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

1. Introduction **4**

1.1. Purpose 4

1.2. Scope 4

1.3. Definitions, Acronyms and Abbreviations 4

1.3.1. Definitions 4

1.3.2. Acronyms 5

1.3.3. Abbreviations 5

1.4. Revision History 5

1.5. Reference Documents 5

1.6. Document Structure 5

2. Architectural Design **6**

2.1. Overview 6

2.1.1. High-level Architecture 7

2.2. Component View 8

2.3. Deployment View 9

2.4. Runtime View 10

2.5. Component Interfaces 17

2.6. Selected Architectural Styles and Patterns

2.7. Other Design Decisions

3. User Interface Design

4. Requirements Traceability

5. Implementation, Integration and Test Plan

6. Effort Spent

7. References

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 Telengana
DD	Design Document
RASD	Requirement Analysis and Specification Document
DBMS	Data-Base Management System
API	Application Programming Interface

1.3.3. Abbreviations

Abbreviations	Description

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.

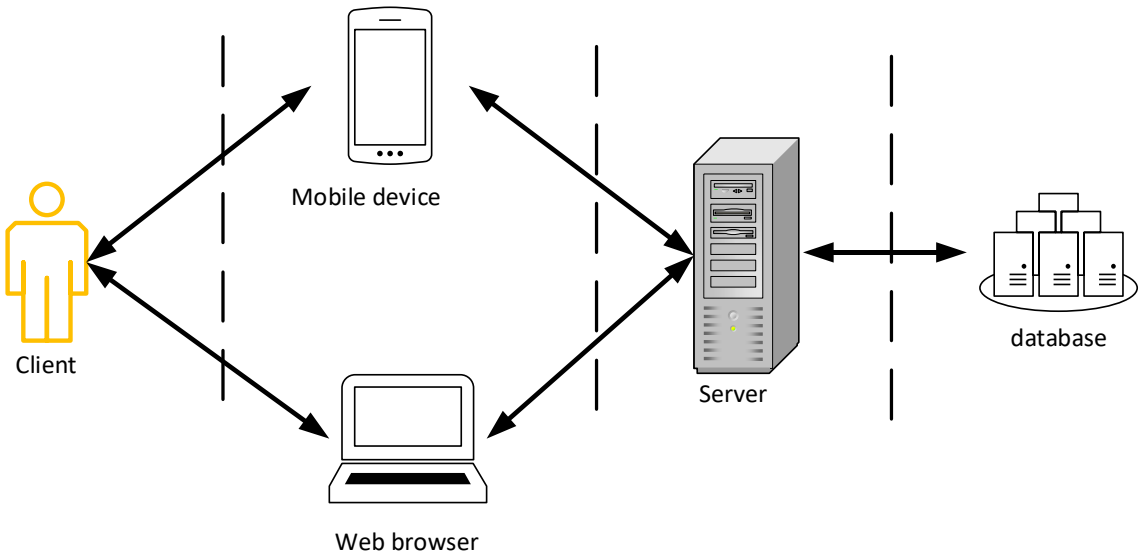

Figure 1 - Tree tier architecture

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.

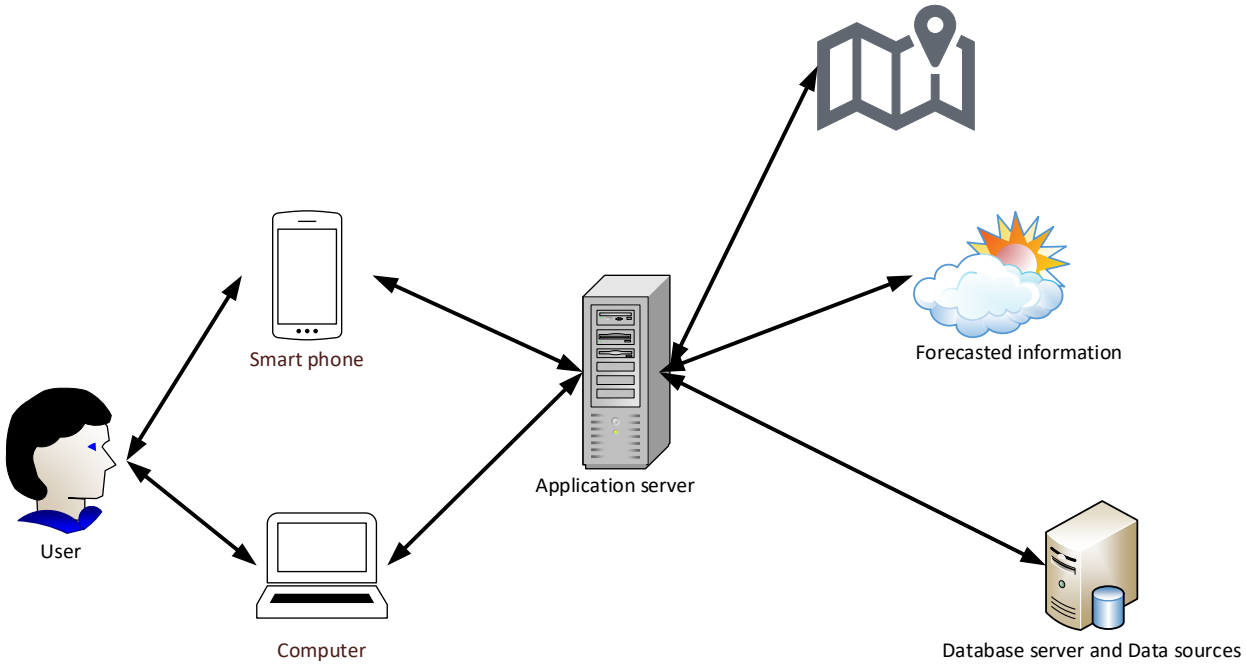

Figure 2 - System architecture

2.2. Component View

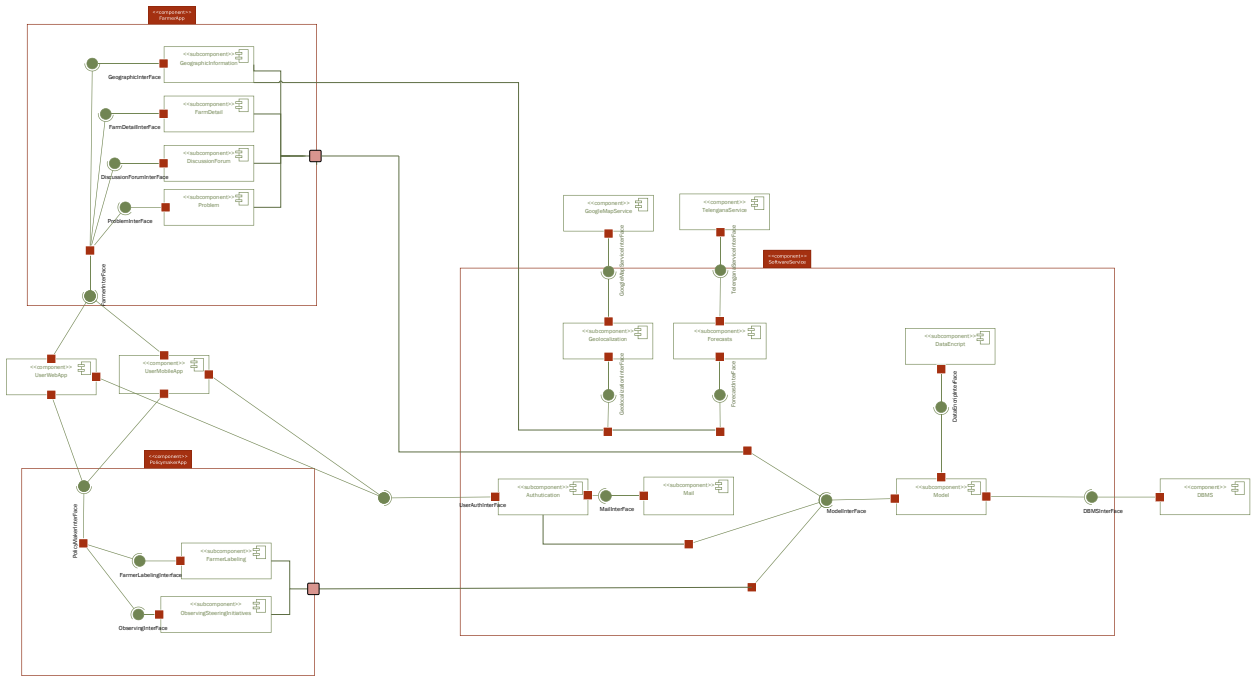

Figure 3 - Component View

2.3. Deployment View

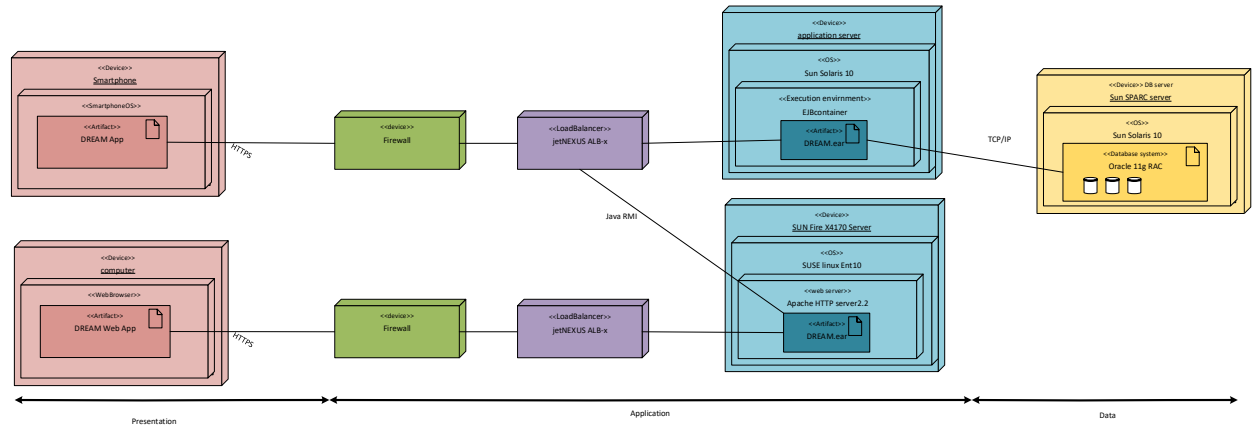

Figure 4 - Deployment View

- **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 the client 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.

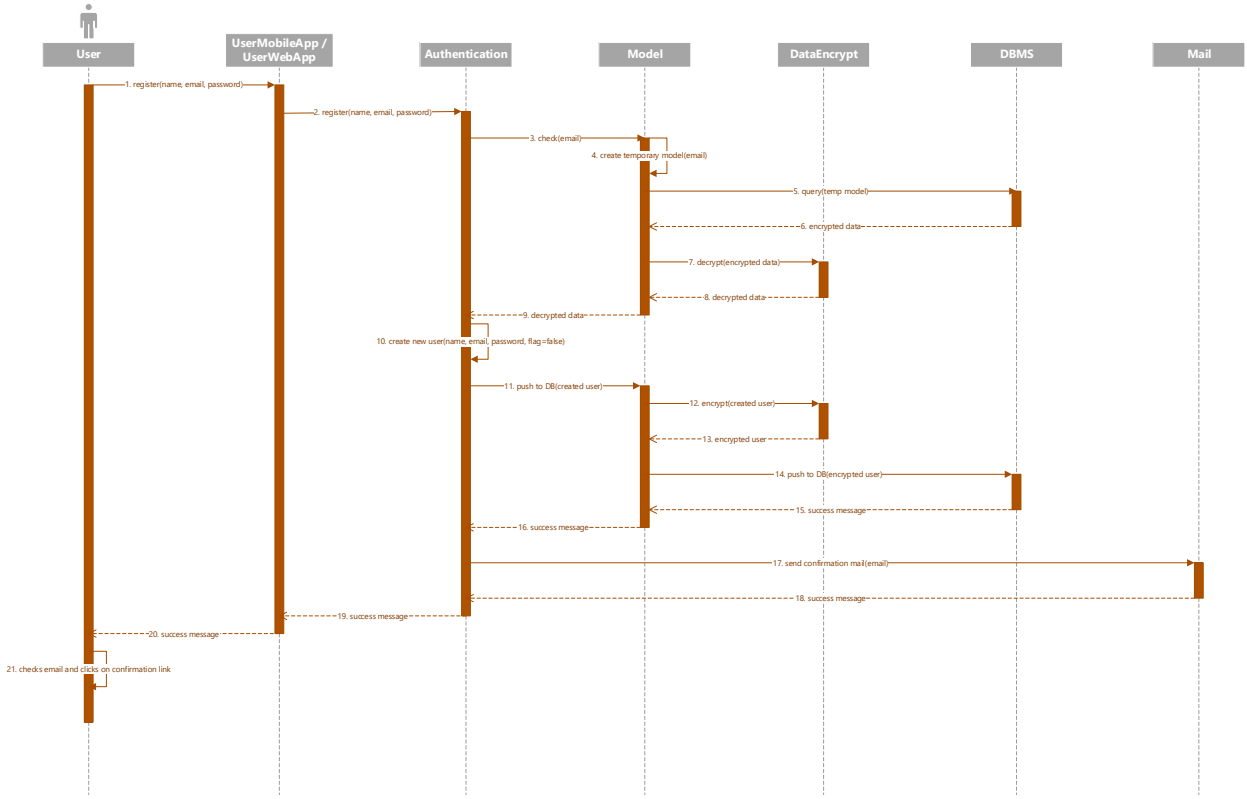

Figure 5 - Runtime View: Registration

- **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.

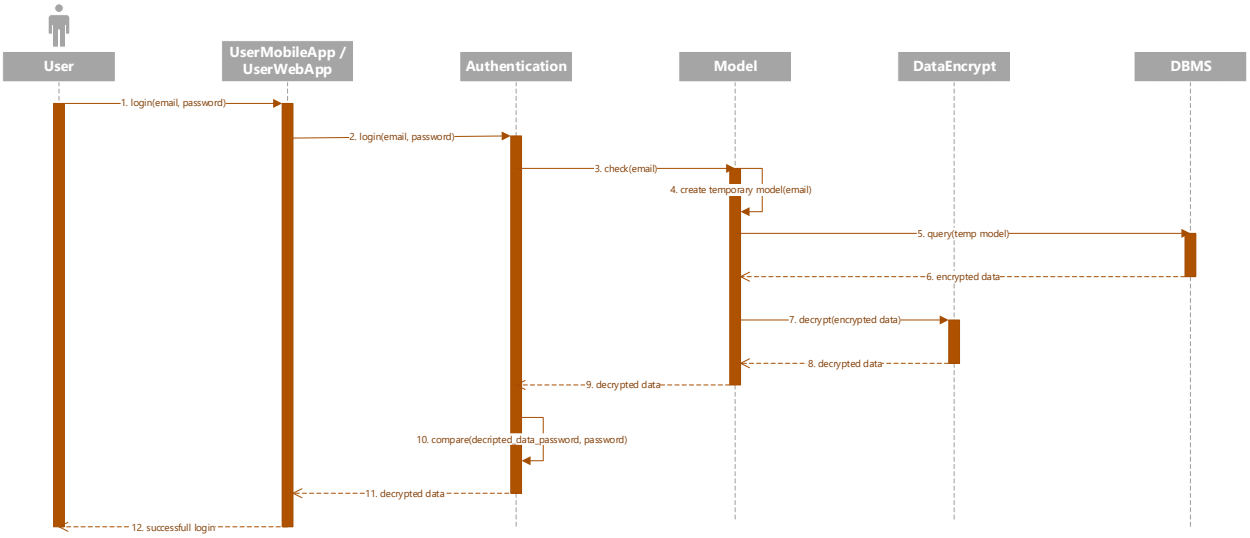

Figure 6 - Runtime View: Login

- **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.

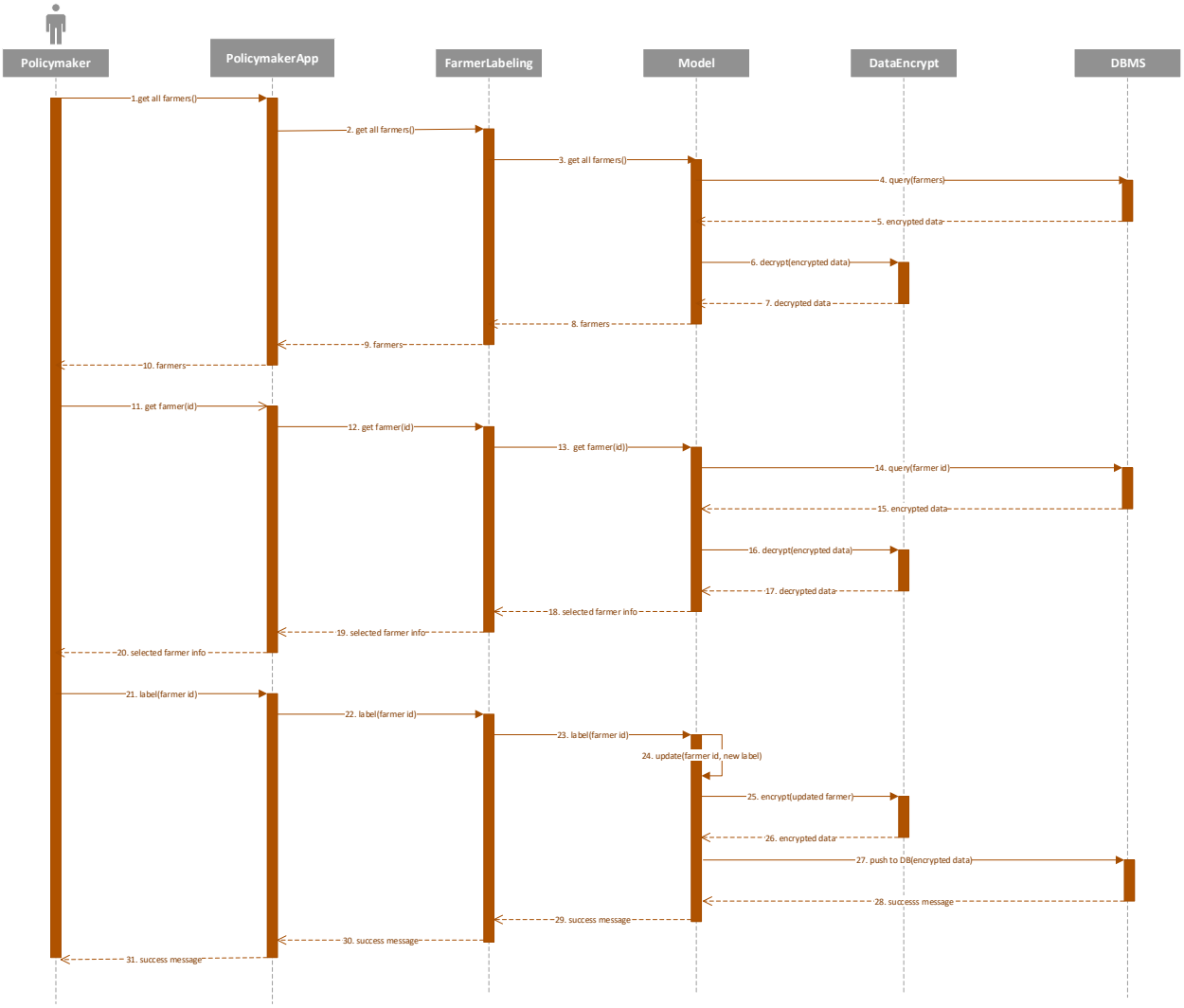

Figure 7 - Runtime View: Labeling Farmers

- **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.

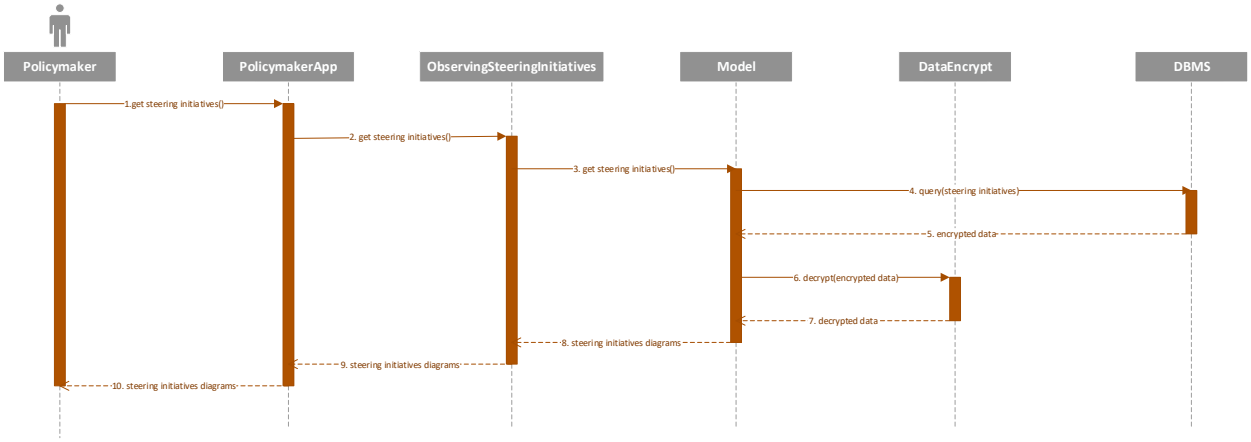

Figure 8 - Runtime View: Steering initiatives

- **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.

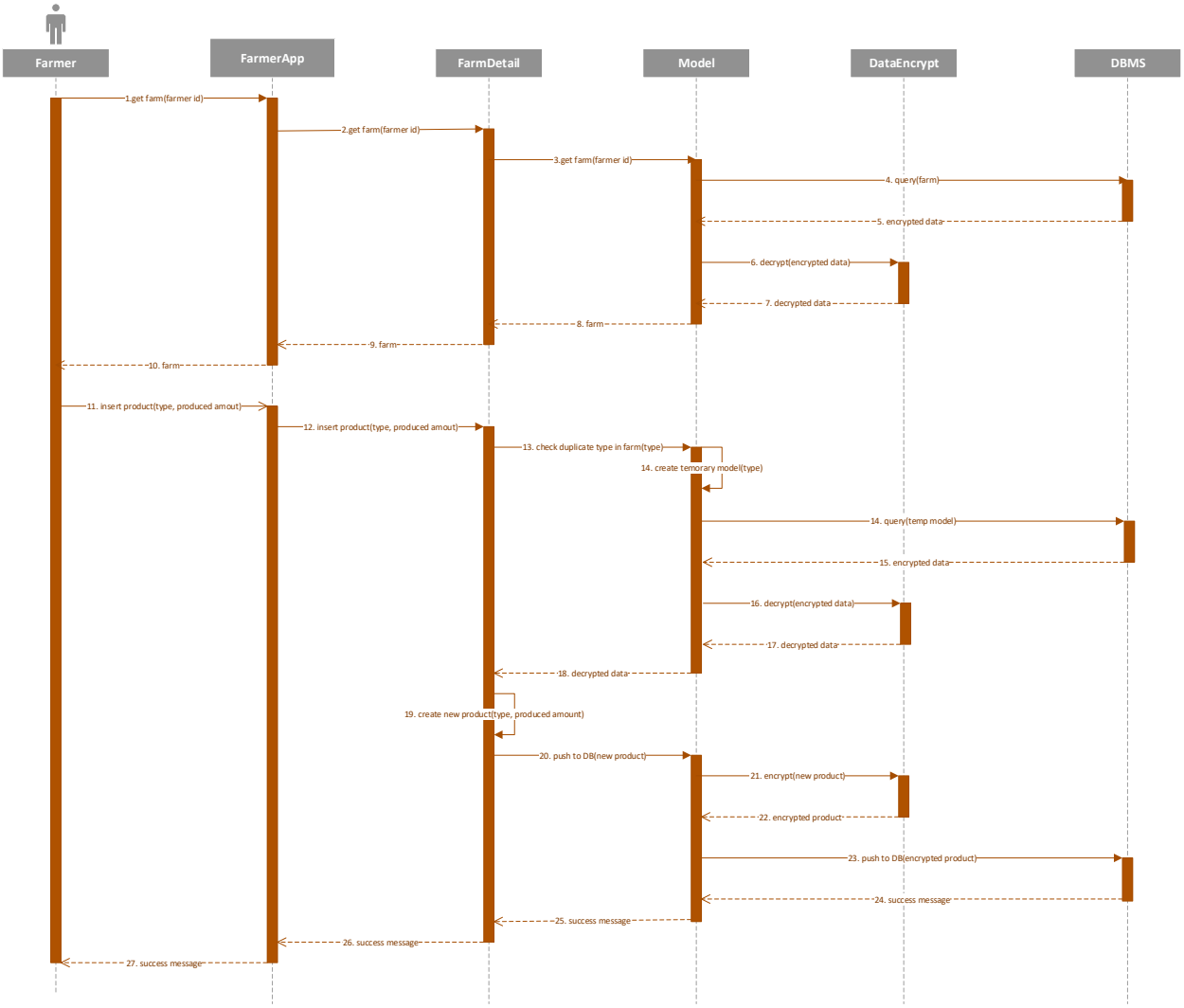

Figure 9 - Runtime View: Inserting Products

- **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.

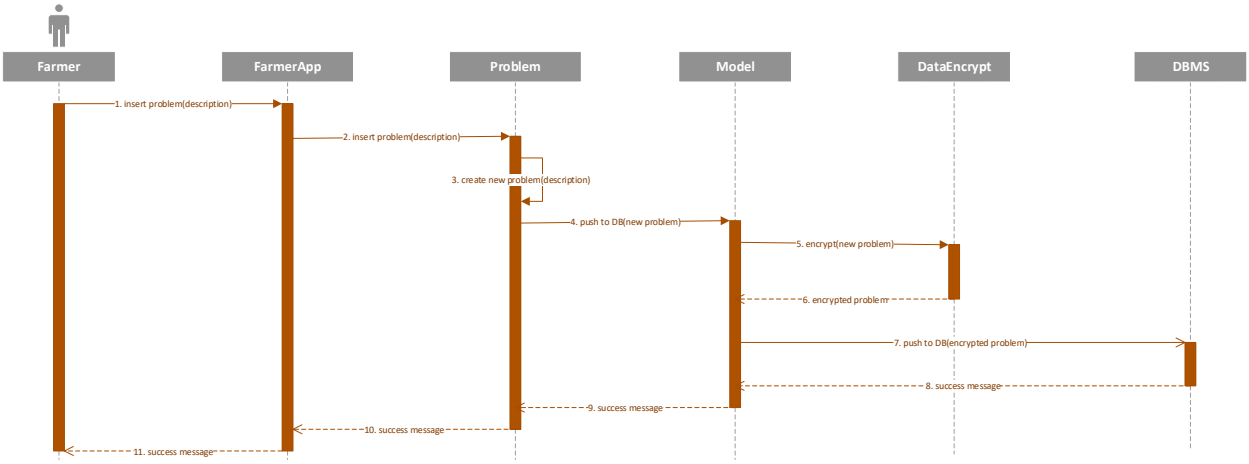

Figure 10 - Runtime View: Requesting for Help

- **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.

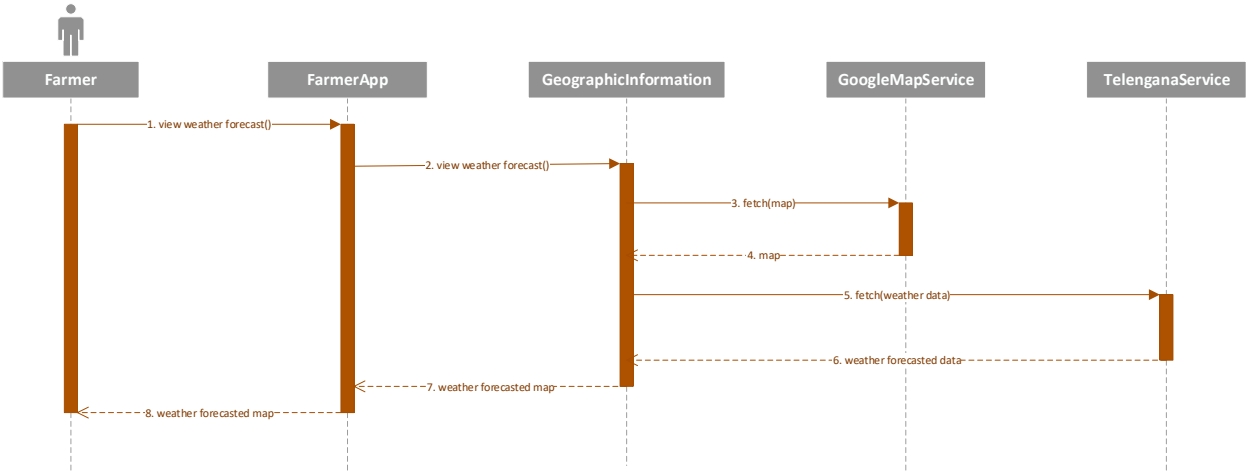

Figure 11 - Runtime View: Geographic Information

- **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.

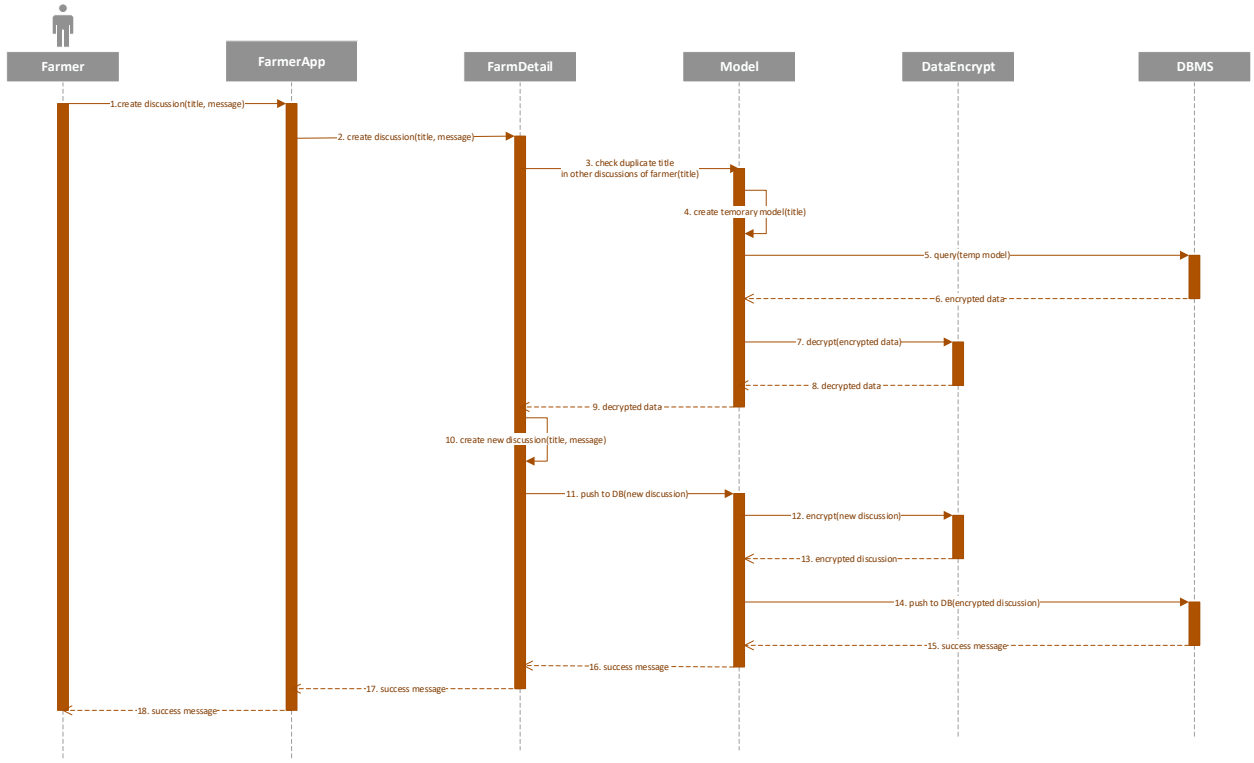

Figure 12 - Runtime View: Discussion forum

2.5. Component Interfaces

The following diagram represents the interfaces that are offered by the different components and their interactions.

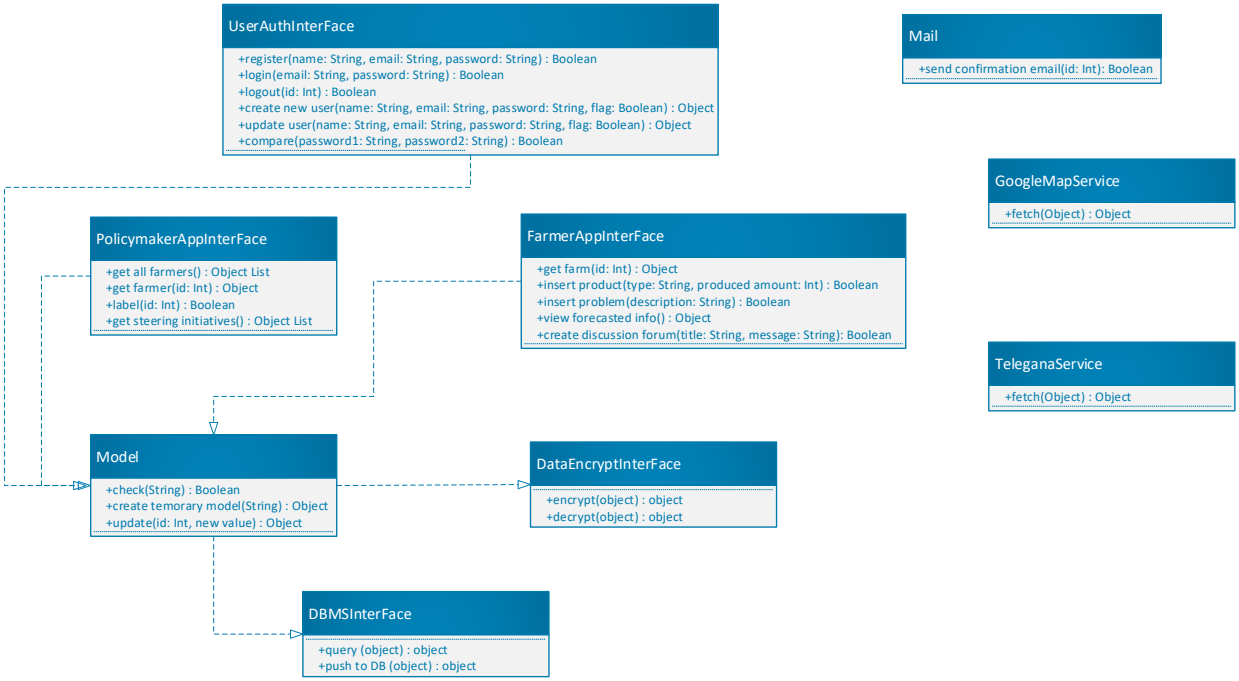

Figure 13 - Component Interface

2.6. Selected Architectural Styles and Patterns

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:

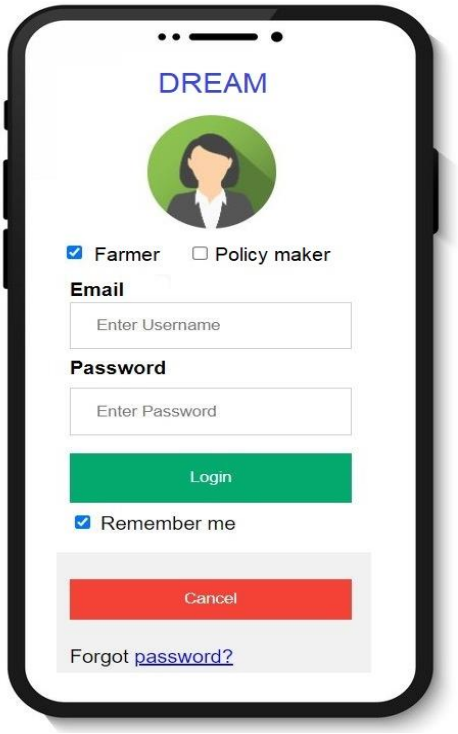

Figure - Mockup: Login

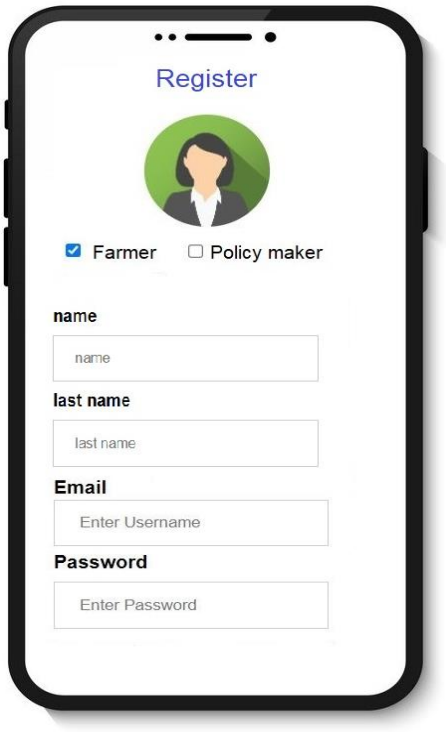

Figure - Mockup: Register

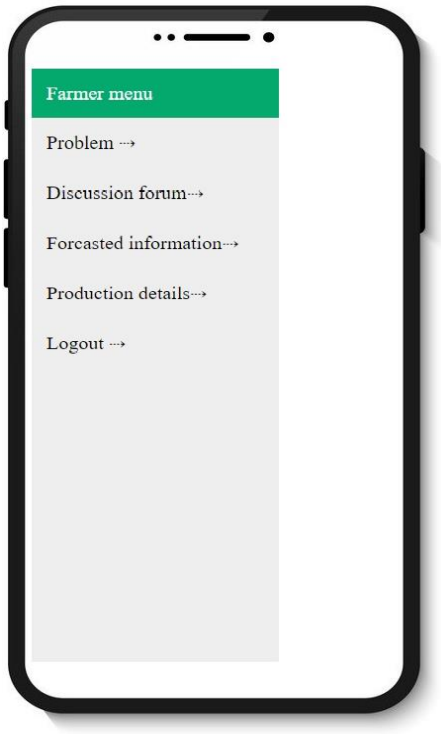

Figure - Mockup: Farmers' Menu

Figure - Mockup: Policymakers' Menu

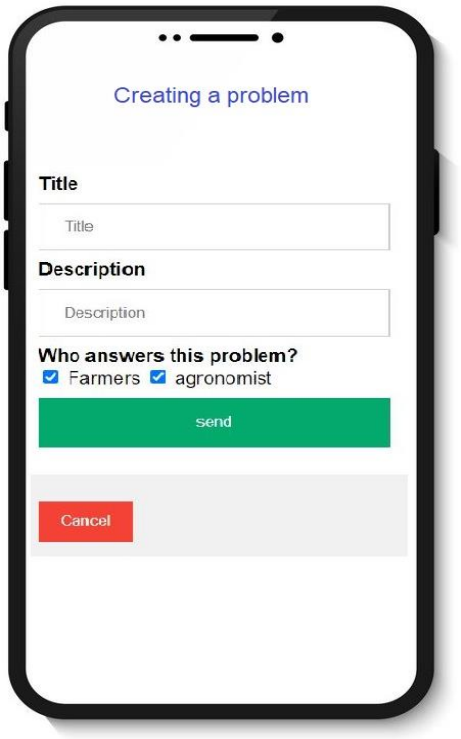

Figure - Mockup: Creating a Problem

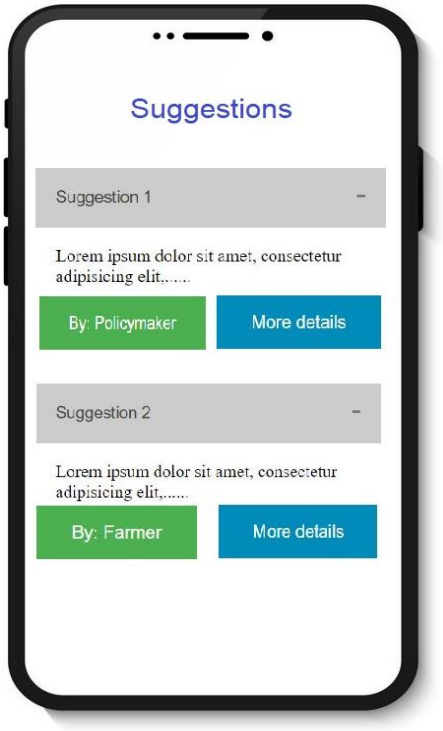

Figure - Mockup: View Suggestions

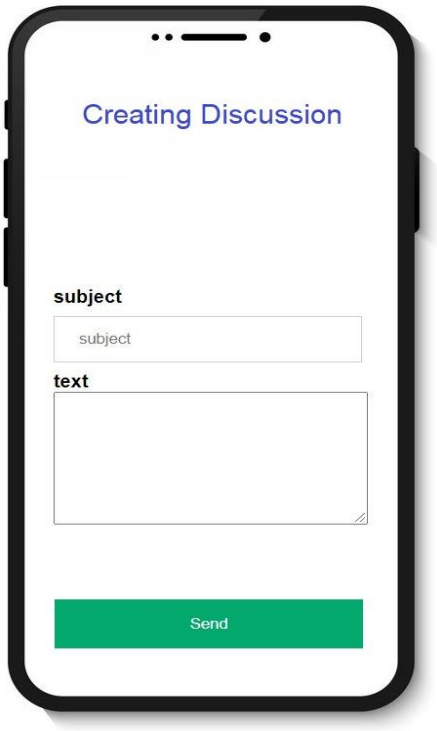

Figure - Mockup: Create a Discussion

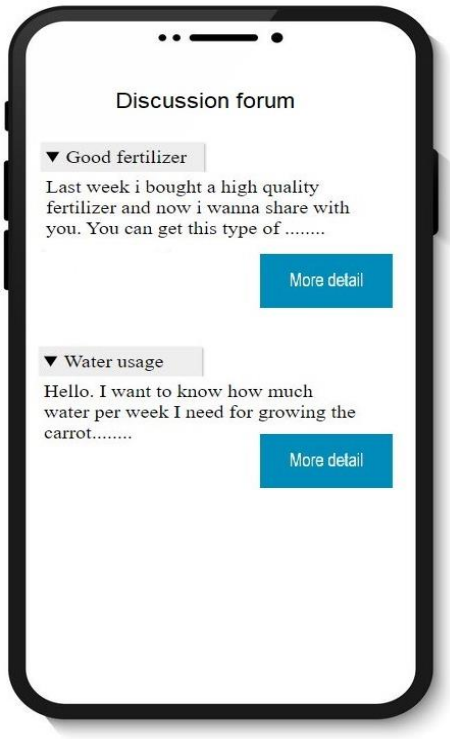

Figure - Mockup: List of Discussions

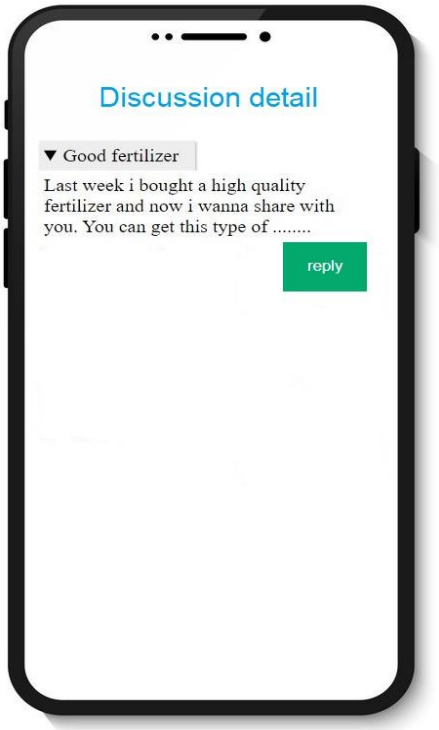

Figure - Mockup: Reply to a Message

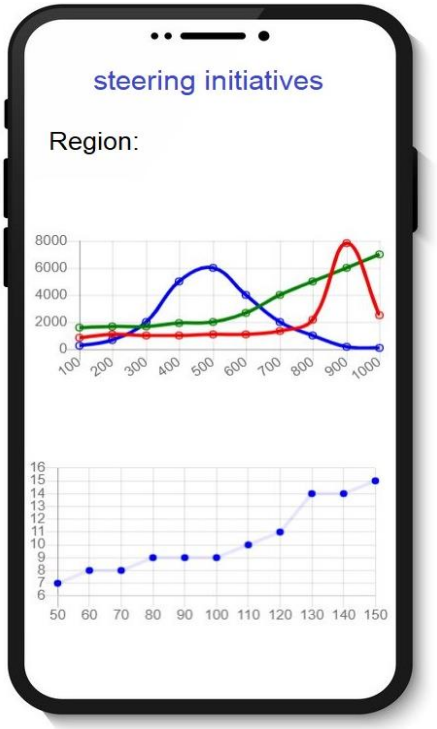

Figure - Mockup: Steering initiatives

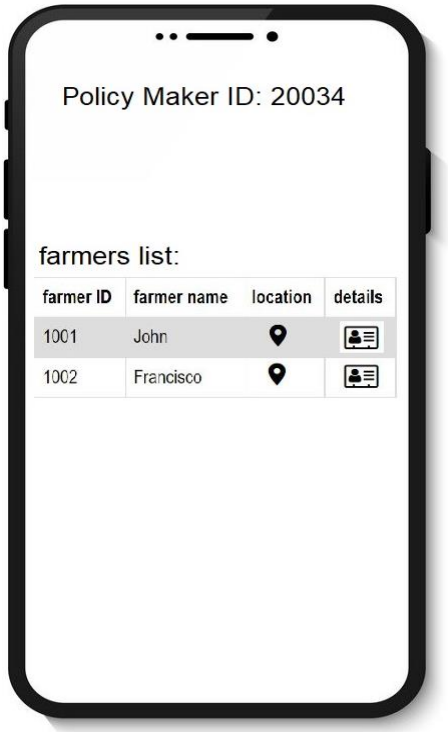

Figure - Mockup: List of Farmers

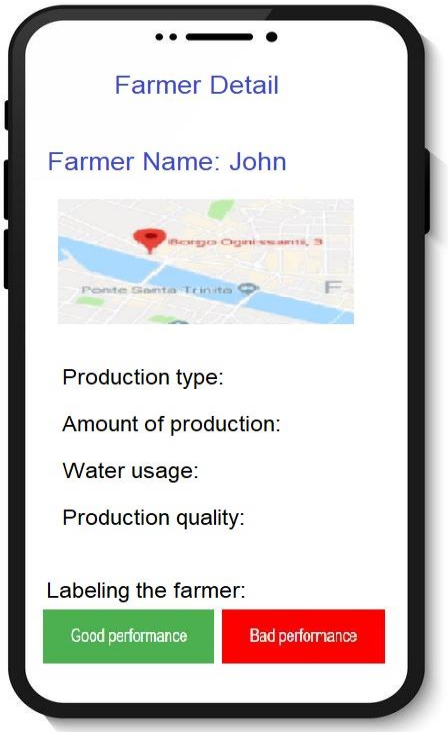

Figure - Mockup: Farmer's Detail

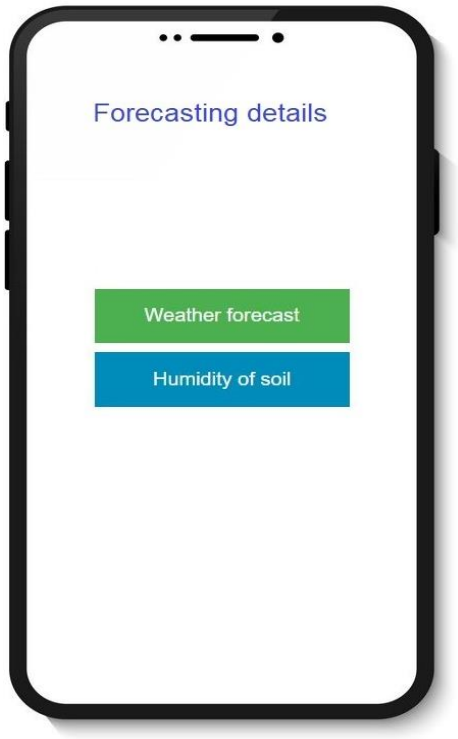

Figure - Mockup: Forecasted Details

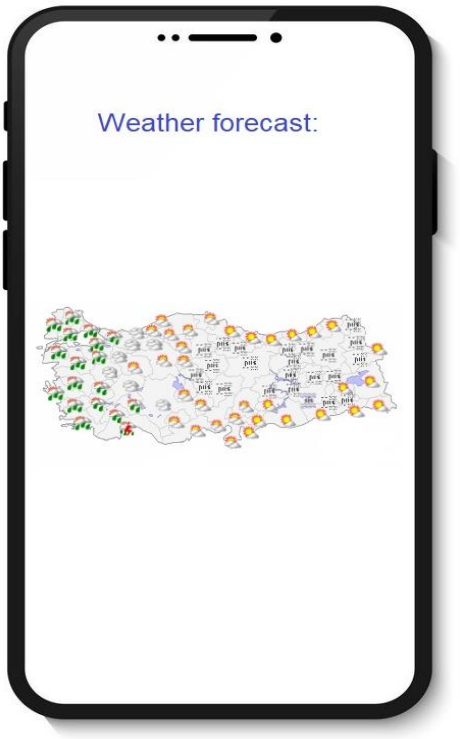

Figure - Mockup: Weather

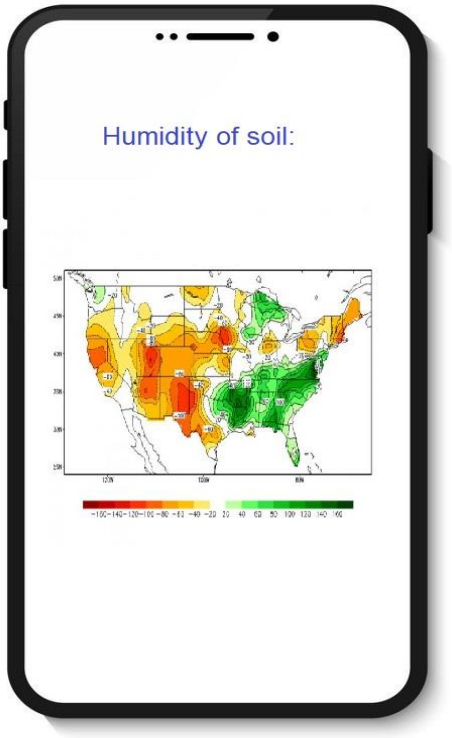

Figure 11 - Mockup: Humidity of Soil

6. Effort Spent

- **Student 1:**

Topics	Hours

- **Student 2:**

Topics	Hours

7. References