



**POLITECNICO**  
**MILANO 1863**

# DREAM

Data-driven Predictive Farming in Telengana

## DD

Design Document

Version 1.0 - 21/12/2021

Fateme Hajizadekiakalaye - 10831743

Reza Paki - 10832693

# Table of Contents

<b>1. Introduction</b>	<b>3</b>
1.1. Purpose	3
1.2. Scope	3
1.3. Definitions, Acronyms and Abbreviations	3
1.3.1. Definitions	3
1.3.2. Acronyms	4
1.3.3. Abbreviations	4
1.4. Revision History	4
1.5. Reference Documents	4
1.6. Document Structure	4
<b>2. Architectural Design</b>	<b>5</b>
2.1. Overview	5
2.1.1. High-level Architecture	6
2.2. Component View	7
2.3. Deployment View	9
2.4. Runtime View	10
2.5. Component Interfaces	16
2.6. Selected Architectural Styles and Patterns	17
2.7. Other Design Decisions	18
<b>3. User Interface Design</b>	<b>19</b>
<b>4. Requirements Traceability</b>	<b>23</b>
<b>5. Implementation, Integration and Test Plan</b>	<b>28</b>
5.1. Overview	28
5.2. Implementation Plan	28
5.3. Integration and Testing	29
<b>6. Effort Spent</b>	
<b>7. References</b>	

# 1. Introduction

## 1.1. Purpose

The main goal of this document is to provide more technical and detailed information about the DREAM application discussed in the RASD document. In the fact, programmers who develop the application could use this document as a strong guide. DD represents deep detail on application design, hardware and software architecture of the system in terms of components and interactions among those components. It also gives a detailed presentation of the implementation plan, integration plan, and testing plan. In general, the main different features listed in this document are:

- The high-level architecture
- Main components of the system
- Interfaces provided by the components
- Design patterns adopted
- Implementation, integration and testing plan

## 1.2. Scope

DREAM is an application that is provided for both farmers and policymakers. Policymakers could manage farmers by this app and farmers could use published data by the application for farming purposes. DREAM uses some specific data about meteorological forecasts, the humidity of the soil, amount of water used for irrigation and then allows farmers to use this information for farming purposes. DREAM also provides a forum for sharing experiences or problems with other farmers. When a farmer creates a problem, he/she is allowed to choose who could answer the problem (farmers or agronomists, or both of those). Also, farmers could create discussions and share experiences with other farmers. Farmers insert their information such as production amount, amount of usage water, quality of production. Then Policymakers observe the farmers' activities and label them as good farmers or bad farmers. If a farmer labeled as a bad farmer gets technical guides from agronomists. This way policymakers allow to manage farmers and by giving special guides help to farmers. More detailed information can be found on the RASD document.

## 1.3. Definitions, Acronyms and Abbreviations

### 1.3.1. Definitions

Definition	Description
Steering initiatives	The provided solutions for farmers by agronomists.
Discussion forum	A meeting at which farmers can exchange ideas and opinions about a specific topic.

### 1.3.2. Acronyms

Acronyms	Description
DREAM	Data-driven predictive Farming in Telangana
DD	Design Document
RASD	Requirement Analysis and Specification Document
DBMS	Data-Base Management System
API	Application Programming Interface
IT	Information Technology
HTTP	HyperText Transfer Protocol
XML	Extensible Markup Language
JSON	JavaScript Object Notation
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
RMI	Remote Method Invocation
TSL	Transport Layer Security

### 1.3.3. Abbreviations

Abbreviations	Description
G	Goal
R	Requirement
C	Component

### 1.4. Revision History

Version	Date	Modification
1.0	21/12/2021	First version

### 1.5. Reference Documents

- Specification Document: "01. Assignment RDD AY 2021-2022.pdf"
- UML diagrams: <https://www.uml-diagrams.org/>
- Slides of the lectures
- IEEE/ISO/IEC 29148-2018 - ISO/IEC/IEEE International Standard - Systems and software engineering - Life cycle processes - Requirements engineering

### 1.6. Document Structure

- **Section1**  
Brief description about DD and introducing purpose and scope. Also, including definitions, acronyms and abbreviations.

- **Section2**

The main part of DD is this section that contains architectural design choice, includes all the components, the interfaces used for the development of the application. Also, it contains deployment view, runtime view. In the end, is explained the architectural patterns chosen with the other design decisions.

- **Section3**

Contains how should be the user interface on mobile applications.

- **Section4**

Contains the traceability matrix that shows which components satisfy certain requirements.

- **Section 5**

Including the implementation plan, integration plan, and testing plan, and shows how these plans are executed.

- **Section 6**

Shows how much time is spent by each member of the group.

- **Section 7**

Includes the reference documents.

## 2. Architectural Design

### 2.1. Overview

The architecture of this application is structured by 3 logic layers:

- **Presentation level (P):** This level manages the user communication with the system.
- **Application layer (A):** This layer handles the business logic of the application, its functionalities and moving data between the other two layers.
- **Data access layer (D):** This layer manages the information, with the corresponding access to the databases and prepares useful information for the users in the database and passes them along with the other layers.

Each user interfaces (farmer or policymaker) by calling methods to provide any functionalities starts the process. Callable methods are like using weather forecasts, creating discussion, sharing the problem, labeling the farmers, and so on.

#### Services for farmers:

- Farmer wants to use forecasted information: server analysis the queries and provides forecasted information from the database for that region and gives those to farmers.
- Farmer wants to create a discussion forum: farmer writes own messages and sends them. The server receives information and saves those in the database as well as replied messages.

- Farmer wants to create a problem: farmer writes own problem, the information about created problem will save in database as well as replied answers.
- Farmer wants to see the suggestions which has received from others: server analysis the query and provides the suggestions from database and gives them to the farmer.
- Each farmer inserts their production details in profile: thus, the server receives a query that contains these details and will send them to the database.

#### Services for policymakers:

- A policymaker could see farmers' performance and information and label them. The server will receive labels for each farmer and update farmer status in the database.
- A policymaker could see steering initiatives progress. The server receives the query and will provide data from the database and gives them to policy maker.

#### 2.1.1. High-level Components

The chosen hardware architecture for this application is **Three-Tier**. The architecture contains three application layers. A tier to interface with the client, the second tier for the application level, and another server for the database management. This division is efficient because the server tier can make a permanent connection with the DBMS, which is less expensive. As well as, having a middle tier between user and server can guarantee more security to the access control of the database from the users.

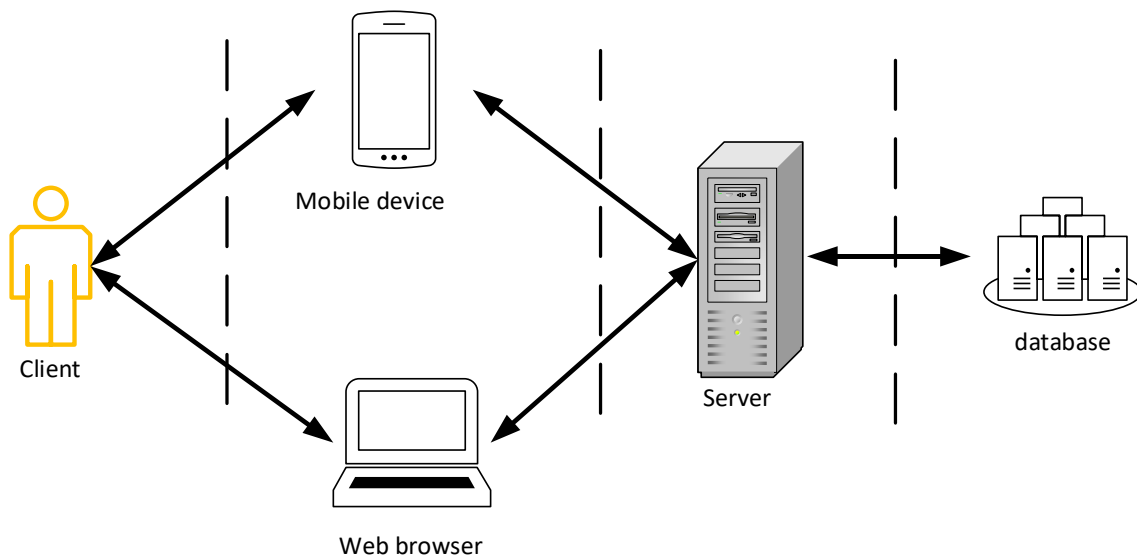


Figure 1 - Tree tier architecture

The following figure represents our system architecture, its high-level components and their interaction. The client could use this application by a smartphone or a computer. Then, user interactions will be processed by the application server. As well as, application server connected

to the database. Finally, the application server is allowed to use APIs for Google Maps and forecasted information.

These figures do not include all details about the architecture of the application, and in the next section, more details are represented.

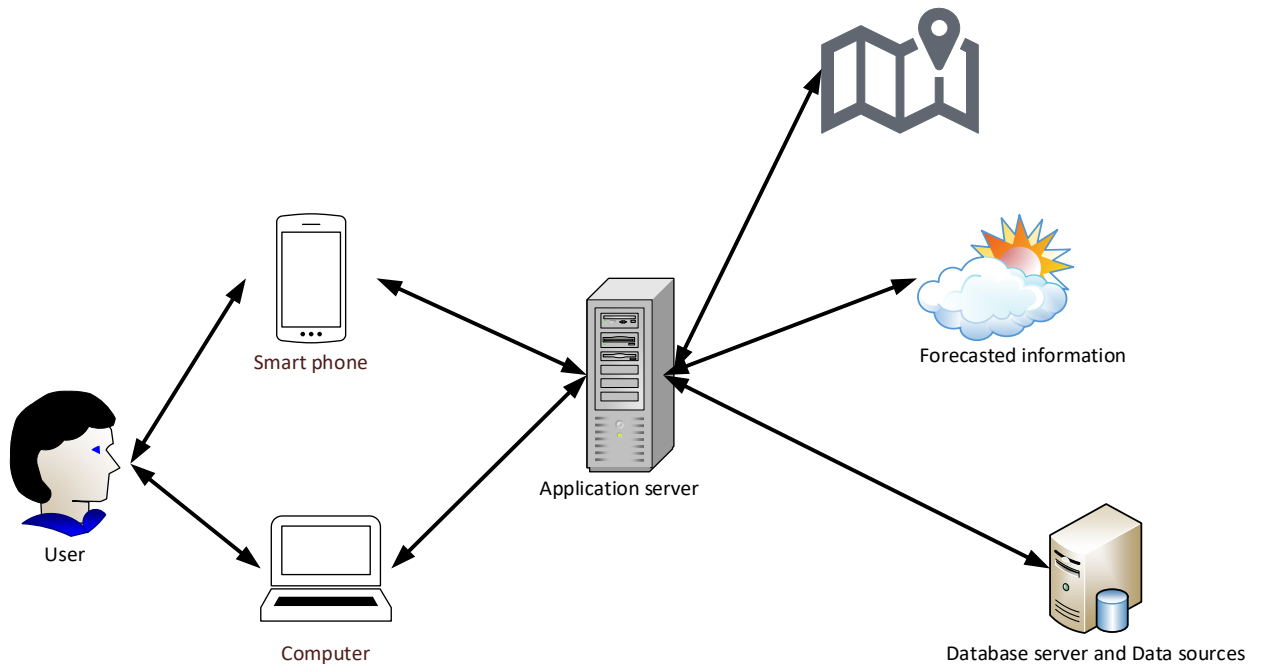


Figure 2 - System architecture

## 2.2. Component View

In the following diagram, each component is more closely examined:

**UserMobileApp:** This component is responsible for interactions between system and users who are not logged in the app. So, it offers registration and login functions. When, users login as a policymaker or farmer, they redirect to PolicymakerApp or FarmerApp through this component.

**UserWebApp:** This component offers all the functions which UserMobileApp offers but it stands for users who do not use smartphones and use a web browser.

**FarmerApp:** All users who login as a farmer redirect to this component as main component. This is responsible for functions such as: requesting for help, creating a discussion forum, inserting products, view suggestions, view forecasted information and so on.

**PolicymakerApp:** All users who login as a policymaker redirect to this component as main component. This is responsible for functions such as: view list of farmers, labeling farmers, view steering initiatives diagrams and so on.

**SoftwareService:** In this component, there are some subcomponents. Model subcomponent and DataEncrypt subcomponent have vital rule for interaction with database. Authentication subcomponent and Mail subcomponent are stands for authorizing users and detecting attacks to application. Also, there are Geolocalization and Forecast subcomponents to interact with GoogleMapService and TelenganaService.

**GoogleMapService:** This component is responsible for gathering maps of different regions.

**TelenganaService:** This component is responsible for gathering data from IT providers.

**DBMS:** This is main component which accesses to database. More details are in following sections.

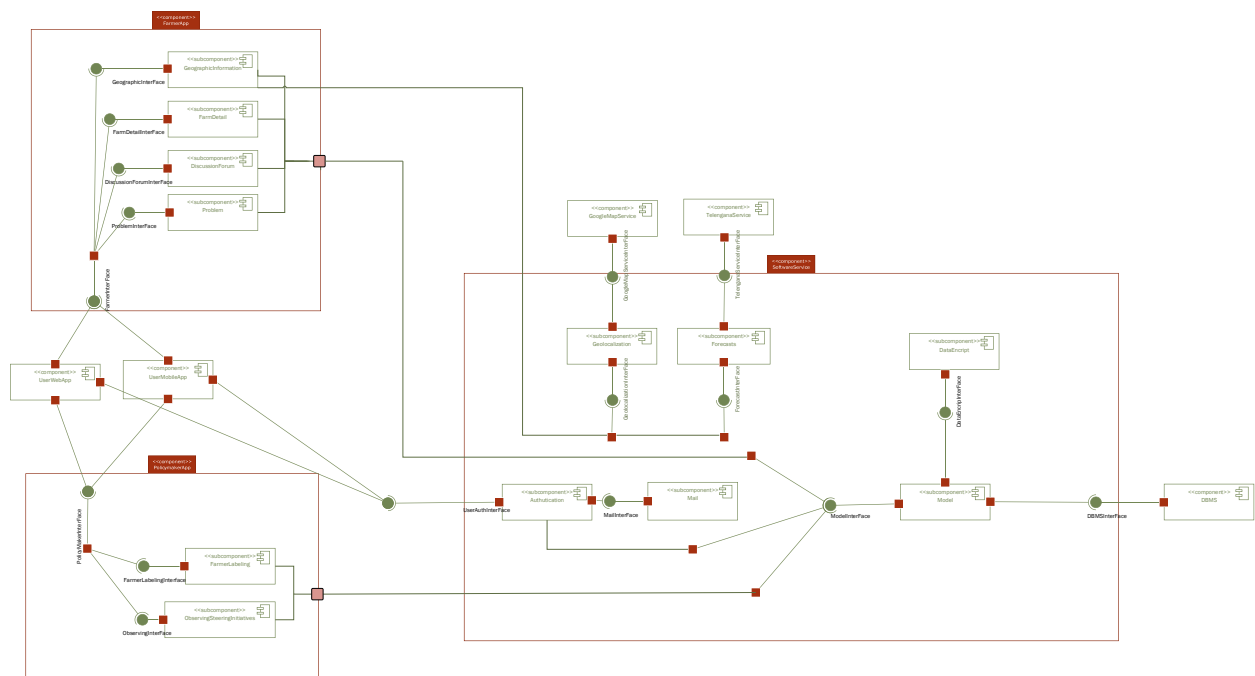


Figure 3 - Component View



## 2.3. Deployment View

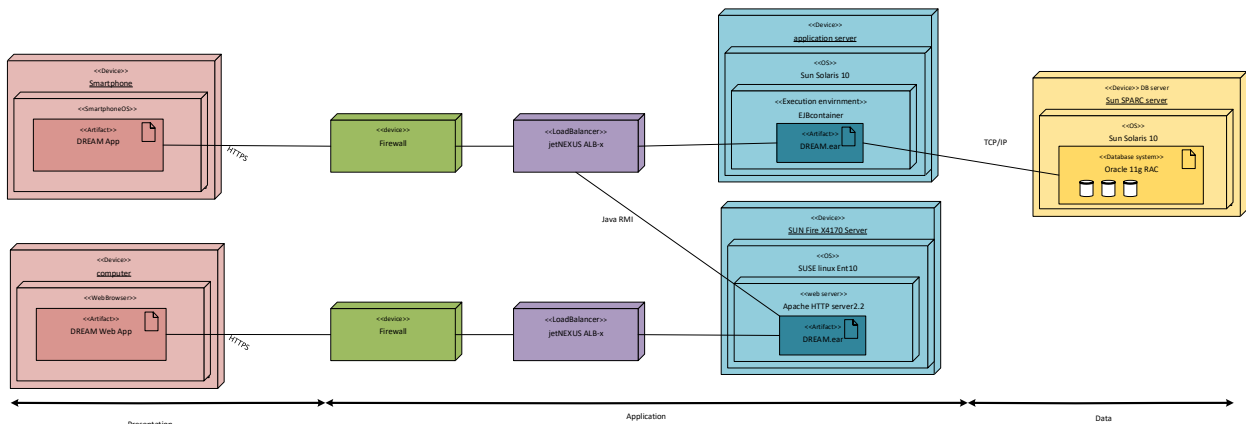


Figure 4 - Deployment View

- **Smartphone:** The user (farmer or policymaker) can use the smartphone to use the application. Farmers can use provided information, create a problem or discussion, and insert their farm information. Policymakers can manage farmers by using the application.
- **Computer:** User can use web application by computer or other devices that contain web browser. The web application contains all features that mobile applications have.
- **Firewall:** Firewall is used for providing safe access to the server and database, as well as, preventing external attacks.
- **Load balancer:** For handling the workload of the application we use a load balancer that distributes the workloads across multiple servers to increase the capacity and reliability of applications trying to avoid the overload of any single server.
- **Web Server:** Receives queries from clients through the web application and sends received data to the application server for processing by using java RMI.
- **Application Server:** This part contains application logic, but not all of those because the client also has a part of this. The Application Server handles all the requests and provides the appropriate answers for all the offered services.
- **Database Management System (DBMS):** For performing the job of DBMS Oracle Real Application Cluster (RAC) has been selected. A cluster comprises multiple interconnected computers or servers that appear as if they are one server to the end-users and applications. Oracle RAC enables you to cluster Oracle databases and it is a configuration of Relation DBMS (RDBMS). Some nodes contain the same instance of the database and the instances are connected to each other and share the cache in the Global Cache (GC). Then, there is a pool of replicated databases that contain data files. Datafiles are common, shared and they are accessed from all instances in a parallel and synchrony way.

## 2.4. Runtime View

In the following sequence diagrams, we discuss the detailed interactions between user and each component. The main interactions between user and system have been already discussed in RASD. In order to keep the diagrams readable, we just visualize the main flow of the events and don't consider the exceptions.

- **Registration**

The user fills the registration form by his/her information through the "UserMobileApp" or "UserWebApp" component. Then, the information are sent to the "Authentication" component. In this step, we should check whether there is another user with same email. So, the "Authentication" component sends the entered email to the "Model" component. Here, a temporary model with entered email is created. Then, a query is sent to database to check the possibility of existing user with same email. The answer of database is encrypted. So, it is passed to the "DataEncrypt" component to be decrypted. If the retrieved data from database is null, it means there is no user with entered email, so the "Authentication" component creates a new user with the inserted information, plus a flag equal to false and pushes it to the database. Then, an email is sent to the registering user with a confirmation by "Mail" component. When the customer clicks on the confirmation link, he/she will be redirect to a web page. Finally, a query will be sent to database to update this user flag to true.

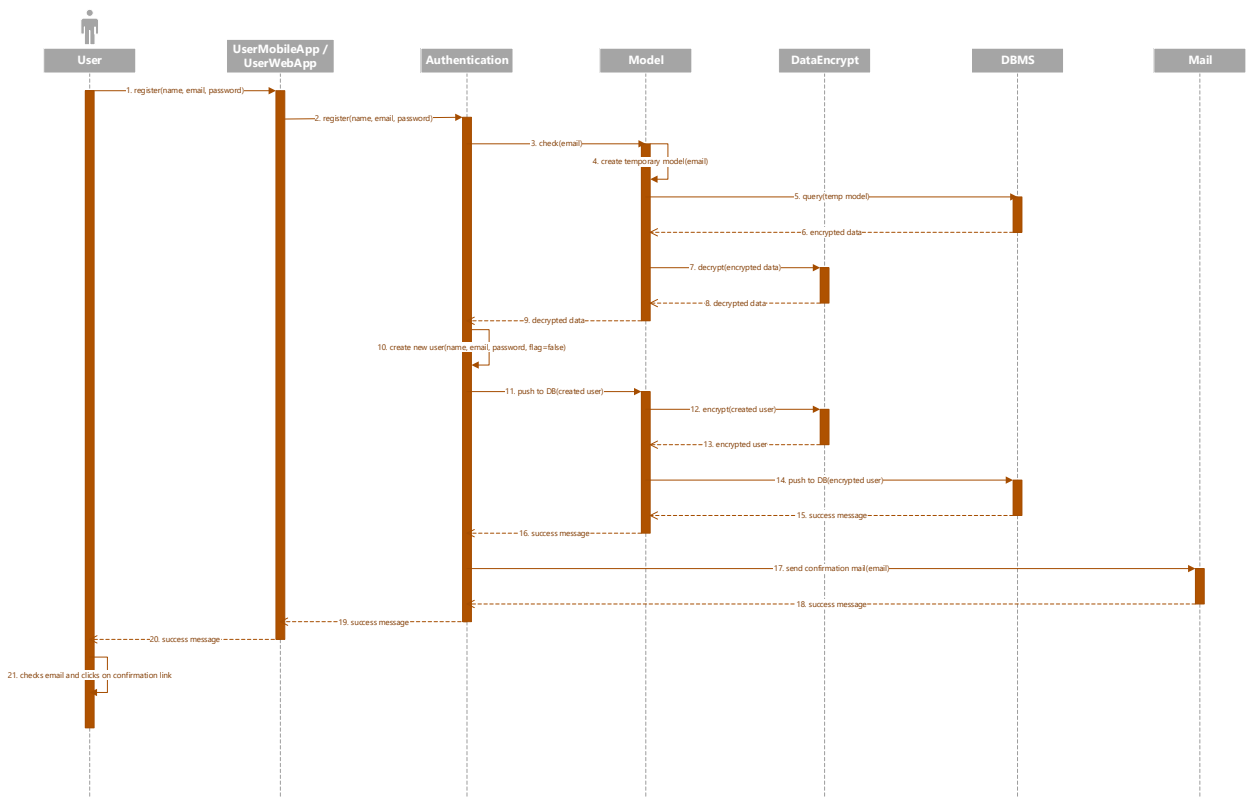


Figure 5 - Runtime View: Registration

- **Login**

The user enters the email and password through the “UserMobileApp” or “UserWebApp” component. Then, his/her information are sent to the “Authentication” component. In this step, we should check whether the user has been registered. So, the “Authentication” component sends the entered email to the “Model” component. Here, a temporary model with entered email is created. Then, a query is sent to database to check the possibility of existing user with same email. The answer of database is encrypted. So, it is passed to the “DataEncrypt” component to be decrypted. If the retrieved data from database isn’t null, it means the user with entered email exists. Then, the “Authentication” component should check the correctness of entered password. In case the check is positive the retrieved user is returned, otherwise a null value is returned.

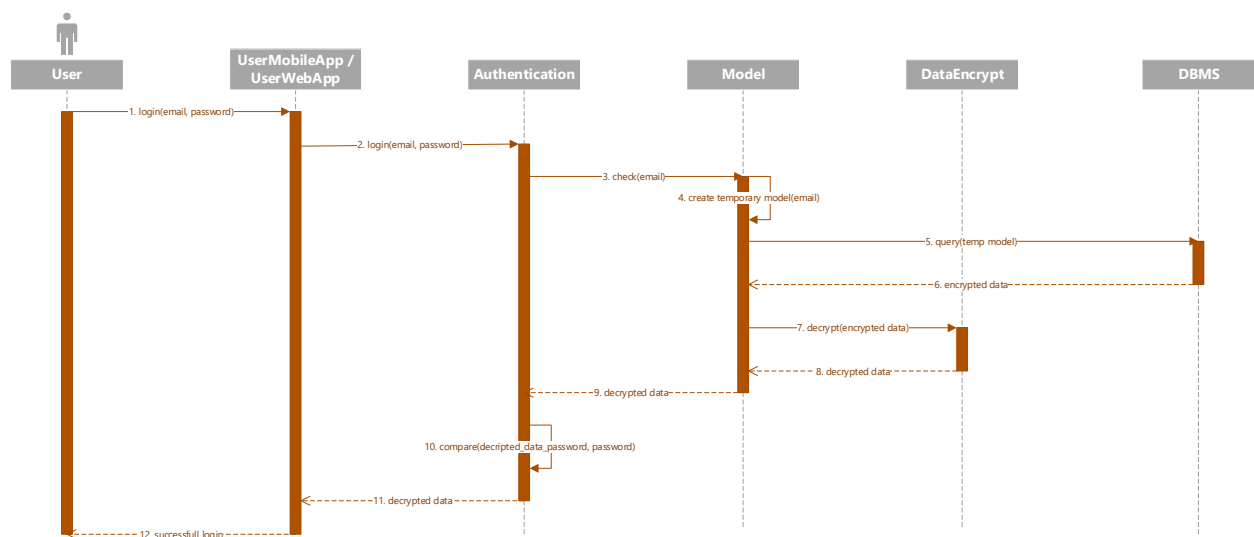


Figure 6 - Runtime View: Login

- **Labeling Farmers**

The user has already logged as a policymaker and now he/she is interacting with “PolicymakerApp” component. First, the policymaker selects the option to view list of farmers through “PolicymakerApp” component. Then, his/her request forwards to the “FarmerLabeling” component. This component send a request to the server asking for all the farmers present in the database. The answer of database is encrypted. So, it is passed to the “DataEncrypt” component to be decrypted. Again, the policymaker selects a farmer and his/her request forwards to the “FarmerLabeling” component and another request asking for the information associated with the selected farmer will be sent to the server. The information of farmer retrieves from database and shows to policymaker. Then, the policymaker clicks on good performance button and this action leads to updating farmer’s information in the database.

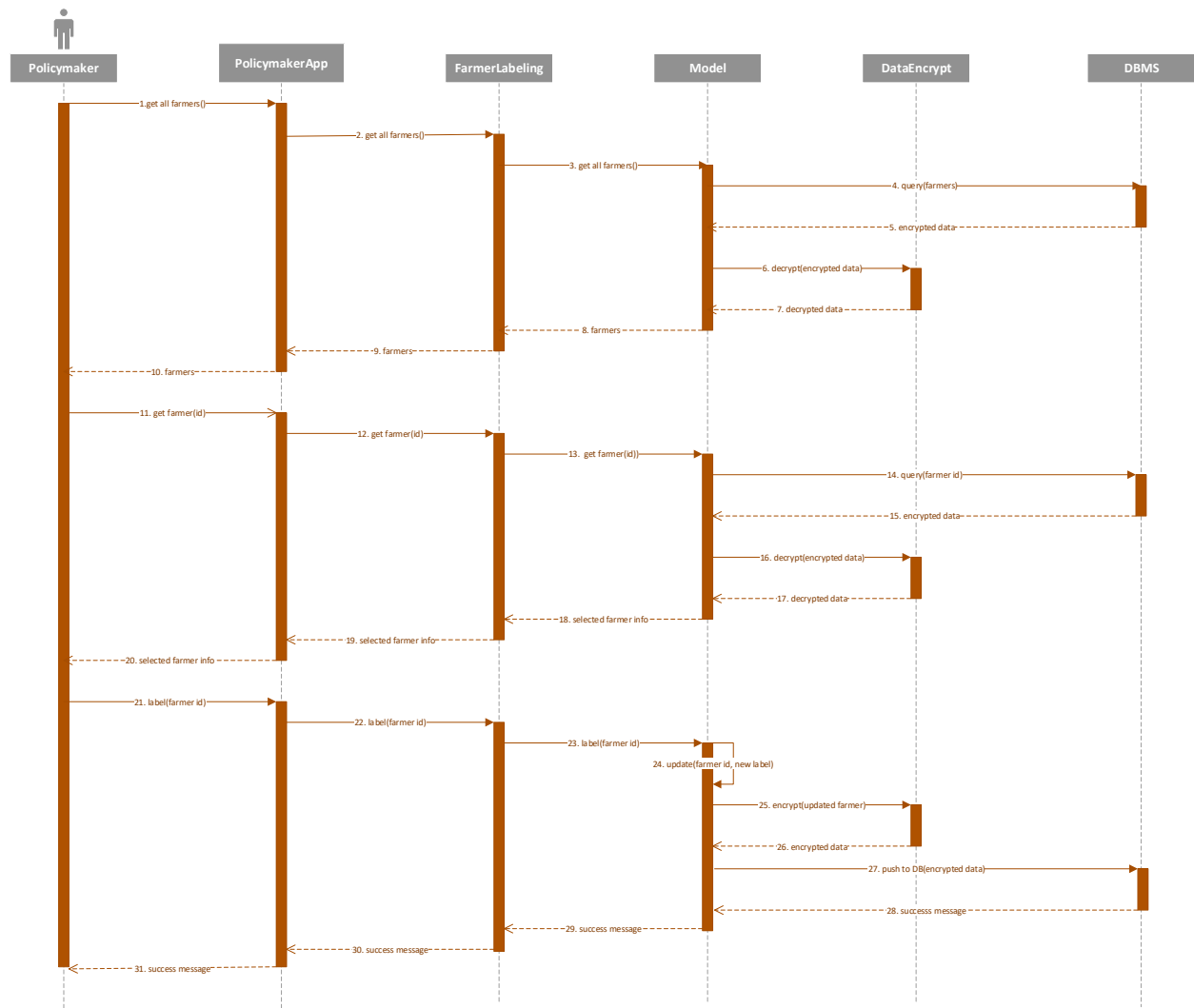


Figure 7 - Runtime View: Labeling Farmers

- **View steering initiatives' result**

The user has already logged as a policymaker and now he/she is interacting with "PolicymakerApp" component. First, the policymaker selects the option to view steering initiatives through "PolicymakerApp" component. Then, his/her request forwards to the "ObservingSteeringInitiatives" component. This component send a request to the server asking for all the corresponding diagrams present in the database. The answer of database is encrypted. So, it is passed to the "DataEncrypt" component to be decrypted. Finally, the diagrams are sent to the policymaker.

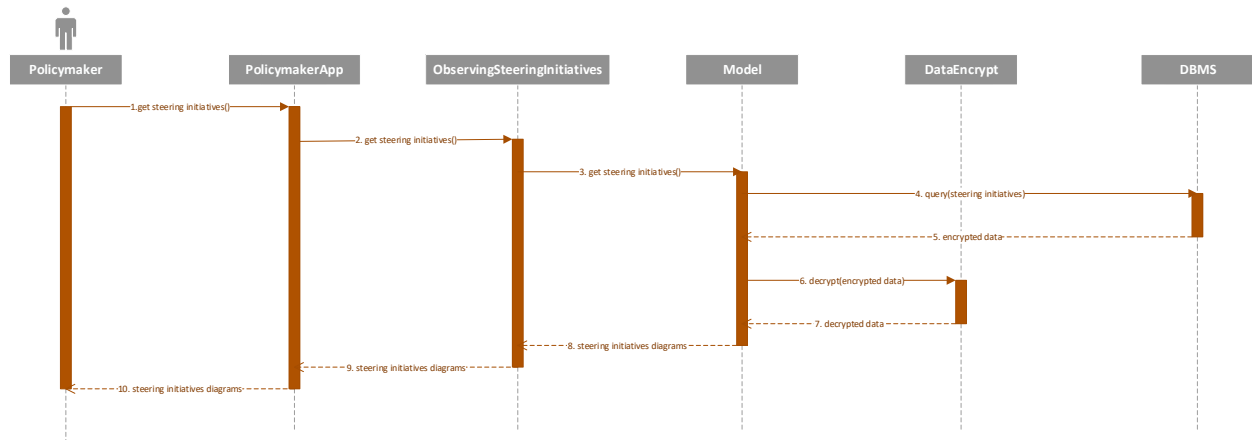


Figure 8 - Runtime View: Steering initiatives

- **Insert products**

The user has already logged as a farmer and now he/she is interacting with “FarmerApp” component. First, the farmer selects the option to view his/her farm detail through “FarmerApp” component. Then, his/her request forwards to the “FarmDetail” component. This component send a request to the server asking for information related to his/her farm. The answer of database is encrypted. So, it is passed to the “DataEncrypt” component to be decrypted. After that, the farmer clicks on insert a new product and enters product information such as: type, produced amount and so on. In this step, the product details are sent to “FarmDetail” component and from there, a request will be sent to the “Model” component to check whether the inserted type is duplicate in the farm. So, the “Model” component creates a temporary product with inserted type and sends a query to database. If the database answer is null, the “FarmDetail” component will create a new product with inserted information and push it to the database. Finally, the information saves in database and success message is send to farmer.

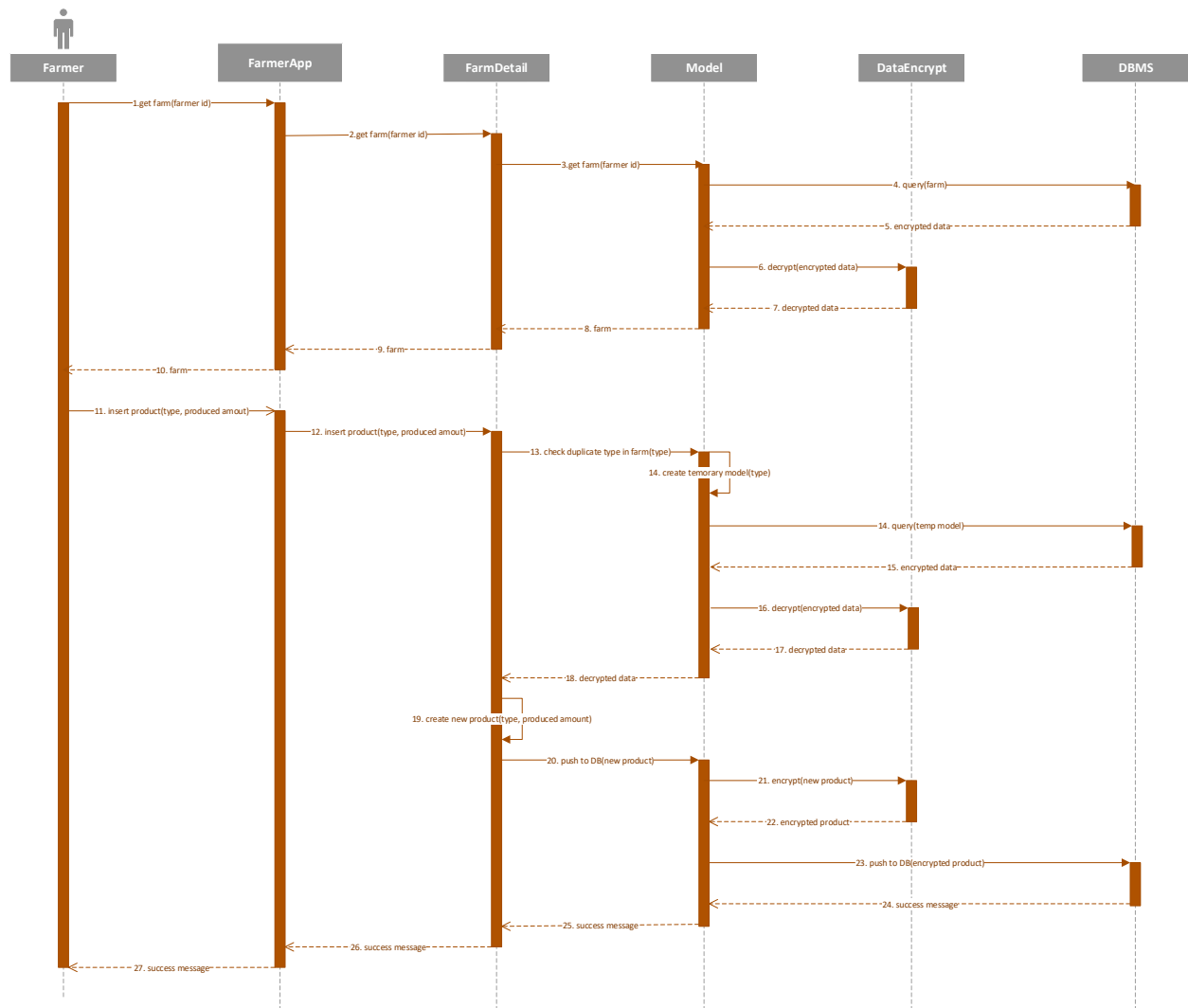


Figure 9 - Runtime View: Inserting Products

- **Request for help**

The user has already logged as a farmer and now he/she is interacting with “FarmerApp” component. First, the farmer selects the option to request for help and inserts his/her problem details through “FarmerApp” component. Then, the problem details are sent to “Problem” component and there, a new problem is created with inserted data and a request will be sent to the “Model” component to update the database with new data. Finally, the information saves in database and success message is send to farmer.

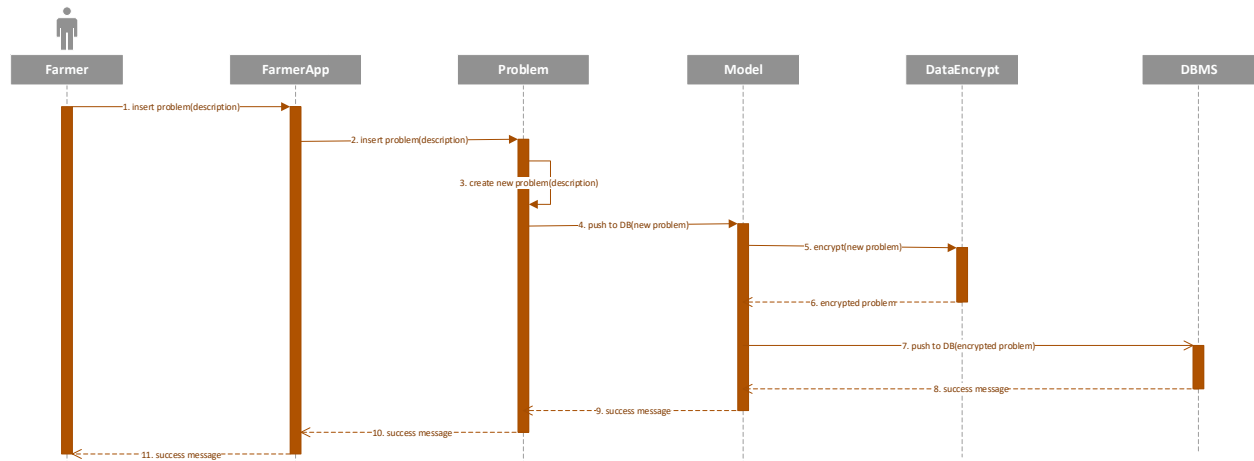


Figure 10 - Runtime View: Requesting for Help

- **View weather forecast and humidity of soil**

The user has already logged as a farmer and now he/she is interacting with “FarmerApp” component. First, the farmer selects the option to view weather forecasting information through “FarmerApp” component. Then, the request is sent to “GeographicInformation” component and there, a request will be sent to the “GoogleMapService” component to get maps and another request will be sent to “TelenganaService” to receive weather information. Finally, the map and weather information are sent to farmer.

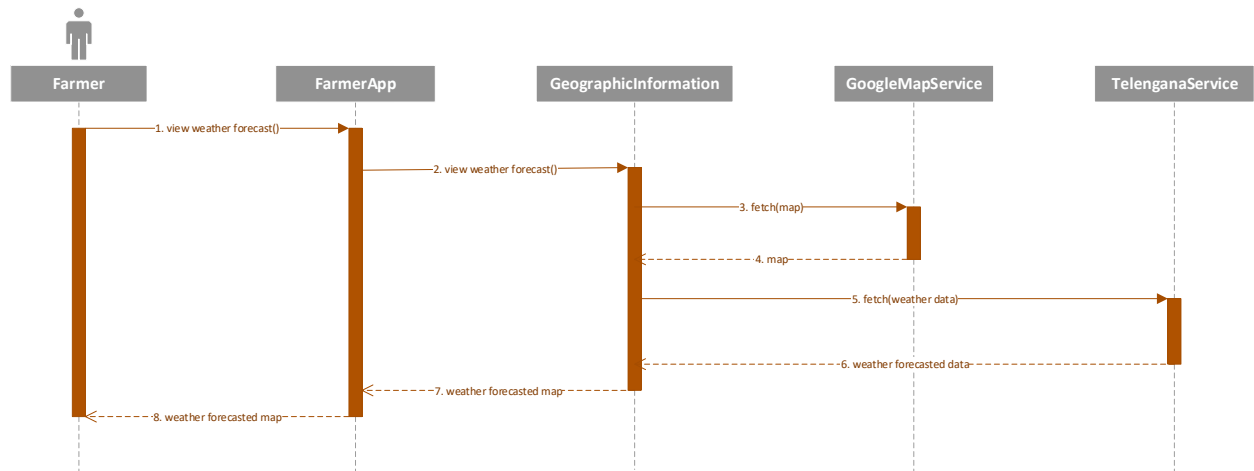


Figure 11 - Runtime View: Geographic Information

- **Create discussion forum**

The user has already logged as a farmer and now he/she is interacting with “FarmerApp” component. First, the farmer selects the option to create a discussion forum and inserts a title and a message through “FarmerApp” component. Then, his/her request forwards to the “DiscussionForum” component. Here, a request will be sent to the “Model” component to check whether the inserted title for discussion forum is duplicate in the other discussion forums which

are created by same farmer. So, the “Model” component creates a temporary discussion forum with inserted title and sends a query to database. If the database answer is null, the “DiscussionForum” component will create a new discussion with inserted information and push it to the database. Finally, the information saves in database and success message is send to farmer.

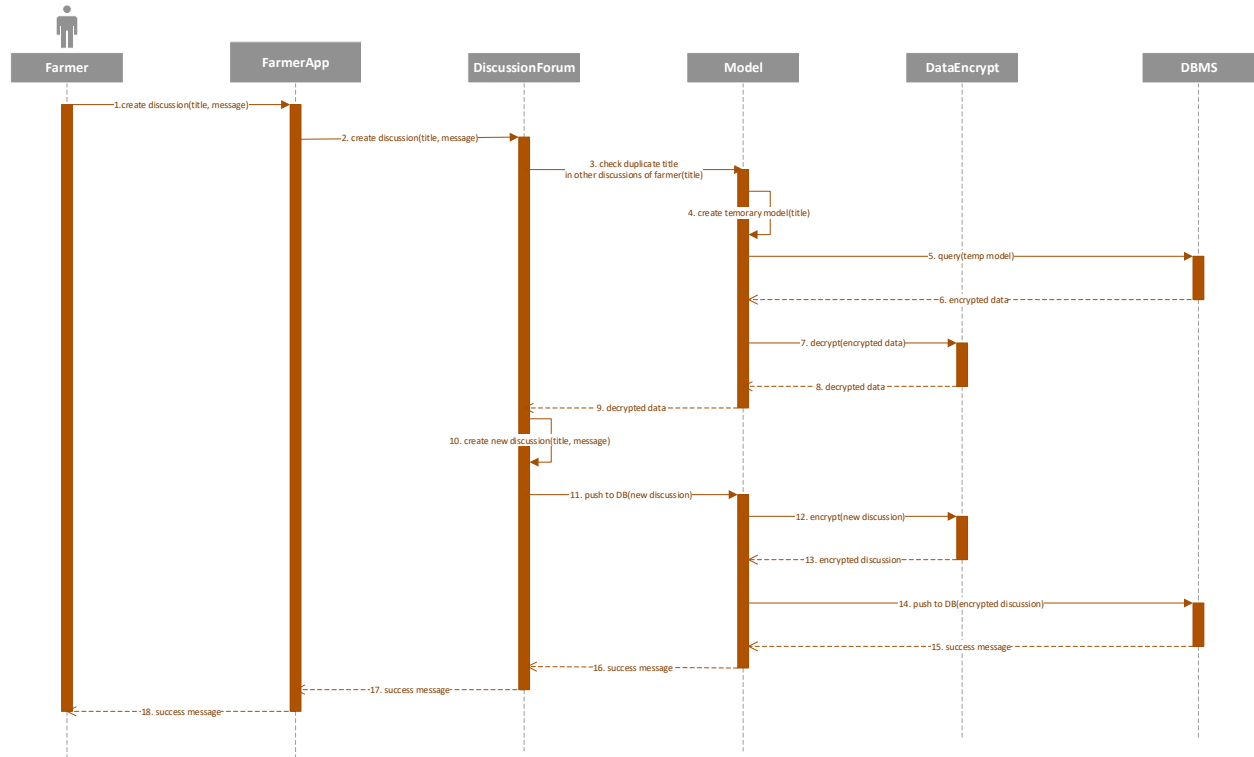


Figure 12 - Runtime View: Discussion forum

## 2.5. Component Interfaces

The following diagram represents the interfaces that are offered by the different components and their interactions. These interfaces are based on the most important processes which are mentioned in the runtime view section.

Note: in FarmerAppInterface and PolicymakerInterface the functions which can be invoked from subcomponents (sub interfaces) are considered.



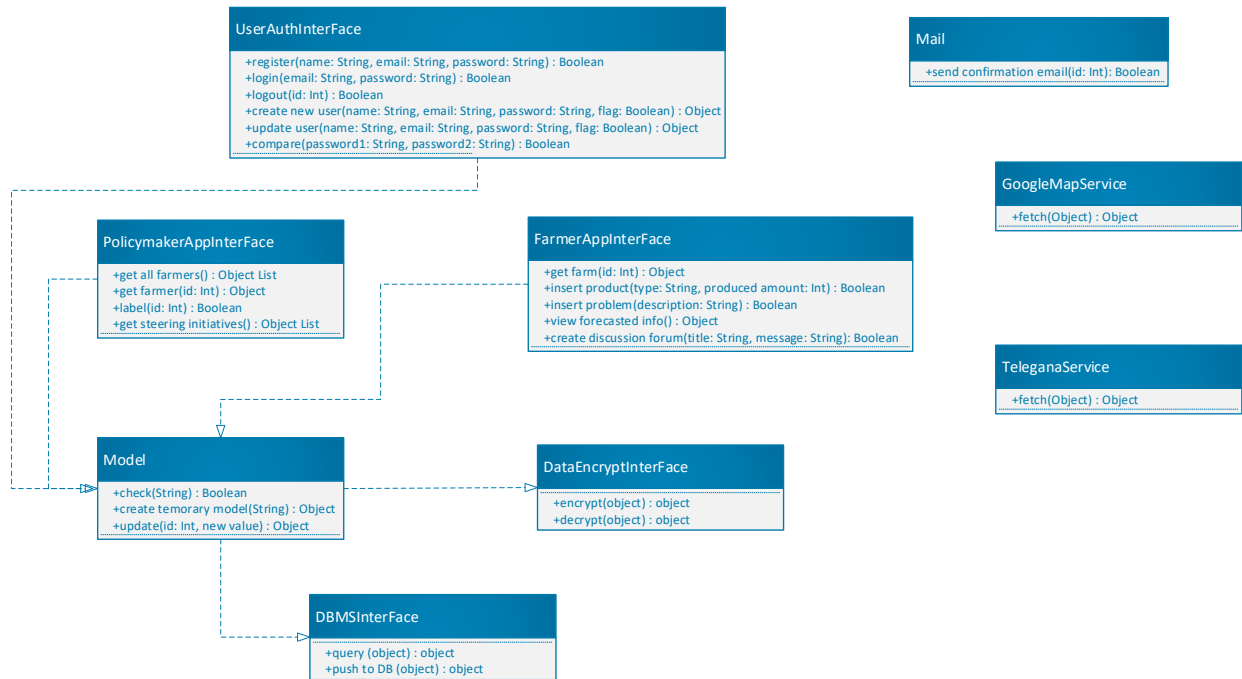


Figure 13 - Component Interface

## 2.6. Selected Architectural Styles and Patterns

As mentioned in the overview, for developing the system is used a three-tier client-server architecture: a presentation tier, an application tier, and a data tier. Dividing architecture into these 3 parts increases its flexibility, scalability, and reusability. As well as, existing components in the application server make the system more comprehensible and modifiable.

In the DREAM application except for user information, we do have not sensitive data, thus HTTP is used to exchange messages. This is also endorsed by the importance that data plays in the system. In this way, it is possible to reduce the coupling among client and server components. When dealing with sensitive data, TLS is used to guarantee the security and reliability of the connection.

Furthermore, the system uses the cache on the client tier: this allows to avoid some interactions between the client and the server and speeds up the communication. Using the cache is useful when the client needs more data during one session.

Regarding the format in which the data are transmitted, JSON is used for its simplicity. It is less verbose and this makes it more readable and allows a much faster parsing. The JSON file is transmitted over the network via XMLHttpRequest.

For using GoogleMaps, the communication protocol used is REST (REST is web standards-based architecture and uses HTTP protocol): GoogleMaps is indeed used as an external service and this

protocol clearly suits this situation. Furthermore, the public REST API provided by GoogleMaps is used: it also gives the possibility to customize the requested contents.

When dealing with Telengana for receiving the forecasted information, the communication protocol is SOAP (SOAP uses HTTP to exchange the XML messages). Although SOAP can be used in a variety of messaging systems and can be delivered via a variety of transport protocols, the initial focus of SOAP is remote procedure calls transported via HTTP. SOAP has an independent security mechanism, due to this is useful for connecting with Telengana service.

The application server works as a client and makes queries and receives responses. To perform the application the following pattern is used:

### **Model View Controller (MVC)**

MVC is an architectural pattern consisting of three parts: Model, View, Controller (actually MVC separates the application into these 3 multiple layers of functionalities). Model: Handles data logic. View: It displays the information from the model to the user. Controller: It controls the data flow into a model object and updates the view whenever data changes. Using this architecture guarantees the maintainability and reusability of the code. This software pattern is particularly adapted for the development of both web and mobile applications in an object-oriented style of programming as java. This method allows components to be created independently and simplifies simultaneous development.

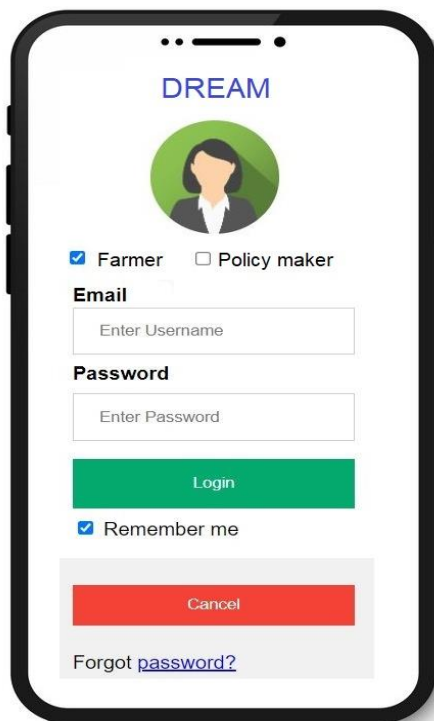
### **Facade pattern**

In this pattern, a component offers an interface that simplifies the use of another component. The Geolocalization and forecasts component uses it to adapt and simplify the use of an external geolocation service and Telengana service.

## **2.7. Other Design Decisions**

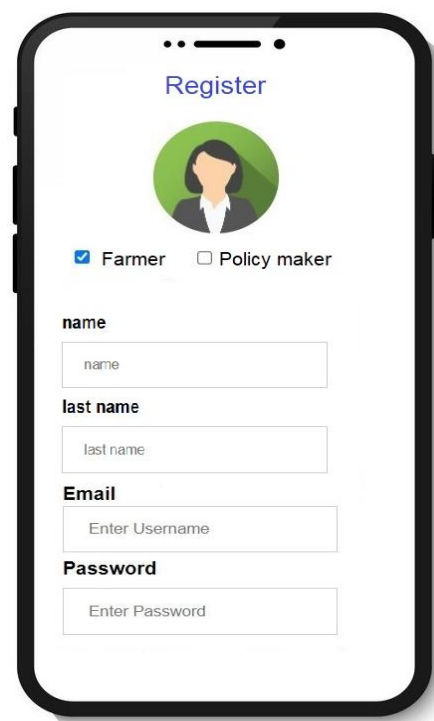
### 3. User Interface Design

The following mockups illustrate the most important sections of the interactions between DREAM application and users:



The mockup shows a mobile app interface for the 'DREAM' application. At the top, the word 'DREAM' is displayed in blue. Below it is a circular profile picture placeholder with a green background and a person icon. Under the profile picture, there are two radio buttons: 'Farmer' (selected with a blue checkmark) and 'Policy maker' (unselected). Below these are two text input fields: 'Email' with the placeholder text 'Enter Username' and 'Password' with the placeholder text 'Enter Password'. A green 'Login' button is positioned below the password field. Below the login button is a 'Remember me' checkbox, which is also selected with a blue checkmark. At the bottom, there is a red 'Cancel' button and a link that says 'Forgot [password?](#)'.

Figure - Mockup: Login



The mockup shows a mobile app interface for the 'Register' screen. At the top, the word 'Register' is displayed in blue. Below it is a circular profile picture placeholder with a green background and a person icon. Under the profile picture, there are two radio buttons: 'Farmer' (selected with a blue checkmark) and 'Policy maker' (unselected). Below these are three text input fields: 'name' with the placeholder text 'name', 'last name' with the placeholder text 'last name', and 'Email' with the placeholder text 'Enter Username'. Below the email field is a 'Password' field with the placeholder text 'Enter Password'.

Figure - Mockup: Register

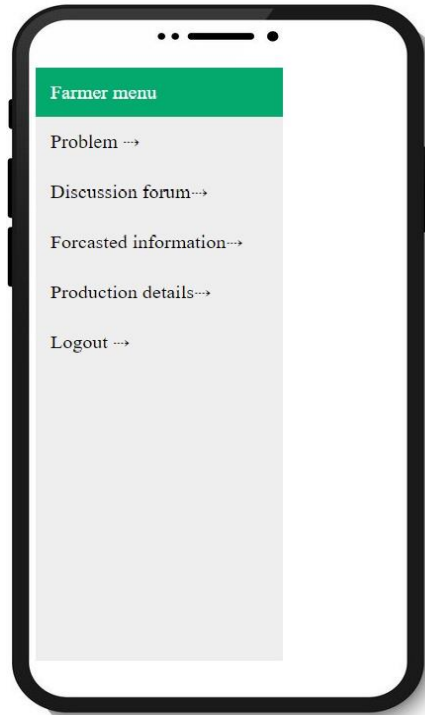


Figure - Mockup: Farmers' Menu



Figure - Mockup: Policymakers' Menu

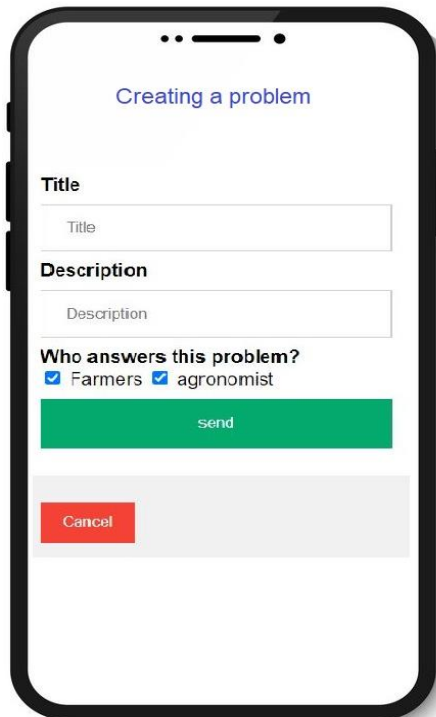


Figure - Mockup: Creating a Problem

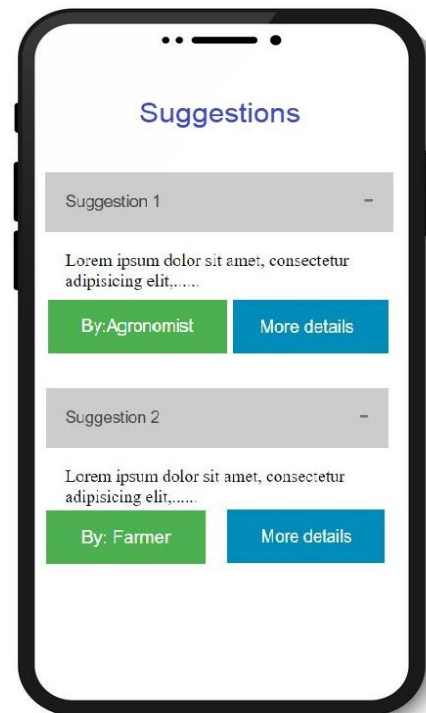


Figure - Mockup: View Suggestions

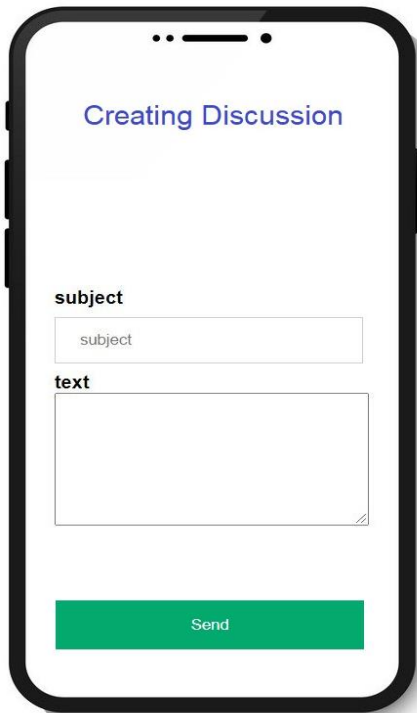


Figure - Mockup: Create a Discussion

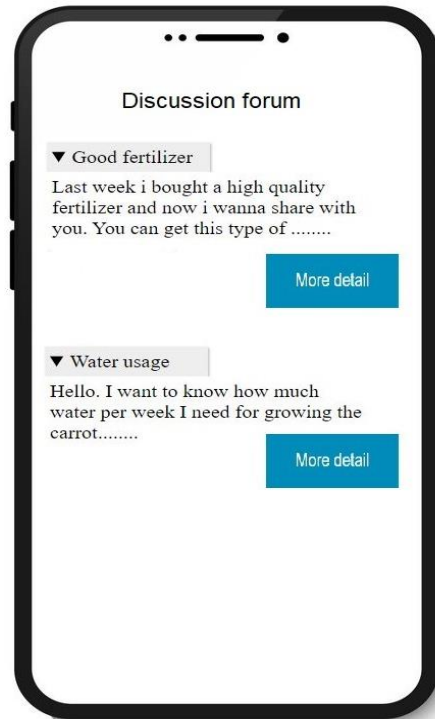


Figure - Mockup: List of Discussions

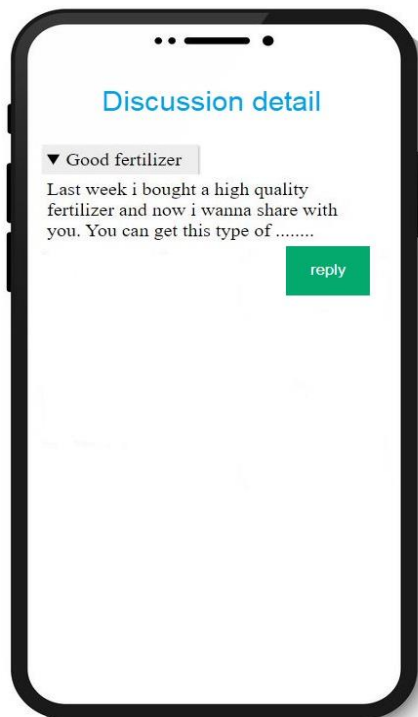


Figure - Mockup: Reply to a Message

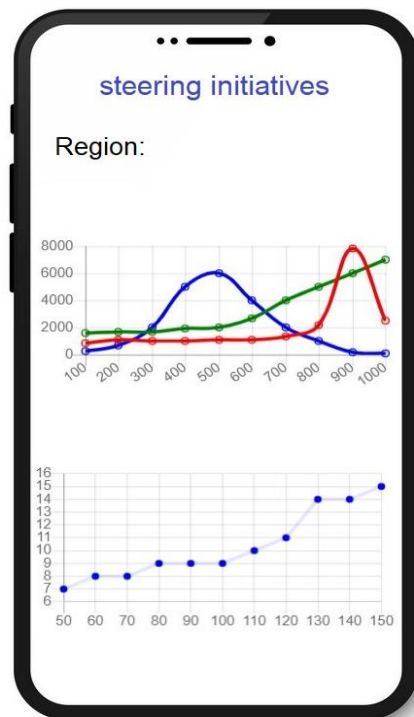


Figure - Mockup: Steering initiatives



Figure - Mockup: List of Farmers

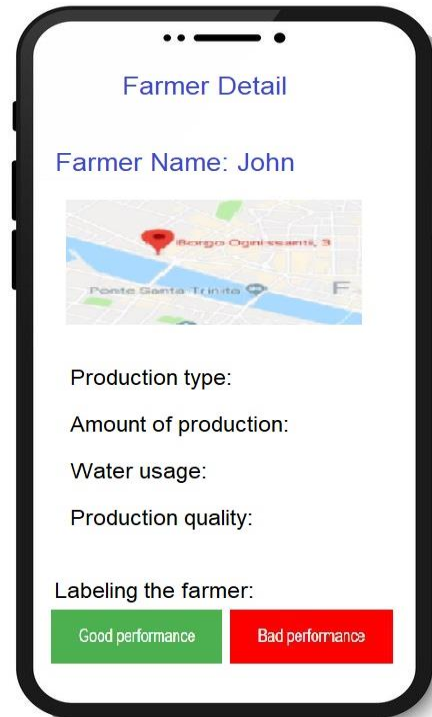


Figure - Mockup: Farmer's Detail

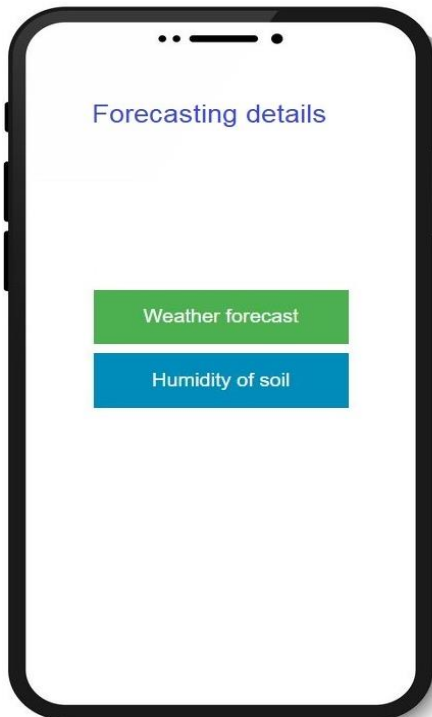


Figure - Mockup: Forecasted Details

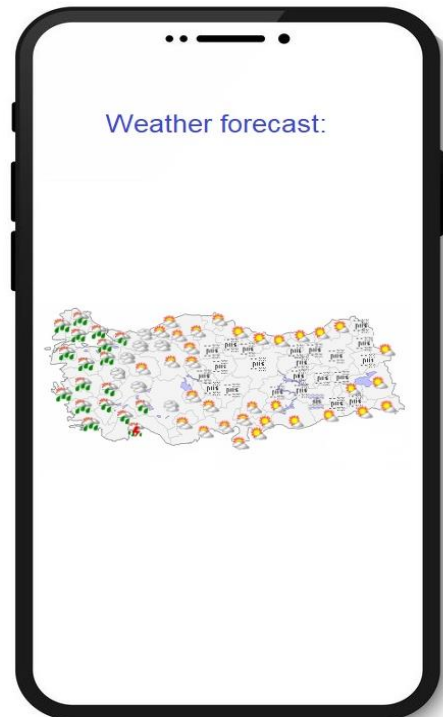


Figure - Mockup: Weather

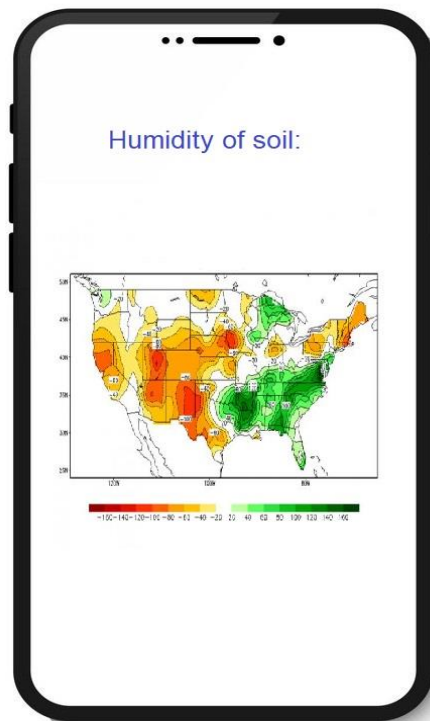


Figure 11 - Mockup: Humidity of Soil

#### 4. Requirements Traceability

In this section, we discuss a mapping between the requirements which have been defined in RASD and the design components. For more readability, we number the components as follows:

- C1: UserMobileApp
- C2: UserWebApp
- C3: PolicymakerApp
- C4: FarmerLabeling
- C5: ObservingSteeringInitiatives
- C6: FarmerApp
- C7: GeographicInformation
- C8: FarmDetail
- C9: DiscussionForum
- C10: Problem
- C11: Geolocalization
- C12: Forecast
- C13: GoogleMapService
- C14: TelenganaService
- C15: Authentication
- C16: Mail
- C17: Model
- C18: DataEncrypt

C19: DBMS

<b>G1</b>	<b>Allow policy makers to identify farmers who are performing well.</b>
R3	The system must allow a logged out policy maker to login.
R4	The system must allow a policy maker to view list of farmers to identify their performance.
R5	During identifying performance process, the system must allow a policy maker to select a farmer among the list and view his/her detailed information.
R6	During identifying performance process, the system must allow a policy maker to click on “poor performance” button or “good performance” button.
R8	After identifying performance process, the system must send a notification to the farmer containing congratulation message, if he/she was labeled as “good performance”.
C1	UserMobileApp
C2	UserWebApp
C15	Authentication
C3	PolycymakerApp
C4	FarmerLabeling
C17	Model
C18	DataEncrypt
C19	DBMS

<b>G2</b>	<b>Allow policy makers to identify farmers who need help.</b>
R3	The system must allow a logged out policy maker to login.
R4	The system must allow a policy maker to view list of farmers to identify their performance.
R5	During identifying performance process, the system must allow a policy maker to select a farmer among the list and view his/her detailed information.
R6	During identifying performance process, the system must allow a policy maker to click on “poor performance” button or “good performance” button.
R7	During identifying performance process, if a policy maker clicks on the “poor performance” button, the system must allow him/her to ask an agronomist to make a few suggestions to the farmer.
R9	After identifying performance process, the system must send a notification to the farmer containing notes and suggestions, if he/she was labeled as “poor performance”.
C1	UserMobileApp
C2	UserWebApp
C15	Authentication
C3	PolycymakerApp



C4	FarmerLabeling
C17	Model
C18	DataEncrypt
C19	DBMS

<b>G3</b>	<b>Allow policy makers to see the result of the steering initiatives.</b>
R10	During viewing steering initiatives process, the system must allow a policy maker to view list of steering initiatives diagrams.
R11	During viewing steering initiatives process, if the results didn't improve, the system must allow a policy maker to ask agronomists to update the steering initiatives.
R12	After viewing steering initiatives process, if the results didn't improve, the system must send a notification to every farmer, containing new updates.
C3	PolymakerApp
C5	ObservingSteeringInitiatives
C17	Model
C18	DataEncrypt
C19	DBMS

<b>G4</b>	<b>Allow farmers to see weather forecast.</b>
R35	During viewing forecasted information process, the system must allow a farmer to select "weather forecast" or "humidity of soil".
R36	During viewing forecasted information process, the system must allow a farmer to see his/her selected forecasted information.
C6	FarmerApp
C7	GeographicInformation
G11	Geolocalization
G12	Forecast
C13	GoogleMapService
C14	TelenganaService

<b>G5</b>	<b>Allow farmers to see humidity of soil.</b>
R35	During viewing forecasted information process, the system must allow a farmer to select "weather forecast" or "humidity of soil".
R36	During viewing forecasted information process, the system must allow a farmer to see his/her selected forecasted information.
C6	FarmerApp
C7	GeographicInformation

G11	Geolocalization
G12	Forecast
C13	GoogleMapService
C14	TelenganaService

<b>G6</b>	<b>Allow farmers to see suggestions relating to specific crop to plan or specific fertilizer to use.</b>
R28	The system must allow a farmer to view his/her suggestions
C6	FarmerApp
C10	Problem
C17	Model
C18	DataEncrypt
C19	DBMS

<b>G7</b>	<b>Allow farmers to insert their type of products and produced amount per product.</b>
R16	The system must allow a logged out farmer to login.
R17	During inserting products process, the system must allow a farmer to insert his/her type of product.
R18	During inserting products process, the system must allow a farmer to insert his/her produced amount.
R19	During inserting products process, the system must allow a farmer to insert his/her crop loss by meteorological adverse events.
R20	After a farmer inserted all the information related to inserting products process, the system must allow him/her to confirm.
R21	After a farmer confirmed all the information related to inserting products process, the system must publish them and allow everyone to see them.
C6	FarmerApp
C8	FarmDetail
C17	Model
C18	DataEncrypt
C19	DBMS

<b>G8</b>	<b>Allow farmers to insert their problems.</b>
R22	During requesting for help process, the system must allow a farmer to insert his/her problem.
C6	FarmerApp
C10	Problem

C17	Model
C18	DataEncrypt
C19	DBMS

<b>G9</b>	<b>Allow farmers to request for help and suggestion.</b>
R23	During requesting for help process, the system must allow a farmer to select who he/she wants help from (agronomists, farmers, both).
R24	After a farmer inserted all the information related to requesting for help process, the system must allow him/her to confirm.
R25	After a farmer confirmed all the information related to requesting for help process, the system must publish them and allow the group which was selected by farmer to see them.
R26	After the system published all the information related to requesting for help process, the system must allow the group which was selected by farmer to give suggestions.
R27	After a suggestion is given to a farmer, the system must send a notification to him/her to inform him/her.
C6	FarmerApp
C10	Problem
C17	Model
C18	DataEncrypt
C19	DBMS

<b>G10</b>	<b>Allow farmers to create discussion forums.</b>
R29	During creating a discussion forum, the system must allow a farmer to insert the title of discussion.
R30	During creating a discussion forum, the system must allow a farmer to insert his/her message.
R31	After a farmer inserted all the information related to creating a discussion forum process, the system must allow him/her to confirm.
R32	After a farmer confirmed all the information related to creating a discussion forum process, the system must publish them and allow every farmer to see them.
R33	After the system published all the information related to creating a discussion forum process, the system must allow every farmer to reply.
R34	After a farmer replied in a discussion forum, the system must send a notification to creator of discussion forum to inform him/her.
C6	FarmerApp
C9	DiscussionForum

C17	Model
C18	DataEncrypt
C19	DBMS

## 5. Implementation, Integration and Test Plan

### 5.1. Overview

It is possible to show the existence of bugs through program testing, but not to show their absence. In order to accomplish this goal, the aim is to find as many bugs as possible before the release date. All the preliminary considerations needed to implement and test DREAM are presented and explained in this section.

### 5.2. Implementation plan

To implement the whole system, the bottom-up approach is selected so that each section of the system can be implemented and tested separately.

The components and subcomponents described in the component view section can be divided in various subsystems:

- User mobile application
- Web application
- Application server
- External systems: DBMS, Google maps, Telengana.

The implementation has to be done from the lower components up to the top ones. Because there are some components that rely on some others. For example, every component and sub-system described in the component diagram, interact with the DBMS. Thus, we implement the **DBMS** component.

On the other hand, **Google Maps** and **Telengana** are two other components which play a role like database. So they should be implement in this step.

After that we can proceed to the implementation of the subcomponents in the **Application Server** component. Here, there are some main subcomponents such as the **Model** and the **DataEncrypt** subcomponents. Most components use them as a connection to access to the database.

Then, it is **Mail** and **Authentication** subcomponents turn. The Mail subcomponent should be implemented before the other one, because the functionalities of Authentication relies on Mail subcomponent and we want to keep using the bottom-up approach while testing.

In the next step, we can go for implementing all the subcomponents in the **User Mobile App** and **User Wen App**. Each of these subcomponents works independently, so they can be implemented at the same level.

### 5.3. Integration and Testing

The following diagrams illustrate how the process of integration testing takes place, according to a bottom-up approach.

The arrows start from the component which “uses” the other one.

- At first the Model, DataEncrypt and DBMS have been integrated and unit tested. They are not complex components, but other components rely on them to provide some services: that’s why they are integrated and tested first.

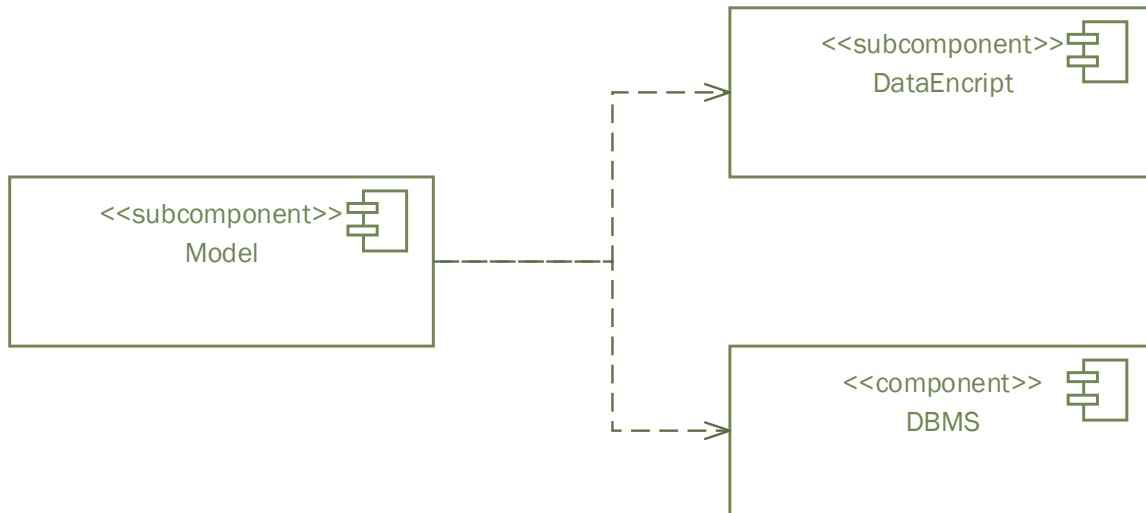


Figure 12 - Integration: Model, DataEncrypt, DBMS

- Then, as we mentioned in the previous section, two other important components named Authentication and Mail, should be integrated with Model component.

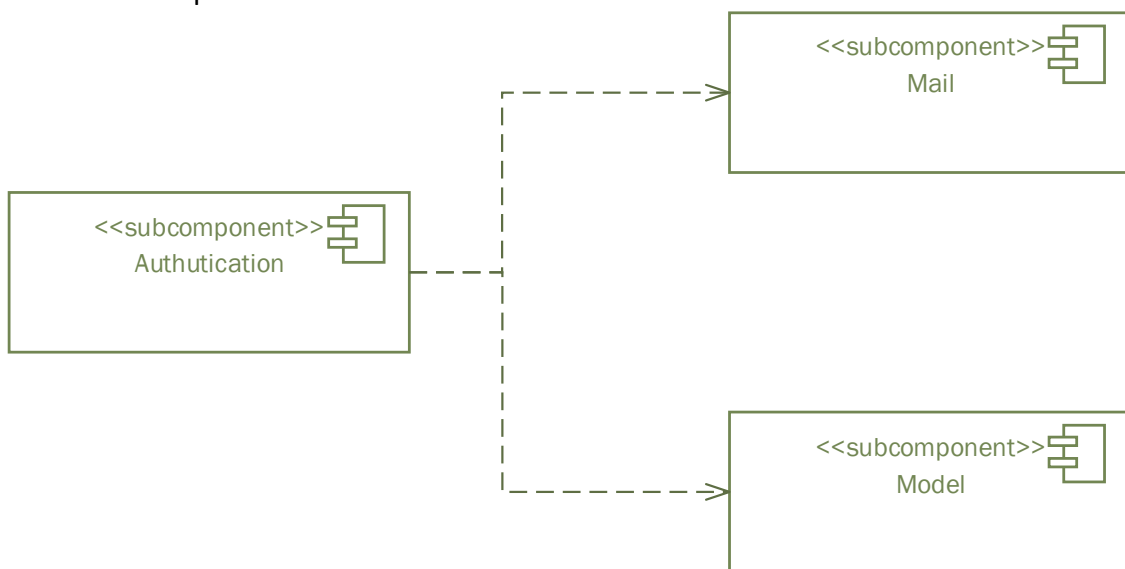


Figure 13 - Integration: Model, Mail, Authentication

- In the next step, the components which responsible for gathering information and maps have been integrated and tested.

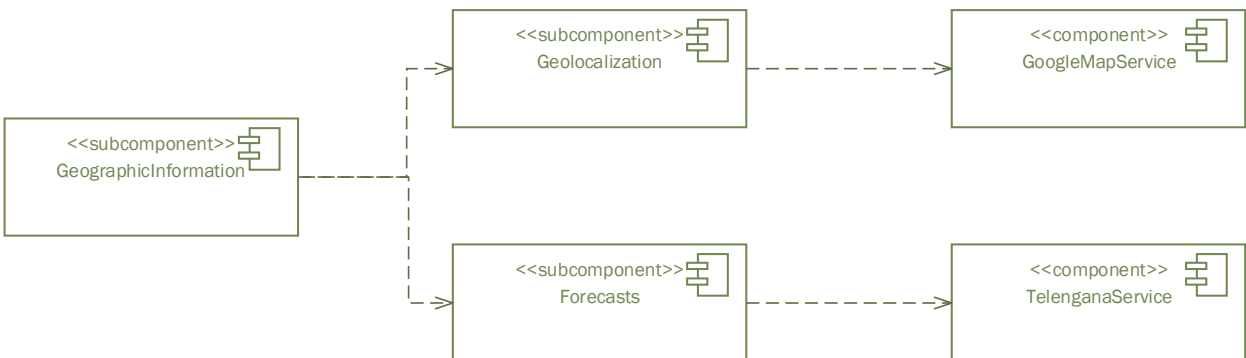


Figure 14 - Integration: GeographicInformation, Geolocalization, GoogleMapService, Forecasts, TelenganaService

- The next two steps can be implement and test parallel. Because their functionalities are independent.

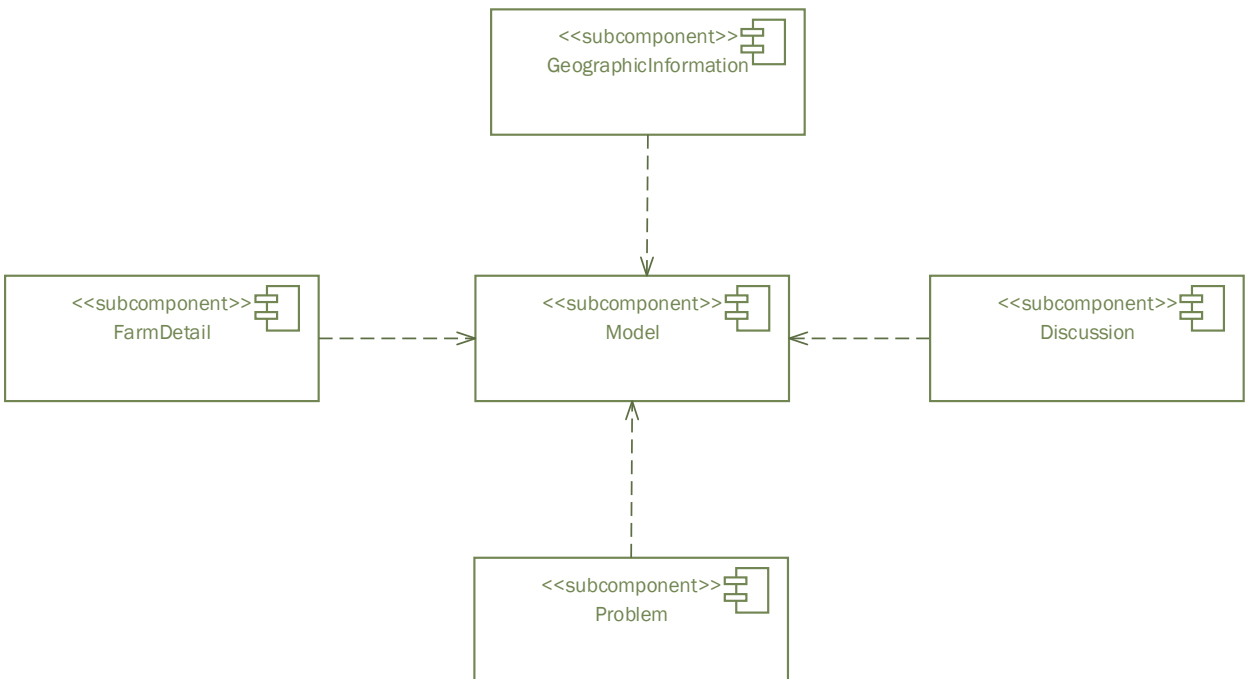


Figure 15 - Integration: Model, FarmDetail, Problem, Discussion, GeographicInformation

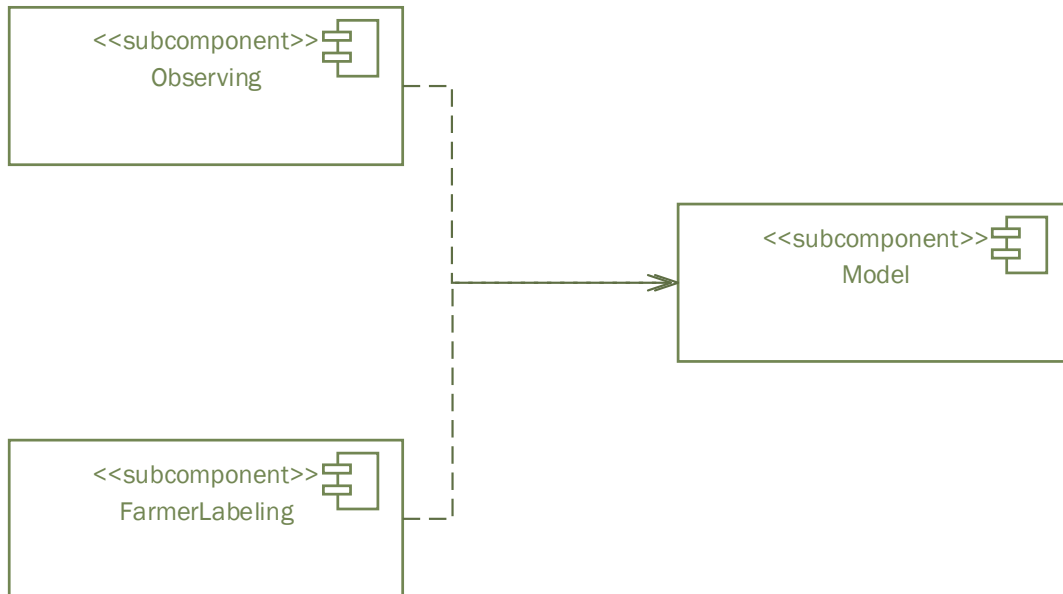


Figure 16 - Integration: Model, FarmerLabeling, Observing

## 6. Effort Spent

- **Student 1:**

Topics	Hours
General reasoning	4h
Preparing Introduction	0h
Overview of Architectural Design	1h
Component View	0.5h
Deployment View	0.5h
Runtime View	3.5h
Component Interfaces	1.5h
Selected Architectural Styles and Patterns	0.5
User Interface Design	2h
Requirement Traceability	1h
Implementing, Integration and Testing Plan	1.5h
Document Organization	4h

- **Student 2:**

Topics	Hours
General reasoning	4h
Preparing Introduction	1.5h
Overview of Architectural Design	0.5h
Component View	2h

Deployment View	1.5h
Runtime View	1h
Component Interfaces	0h
Selected Architectural Styles and Patterns	1.5h
User Interface Design	2h
Requirement Traceability	1h
Implementing, Integration and Testing Plan	1.5h
Document Organization	4h

## 7. References

- Specification Document: "01. Assignment RDD AY 2021-2022.pdf"
- UML diagrams: <https://www.uml---diagrams.org/>
- Slides of the lectures
- IEEE/ISO/IEC 29148-2018 - ISO/IEC/IEEE International Standard - Systems and software engineering - Life cycle processes - Requirements engineering