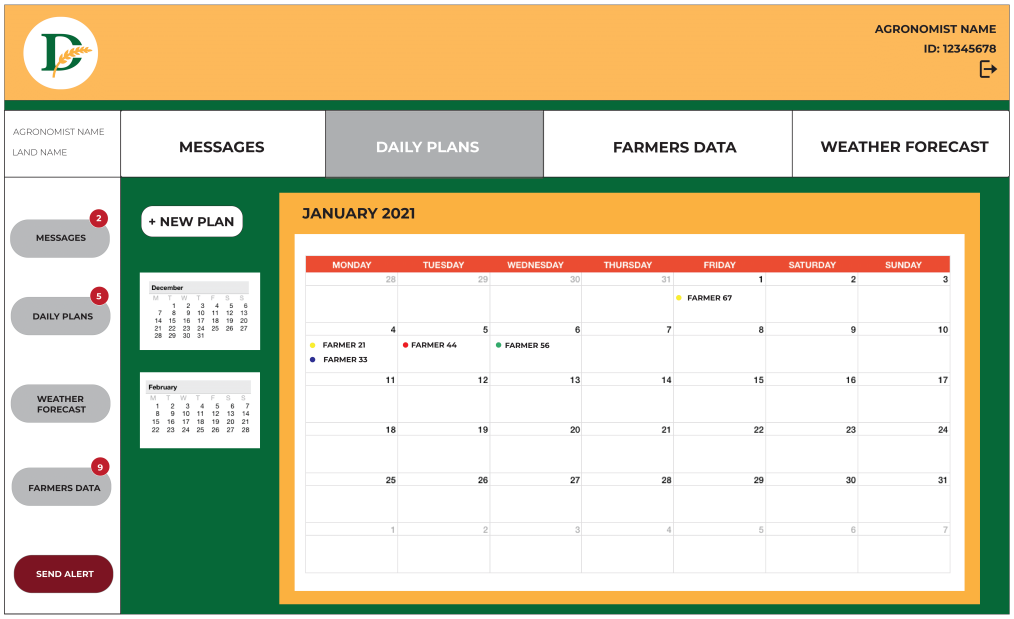


Figure 9: Daily Plan calendar

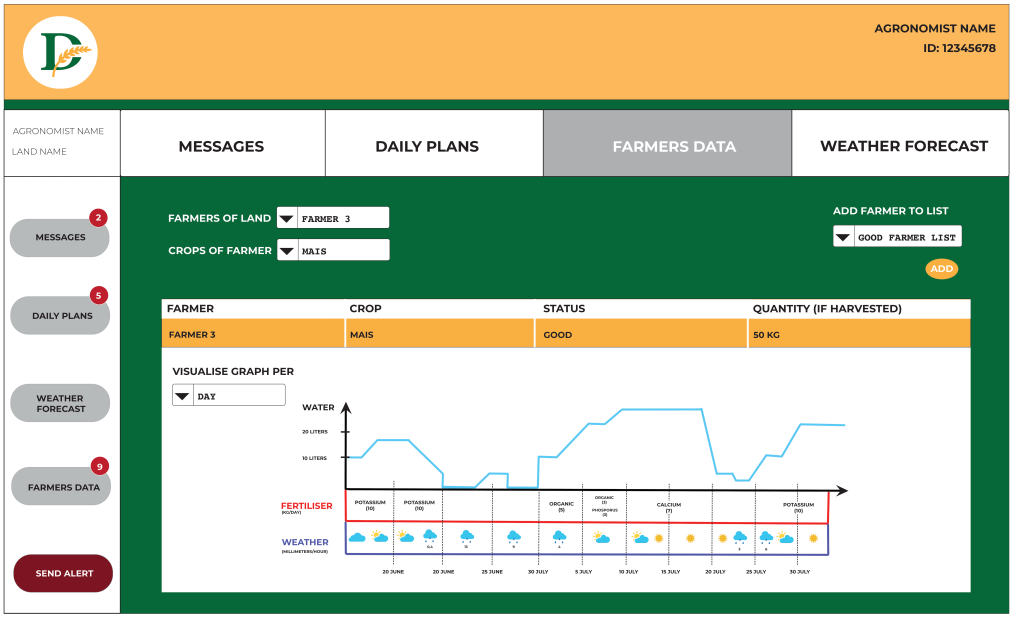


Figure 10: Farmer Performance

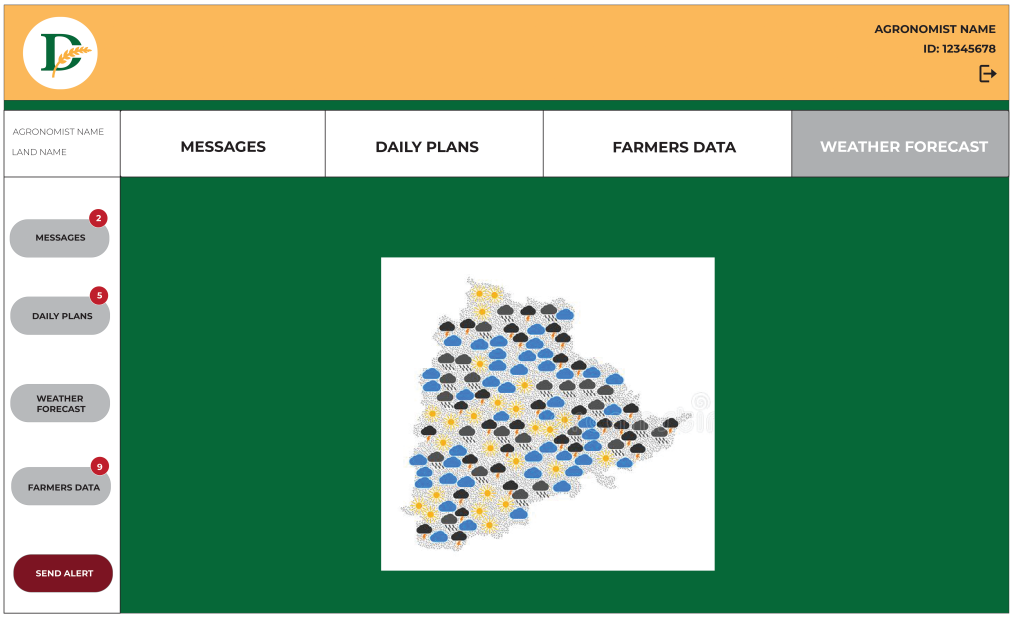


Figure 11: Weather map of the region

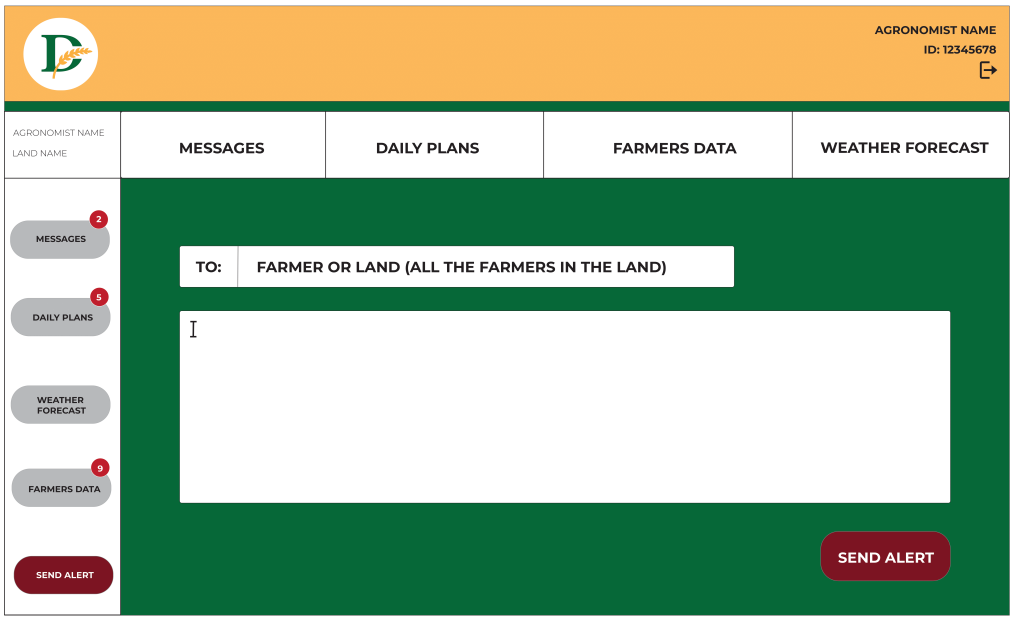


Figure 12: Alert form to send

D Policy Maker

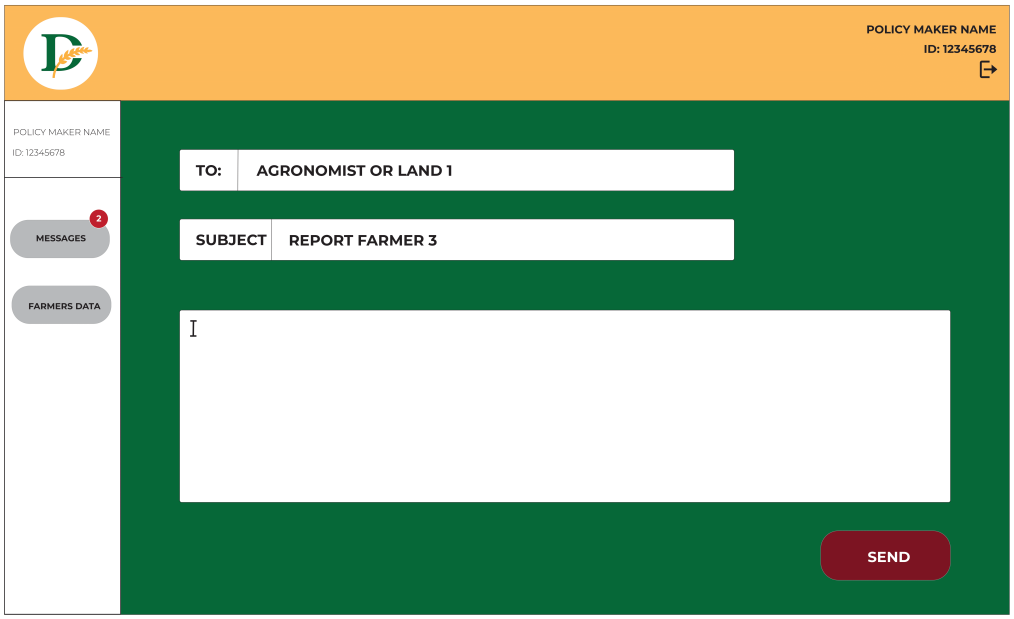


Figure 13: Farmer report form

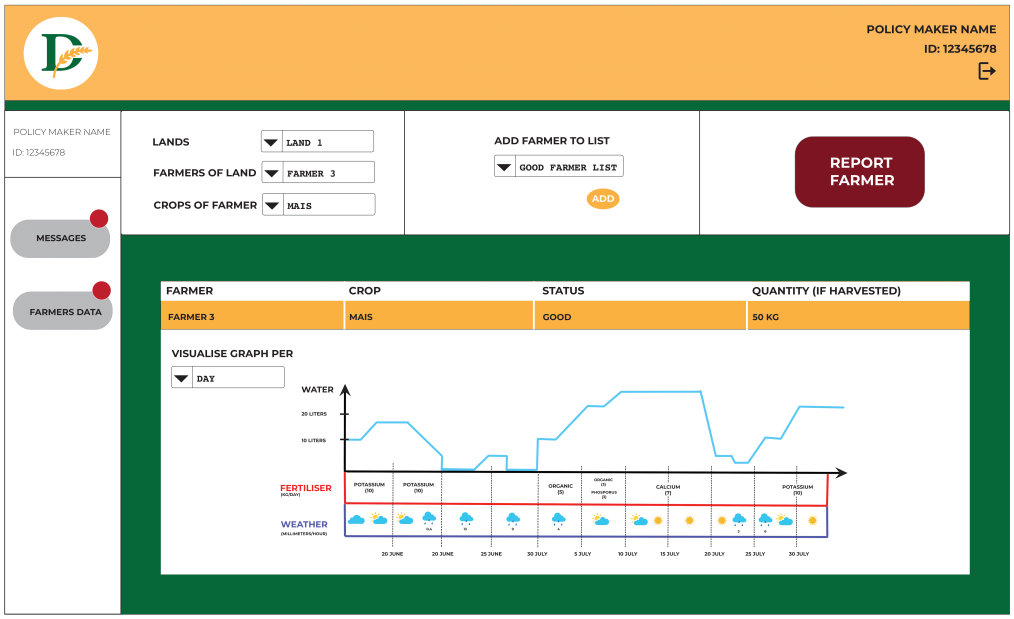


Figure 14: Farmer Perfomance Dashboard

4 Requirements Traceability

The following matrix shows which component contributes to the realization of a certain requirement. In particular it contains components as columns and requirements as row and the X sign on a cell indicates that the components corresponding to the cell's column is contributing to realize the requirement on the cell's row.
To save space the following abbreviations have been used.

- MOD: Model
- F: Forum
- DM: Data Manager
- RSV: Report SV
- DP: Daily Plan
- N: Notification
- TS: Ticket Service
- UF: User Farmer
- UA: User Agronomist
- UPM: User Policy Maker
- DBMS: Database Management System
- A: Alert
- AUT: Authentication
- W: Weather

	MOD	F	DM	RSV	DP	N	TS	UF	UA	UPM	DBMS	A	AUT	W
R1	X		X					X			X			
R2								X						X
R3	X						X	X	X		X			
R4	X	X					X	X						
R5	X		X						X		X			
R6						X	X	X						
R7	X				X	X			X		X			
R8					X									
R9	X				X						X			
R10	X				X				X					
R11					X	X		X						
R12						X		X				X		
R13	X						X		X		X			
R14	X	X									X			
R15					X	X			X		X			
R16						X		X						
R17	X	X						X			X			
R18	X										X		X	
R19						X	X		X					
R20		X				X		X						
R21	X					X								
R22	X		X							X	X			
R23	X			X						X	X			
R24	X			X						X	X			
R25	X			X						X				
R26					X	X			X					
R27					X	X		X	X					
R28	X				X				X		X			

5 Implementation, Integration and Test Plan

A Overview

A key feature of the system architecture and project plan is the fact that the integration test plan drives and is driven by the integration plan.
Since it is almost impossible to develop error free software and, quoting Dijkstra, program testing can be used only to show the presence of bugs, but never to show their absence, the phase of verification and validation takes a crucial role, in particular the necessity of having a plan to integrate components while testing them. Thus, for this reason, the aim is to find as many bugs as possible until the release date of the application.

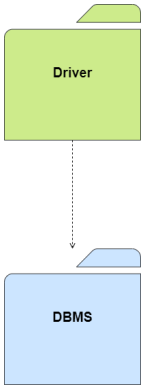
B Implementation Plan

The next sections explains the implementation and integration plan in order to test all the sub-components of the system. An incremental approach is considered in this phase since it gives a better fault understanding and can be done as soon as a component is released.
In particular, a bottom-up approach is used, requiring that the implementation proceed gradually from the components close to the Data Tier up to those that interact with the Client one.
This also allow to proceed in parallel when testing different components that work independently from each other, reducing the steps to be taken for this part, which are listed here:

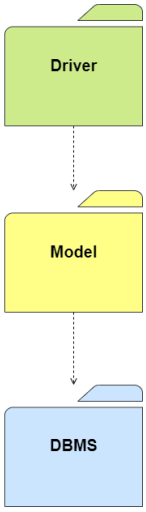
- The first step will be to implement the DBMS component of the system by testing the database communications.
- The Model component will be the second one to be implemented given its role play in the entire application.
- At this point, many components can be implemented and integrated autonomously as they will provide different services for different users. External components can also be tested in this phase.
- The fourth part is dedicated to the Daily Plan component and its sub-components while the fifth component will be the Notification Service, since further components rely on it.
- Then the Alert Service and the Weather Service, along with the external provider for this one, are implemented.
- At this point, the client components are implemented with their services.
- Finally, the Authentication Manager is tested and the Weather Service is implemented as the system and the client components are fully tested.

C Integration Strategy

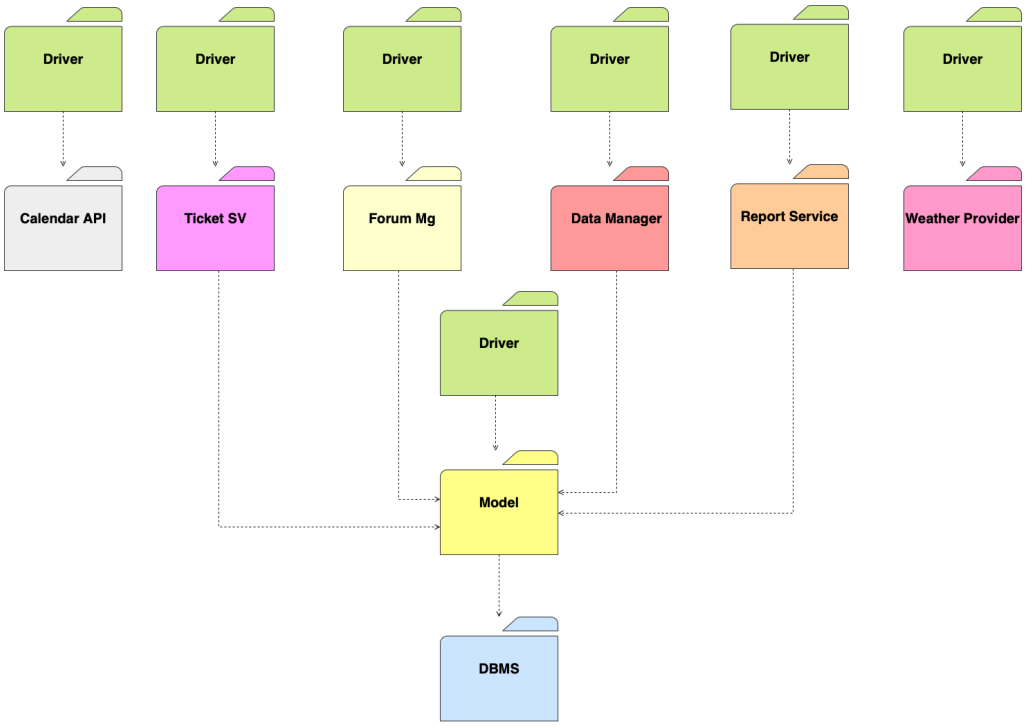
The following schemes represent the implementation and integration plan according to the chosen approach.
Firstly, communication with the Database is unit tested with a driver to check if all the functionalities offered work correctly. Checking the network can be done in this phase since it will be used next.



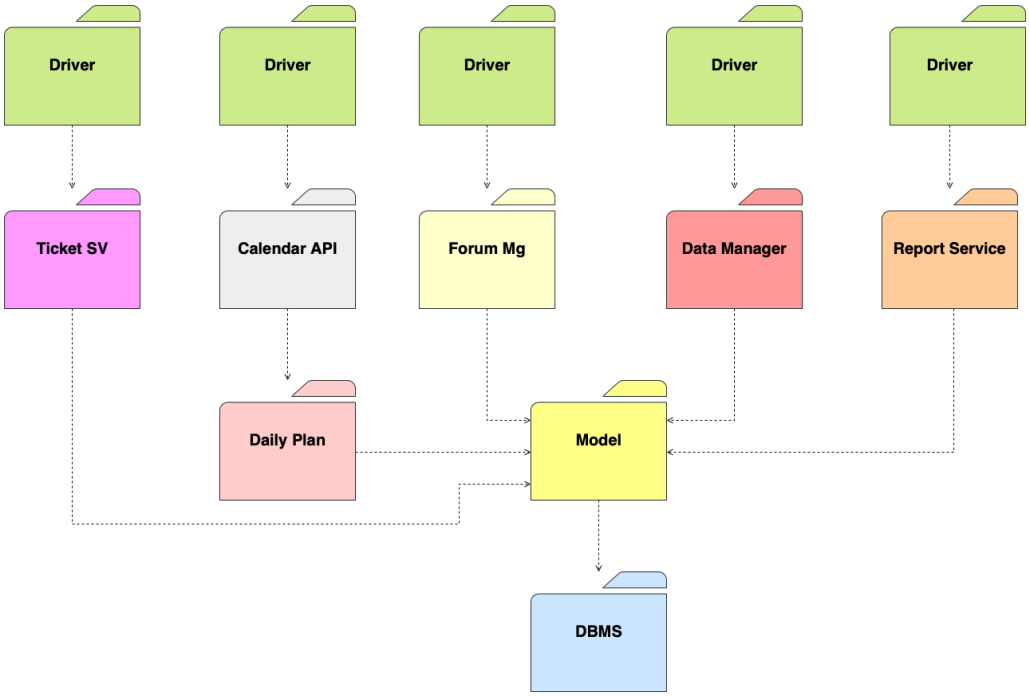
The second phase is entirely dedicated to the Model Component since it can be considered the single point of entry to the DBMS. As the majority of the components rely on this, different type of data must be stored, updated or deleted in the database, so it's fundamental to check with a driver all these functions to ensure consistency and availability among other functions.



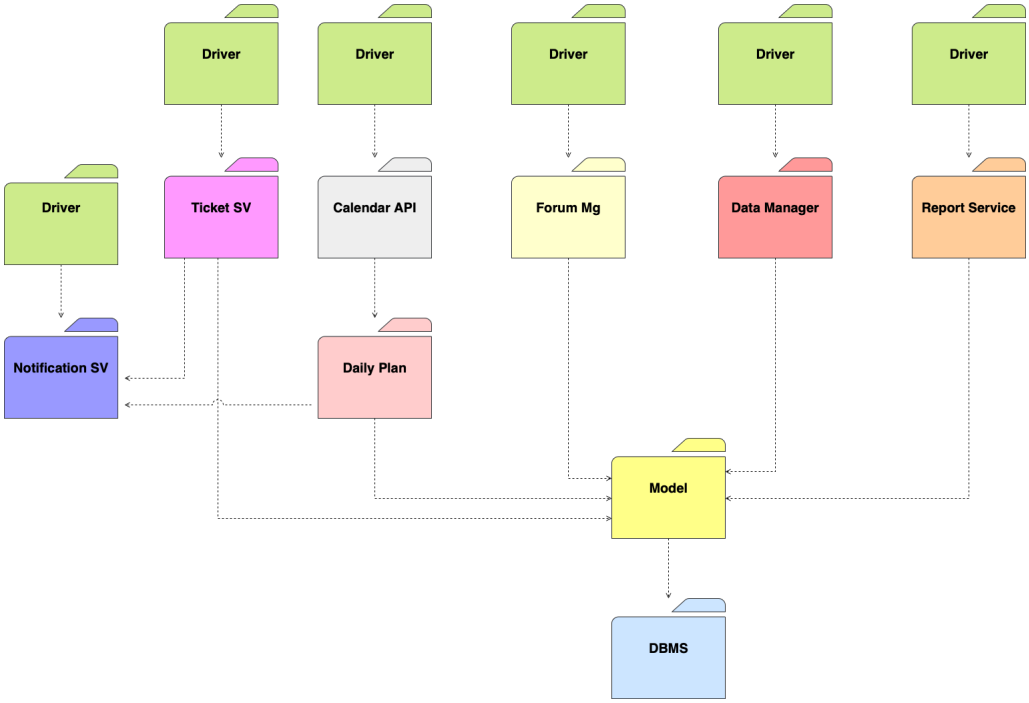
At this point, many components can be tested in parallel as they offer different services to different users and they don't rely on each others. The Forum Component will be implemented along with its subcomponents (Notification Service excluded) while the other components (Ticket Service, Data Manager and the Report Service) are unit tested with a single driver for each one. At the same time, the two external services (Calendar API and Weather Provider) are tested since they will be used in the next phases.



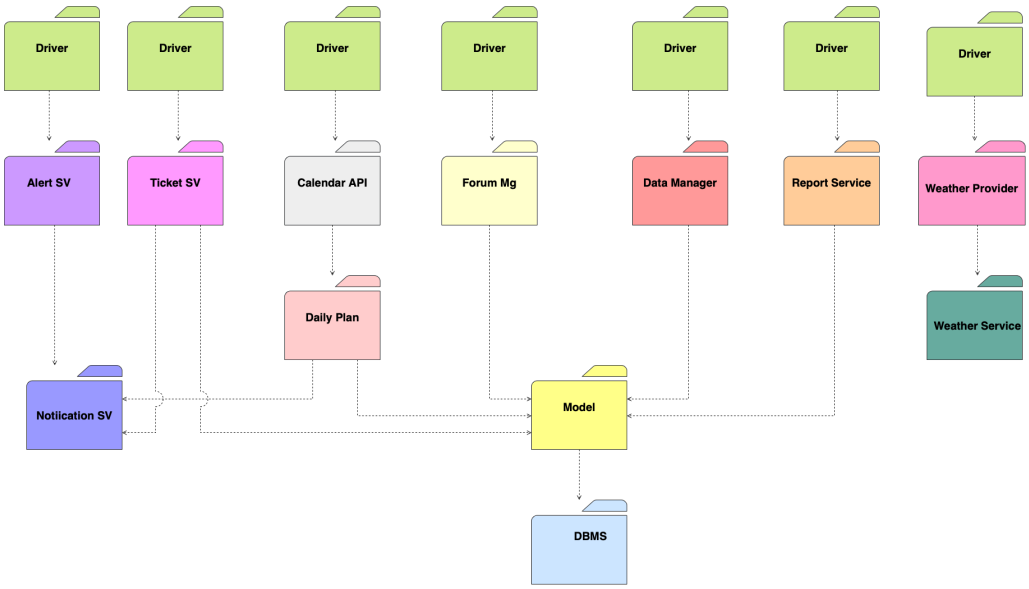
As the Calendar API is integrated, the next step is dedicated to the Daily Plan as it offers many functions to the Agronomist. A driver is needed and all the sub-components that compose it are implemented and integrated.



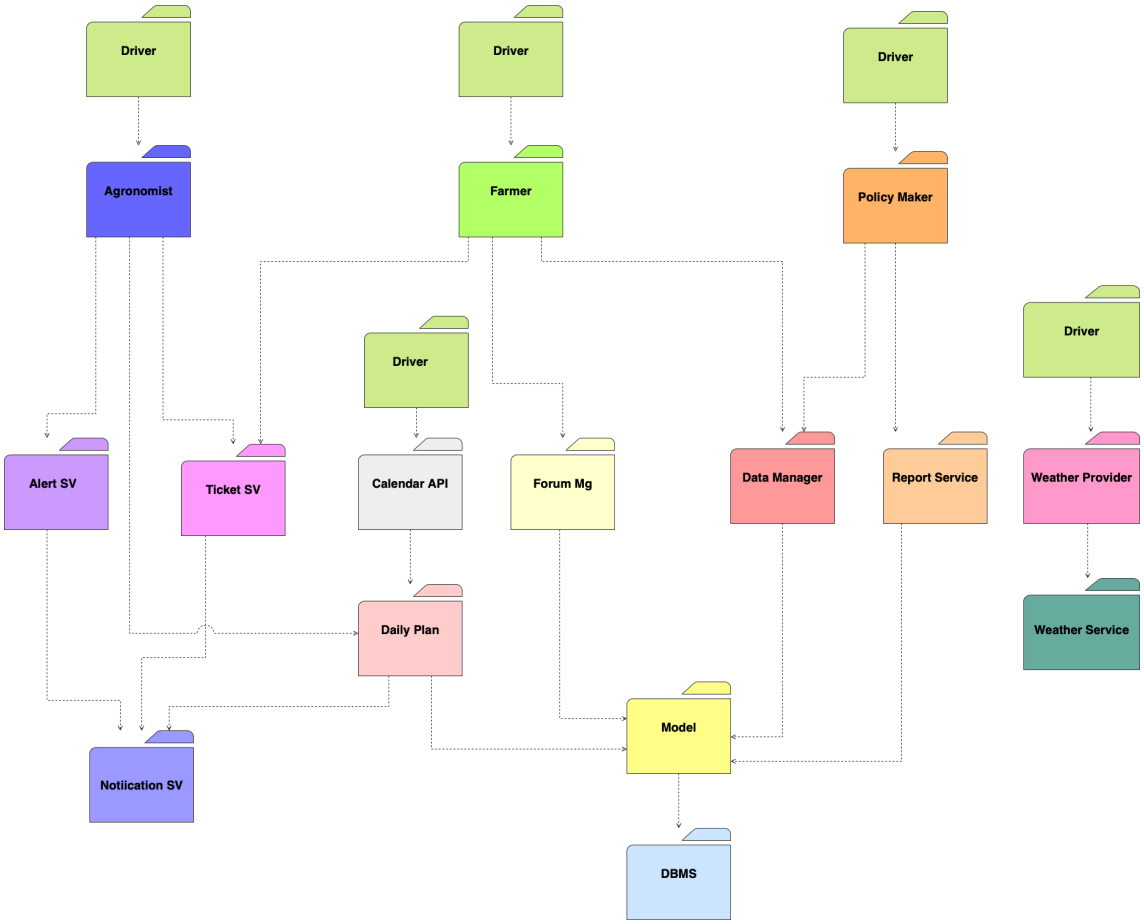
Then the Notification Service is unit tested, focusing on the interaction with the components that will make use of it such as the Ticket Service, that will exploit it to send answers and reply to the users, and the previously tested Daily Plan, fundamental to send the visit plan to the farmers.



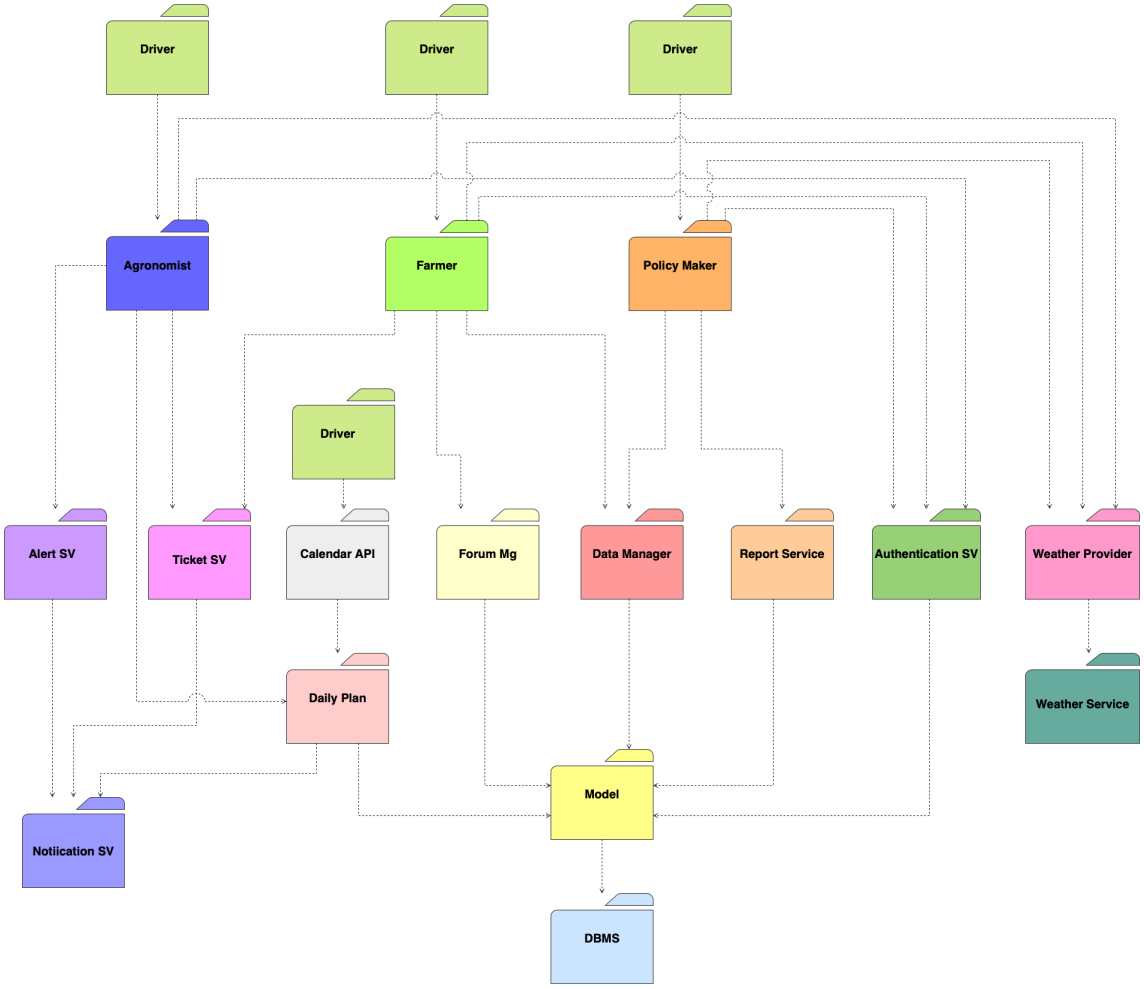
The next part is dedicated to the implementation of the Alert Service, mainly focusing on the integration with the already implemented notification system since it the core part of the functionality offered by this component. Also the Weather Service is unit tested and implemented with its external provider.



At this point, the system is almost tested and all the functionalities are implemented so the three client components can be added in the integration plan. Each actor has its own component linked to the services available for him, checking the integration between them.



Finally, the Authentication Manager and the Weather Service are implemented as they are both used by all the actors. In particular, it's tested the ability to authenticate every user registered to the system and the correctness of the weather data provided to the users.



D System Testing

This final chapter describes other guidelines for the verification process. When the system is completely tested, the next step will be the implementation as a whole of the entire application in order to verify all the requirements specified in the RASD document, both functional and non-functional.
The environment in which the testing will be developed must be similar to the real one, in order to identify and resolve any issue on performance and reliability.
At this point, different testing can be done, focusing on specific requirement that were not considered in the previous steps. In particular:

- Functional Testing: following the RASD document (Chapter 3, Section B), all the functional requirements must be satisfied along with design constraints and other limitations reported. The traceability matrix in Chapter 4 of this document can help to establish a link from the system and its component to the requirements.

- Environment Testing: it's important to test the final interaction with the system on the real world. The tablet app and the web app must be tested

before any public release in order to meet QoS constraints and to avoid later intervention that can cost up to 200 times the cost of early correction

- Performance Testing: this part in fundamental to identify the presence of query optimisation possibilities or any inefficiency with the architectural style and patterns chosen. Also, as seen in the deployment view, TCP/IP connections plays a major role in the communication part of this platform, so they must be checked in order to find any issues or possible bottlenecks that can interfere with the correctness of the application.

6 Effort Spent

In this section you will include information about the number of hours each group member has worked for this document.

Gabriele	
Topic	Ore
Purpose, scope, definitions	6
high level components and interaction	2
Deployment view	3
Runtime view - sequence diagrams	18
selected architectural style and patters	2
implementation, integration and test plan	1
Document organization	2
User Interface Design	5

Marco	
Topic	Ore
Purpose, scope, definitions	2
high level components and interaction	5
Deployment And Component view	11
Runtime view - sequence diagrams	3
selected architectural style and patters	2
implementation, integration and test plan	4
Document organization	3
User Interface Design	5

Michele	
Topic	Ore
Purpose, scope, definitions	3
high level components and interaction	7
Component view	4
Runtime view - sequence diagrams	2
selected architectural style and patters	3
implementation, integration and test plan	8
Document organization	2
User Interface Design	5

8

7 References

Specification Document: "RDD Assignment AY 2021-2022.pdf"
RASD Document: RASD.pdf