



POLITECNICO
MILANO 1863

SOFTWARE ENGINEERING II

A.Y. 2021/22

DREAM

Data-dRiven prEdictive fArMing

Design Document

Authors:

Mauro FAMÀ

Giacomo LOMBARDO

January 10, 2022

Contents

1	Introduction	3
1.1	Purpose	3
1.2	Scope	3
1.3	Definitions, Acronyms, Abbreviations	4
1.4	Revision History	4
1.5	Reference Documents	5
1.6	Document Structure	5
2	Architectural Design	6
2.1	Overview	6
2.2	Component view	7
2.2.1	Application Server	8
2.2.2	Client Handler	10
2.2.3	Dashboard Manager	11
2.3	Deployment view	12
2.4	Runtime view	13
2.4.1	Registration	13
2.4.2	Login	14
2.4.3	Report insertion	15
2.4.4	Creation and handling of a help request	16
2.4.5	Forum interaction	18
2.5	Component Interfaces	19
2.6	Selected Architectural styles and patterns	21
2.7	Other design decisions	23
3	User Interface Desgin	24
3.0.1	Login	24
3.1	Farmers User Interface	25
3.1.1	Homepage	25
3.1.2	Report submission	27
3.1.3	Forum	28
3.1.4	Help Requests	31
3.2	Policy maker User Interface	32
3.2.1	Homepage	32
3.2.2	Help Requests	39
3.2.3	Forum	41
3.3	Flow of the User Interface	42
4	Requirements Traceability	43
5	Implementation, Integration and Test Plan	46
5.1	Plan definition	46
5.2	Development tools	50
5.3	System testing	51

5.4	Testing tools	52
6	Effort spent	53
6.1	Team Effort	53
6.2	Mauro's Effort	53
6.3	Giacomo's Effort	54

1 Introduction

1.1 Purpose

The purpose of this document is to comprehensively describe the design of the DREAM platform[1]. This document will describe the architecture of the system and its characteristics, analyzing its components explaining their functionalities. This document was written following the RASD[2] of the same project, in fact the assumptions and design choices are based on the assumptions already explained, however maintaining independence between the documents. This document focuses on the explanation about the Software-To-Be, its modules, its interfaces and its implementation.

1.2 Scope

The main goal of the platform is to create a support tool and communication channel for farmers in Telangana, leveraging the IT infrastructure of government resources. The platform also aims to be a supervisory and control system for the authorities of the Ministry of Agriculture in Telangana, providing accurate and bulk data of farmers' performance, reducing the overhead due to manual retrieval of this information.

The system will have two separate and exclusive interfaces, one dedicated to PMs in Telangana and one dedicated to farmers in the region. Farmers will be provided with a browser executable web application which provides a dashboard to monitor the sensors embedded in their fields, weather forecasts for the local area and summarize the resources obtained and consumed by the plantations. In addition to the monitoring functionality, from the farmer's dashboard it will be possible to compile reports to be sent to the PM responsible for the area, combining data entered manually by the farmer, data created by IoT devices and data retrieved from the internet.

The web application also provides access to the forum dedicated to the Telangana farming community, connecting all participants in the DREAM program through its threads. Farmers will also be able to submit their help requests into the system, which will be automatically forwarded to the PM responsible for the area, who in turn will have a dedicated dashboard exclusively for the administrative side. From his application, the PM will be able to forward responses to help requests, monitor the performance of farmers in his area, all while viewing data from sensors and weather forecasts. From his dashboard, the PM will have the tools to perform end-of-quarter performance evaluations, including direct communication channels to affected farmers.

There is no platform that currently provides these functions, in fact this document refers to the design of a system created from scratch, without any integration of existing systems, but exploiting the IT infrastructure already physically installed in the Telangana region.

1.3 Definitions, Acronyms, Abbreviations

- **DREAM:** Data-dRiven PrEdictive FArMing
- **DD:** Design Document
- **RASD:** Requirements and Specification Document
- **IT:** Information Technologies
- **PM/TPM:** Policy Maker or Telangana Policy Maker
- **HR:** Help Request
- **DMZ:** De-Militarized Zone
- **UI:** User Interface
- **GUI:** Graphical User Interface
- **REST:** REpresentational State Transfer
- **API:** Application Programming Interface
- **MVC:** Model View Controller
- **R:** Requirements
- **HTTPS:** HyperText Transfer Protocol over Secure Socket Layer
- **MQTT:** Message Queue Telemetry Transport
- **IMAP:** Internet Message Access Protocol
- **SMTP:** Simple Mail Transfer Protocol
- **JSON:** JavaScript Object Notation
- **IoT:** Internet of Things
- **DBMS:** Database Management System
- **CRUD:** Create Read Update Delete
- **JDBC:** Java DataBase Connectivity
- **JPA:** Java Persistence API

1.4 Revision History

- **Version 1.0** January 10, 2021
First version of the complete DD document

1.5 Reference Documents

1.6 Document Structure

1. **Introduction:** a brief introduction to the document, the analyzed problem and the proposed solution.
2. **Architectural Design:** overview of the system architecture and analysis of its design, components, deployment and runtime behaviour.
3. **User Interface Design:** overview of the system's User Interface.
4. **Requirements Traceability:** mapping of the requirements defined in the RASD document to the design elements defined in this document.
5. **Implementation, Integration and Test Plan:** description of the implementation procedure of the system's subcomponents, detailing their order and the testing procedure.
6. **Effort Spent:** a brief recap of the time spent to complete this document.

2 Architectural Design

2.1 Overview

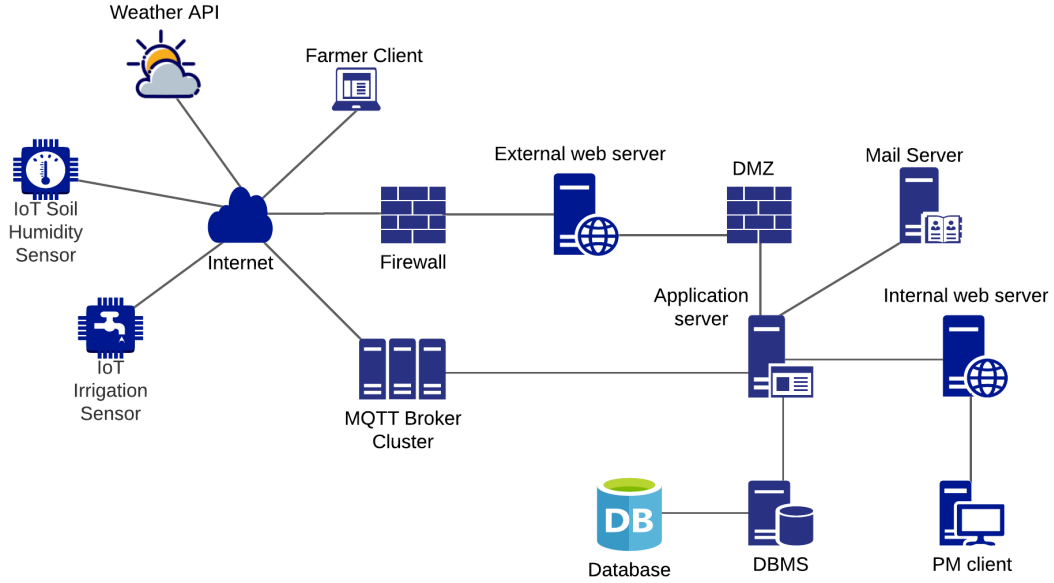


Figure 1: System architecture

The system will be developed from scratch and it will entirely replace the legacy system.

The DREAM application is designed with as a three-tier application with web servers, application server, and a database functioning as the three tiers of the application. The application can be accessed via a webapp provided both to internal and external users. The internal users of the application are the policy maker, that will log into and use the platform from their workstation, connected to an internal web server. The external users are the farmer, connected to the webapp from their personal computer. This separation between internal and external users is meant first and foremost for security requirements, in order not to expose sensitive data to potential risks.

Other significant components in the DREAM ecosystem are the IoT sensors, crucial to automatically retrieve and collect data from local farmers and to better evaluate their performances. IoT sensors are connected via MQTT protocol to a MQTT Broker Cluster which periodically retrieves the data collected by sensors. All the weather informations are taken from the weather forecasting system of Telangana through its APIs.

Finally, the application server communicate with a database through a Database Management System.

A more detailed description of the components will be given in the following sections.

2.2 Component view

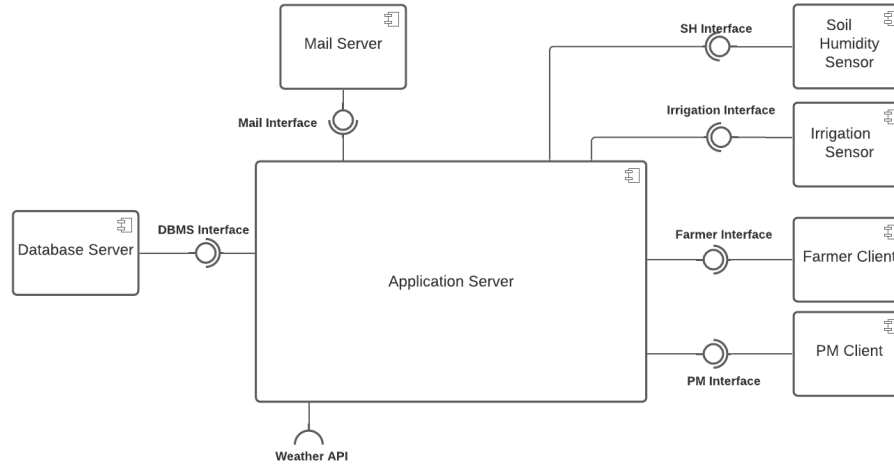


Figure 2: High Level Component Diagram

- **Database server**

This component is responsible for data management, it provides the interface of the Database Management System to the Application Server, in order to deal with the data management, storing and modification process.

- **Mail server**

This component allows the Application Server to create and manage email accounts, send and receive email using standard email protocols. The SMTP protocol is used to send messages and handle outgoing mail requests, The IMAP protocol is used to receive messages and to process incoming mail.

- **Application server**

It is the main component of the system, all the business logic resides in it. This component offers both functionality dedicated to farmers and PMs, interfacing directly with clients. It receives the data entered by the users and sent by the weather forecast system, cross-referencing them and providing for processing, back forwarding and transferring them to the

data management system. Its components are described in detail in the following sections.

- **Farmer client**

It is the component thanks to which the farmers can interface with the Application Server. It represents the web application dedicated to farmers that allows them to access the services of DREAM. This component also receives data from the IoT sensors that monitor the farmer's possessions, making them available to be retrieved from to the Application Server.

- **PM client**

This is the component thanks to which the Policy Makers can interface with the Application Server. It represents the web application dedicated to them that allows them to take advantage of the services offered by DREAM to monitor and manage agriculture in Telangana. All the functions are accessible thanks to the PM Interface exposed by the Application Server.

- **Soil Humidity and Irrigation sensors**

Sensors for measuring soil humidity and irrigation are internet-connected components responsible for creating and sending data about farmers metrics to the Application Server. They both expose their own interface, through which the Application Server can retrieve the data of interest.

2.2.1 Application Server

The application server contains the business logic of the application. The following diagram describe its internal structure: The components of the application server are:

- **Database Manager:** deals with management of the data model and data operations, working as a bridge between the physical database and the other components of the server.
- **Mail Manager:** interfaces between the server components and the mail server in order to provide mailing services.
- **Weather Manager:** interfaces between the external weather forecasting API and the other server components.
- **Access Manager:** handles the authentication procedure of the application's users. It is also responsible for the generation of access credentials.
- **Sensor Manager:** handles the communications between the application and the sensors installed in the Telangana territory. Manages the retrieving of collected data from the sensors and its processing and storing procedure.
- **Report Manager:** handles the creation and the visualization of reports.

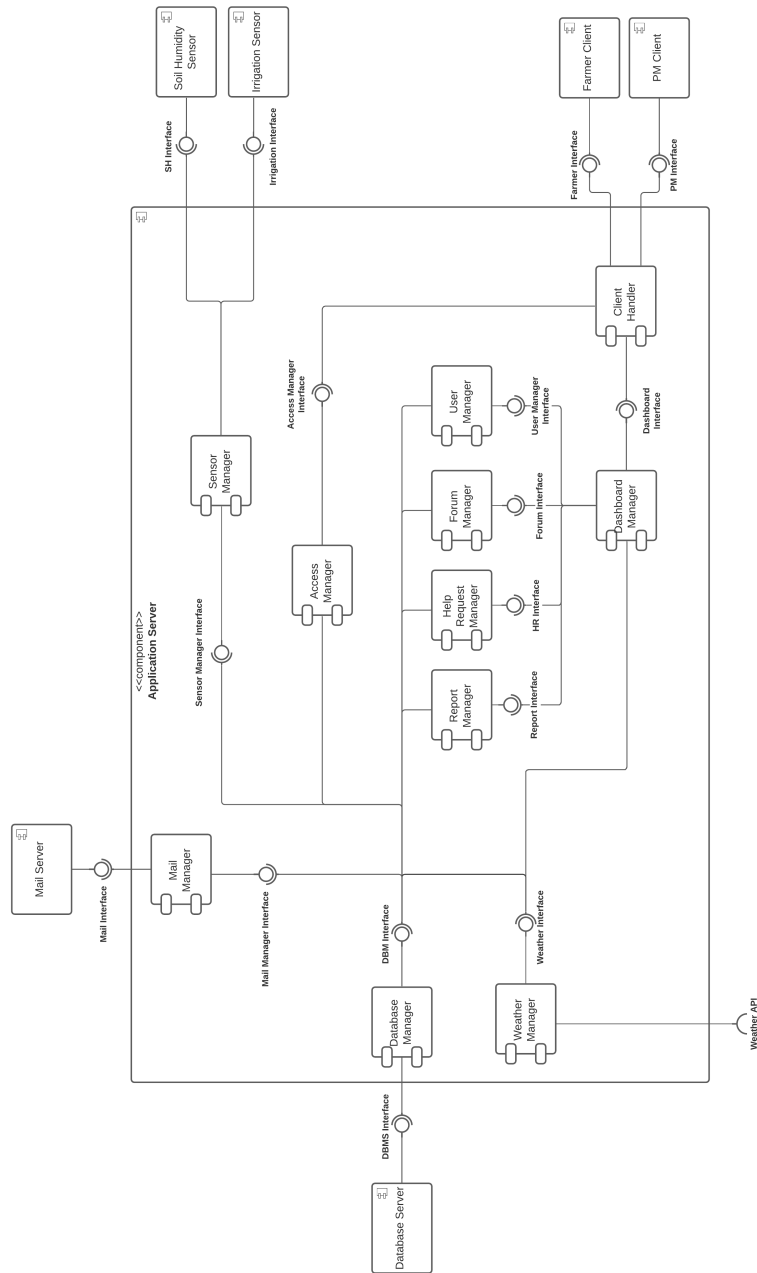


Figure 3: Application Server Component Diagram

- **Help Request Manager:** handles the creation, the handling and the visualization of help requests.
- **Forum Manager:** handles forum operations such as creation of threads, answering, visualization and moderation.
- **User Manager:** manages the retrieving of user information from the database.
- **Dashboard Manager:** manages the process of generation of web pages, retrieving the necessary data from the other interfaces, and the handling of user requests. This component will be further analyzed in the following part of the document.
- **Client Handler:** handles clients' connection, interfacing between the application server and the clients.

2.2.2 Client Handler

Figure 4 describes the internal structure of the Client Handler component. This component receives requests from clients and forwards them to the Dashboard Manager or to the Access Manager, accordingly to the type of request. The internal component that is key to this operation is the Request Handler, which unpacks the client request and calls the corresponding method provided by the Dashboard Interface or the Access Manager Interface.

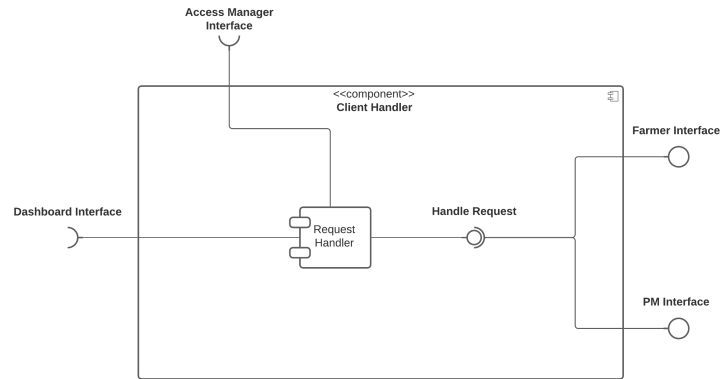


Figure 4: Client Handler Component Diagram

2.2.3 Dashboard Manager

Figure 5 describes the internal structure of the Dashboard Manager component. The Dashboard Manager works as a provider of webpages, forwarded then to the user by the Client Handler. Since each webpage is dynamically built by fetching data from the model, this component is interfaced with the various controller components such as Report and Help Request Managers through their external interfaces. Once a webpage is requested through the Dashboard Interface, the Dashboard Manager internal components proceed to build the webpage by making requests to various components and once the page has been built it is returned from the interface.

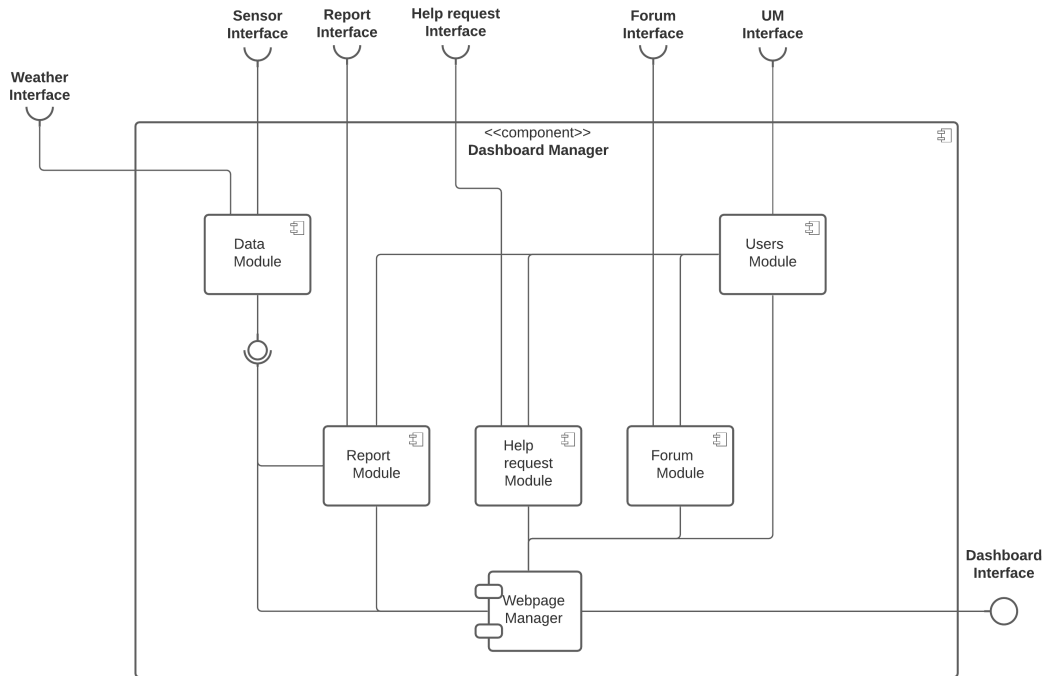


Figure 5: Dashboard Manager Component Diagram

2.3 Deployment view

Figure 6 describes the deployment components of the application. The system is implemented as a three-tier architecture, a standard software application architecture dividing the application into three logical and physical tiers that is present in most of client-server applications. The three tiers of the architecture consist in:

- **Tier 1: Presentation tier:** the presentation tier comprises the user interfaces and the communication layers, providing services to display data to end users and allowing them to interact with the application. In the DREAM deployment, this tier is composed of a web server installed on a dedicated server instance with Apache TomEE 8, responsible for the interaction with end users, and an MQTT broker that periodically interacts with the sensors installed on a dedicated broker server.
- **Tier 2: Application tier:** the application tier implements the business logic of the application, providing the necessary back-end services. This tier collects information from the presentation tier and interacts with the data tier through CRUD operations. The DREAM application server is installed on a dedicated server instance, running a Java Virtual Machine.
- **Tier 3: Data tier:** the data management tier provides database services accessed by the application tier and the presentation tier. The DREAM data management tier is installed on a dedicated MySQL server and accessed via JDBC and JPA.

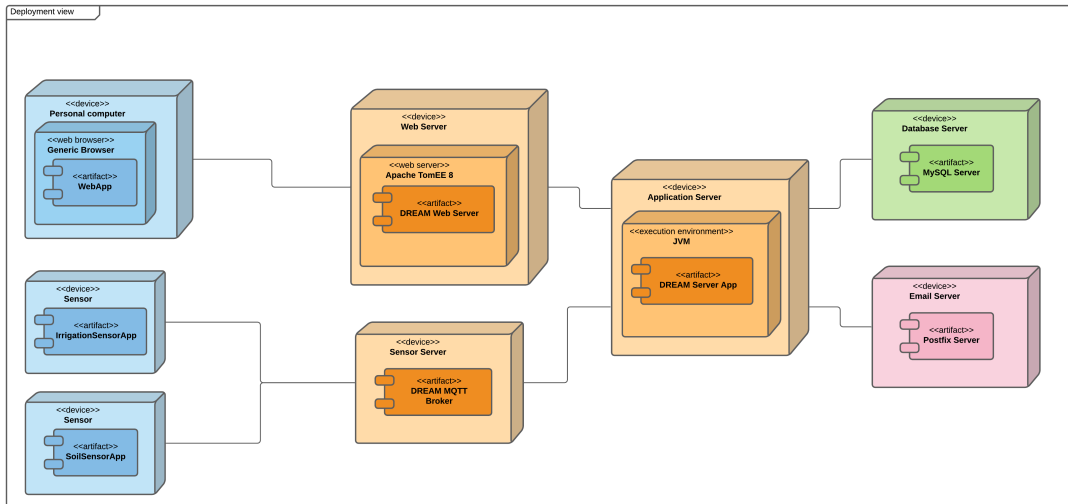


Figure 6: Deployment Diagram

2.4 Runtime view

This section aims to describe the interactions between components and their interfaces at runtime, detailing some crucial operations such as login and report insertion. It should be noted that these diagrams are meant to give an idea of the behaviour of the system and to provide a blueprint for the implementation of these processes. The methods used in the diagrams are the ones provided by each component's interface. Internal methods of a component are only shown if particularly relevant to the operation.

2.4.1 Registration

Figure 7 shows the registration procedure for both farmers and policy makers. These procedures differs in some points but the overall system behaviour is similar. For what concerns the farmer registration process, it begins with a request from the policy maker handled by the Access Manager: new credentials are generated, stored into the database and sent via e-mail by the Mail Manager component to the farmer. The policy maker registration process is almost the same, with the only difference being that the registration procedure is started by a system admin, requesting directly to the Access Manager to generate credentials for the PM. The generated credentials are then stored in the DB and sent via e-mail to the policy maker.

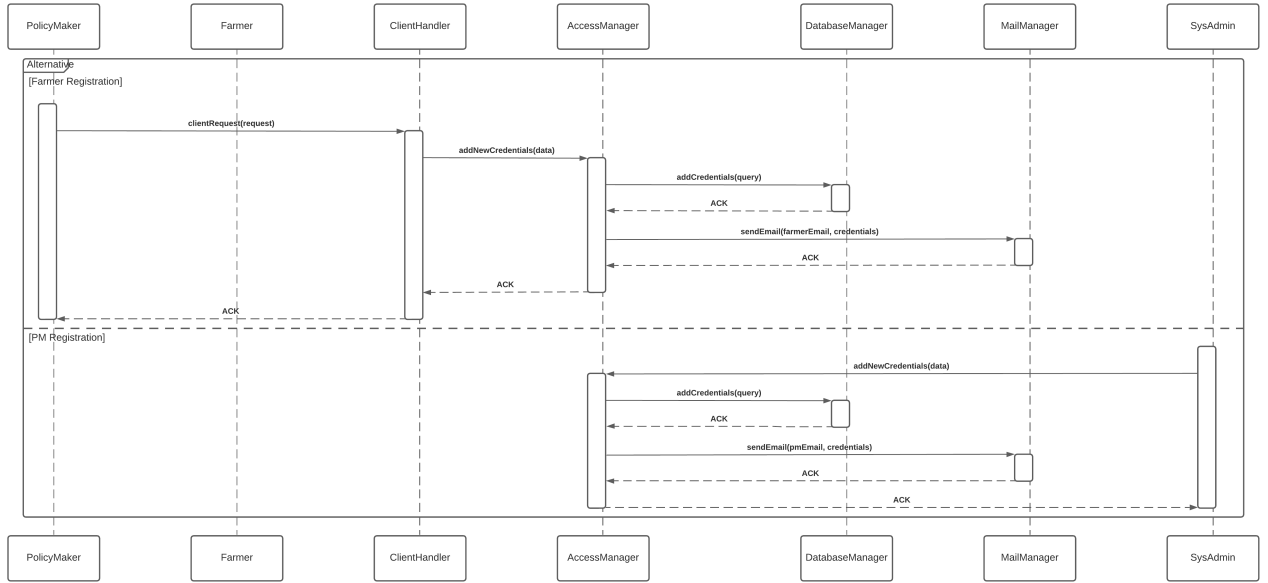


Figure 7: Registration runtime diagram

2.4.2 Login

Figure 8 describes the login procedure: the first request to the system is to get the login page, generated by the Dashboard Manager. Once the client has been provided with the page, it enters his login credentials and the login request is then forwarded to the Access Manager, that verifies the credentials through a DB query and returns the outcome of the request: if the credentials are correct, the user is redirected to his homepage, otherwise the login page gets newly displayed, this time with an error message.

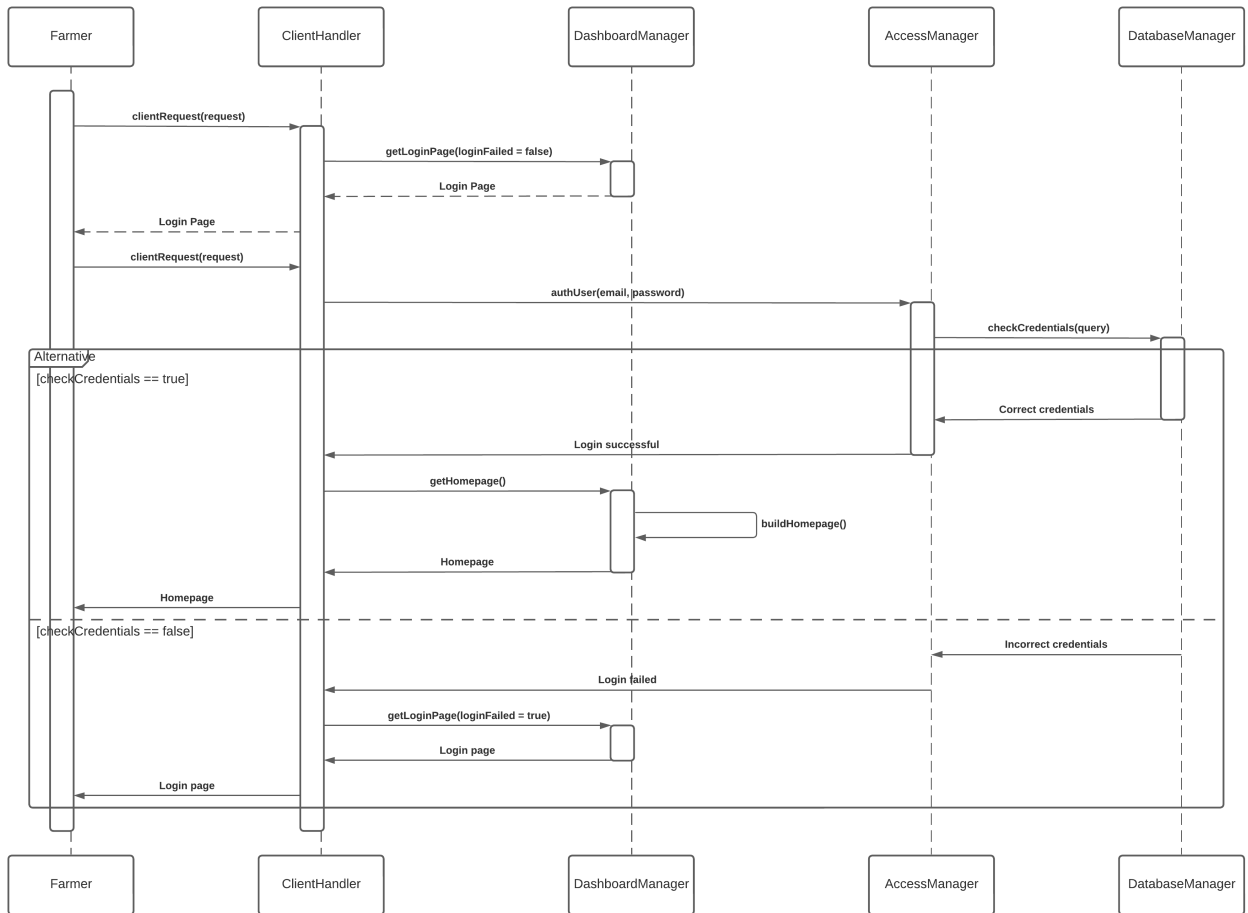


Figure 8: Login runtime diagram

2.4.3 Report insertion

In Figure 9 is described the report insertion procedure. Such as almost every other process, it starts with a request to the Dashboard Manager to get the webpage containing the form to insert a report. Once the user has compiled such form, the report is submitted and handled by the Report Manager. In this phase sensor data and weather data are retrieved from the database through the respective Sensor and Weather Managers and, through the `crossData()` operation, are binded to the farmer's report. Once this operation succeeds, the report is stored into the database via the Database Manager and the user is acknowledged of the success of the operation.

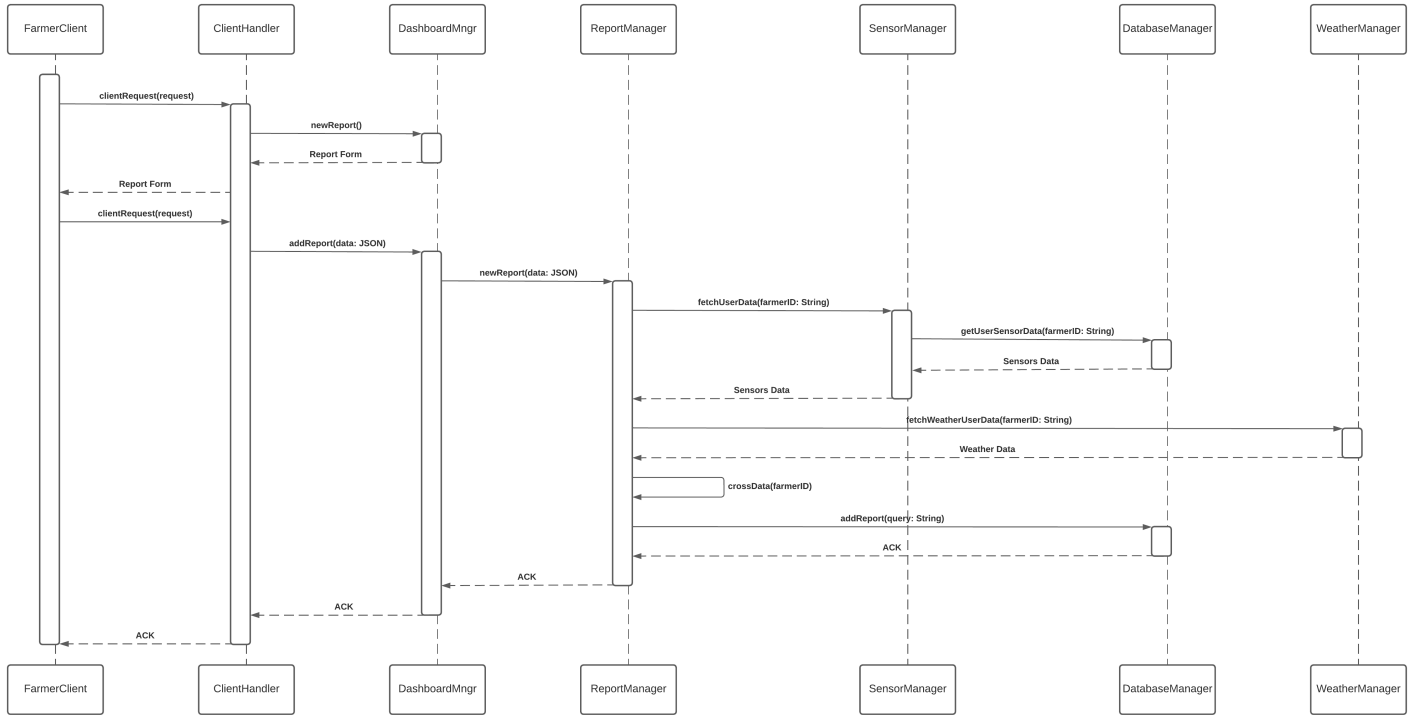


Figure 9: Report insertion runtime diagram

2.4.4 Creation and handling of a help request

In Figure 10 and Figure 11 is described the process of creation and handling of a help request.

Figure 10 describes the creation process, which starts with a request from the farmer to get the report creation form webpage, provided by the Dashboard Manager. Once the form is compiled, the help request is submitted and handled by the Help Request Manager, which stores it into the database. The user is then acknowledged of the success of the operation.

Figure 11 describes the handling of a newly created help request by the policy

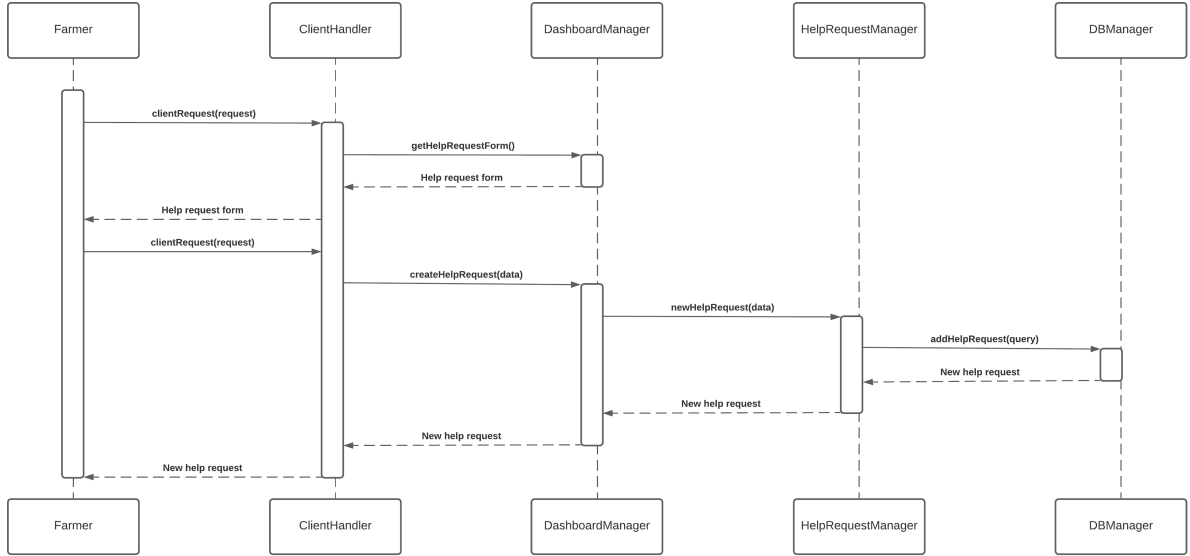


Figure 10: Help request creation runtime diagram

makers. This process starts by getting the list of help requests, provided as a webpage by the Dashboard Manager after fetching help requests from the database through the Help Request Manager. Once a help request is selected, the first operation is to open it. After that, the policy maker proceeds to evaluate its severity. If the severity is low, the Help Request Manager fetches from the database suggestions related to the help request farmer, and these suggestions are automatically sent to the farmer via e-mail by the Mail Manager. If the severity is high, control visits to the farmer are newly scheduled and the farmer is notified via e-mail.

The process ends with an evaluation of the farmer performances: if they are improved the help request is closed, otherwise its severity is increased and new actions are taken.

2.4.5 Forum interaction

Figure 12 describes a user interaction with the DREAM forum. A typical user/-forum interaction would start with the opening of the forum webpage, provided by the Dashboard Manager, where threads fetched by the Forum Manager are displayed. The next step is to open a thread discussion, which consists in a similar interaction as the previous one. If then the user intends to contribute to the discussion by posting an answer, the user request is handled by the Forum Manager, updating the thread and storing the contribution into the database. Once the operation has succeeded, the user is provided with the updated thread discussion webpage.

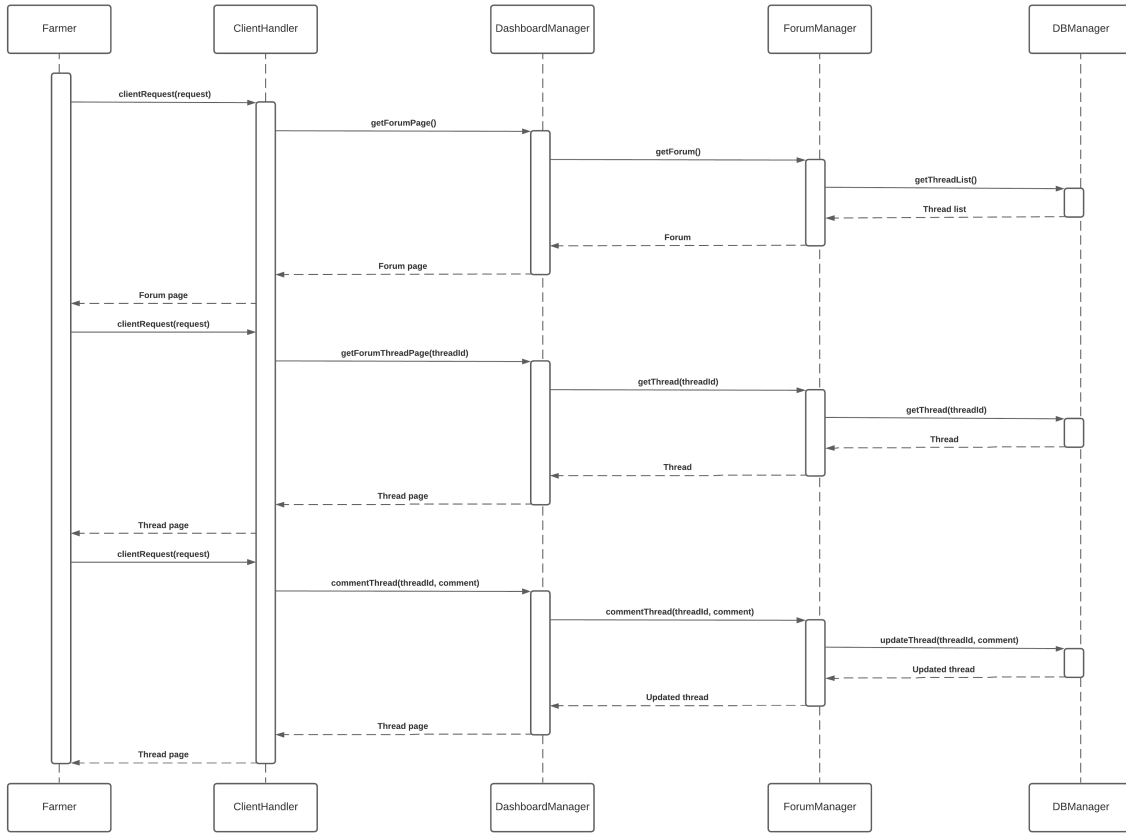


Figure 12: Forum interaction runtime diagram

2.5 Component Interfaces

This section lists all the methods exposed by each interface to the external components. Only internal components of the application server are described. For each method, its parameters and its return value are only meant to give an idea of its behaviour. The list of methods may not comprehend every method used by each interface at the end of the implementation, but should give an overall description of the main functionalities of the interfaces. Additional methods may be added, if necessary, during the implementation phase.

- **Farmer Interface**
 - clientRequest(httpRequest): HttpResponse
- **PM Interface**
 - clientRequest(httpRequest): HttpResponse
- **SH Interface**
 - postSensorData(sensorData): void
- **Irrigation Interface**
 - postSensorData(sensorData): void
- **Sensor Manager Interface**
 - fetchUserData(farmerID, date): List<SensorData>
 - storeSensorData(sensorData): void
- **Access Manager Interface**
 - authUser(email, password): User
 - getAuthenticatedUser(): User
 - addnewCredentials(data): void
- **Dashboard Interface**
 - getLoginPage(loginFailed): DashboardPage
 - getHomepage(): DashboardPage
 - buildHomepage(): DashboardPage
 - getReportPage(): DashboardPage
 - getReportForm(): DashboardPage
 - addReport(data): DashboardPage
 - getProductListPage(): DashboardPage
 - getProductPage(): DashboardPage
 - getForumPage(): DashboardPage

- getForumThreadPage(): DashboardPage
- createThread(data): DashboardPage
- commentThread(threadId, data): DashboardPage
- getFarmerPage(farmerId): DashboardPage
- getHelpRequestListPage(): DashboardPage
- getHelpRequestPage(helpId): DashboardPage
- getHelpRequestForm(): DashboardPage
- getForecastPage(): DashboardPage
- createHelpRequest(data): DashboardPage
- **User Manager Interface**
 - getUser(userId): User
 - getUserList(limit): List<User>
 - fetchFarmerPerformance(farmerID)
- **Forum Interface**
 - getForum(): Forum
 - getThread(threadId): Thread
 - commentThread(threadId, comment): Thread
- **HR Interface**
 - newHelpRequest(data): HelpRequest
 - fetchHelpRequests(): List<HelpRequest>
 - forwardVisitScheduling(farmerID, visitData): Date
 - forwardSuggestion(farmerID, suggestion)
 - closeHelpRequest(requestID)
 - openHelpRequest(requestId): HelpRequest
 - closeHelpRequest(requestID): HelpRequest
 - setSeverity(requestID, severity): HelpRequest
 - increaseSeverity(requestID): HelpRequest
- **Report Interface**
 - newReport(data): Report
 - crossData(farmerId): Report
- **Weather Interface**
 - fetchWeatherData(locationId): WeatherData

- **Mail Manager Interface**

- `sendEmail(emailAddress, mailData): JSONObject`

- **DBM Interface**

- `addCredentials(String query): void`
- `checkCredentials(email, password): boolean`
- `addReport(query): Report`
- `getUserSensorData(farmerID): List<SensorData>`
- `saveIrrigationData(query): SensorData`
- `getFarmerPerformance(query): List<FarmerPerformance>`
- `getHelpRequests(): List<HelpRequest>`
- `addHelpRequest(query): HelpRequest`
- `setHelpRequestStatus(query): HelpRequest`
- `setHelpRequestSeverity(query): HelpRequest`
- `getThreadList(): List<Thread>`
- `getThread(threadId): Thread`
- `updateThread(threadId, comment): Thread`

2.6 Selected Architectural styles and patterns

- **Three-Tier Architecture**

A three-tier architecture was chosen for the implementation of the DREAM platform: presentation, application, and data tier. The key three-tier benefit is improved scalability, since each element can scale horizontally and the application servers can be deployed on many machines. Also, the database does not make connections with every client, it only requires connections from a smaller number of application servers. This architecture also leads to implement thin clients.

Change management is easier and faster to execute, because program logic is implemented on the centralized server, so the modularity makes it easier to modify or replace one tier without affecting the other tier.

- **REST API**

It was decided to implement the interfaces intended for web services following the standards imposed by the REST architectural style. This choice was made for ease of development and increased scalability of the system, in fact, thanks to the flexibility of RESTful APIs, it will be possible to add components to the system or modify them easily.

- **Model-View-Controller**

The application software is implemented following the MVC design pattern, splitting the application logic into presentation logic (view), business

logic (controller) and data management logic (model). This pattern follows the deployment architecture and allows an easy defining of each component's tasks. In detail, the design pattern implemented in the DREAM application does not follow the standard MVC, where there is also interaction between the View and the Model, but the Cocoa MVC pattern where this kind of interaction does not exist and each communication between View and Model is first processed by the Controller layer.

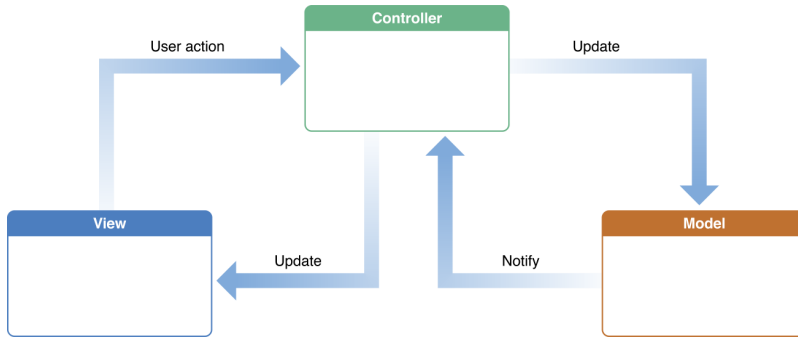


Figure 13: Cocoa Model-View-Controller

- **Event based communication for sensors**

The communication protocol chosen to retrieve data created by IoT sensors is Message Queue Telemetry Transport (MQTT), a standard messaging protocol for the Internet of Things that is extremely lightweight and ideal for connecting remote devices with a small code footprint and minimal network bandwidth. MQTT is a publish/subscribe communication protocol: A topic will be created for each farmer, then the sensors, depending on their geographical location, will publish data to the topics corresponding to the farmers of their interest. In this way it will be possible to access the data created by the sensors divided by each farmer. A MQTT Broker Cluster distributed in the region of Telangana, will take care of the management of sensor data. The application server will be subscribed to all topics and will receive periodically the data, it will then store them in the database to make them always accessible for processing. This design provides high scalability because, if we had to resort to replication of the application server, it will be enough to subscribe to all topics the new server to receive all sensor data.

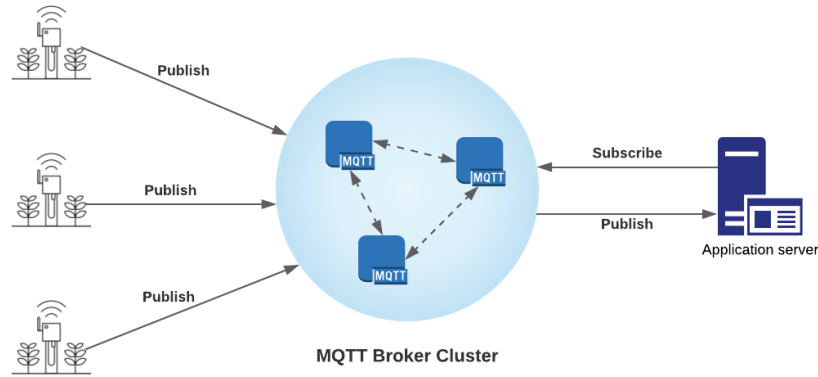


Figure 14: Distributed MQTT Broker System

2.7 Other design decisions

- **Thin Clients**

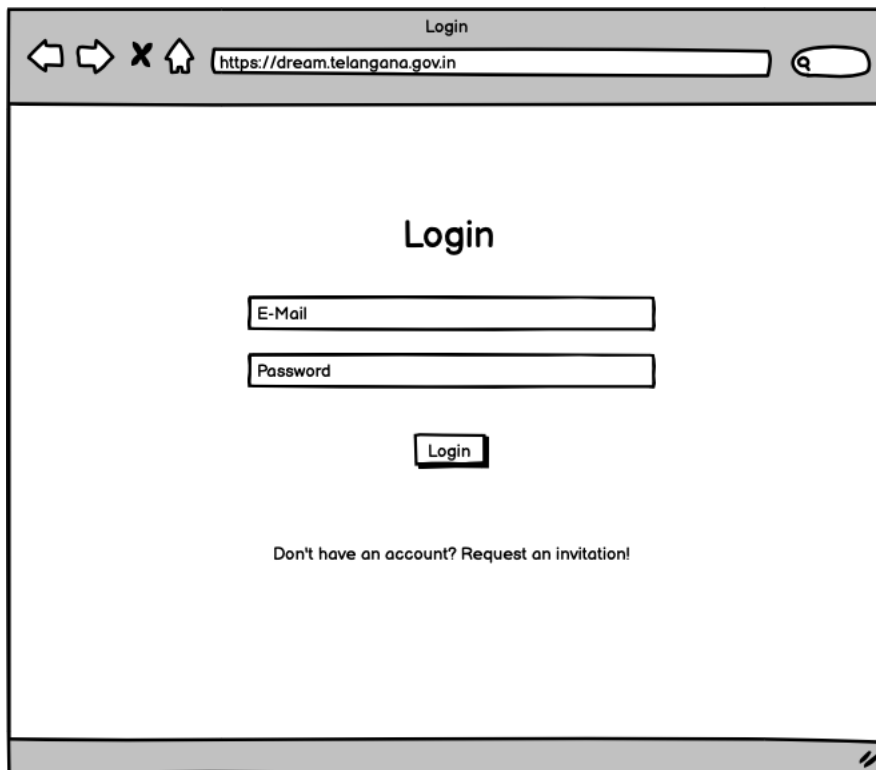
The architecture has been designed so that the end-points of the system are thin clients, i.e. they use resources hosted within a central server, which therefore do not need an high computational power. A thin client connects to a server-based environment that hosts the majority of applications, memory, and sensitive data the user needs. It was decided to undertake this solution to ensure that farmers who do not have powerful devices can use the platform without complications.

3 User Interface Desgin

The following section describes the user interface of the DREAM webapp. Its goal is to motivate design choices, describe each mockup, its content and the flow between the various pages of the application. The section is divided in three subsections: the first one focuses on the farmer's webapp, the second one on the Policy Maker's webapp and the third on the flow of the webapp.

3.0.1 Login

The first page for both farmers and policy makers will always be the login page, requiring e-mail and password to login into the platform. For farmers, this page will be accessible from any web browser and will also show an option to request an invitation: a simple popup will open up requesting the user to insert his e-mail address, which will be then submitted to the policy makers who will handle the registration request and procedure.



The mockup shows a web browser window with the title 'Login'. The address bar contains 'https://dream.telangana.gov.in'. The main content area has the heading 'Login' centered. Below the heading are two input fields: 'E-Mail' and 'Password'. A 'Login' button is positioned below the password field. At the bottom of the form area, there is a link that says 'Don't have an account? Request an invitation!'. The browser window has a standard toolbar with back, forward, and search icons.

Figure 15: Login page

3.1 Farmers User Interface

3.1.1 Homepage

The first page visualized by the farmer once logged in the platform is the homepage. The farmer's homepage contains a brief overview of weather forecasts and sensor data, and allows users to keep track of their crop in a table containing information about crop's type, eta and comments. The user can add crops to the table by clicking "Add crop", which will open a popup to insert the new crop and its info. The page then allows to navigate to the pages for creating reports and help request, to the forum or to a specific product page by searching the product by its name.

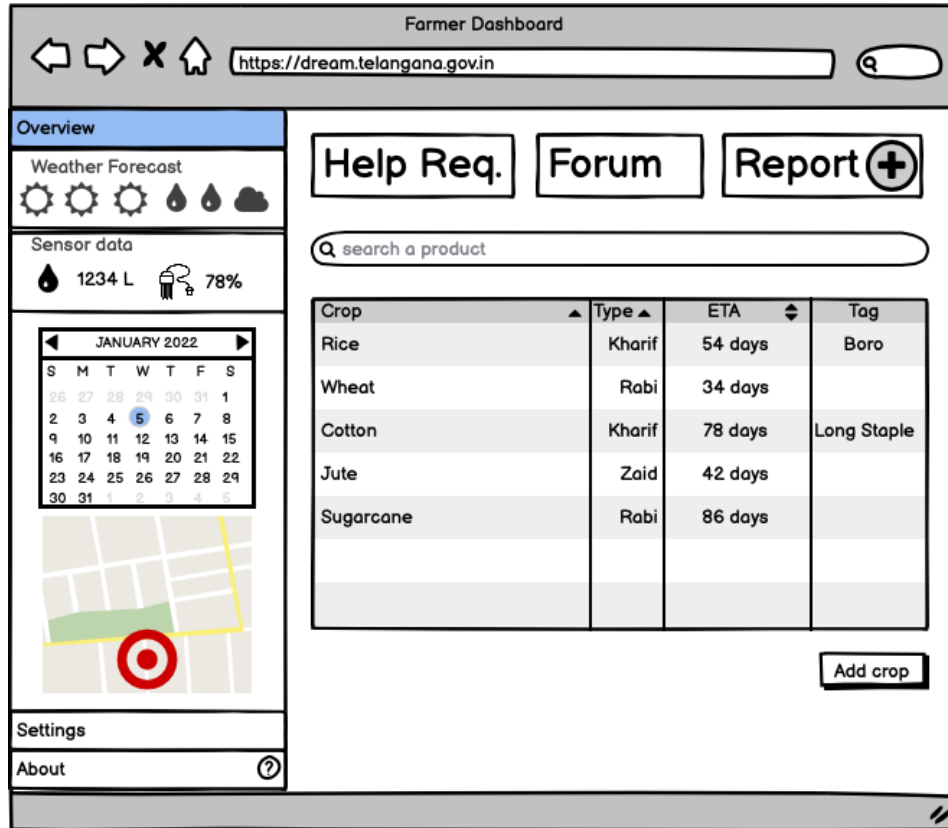


Figure 16: Farmer homepage

Farmer Dashboard

X

https://dream.telangana.gov.in

Product details

Product ID

P1555

Product Name

Carrot

Suggestions

Sugg.ID	Date	Suggestion
SG123	2021-05-12	Lorem ipsum dolor sit amet
SG456	2021-07-13	Lorem ipsum dolor sit amet
SG789	2021-08-01	Lorem ipsum dolor sit amet

Figure 17: Farmer's product page

3.1.2 Report submission

The report creation page consists of a simple form to insert the harvested products and their quantity. Each product added to the report is visualized on a table, that also allows to edit or delete previously added products. To add a product, the user selects the type of product by an interactive dropdown menu that also allows searching the product, then adds seeds and produced quantity. Clicking on "Auxiliary Products" will open a popup showing the auxiliary products used in the harvesting process, allowing the user to add, edit or remove them. Once the user has inserted all the necessary information, by clicking on "Add product" the product will be added to the report's products. The page also contains a text area where the user can insert the problems encountered during the harvesting process. Once the report is fully compiled, clicking on "Submit" will create the report and send it to the DREAM server. The webapp will then redirect the user to the homepage.

Farmer Dashboard

https://dream.telangana.gov.in

Insert report 29/02/2022

Select crop

Seeds Quantity

Produced Kg

Auxiliary products Add product

Product	Edit	Delete
Carrot	Edit	Delete
Wheat	Edit	Delete
Tomato	Edit	Delete

Problems

B I U style

Submit

Figure 18: Farmer's report creation page

3.1.3 Forum

The forum page shows all the open threads, allowing the user to open them and visualize the discussion. A farmer can also create a new thread by clicking "Create thread", which will redirect the user to the thread creation page.

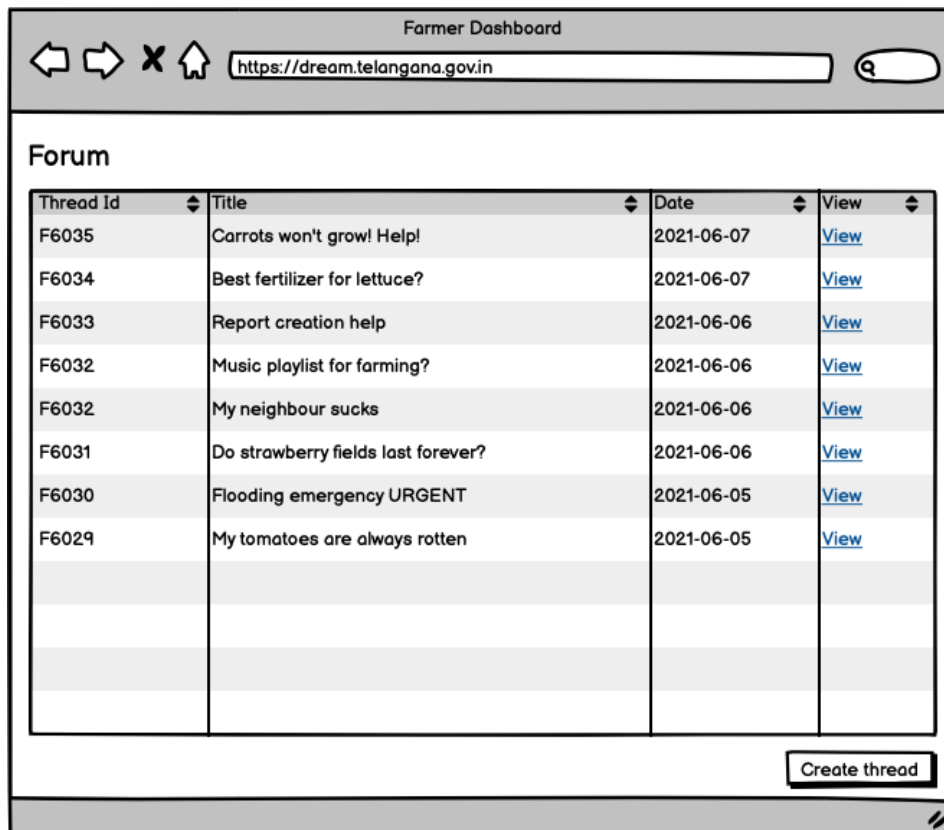


Figure 19: Farmer's forum page








Farmer Dashboard

https://dream.telangana.gov.in

Create thread

Thread title

Comment

B I U  style      

Create thread

Figure 20: Farmer's thread creation page

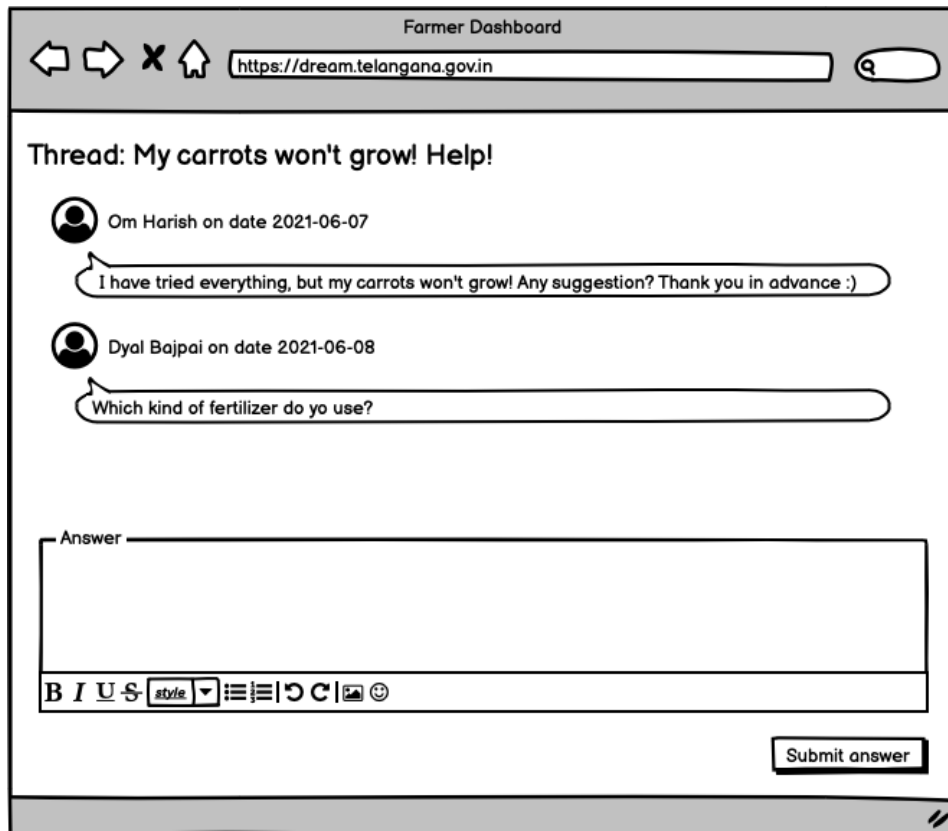


Figure 21: Farmer's thread page

3.1.4 Help Requests

The screenshot shows a web browser window titled "Farmer Dashboard". The address bar displays "https://dream.telangana.gov.in". The main heading is "Create help request". To the right of the heading is a date field showing "29/02/2022" and a calendar icon. Below the heading is a large text area labeled "Description of the help request". At the bottom of this text area is a rich text editor toolbar with buttons for Bold (B), Italic (I), Underline (U), Strikethrough (ABC), a style dropdown menu, bulleted list, numbered list, indent, outdent, link, unlink, image, and emoji. A "Submit" button is located at the bottom right of the form area.

Figure 22: Farmer's help request creation page

3.2 Policy maker User Interface

3.2.1 Homepage

The first page visualized by the policy maker once logged into the platform lists the farmers and the latest reports submitted, allowing the user to search farmers/reports. The user can then navigate to the farmer/report details. On the top of the page there is a navigation bar that allows the user to switch dashboard page.

The screenshot displays the 'Policy Maker Dashboard' interface. At the top, there is a navigation bar with tabs for 'Farmers', 'Forecasts', 'Help Requests', 'Forum', and 'Products'. The 'Farmers' tab is currently selected. Below the navigation bar, the dashboard is divided into two main sections: 'Farmers List' and 'Latest Reports'. Each section has a search bar and a table of data.

Farmers List

Search by ID or Name

Id	Name	Profile
1020	Jeet Valimbe	View Profile
1021	Krishan Bajpai	View Profile
1022	Om Harish	View Profile
1023	Dyal Bajpai	View Profile
1024	Arjuna Bandyopadhyaya	View Profile
1025	Ishwar Goyal	View Profile
1026	Aman Mahanti	View Profile

Latest Reports

Search by ID or Date

Id	Date	Farmer ID	View
R4501	2021-08-10	1020	View
R4502	2021-08-10	1021	View
R4503	2021-08-10	1022	View
R4504	2021-08-11	1025	View
R4505	2021-08-11	1021	View
R4506	2021-08-11	1026	View
R4507	2021-08-12	1020	View

Figure 23: Policy maker's farmers page

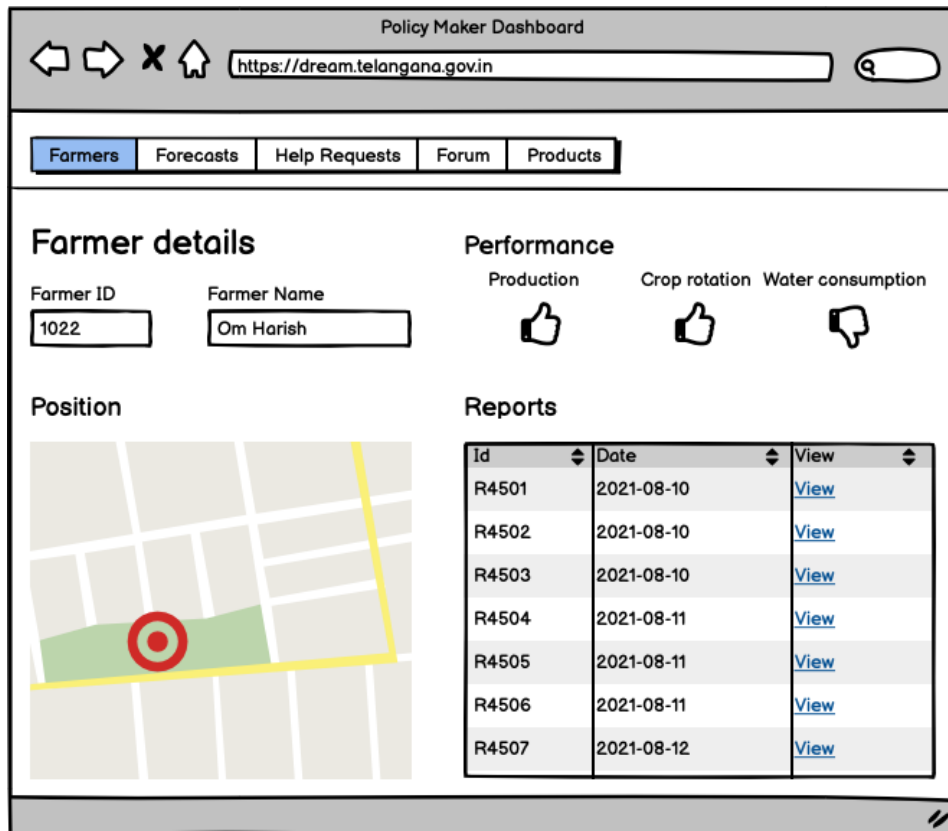


Figure 24: Policy maker's farmer detail page

Policy Maker Dashboard

https://dream.telangana.gov.in

Farmers

Forecasts

Help Requests

Forum

Products

Report - ID: R4501

Farmer ID

Farmer Name

Date

1020

Jeet Valimbe

2021-08-10

Prod.ID	Product	Quantity	Details
P1555	Carrots	15kg	View
P1302	Tomatoes	5kg	View
P1001	Wheat	50kg	View

Sensor Data

Weather Data

View farmer

Figure 25: Policy maker's report page

34

Policy Maker Dashboard
https://dream.telangana.gov.in

Farmers
Forecasts
Help Requests
Forum
Products

Report - ID: R4501 - Product details

Product ID

Product Name

Quantity

P1555

Carrot

15kg

Auxiliary products

Prod.ID	Product	Quantity
P1201	Fertilizer	0.2kg
P1002	Water	30lt
P1403	Pesticide	0.1kg

Back to report

Figure 26: Policy maker's report production detail page

The forecast page contains basic information about the future weather forecasts. Production forecasts are calculated and then shown to the user through graphs.

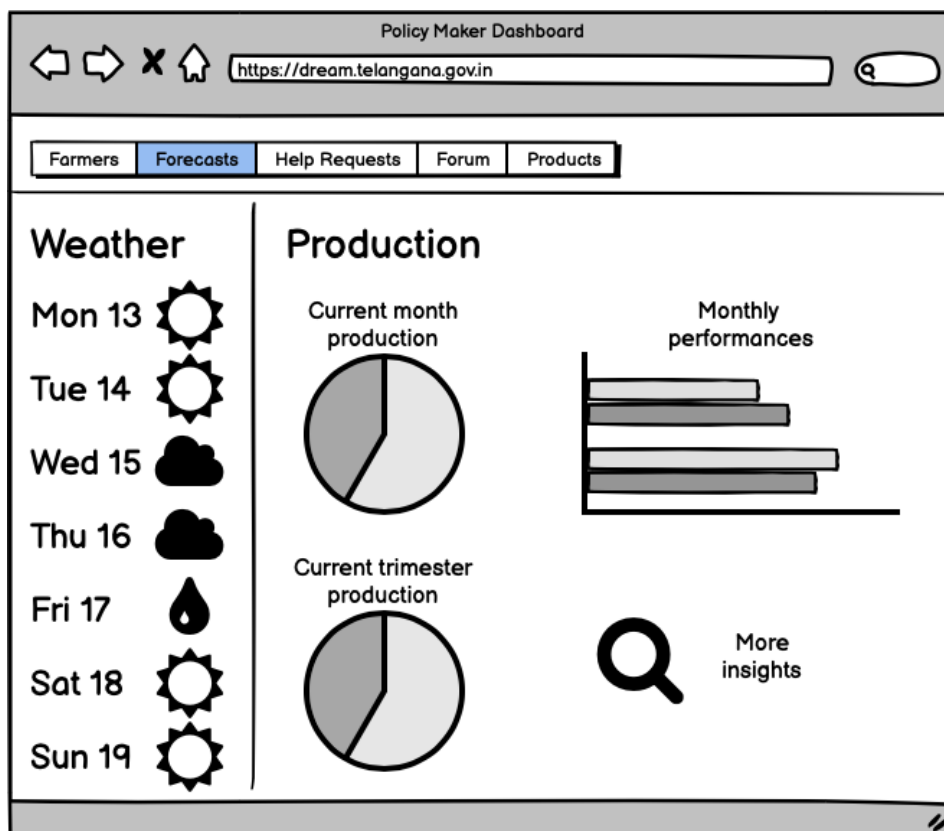


Figure 27: Policy maker's forecasts page

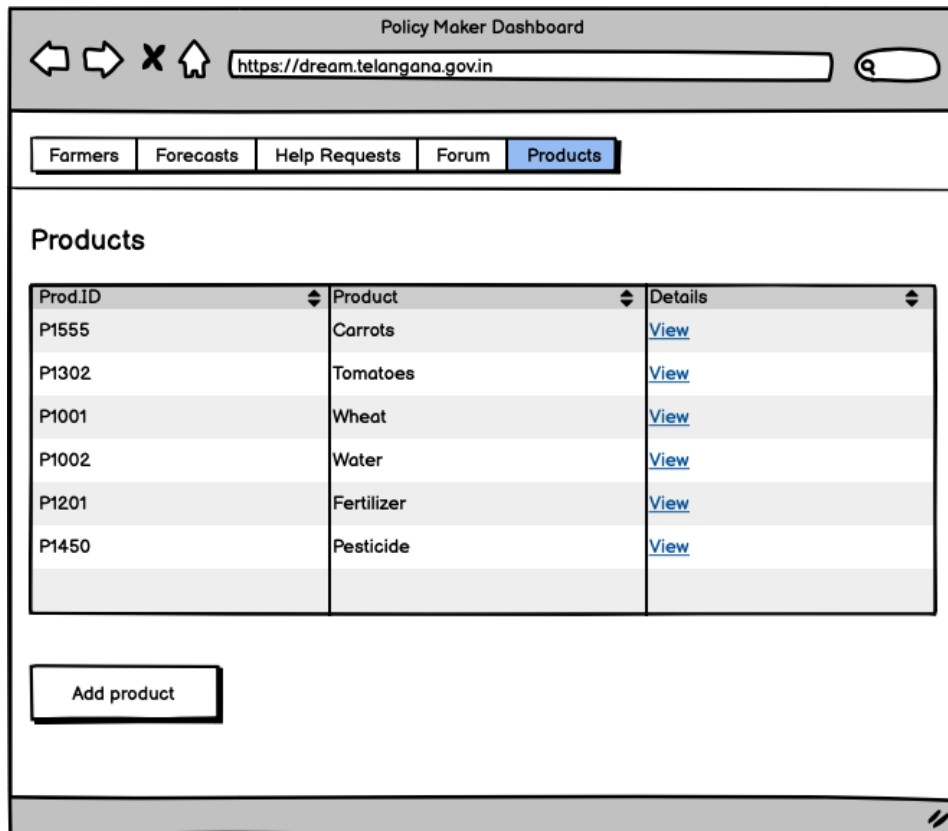


Figure 28: Policy maker's products page

Policy Maker Dashboard

https://dream.telangana.gov.in

Farmers

Forecasts

Help Requests

Forum

Products

Product details

Product ID

Product Name

Quantity

P1555

Carrot

15kg

Suggestions

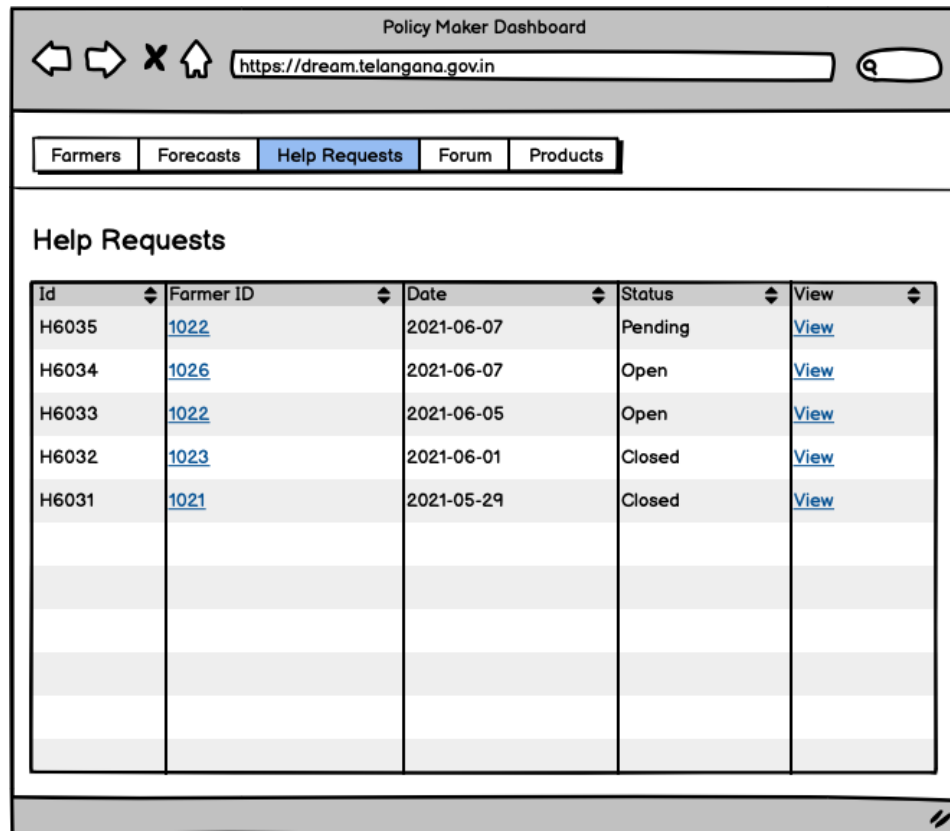
Sugg.ID	Date	Suggestion
SG123	2021-05-12	Lorem ipsum dolor sit amet
SG456	2021-07-13	Lorem ipsum dolor sit amet
SG789	2021-08-01	Lorem ipsum dolor sit amet

Add suggestion

Figure 29: Policy maker's product detail page

38

3.2.2 Help Requests



Policy Maker Dashboard

https://dream.telangana.gov.in

Farmers Forecasts **Help Requests** Forum Products

Help Requests

Id	Farmer ID	Date	Status	View
H6035	1022	2021-06-07	Pending	View
H6034	1026	2021-06-07	Open	View
H6033	1022	2021-06-05	Open	View
H6032	1023	2021-06-01	Closed	View
H6031	1021	2021-05-29	Closed	View

Figure 30: Policy maker's help requests page

Policy Maker Dashboard

https://dream.telangana.gov.in

Farmers

Forecasts

Help Requests

Forum

Products

Help Request - ID: H6035

Farmer ID

Farmer Name

Date

Severity

1022

Om Harish

2021-06-07

High

Help Request

Hello,

I am Om Harish. The last week bad weather...

....

....

Sincerely, Om Harish

Open help procedure

Figure 31: Policy maker's help request detail page

3.2.3 Forum

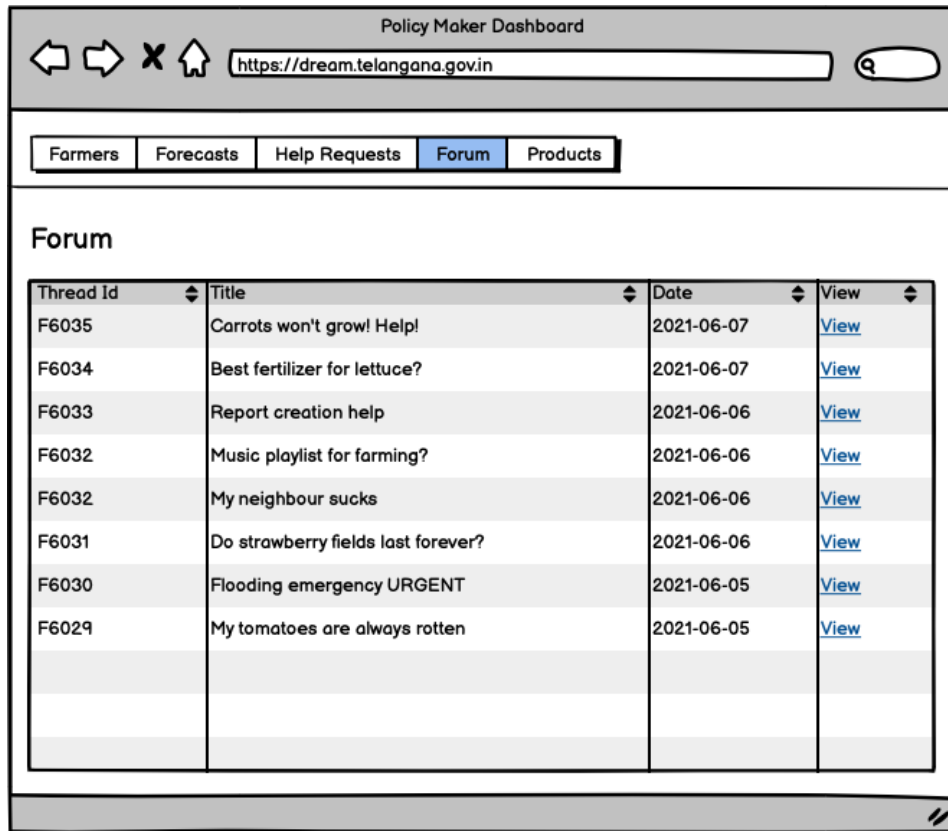


Figure 32: Policy maker's forum page

3.3 Flow of the User Interface

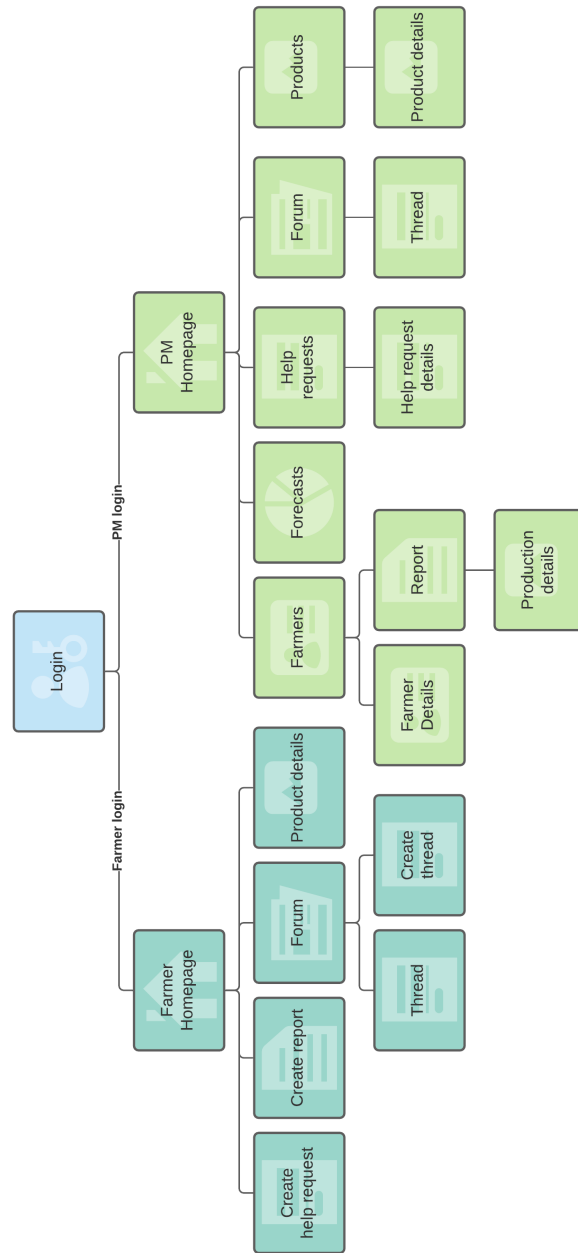


Figure 33: User interface flow

4 Requirements Traceability

This section associates the requirements set forth in the DREAM RASD document with the components designed to meet them.

Please note that, for each requirement, only the main components designed to satisfy it have been indicated, in fact the auxiliary components (e.g. Client Handler, Farmer Client, ...) and the sub-components are omitted since they are already described in the previous sections.

- R1** The system shall allow system admins to register policy maker.
Access Manager, Mail Manager
- R2** The system shall allow policy makers to login.
Access Manager
- R3** The system shall allow policy makers to register farmers.
Access Manager
- R4** The system shall provide farmers credentials to log in.
Access Manager, Mail Manager
- R5** The system shall allow farmers to complete their profile information.
Access Manager
- R6** The system shall allow farmers to log into the platform.
Access Manager
- R7** The system shall allow farmers to fill reports with data about goods production.
Report Manager
- R8** The system shall allow farmers to fill reports with data about auxiliary products used for goods production.
Report Manager
- R9** The system shall allow farmers to fill reports with problems faced during goods production.
Report Manager
- R10** The system shall retrieve data from sensors and attach it to reports.
Report Manager, Sensor Manager, Soil Humidity Sensor, Irrigation Sensor
- R11** The system shall retrieve data from weather stations and attach it to reports.
Report Manager, Weather Manager
- R12** The system shall allow farmers to visualize weather forecasts.
Weather Manager

- R13** The system shall allow farmers to visualize data retrieved from humidity and irrigation sensors.
Sensor Manager, Soil Humidity Sensor, Irrigation Sensor
- R14** The system shall allow farmers to create and contribute to forum discussions.
Forum Manager
- R15** The system shall allow farmers to create help requests.
Help Request Manager
- R16** The system shall allow farmers to visualize personalized suggestions about goods and techniques.
Help Request Manager
- R17** The system shall allow PMs to monitor farmers' performances.
User Manager
- R18** The system shall provide PMs parameters to evaluate farmers' performances.
User Manager
- R19** The system shall allow PMs to compare and aggregate farmers' performances.
User Manager, Dashboard Manager
- R20** The system shall allow PMs to filter farmers based on their information.
User Manager, Dashboard Manager
- R21** The system shall allow PMs to filter farmers based on their performance.
User Manager, Dashboard Manager
- R22** The system shall allow PMs to sort farmers based on their information.
User Manager, Dashboard Manager
- R23** The system shall allow PMs to sort farmers based on their performance.
User Manager, Dashboard Manager
- R24** The system shall allow PMs to reward best performing farmers.
User Manager, Dashboard Manager, Mail Manager, Mail Server
- R25** The system shall allow PMs to visualize weather forecasts.
Weather Manager
- R26** The system shall allow PMs to visualize help requests.
Help Request Manager
- R27** The system shall allow PMs to take charge of help requests.
Help Request Manager

- R28** The system shall allow PMs to visualize farmers who have requested help and their performances.
Help Request Manager, User Manager
- R29** The system shall allow PMs to close help requests.
Help Request Manager
- R30** The system shall allow PMs to fill reports about farmers interviews.
User Manager
- R31** The system shall allow PMs to re-schedule agronommist's control visit plans.
Help Request Manager
- R32** The system shall allow PMs to create and contribute to forum discussions.
Forum Manager
- R33** The system shall allow PMs to create suggestions.
Help Request Manager

5 Implementation, Integration and Test Plan

After giving a complete and detailed description of the system, its structure and its functionalities, this section provides a complete plan to develop, implement and test the application.

The implementation, integration and testing of the system will follow a bottom-up approach, keeping track of the dependencies between components and their interactions. Since the proposed solution involves a thin client, the main focus of this section will be on the application server and its sub-components. External services will not be analyzed and do not require unit testing.

5.1 Plan definition

In this section is described the implementation plan of the server-side of the application. The server implementation is divided into stages to facilitate the testing of the implemented components, and the testing of the server in its totality will be only done after the last stage of the implementation plan.

The following table presents the development stages, summarizing the development process and the execution order. For each step, a forecast of development difficulties has also been noted.

Order	Development stage	Dev. difficulty
1	Database Manager	High
2	Access Manager and User Manager	Low
3	Sensor Manager	Medium
4	Weather Manager	Low
5	Mail Manager	Low
6	Report Manager, Help Request Manager and Forum Manager	Medium
7	Dashboard Manager	High
8	Client Handler	Medium
9	Client-side and Server-side integration	Low

Table 1: Development stages

The first step (Figure 34) is the implementation of the Database Manager. It is crucial to implement this component as soon as possible since it is responsible for every operation involving the database.

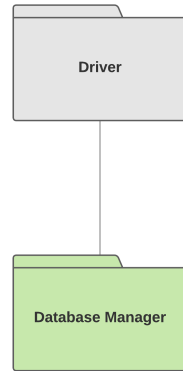


Figure 34: Development stage 1

The second step (Figure 35) is the implementation of Access Manager and User Manager. These components do not require any dependencies if not the database manager already implemented. The same can be said for the next 3 implementation stages (Figure 36, Figure 37, Figure 38), which involve the implementation of Sensor Manager, Weather Manager and Mail Manager.

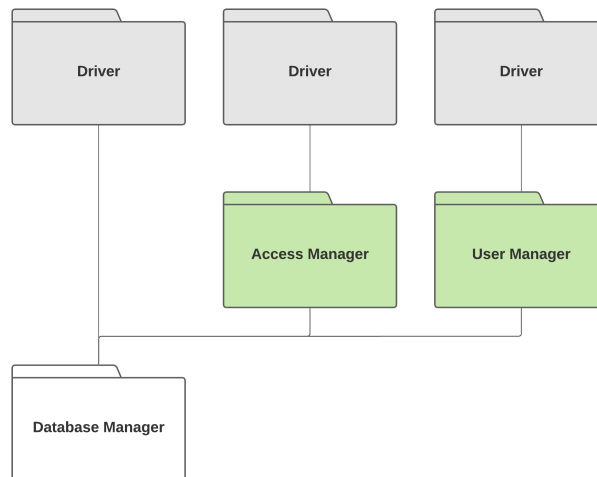


Figure 35: Development stage 2

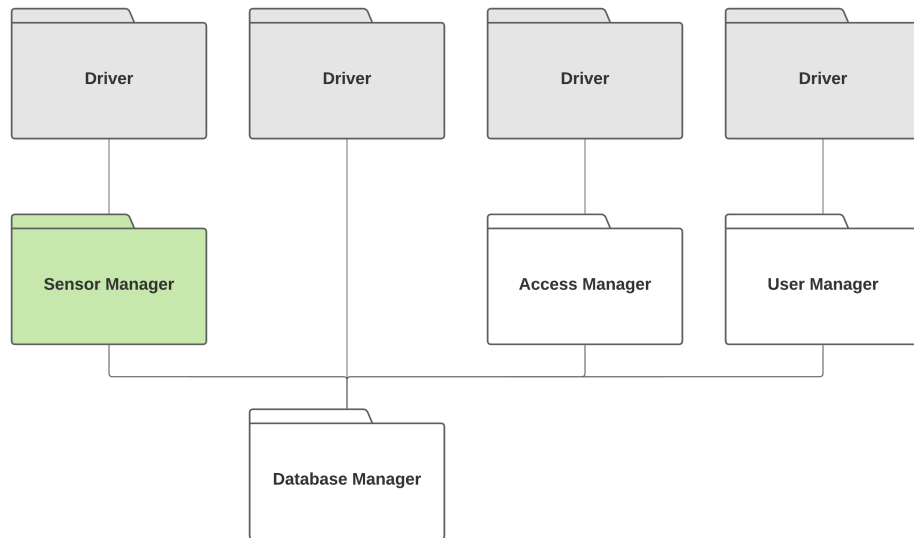


Figure 36: Development stage 3

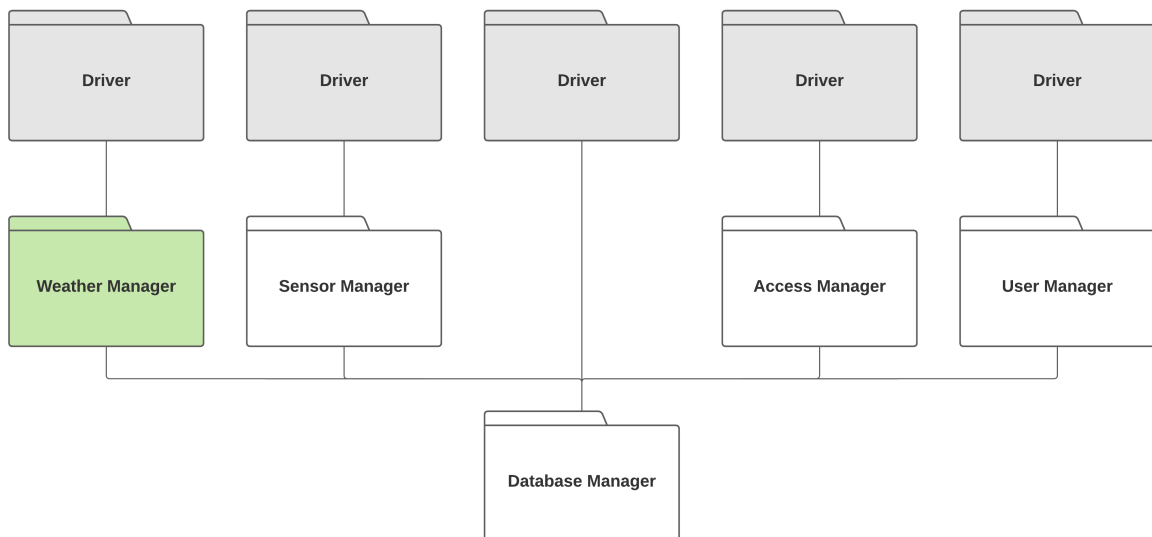


Figure 37: Development stage 4

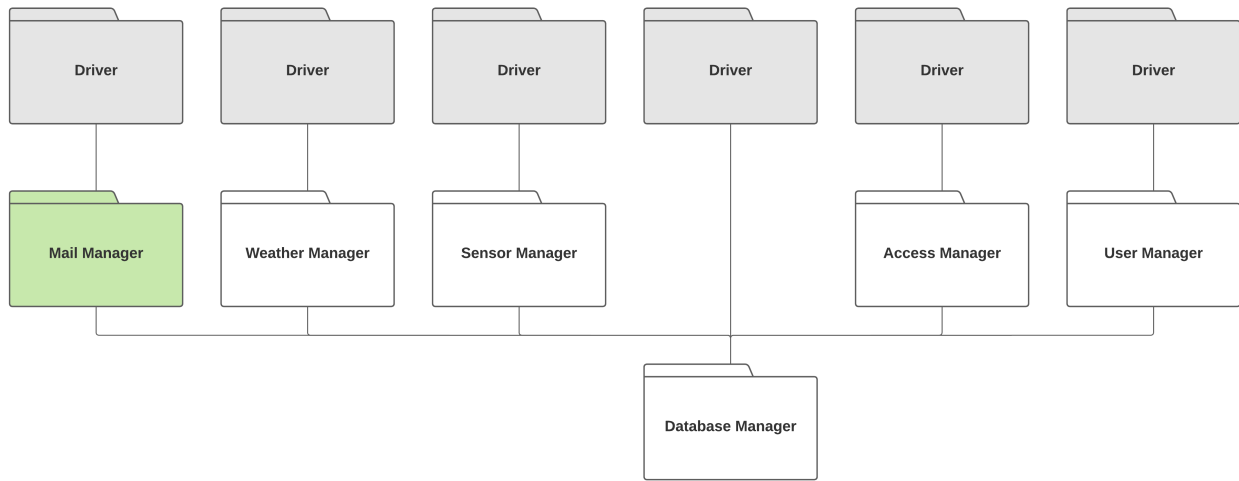


Figure 38: Development stage 5

The sixth step (Figure 39) consists of the implementation of the main controllers of the system: Report Manager, Help Request Manager and Forum Manager.

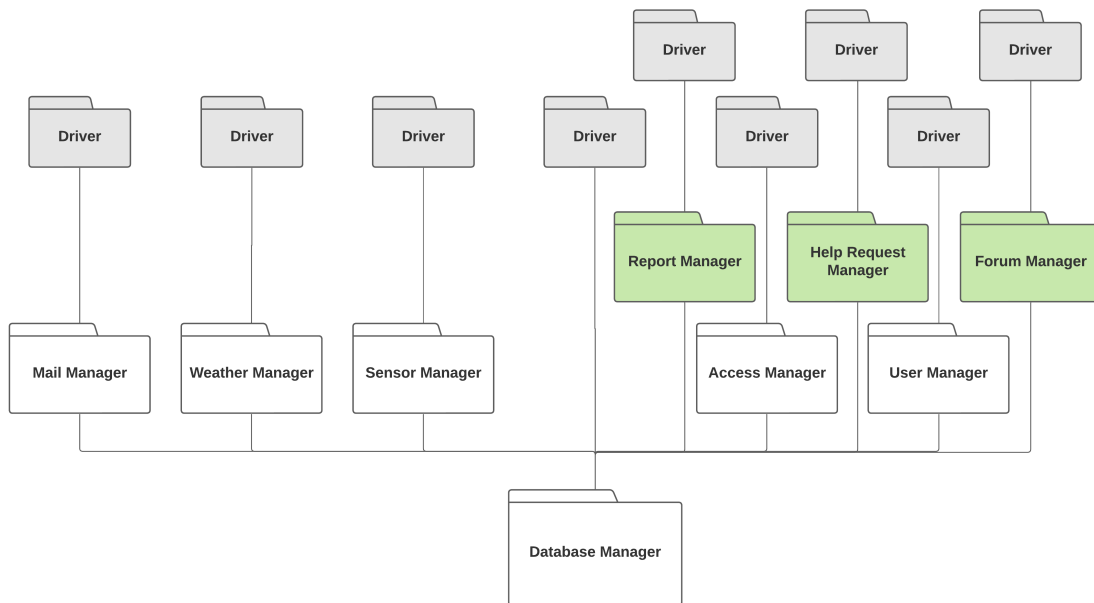


Figure 39: Development stage 6

Once the controllers have been implemented, it is possible to start implementing the components responsible for the user interaction (MVC): in the seventh step (Figure 40) the Dashboard Manager is implemented, and after that it is possible to implement also the Client Handler (Figure 41).

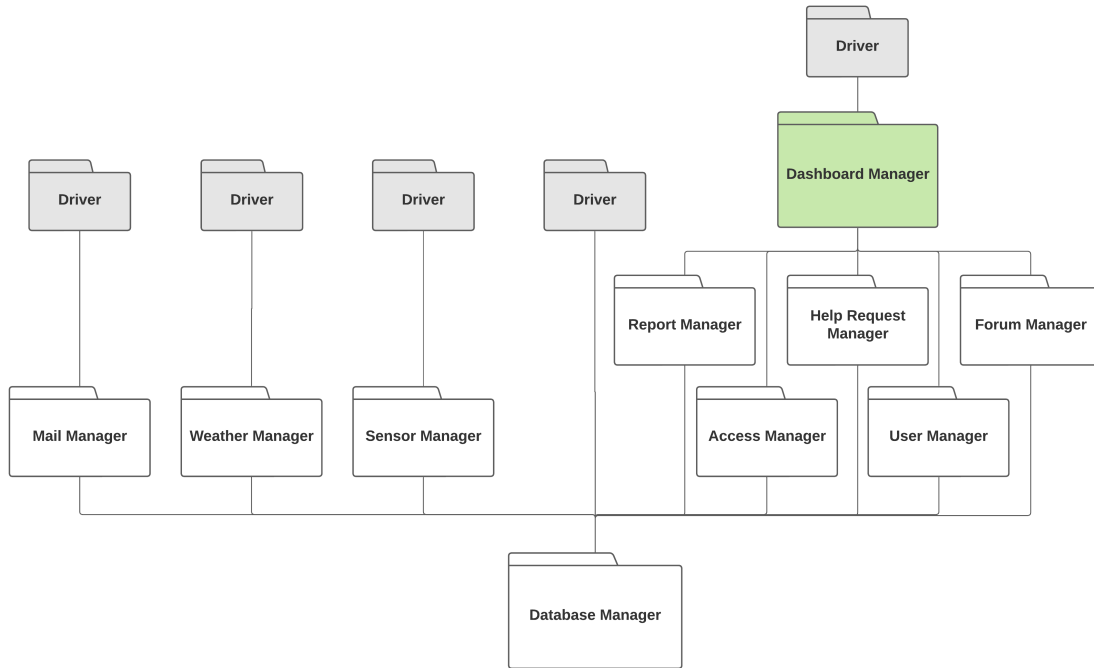


Figure 40: Development stage 7

After every server component has been implemented, it is possible to integrate the client-side and the server-side.

5.2 Development tools

The main language chosen for the implementation of the back-end of the application is **Java**. Other programming languages are also worth considering, but the decision to use Java is based on the popularity and the reliability of Java applications. Furthermore, one of Java main strengths is the support for database interaction: tools such as **JPA** allow an easy and reliable database-application interaction.

Java is a viable option for the realization of the application front-end aswell, but given its actual popularity for this kind of softwares, the chosen language is **JavaScript** with the **Angular** framework. This grants an up-to-date solution, a modern framework studied for modern web applications but with enough support to be considered reliable.

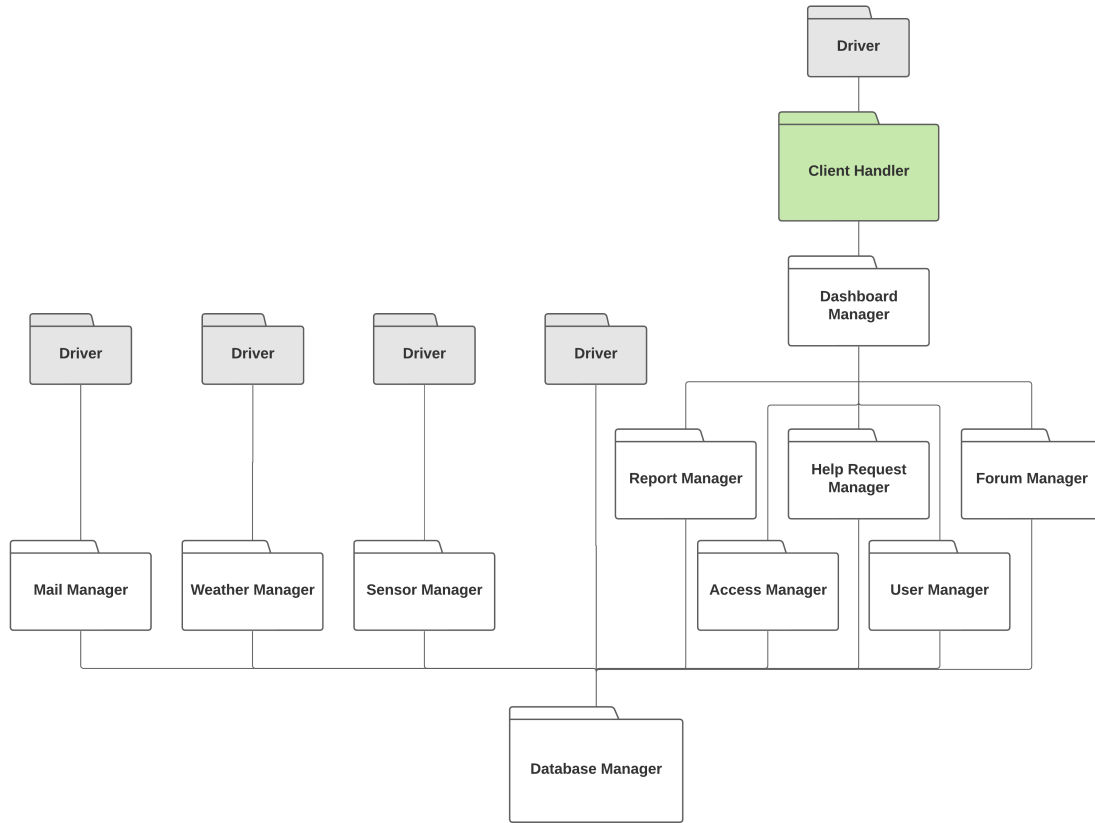


Figure 41: Development stage 8

The communication between back-end and front-end will be implemented via **JSON**, already implemented in JavaScript and heavily supported in Java through libraries such as Google's *GSON*.

5.3 System testing

During implementation, the developed software must be tested step-by-step with unit tests before being incorporated into the system. After the entire system has been properly developed and integrated, testing of the entire infrastructure will be required to verify its integrity and correctness. At this point the test environment should be as close to the production environment as possible, the system tests to be performed are[3]:

- **Functional Testing**

The application is tested to verify that it meets all requirements, i.e., that it is able to function properly under certain conditions. This is a black-box

testing methodology, i.e., the internal workings of the tested code are not taken into account, nor are the steps that lead to the result, but only the final result is focused on.

The tool used for this phase must support running scripted test automation, simulating the behavior of farmers and PMs in all scenarios.

- **Performance and Load Testing**

This testing phase is needed to identify bottlenecks affecting response time, utilization, throughput and to do benchmarking, in order to establish a baseline thanks to which it will be possible to make comparisons for possible future integrations.

Using the data available to the Telangana Minister of Agriculture, an expected workload will be calculated and acceptable performance will be formulated based on it. The whole system will be under this load, increasing it periodically in order to calculate load threshold and the maximum time the system can be exposed to it.

- **Stress Testing**

After calculating a solid performance baseline, the system will be subjected to loads well beyond expectations to evaluate response and make sure that the system recovers gracefully after failure.

5.4 Testing tools

- **JUnit**

Unit testing framework for the Java programming language.

- **TestComplete**

GUI test automation tool that tests every desktop and web application. Integrates tightly with the tools in your ecosystems, giving you a complete testing lifecycle.

- **LoadNinja**

LoadNinja allows you to build scriptless load tests. It replaces load emulators with real browsers and generating actionable metrics.

6 Effort spent

This section keeps track of team and personal effort spent to produce the DD document and its sections. The activities that involved crucial decisions such as design choices were performed in group, while the individual work mainly consists in activities that do not involve decision-making and team discussions.

6.1 Team Effort

Task	Hours
Initial briefing	2
Architectural design briefing	1
Component View	2
Deployment View	1
Implementation, integration and test plan	1
Document Revisioning	1
Total	8

Table 2: Team effort

6.2 Mauro's Effort

Task	Hours
Introduction	3
Architectural overview	5
Runtime view	3
Component interfaces	1
Architectural & Design decisions	2
Requirements Traceability	3
System testing	1
Document revisioning	2
Total	20

Table 3: Mauro's effort

6.3 Giacomo's Effort

Task	Hours
Introduction	1
Component Views	3
Deployment View	2
Runtime View	3
Component interfaces	2
User interface design	4
Implementation and integration	3
Document revisioning	1
Total	19

Table 4: Giacomo's effort

References

- [1] *RDD Project Assignment*, A.Y. 2021/22. URL: https://webeep.polimi.it/pluginfile.php/306783/mod_folder/content/0/01.%20Assignment%20RDD%20AY%202021-2022.pdf?forcedownload=1.
- [2] Giacomo Lombardo Mauro Famà. *DREAM Requirements Analysis and Specification Document*, 2021. URL: <https://github.com/giacomolmb/FamaLombardo/blob/main/DeliveryFolder/RASD1.pdf>.
- [3] Matteo G. Rossi. *Verification and Validation: Testing. Retrieved from Course Slides*, 2021. URL: https://webeep.polimi.it/pluginfile.php/196408/mod_folder/content/0/06b.VerificationAndValidationTesting.pdf?forcedownload=1.