Politecnico di Milano

Computer Science and Engineering

Design Document

DREAM

Software Engineering 2

Optional Project AY 2021-2022

Curated by: Francesco Mazzola and Alessio Ferrara

CONTENTS

1 Introduction

1.1 Purpose

The purpose of this document is to provide an overall guidance to the architecture of this software product and therefore it is primarily addressed to the development team.

The document presents an overview of the high-level architecture and adds details on the runtime behaviour and the user interface of the application.

Finally, it also includes a plan for the implementation, integration and testing activity.

Together with the RASD, this document has the purpose to guide the developers in the realization of the DREAM software.

1.2 Scope

DREAM is an **easy-to-use** application which aims to improve the production of the farmers on Telangana and at the same time to help the policy makers have an overview on the performance of the farming and agronomists.

Both farmers and policy makers will receive the credentials to login via email by a Telangana's government employee.

Farmers can insert productions both from computer or mobile devices, moreover they can read personalized information and create/participate in a discussion at any moment.
In addiction Farmers can update their personal information to keep receiving the correct personalized information by the application.

Policy maker, through the Web App, will have an overview on how the farmer is performing and can evaluate both the farmer and the steering initiatives at any moment, having all the needed data to do correct evaluations.

1.3 Definitions, acronyms and abbreviations

1.3.1 Definitions

- **User:** person who has credentials to log in into the system, it abstracts the concepts of farmer and policy maker.
- **Farmer:** person who can use the farmer service provided by DREAM.
- **Policy maker:** person who can use the policy maker services offered by DREAM.
- **Steering Initiative:** A set of instructions that are provided by an Agronomist that is assigned to a farmer that is performing badly
- **Agronomist:** A person that provides useful practices to those farmers that are having problems

1.3.2 Acronyms

- **RASD:** Requirement Analysis and Specification Document.
- **DD:** Design Document.
- **JSON:** JavaScript Object Notation
- **REST:** Representational State Transfer
- **API:** Application Programming Interface
- **GPS:** Global Positioning System

1.3.3 Abbreviations

- **DREAM:** Data-dRiven PrEdictive FArMing in Telangana.
- **Gx:** Goal number x.
- **Dx:** Domain assumption number x.
- **Rx:** Functional requirement number x.

1.4 Reference documents

- Project assignment specification document
- DREAM, Requirements Analysis and Specification Document.
- Course slides on WeBeep.

1.5 Document structure

- **Section 1: Introduction:** The first section introduces the purpose of the document and the scope of the DREAM system. Included here is a glossary with the definitions, acronyms and abbreviations used in this document.
- **Section 2 Architectural design:** This is the core section of the DD; it gives an overview of the main components of the system and the relationship between them providing the most relevant views. This section also focusses on the main architectural styles and patterns adopted in the design of the system.
- **Section 3 User interface design:** In this section the UI mock-ups and UI Flow diagrams are provided to better understand the paths that each user can follow while using the system.
- **Section 4 Requirements traceability:** This section associates the decision taken in the RASD with the ones taken in this DD.
- **Section 5 Implementation, integration and test plan:** Here is specified in which order the different components of the application are developed and integrated with each other. This section also specifies a strategy to follow to correctly test the implementation of the system.
- **Section 6 Effort spent:** This section includes information on the number of hours each group member worked for this document.

2 Architectural designs

2.1 Overview

The DREAM system has a four-tier architecture that can be grouped into 3 logical layers: *presentation, application and data*.

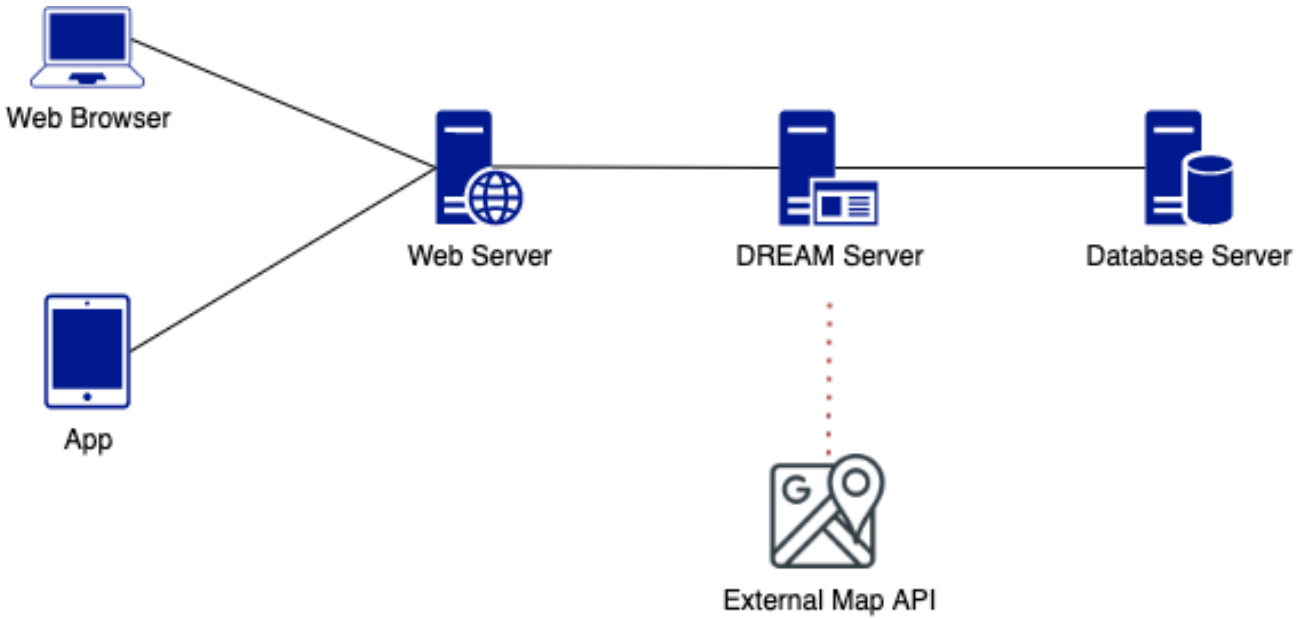

Figure 1: DREAM system diagram

On the client side we have browser used to connect to the web server and to dynamically update the *Web View* when needed.

The web server acts as a middleware between the client interface and the system logic, communicating with the user via the standard HTTP protocol, receiving requests and giving responses.

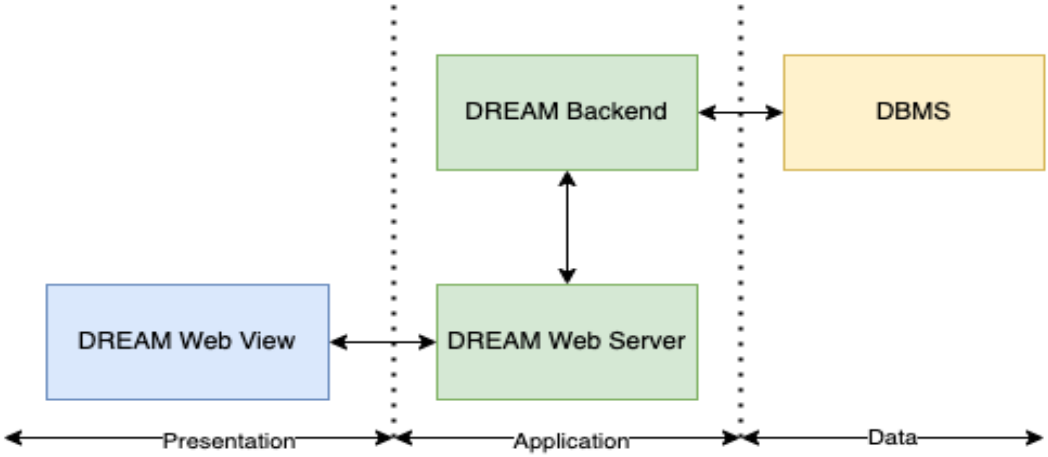

Figure 2: High level layers diagram

Finally, the business logic of the system resides on the *Back end.* This node can manage the connection with the DBMS and takes care of performing the necessary processing for the correct functioning of the system.

2.2 Component view

The diagrams below show the main components of the DREAM system and the interfaces through which they interact.

The first diagram shows a high-level view of the components, which will be further explored individually.

The system exposes a RESTful API with multiple public endpoint and resources, some of them require a proper authentication and authorization level to be used.

2.2.1 High level component

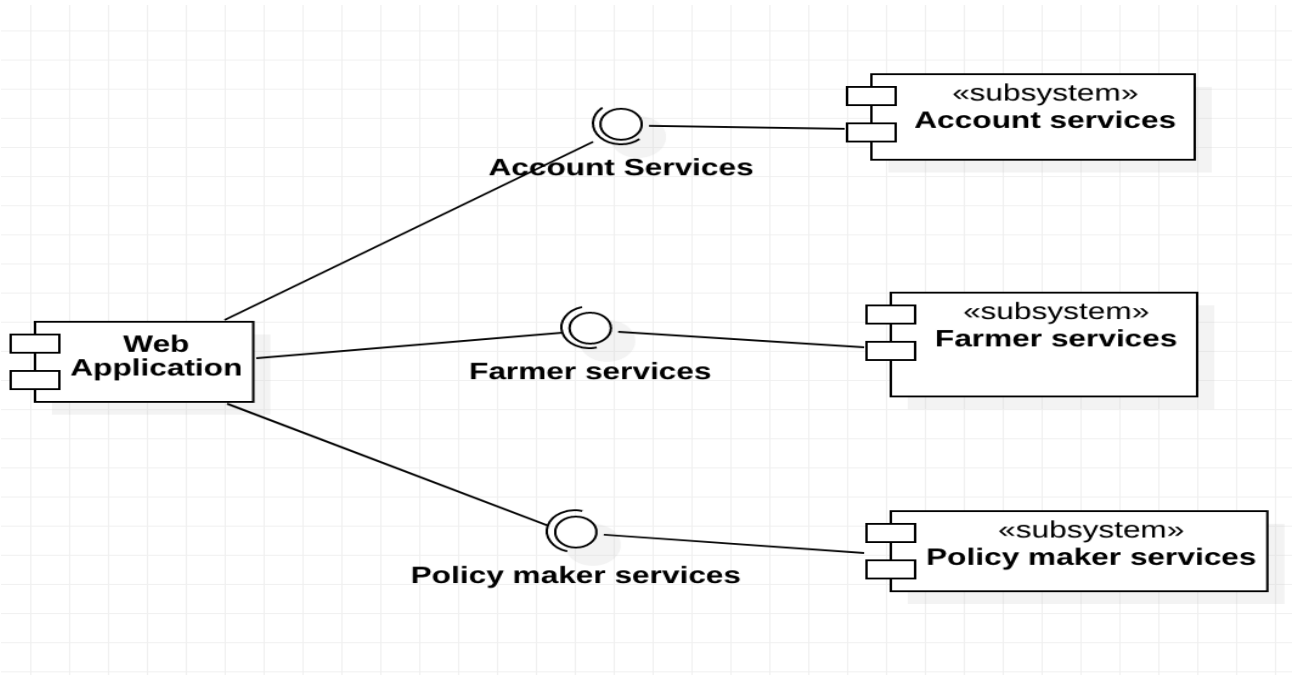

Figure 3: High level component diagram

The client side consists in a single component that refers to the web application, the only access point to this system.

The server side, instead, is composed of three subsystems:

- **Account Services:** provides support to login and logout operations for any type of user.
- **Farmer Services:** provides access to services of the application reserved to the farmers.
- **Policy maker Services** provides access to the decisional services of the application, these services are reserved to the policy makers.

2.2.2 Account Service

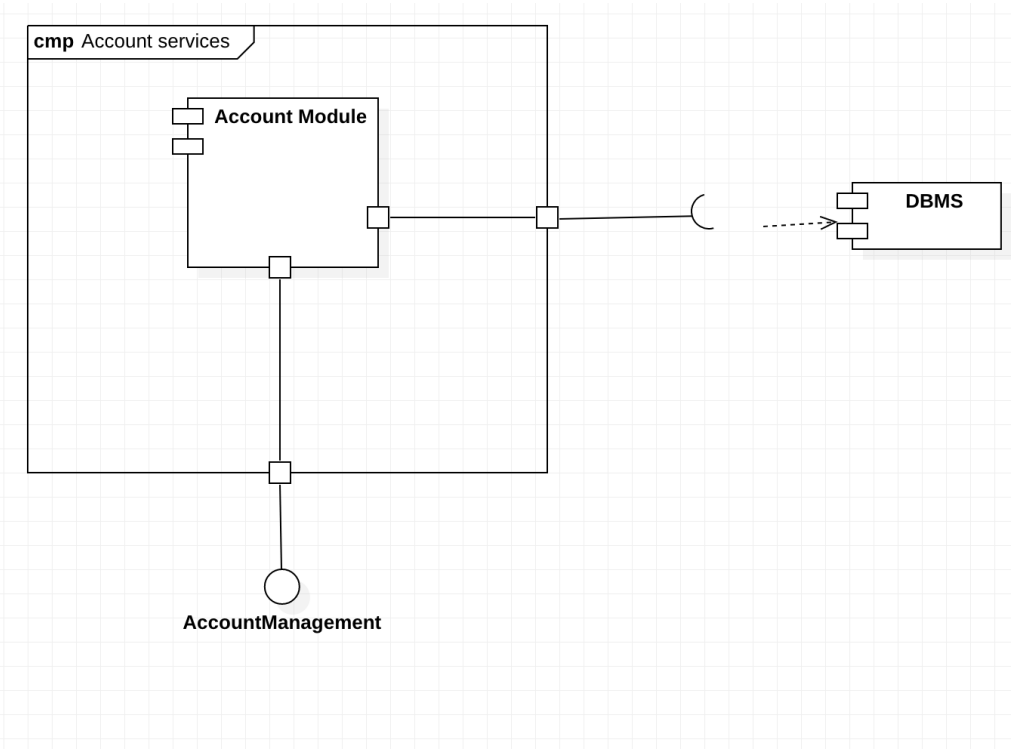

Figure 4: Account Service component diagram

The Account Services subsystem contains one component:

- **Account Module:** provides the *AccountManagement* interface to handle the operations of login and logout.

2.2.3 Farmer Service

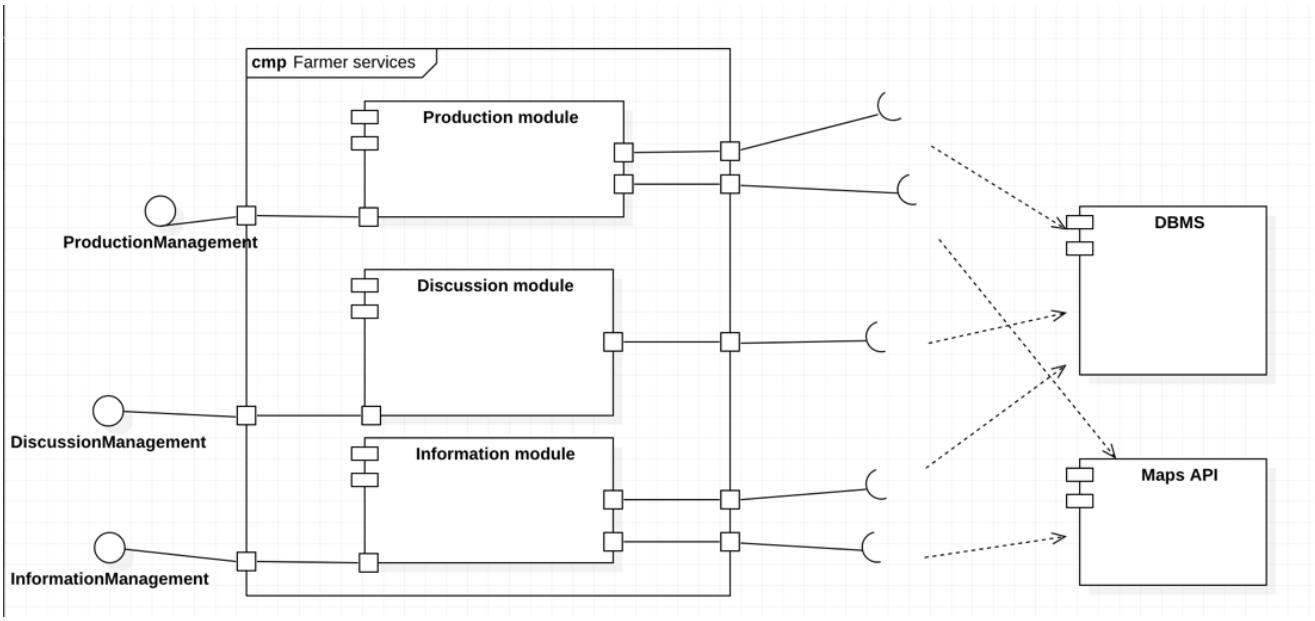

Figure 5: Farmer Service component diagram

The Farmer Service subsystem contains three components:

- **Production Module:** offers the *ProductionManagement* interface to handle all the operations related to the insert production functions.
- **Discussion Module:** offers the *DiscussionManagement* interface to handle all the operations related to the discussions.
- **Information Module:** offers the *InformationManagement* interface that allow the farmer to check personalized data and manage his informations.

In order to fulfill their goals, these components need to communicate with the DMBS and the Maps API through the corresponding interfaces.

2.2.4 Policy maker service

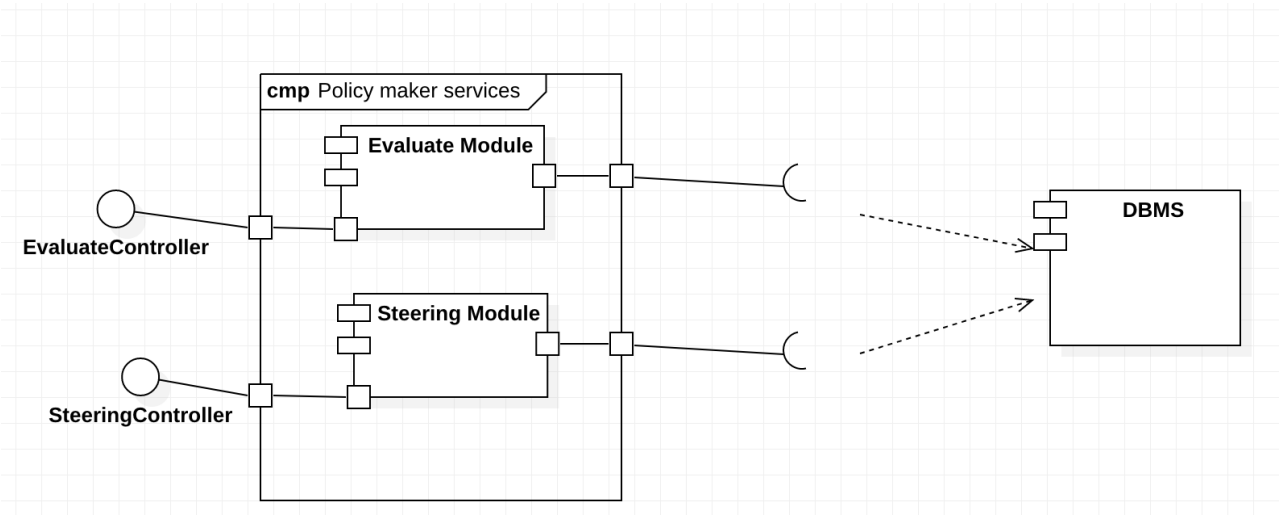

Figure 6: Policy maker Service component diagram

The policy maker Service contains two components:

- **Evaluate Module** offers the *EvaluateController* interface to perform the evaluation of both the farmer and the steering initiatives.
- **Steering Module** offers the SteeringController interface to assign a badly performing farmer to a new steering initiative.

In order to fulfill their goals, these component needs to communicate with the DBMS.

2.3 Deployment view

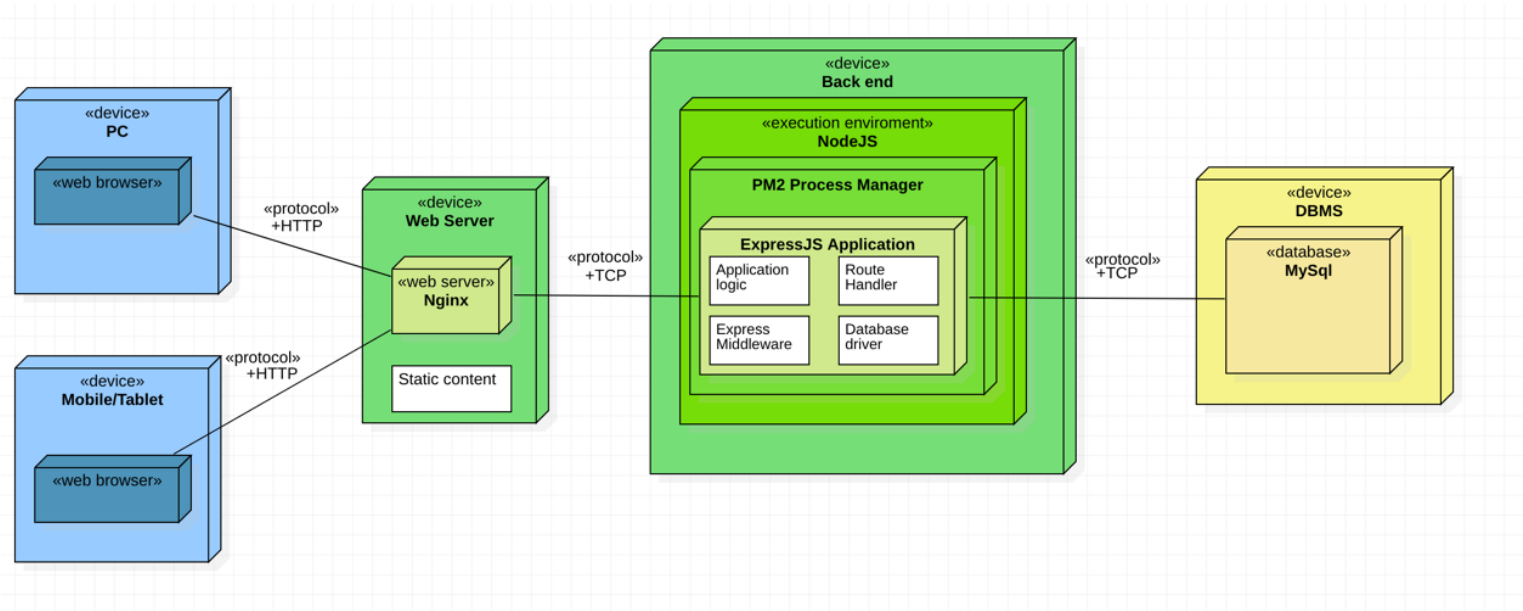

Figure 7: Deployment diagram of DREAM

The system architecture is divided in 4 tiers:

- The first tier is the *client tier*: it is composed by any device capable of rendering a web page. It communicates with the web tier through the HTTP protocol.
- The *web tier* contains the web server implemented with Nginx.
- The third tier is the *application tier.* The DREAM's Back end is built on an Express application which is managed by a process manager to spawn multiple instances (PM2). The execution environment is the NodeJS runtime.
- The *data tier* is the fourth tier composed by the Database server.

2.3.1 Recommended implementation

- **Client tier:** The client web browser may be an arbitrary one, but it should be able to render HTML5 and CSS3 web pages and execute JavaScript.
- **Web tier:** The web tier must be implemented with Nginx web server.
- **Application tier:** The runtime engine where the backend is hosted is NodeJS and the docker-like process manager is PM2, it is used to create multiple instances of the application, which uses ExpressJS to expose services.

- **Data tier:** The database is relational and implemented using MySQL.

2.4 Runtime view

The following sequence diagrams describe the interactions between the main components of the product when utilizing the most common features.

This is still a high-level description of the actual interactions so that they can be slightly modified during the development process.

2.4.1 User Login

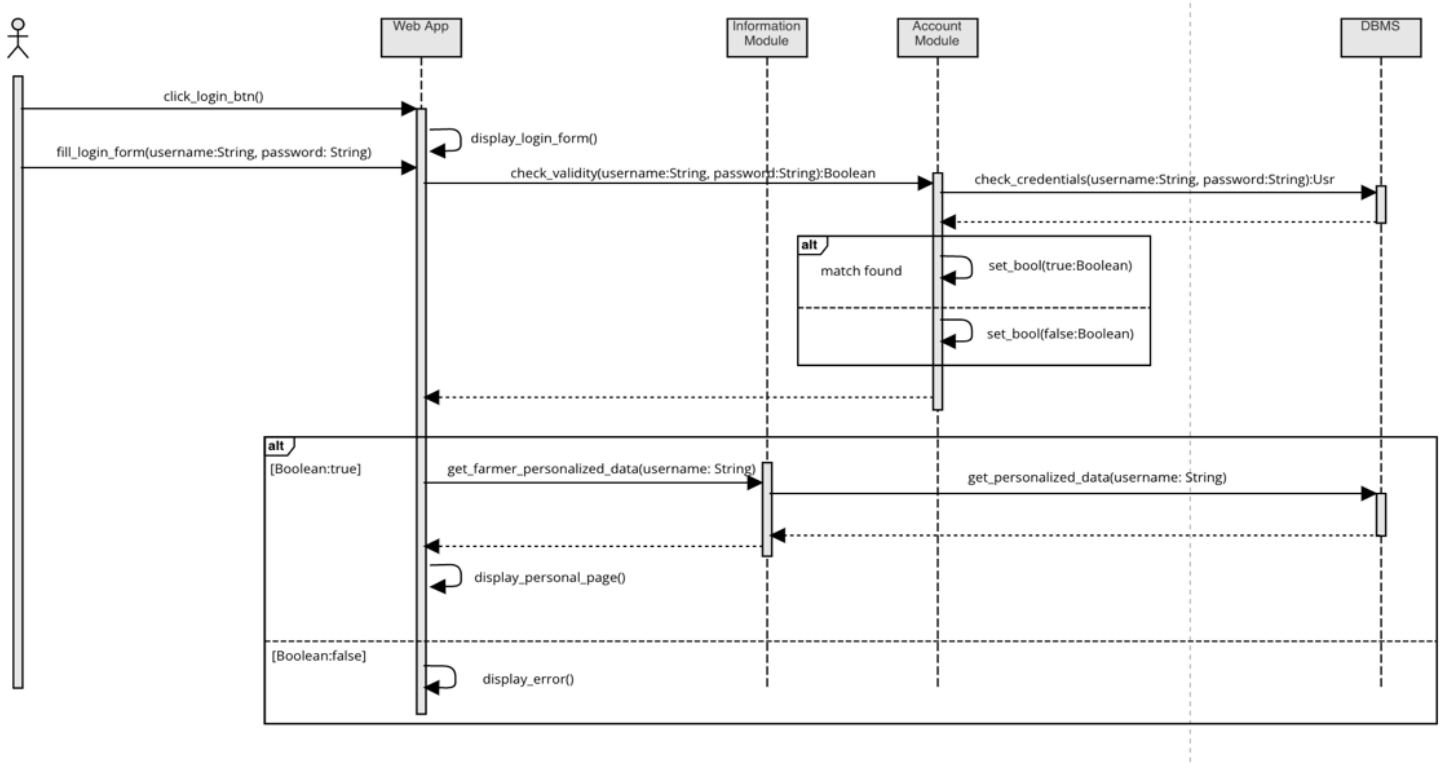

Figure 8: User login request

2.4.2 Inserting Production

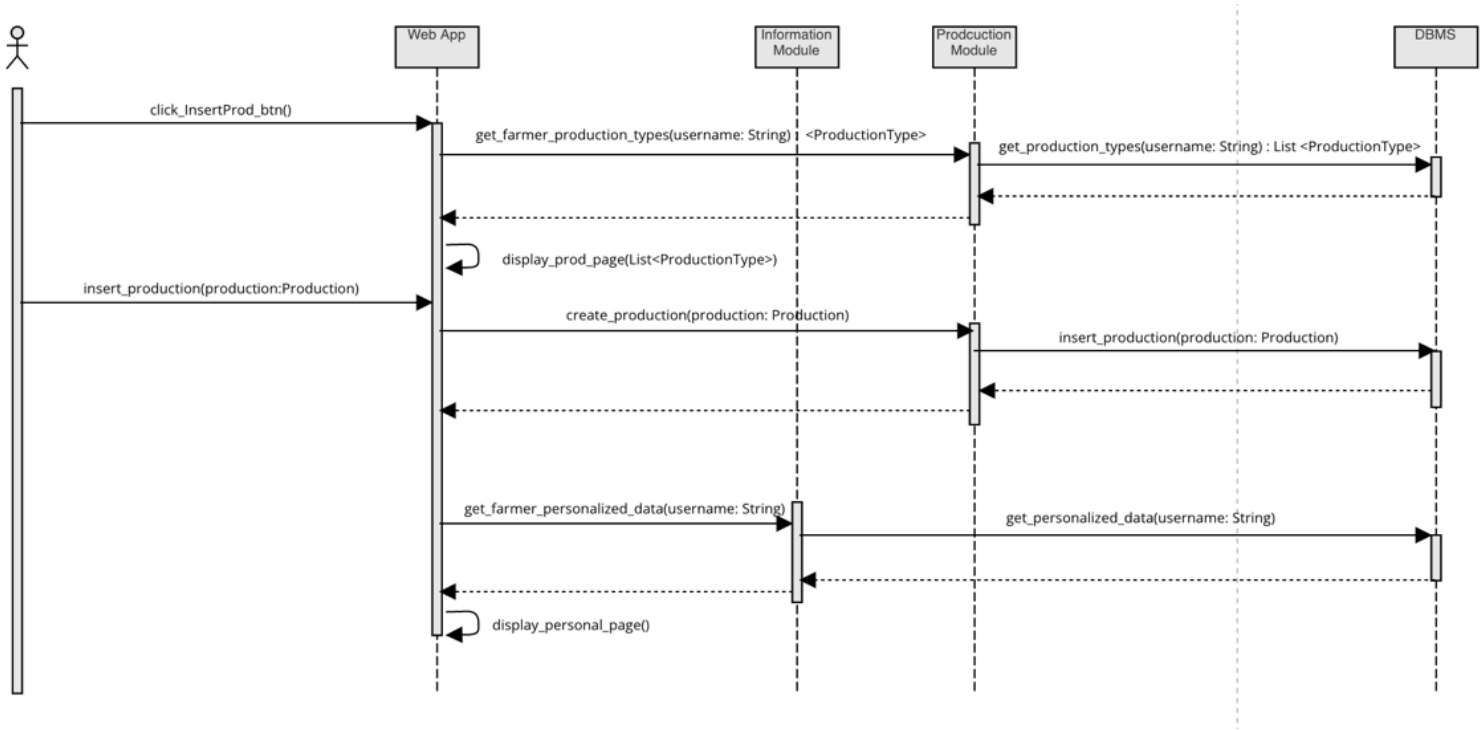

Figure 9: Insert production request

2.4.3 Evaluating Farmer

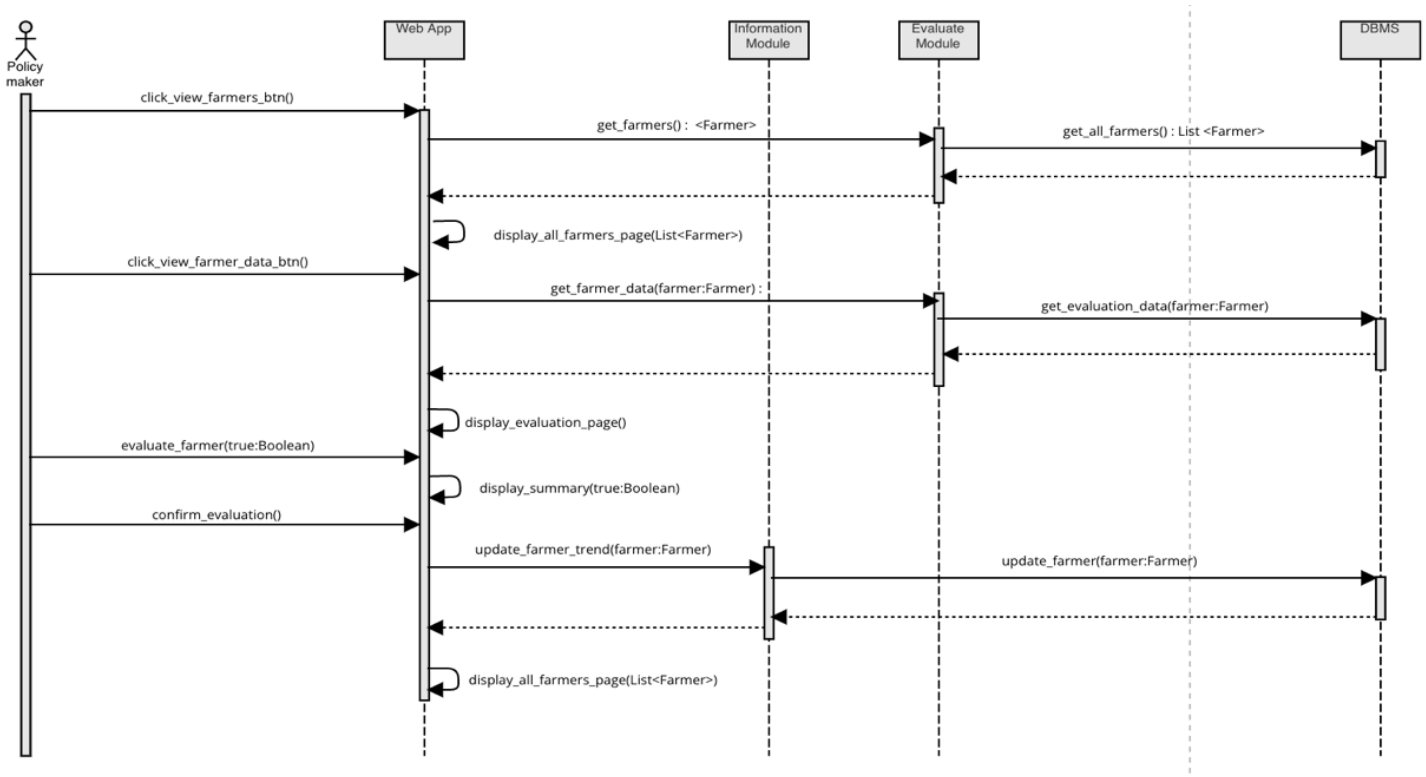

Figure 10: Evaluating Farmer request

2.5 Component interface

This section lists the main methods that each component interface provides to the other components.

- *AccountServices* is an interface accessible by users. It allows the users to login, and logout.
- *FarmerServices* is an interface that provides every method the Farmer could need to access application's functionalities. There are methods to insert production, manage personal information and interacting in discussions.
- *PolicyMakerServices* is an interface that provides every method the Policy Maker needs in order to use the application. There are methods to get information about farmers and steering initiatives, to evaluate them and to assign a farmer to a steering initative.

2.5.1 AccountServices interface

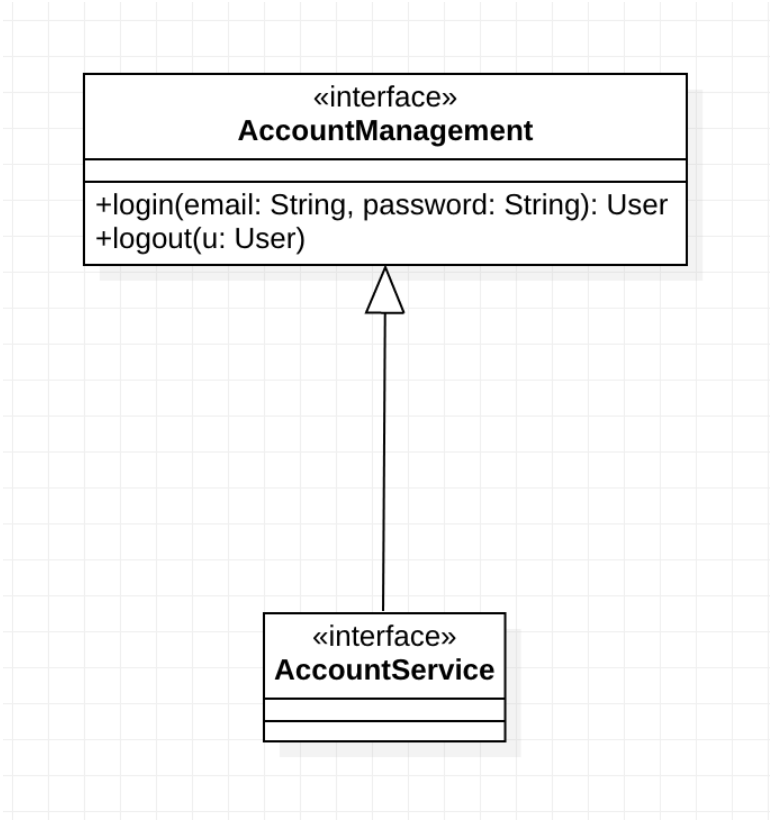

Figure 11: AccountService interface diagram

2.5.2 FarmerServices interface

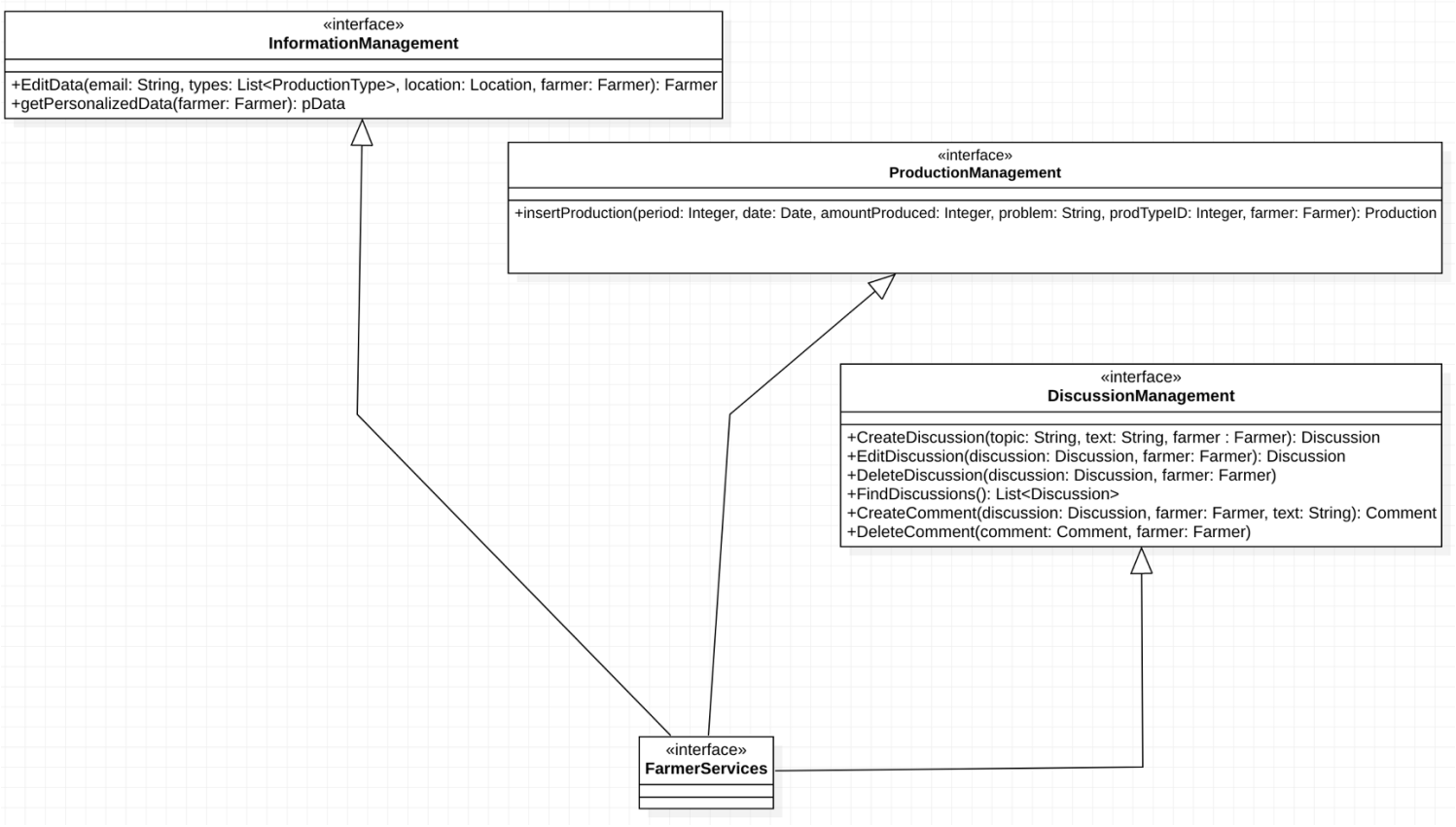

Figure 12: FarmerServices interface diagram

2.5.3 PolicyMakerServices interface

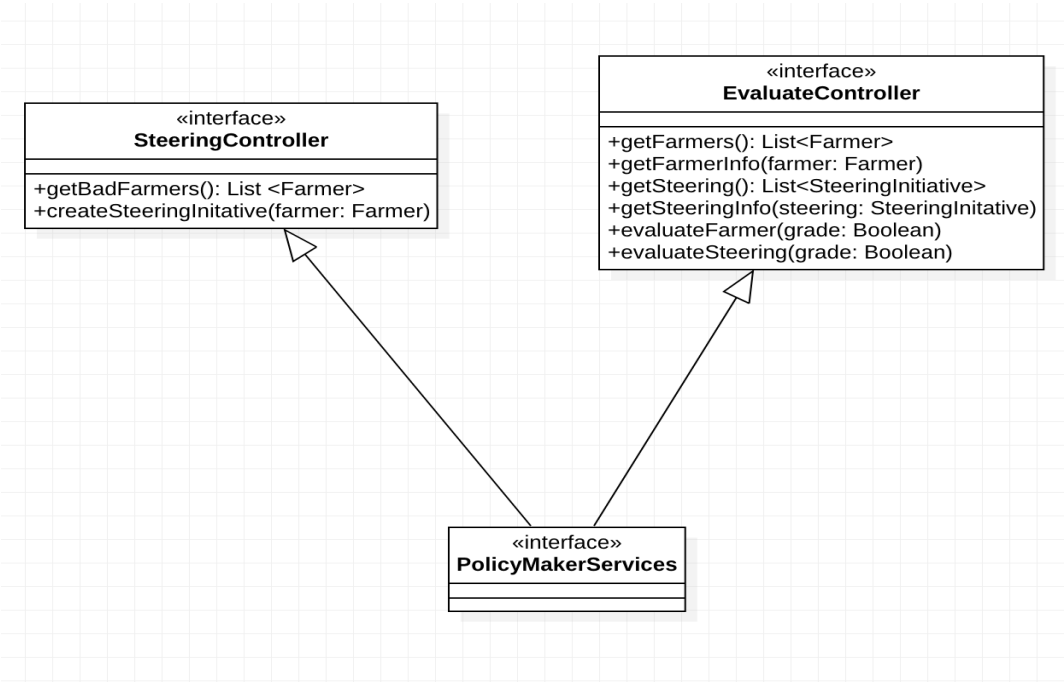

Figure 13:PolicyMakerServices interface diagram

2.6 Selected architectural styles and patterns

2.6.1 MVC pattern

The Model-View-Controller pattern is used in order to obtain a clear division between the internal representation of information, the way in which that information is presented to the user and the business logic of the system.

2.6.2 RESTful API with JSON

The application layer of the system offers a set of service that each client can reach via HTTP protocol.

As prescribed by the REST principles, for each incoming request the server provides a response that can also include a JSON representation of one or more objects

2.7 Other design decisions

2.7.1 Database Structure

As already mentioned, all the data will be stored in a database. The chosen DBMS is MySQL since it is an open-source relational database management system (RDBMS). The following diagram explains how the database is structured.

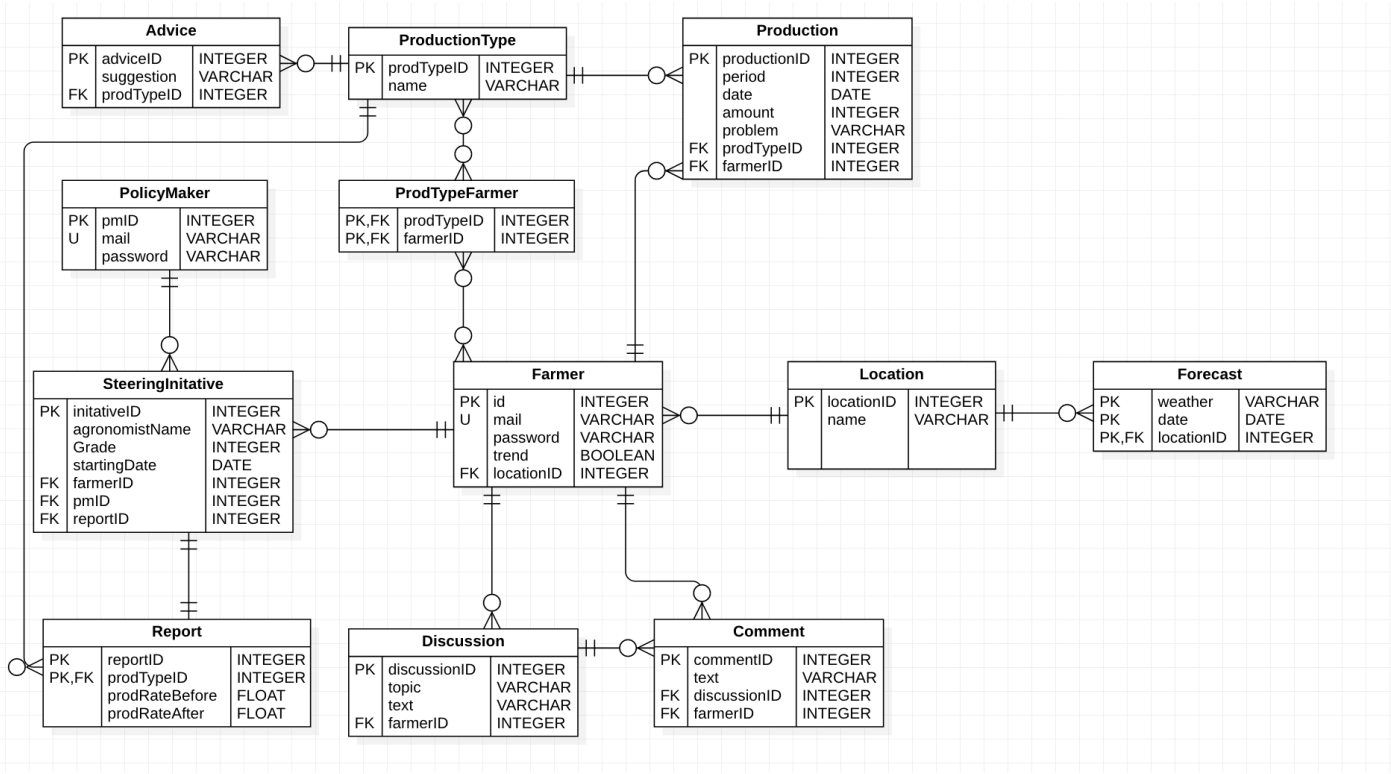

Figure 14: Database structure

2.7.2 Report creation

To create a report the system runs a script 3 months after the *startingDate* of the Steering Initiative and generate a *prodRateBefore* (As sum(amount) / sum(period)) for each type of production completed 2 months before the SteeringInitative and a *prodRateAfter* for each type of production completed 3 months after the SteeringInitative.

2.7.3 Maps API

To implement some of the system functions, which require GPS information, the use of an external maps service is necessary.

In order to use a trusted source, DREAM will use the API offered by Google Maps. Since the system will be developed as a web application, the *Google Maps JavaScript API* is recommended.

3 User interface design

3.1 UI mock-ups

The WebApp is the interface that allows the customers to enjoy DREAM's services.
The application can be accessed through different devices (smartphone, pc, tablet...) and it's the only way for a customer to use DREAM. The application interface mock-ups of the most important pages are shown below.

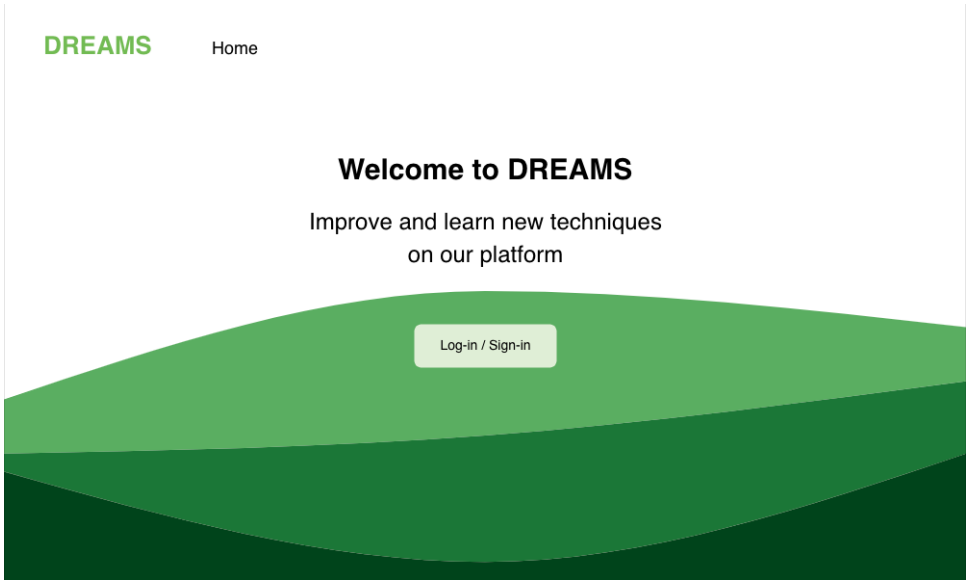

Figure 15: Home Page

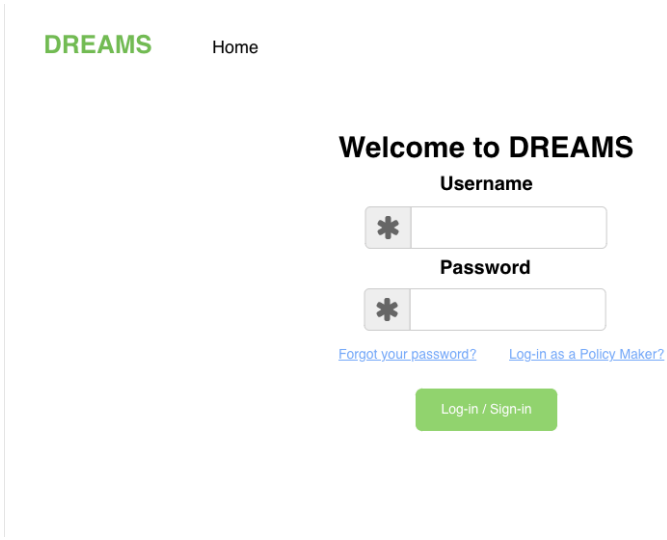

Figure 16: Login Page

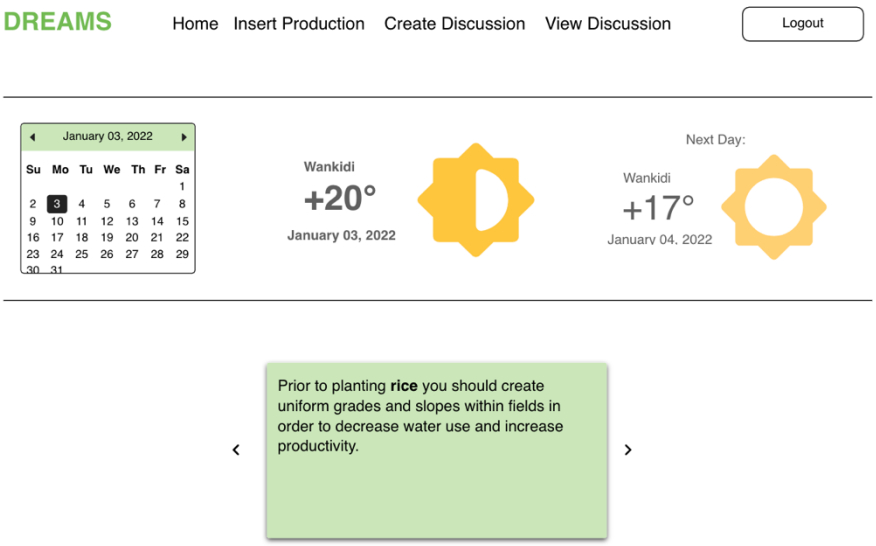

Figure 17: Farmer Personalized Data Page

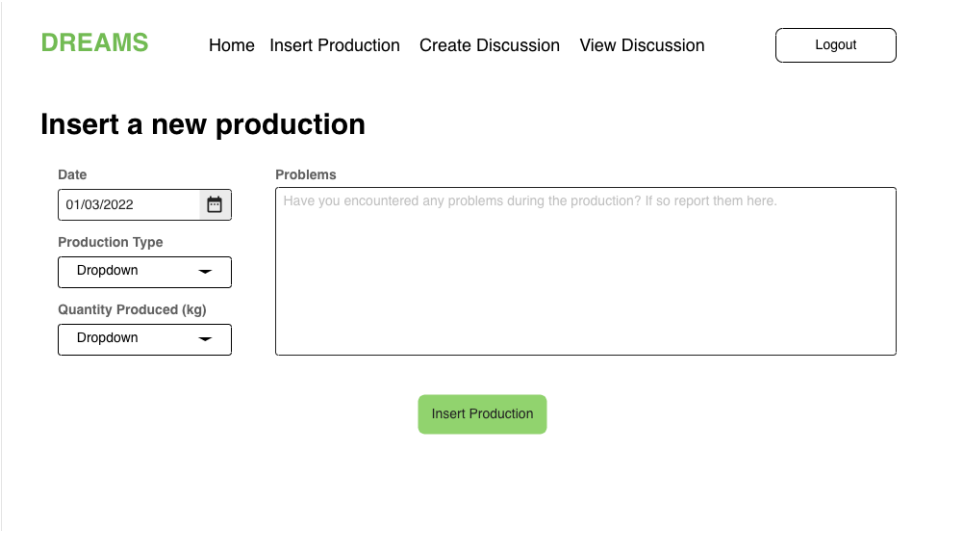

Figure 18: Inserting a Production

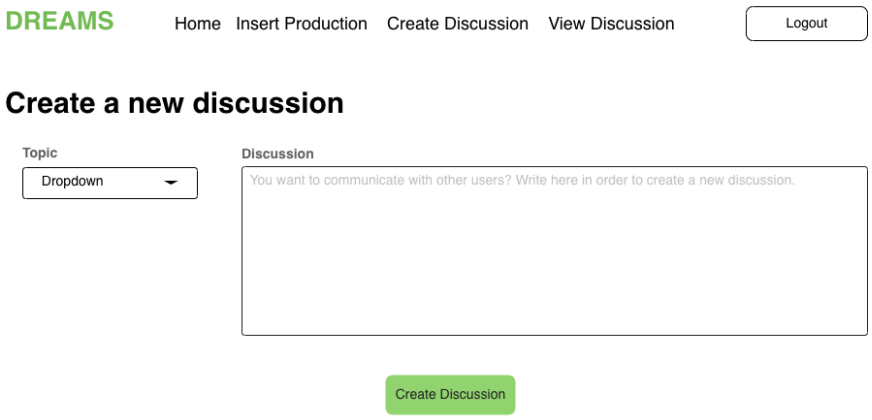

Figure 19: Creating a Discussion

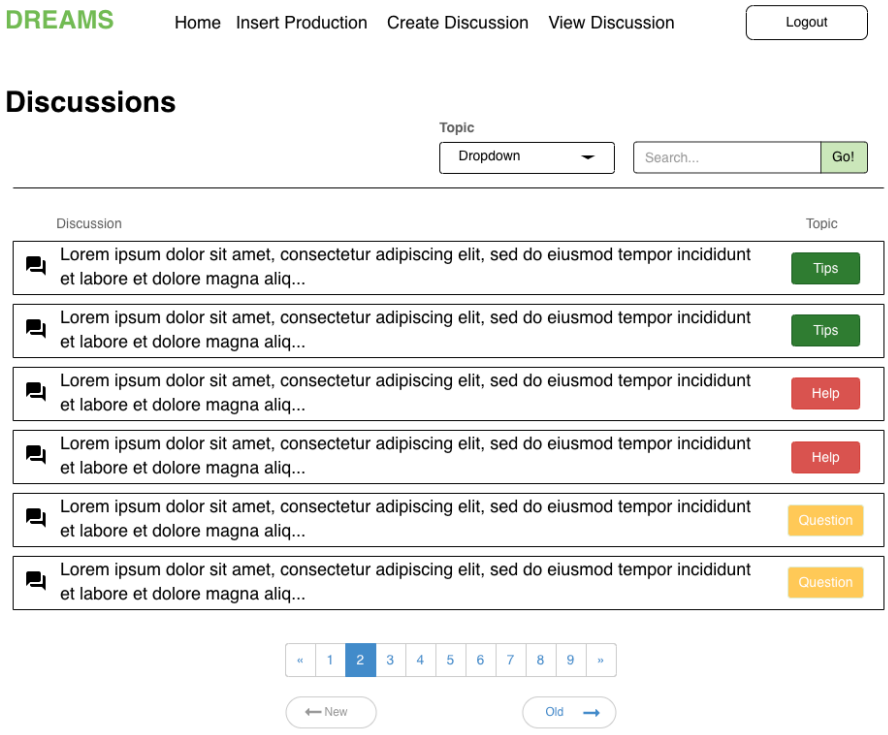

Figure 20: Viewing a Dicussion (1/2)

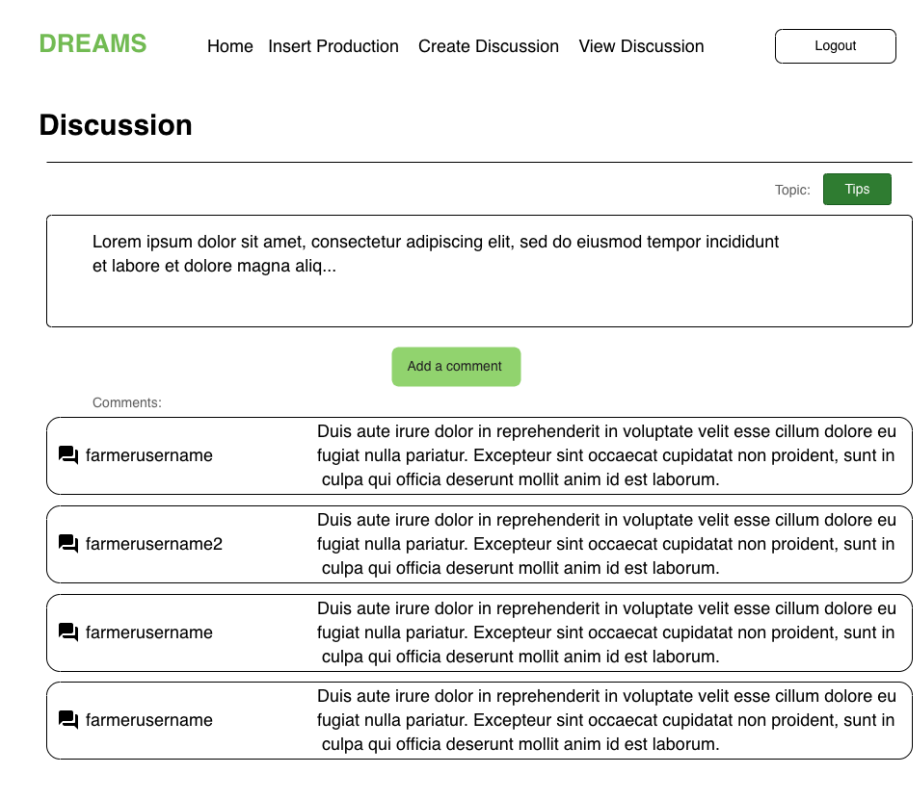

Figure 21: Viewing a Discussion (2/2)

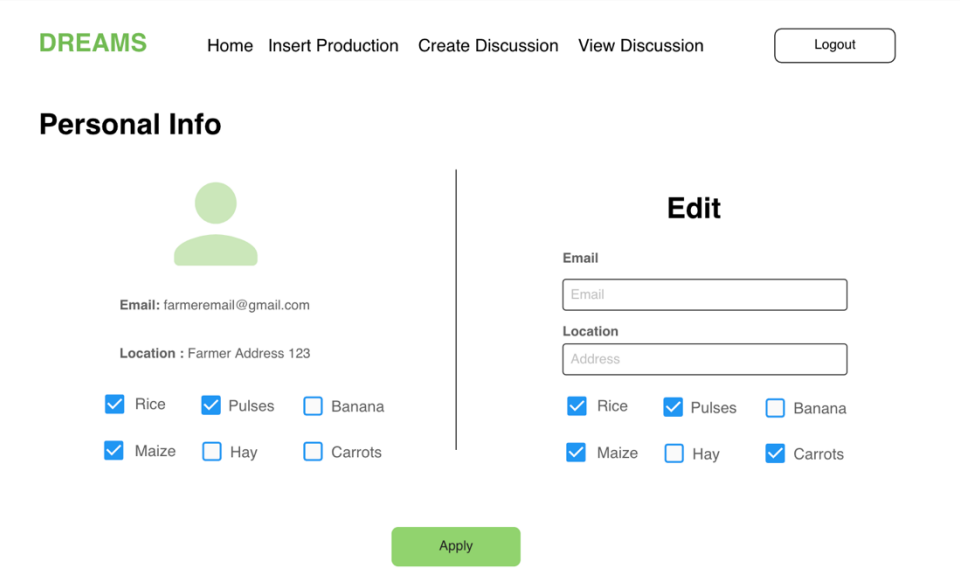

Figure 22: Changing Personal Info

The employees access some interfaces different than the one available to a standard user.
The most important pages of the application are shown below.

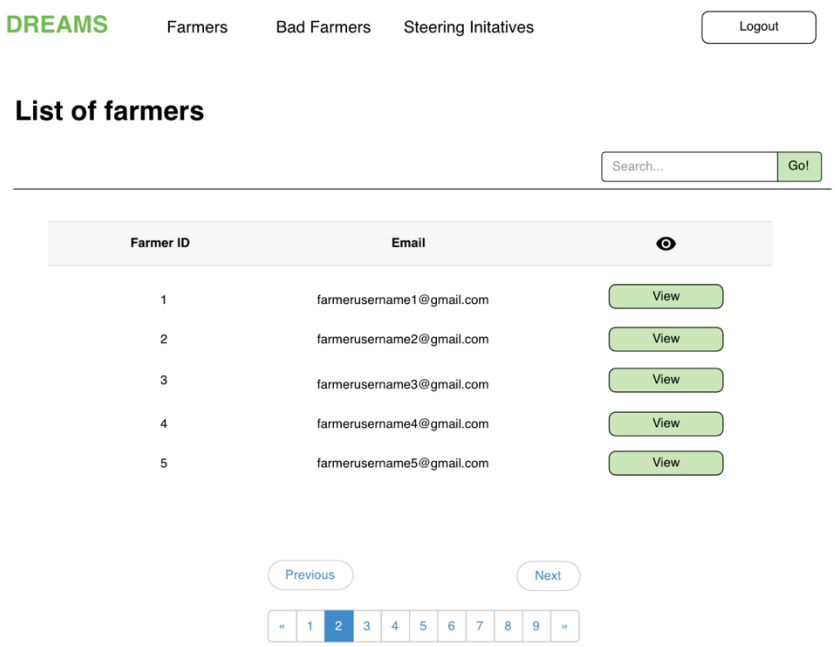

Figure 23: List of Farmers

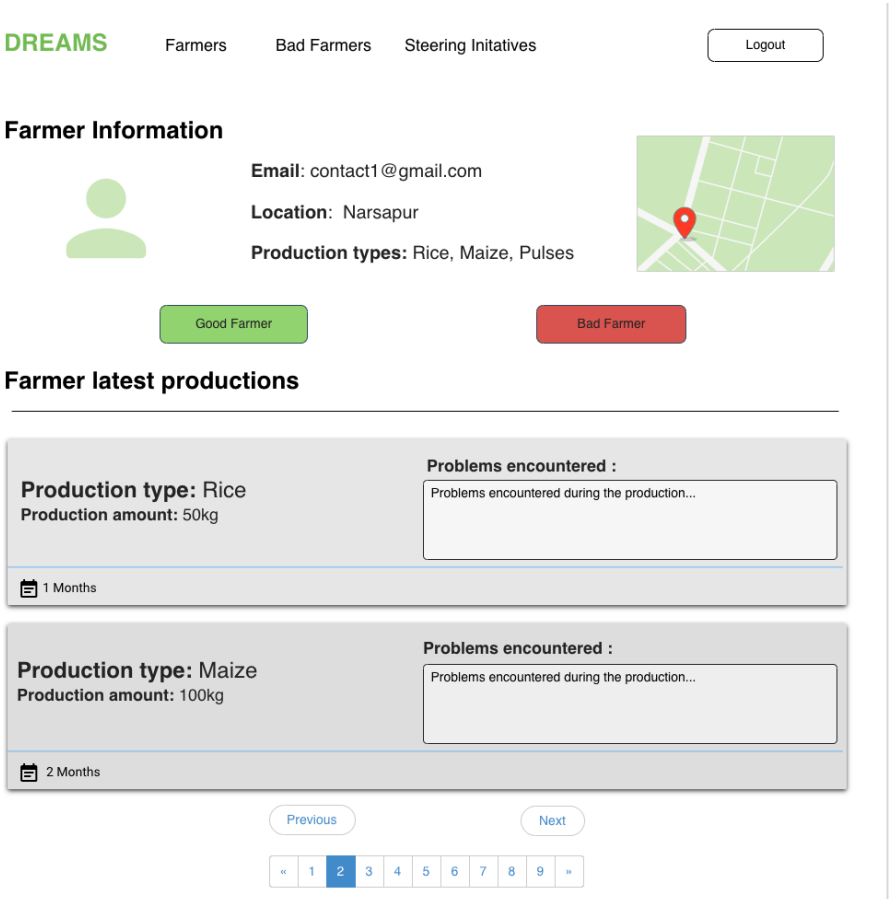

Figure 24: Farmer Evaluation Page

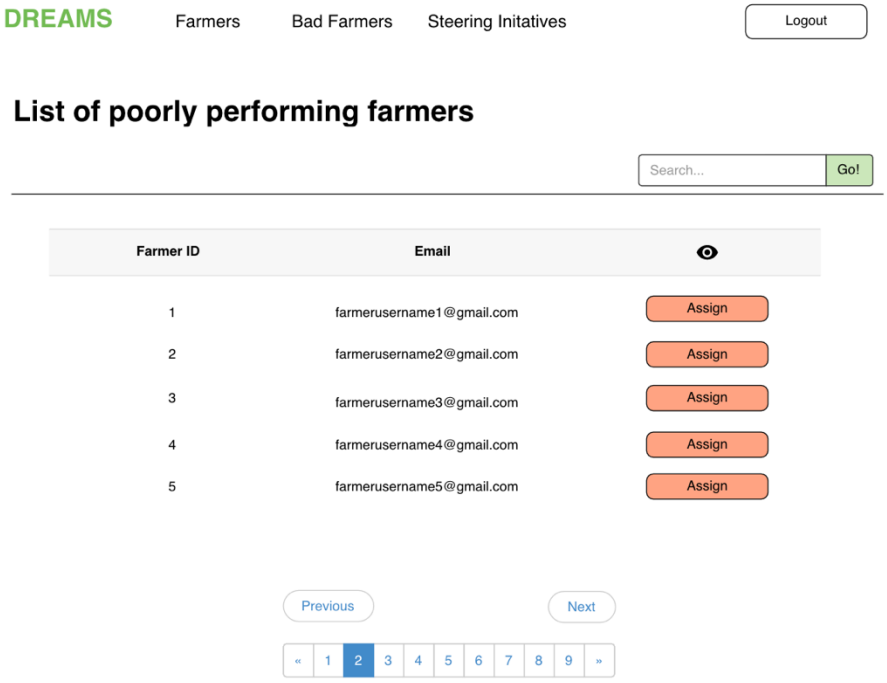

Figure 25: List of Bad Farmers

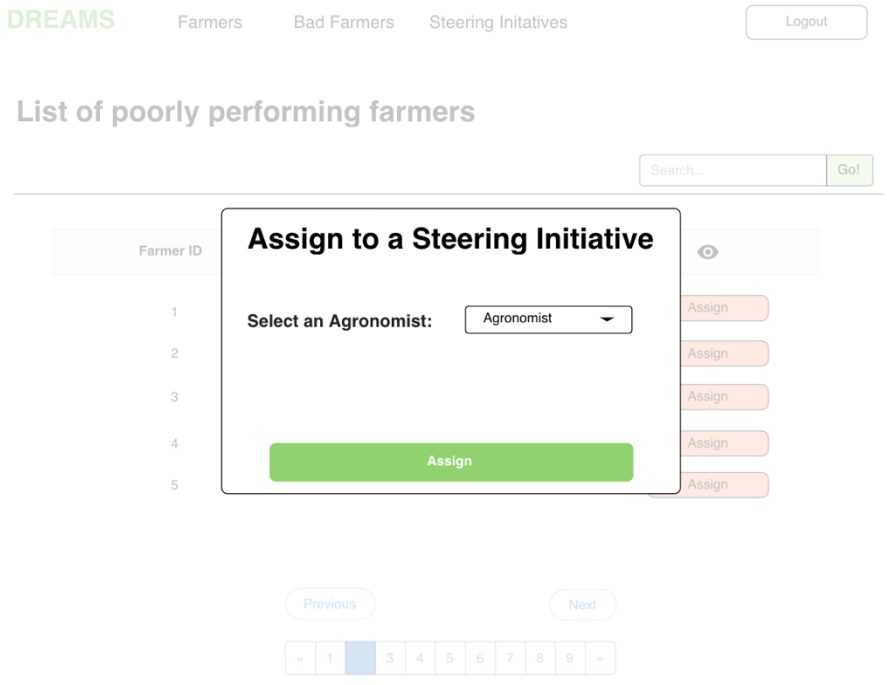

Figure 26: Assign Farmer form

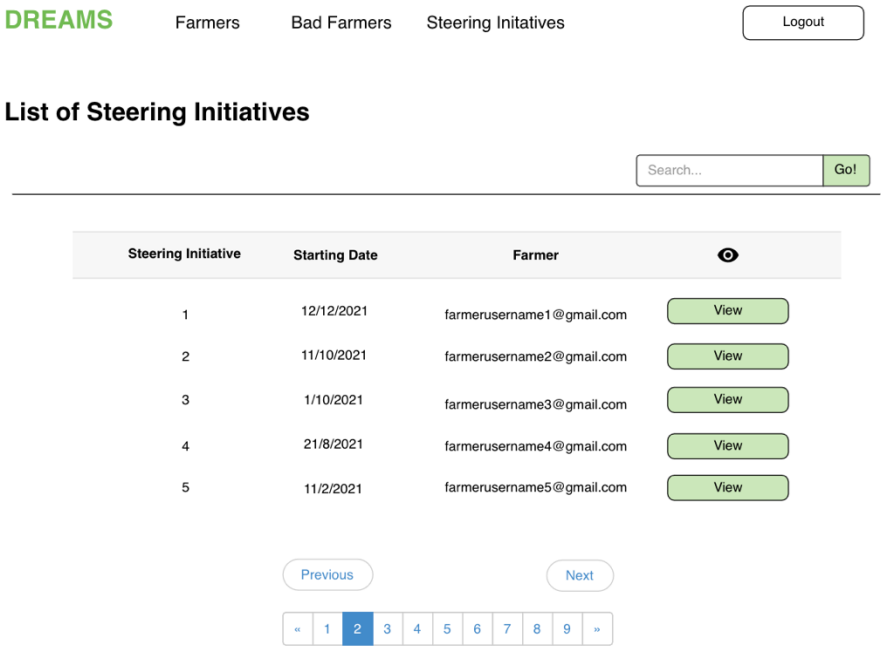

Figure 27: List of Steering Initiatives

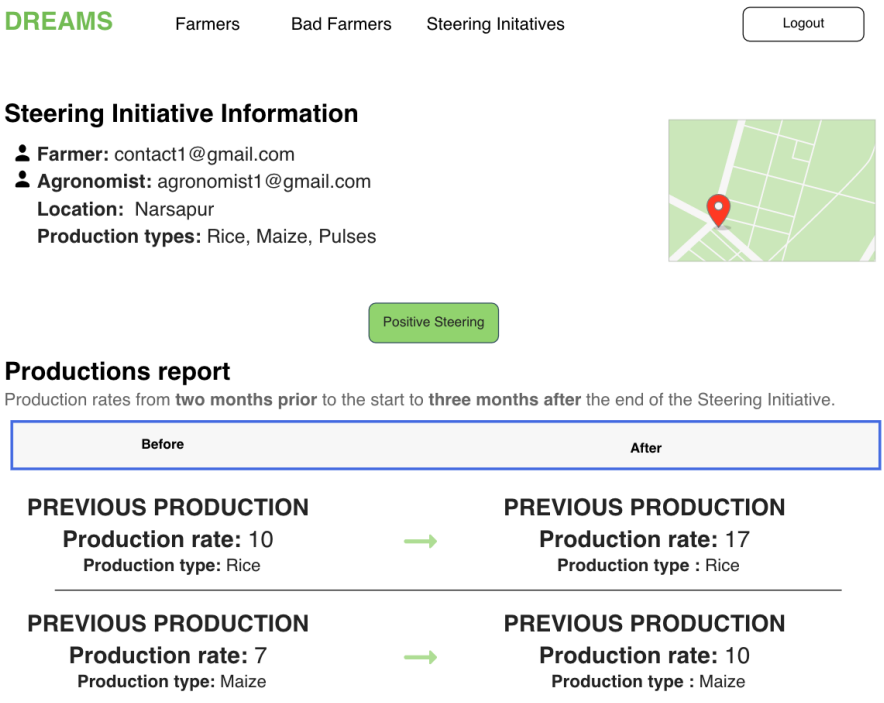

Figure 28: Steering Initiative Evaluation Page

3.2 User Interface Flow Diagrams

In this section are shown the flow diagrams of the most important User Interfaces that were presented in the section 3.1 above.

3.2.1 Farmer

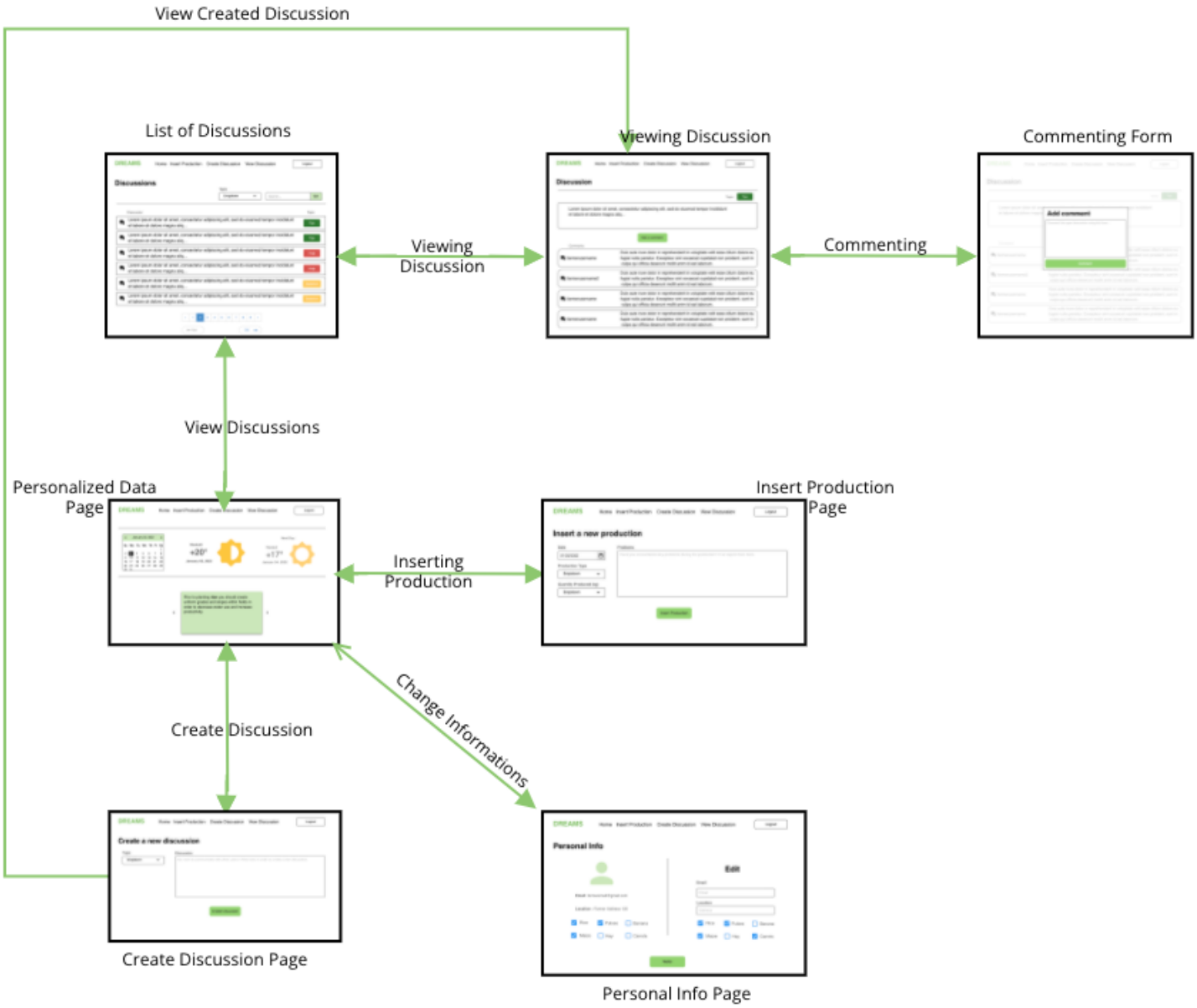

Figure 29: UI Flow Diagram for the Farmer

3.2.2 Policy Maker

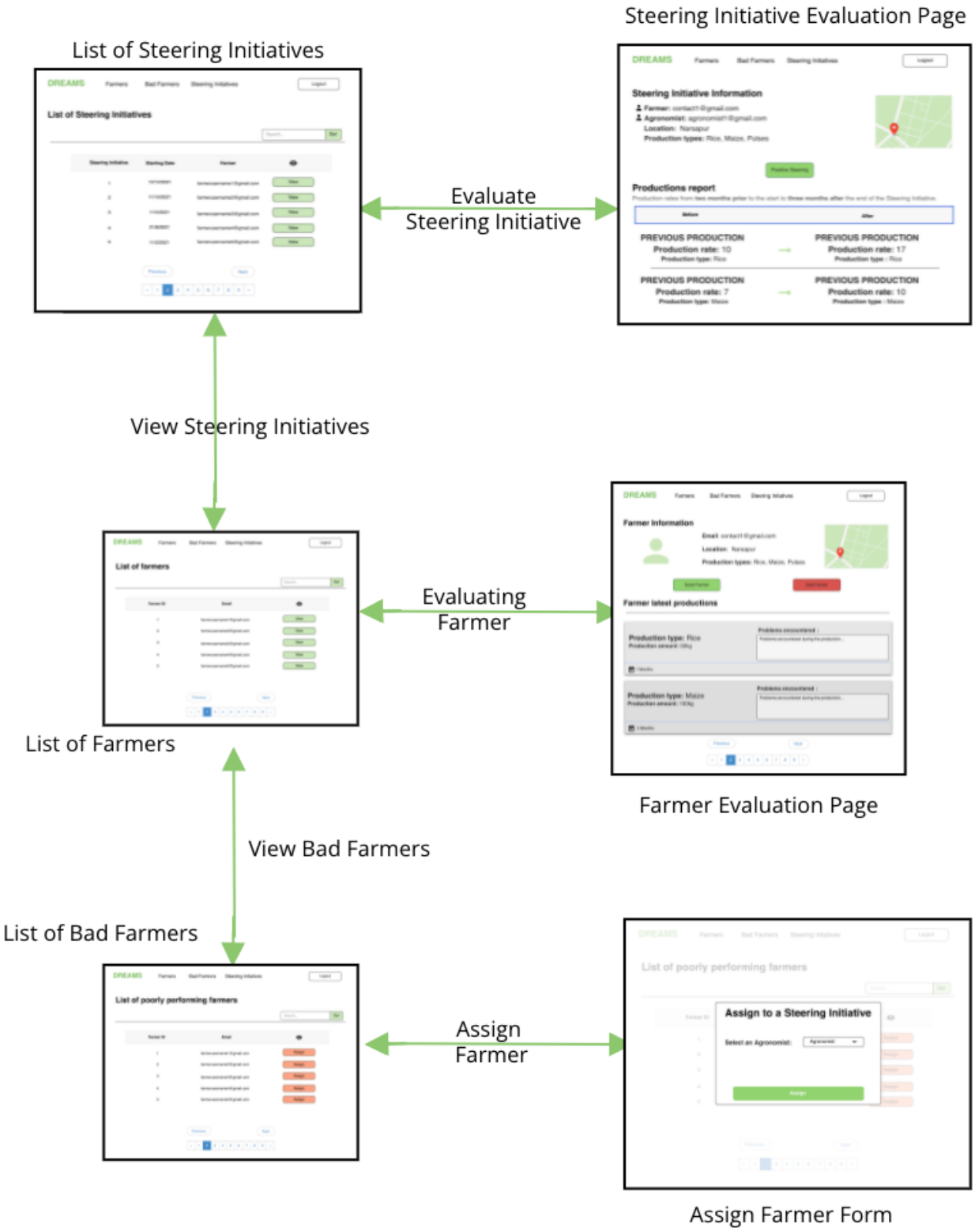

Figure 30: UI Flow Diagram for the Policy Maker

4 Requirements traceability

While making the design choices presented in this document, the main object was to fulfill the previously identified requirements of the system.

Here is provided a map between the requirements defined in the RASD and the system components presented in the DD.

- **R1:** The system should allow the user to log-in.
  - Account Services (*Account Module)*
- **R2:** The system should allow the farmer to view his personalized data.
  - Farmer Services (*Information Module*)
- **R3:** The system shall allow the farmer to edit his personal information.
  - Farmer Services (*Information Module)*
- **R4:** The system shall allow the farmer to edit his location.
  - Farmer Services (*Information Module*)
- **R5:** The system shall allow the farmer to edit his type of production.
  - Farmer Services (*Information Module*)
- **R6:** The system shall allow the farmer to insert data about his production and any problems he had.
  - Farmer Services (*Production Module)*
- **R7:** During the insertion of the production data the system shall allow the user to select the production type.
  - Farmer Services (*Production Module)*
- **R8:** The system shall allow the farmer to request for help and suggestions to other farmers.
  - Farmer Services (*Discussion Module)*
- **R9:** The system shall allow the farmer to create discussion forums.
  - Farmer Services (*Discussion Module)*
- **R10:** The system shall allow the farmer to view discussion opened by other farmers.
  - Farmer Services (*Discussion Module)*
- **R11:** The system shall allow the user to comment the opened discussion.
  - Farmer Services (*Discussion Module)*
- **R12:** The system should be able to create a report on the data before and the after the Steering Initiative.
  - Policy Maker Services (*Evaluate Module)*
- **R13:** The system should display to the Policy Maker all the farmers.
  - Policy Maker Services (*Evaluate Module)*

- **R14:** The system should display to the Policy Maker the information about a specific farmer.
  - Policy Maker Services (*Evaluate Module)*
- **R15:** The system should let the Policy Maker decide whether the Farmer is performing well or not.
  - Policy Maker Services (*Evaluate Module)*
- **R16:** The system shall allow the Policy Maker to see the farmers that are performing particularly badly.
  - Policy Maker Services (*Steering Module)*
- **R17:** The system shall display the Policy Maker all the Steering Initiatives.
  - Policy Maker Services (*Evaluate Module)*
- **R18:** The system shall display the data concerning a specific Steering Initiative.
  - Policy Maker Services (*Evaluate Module)*
- **R19:** The system shall allow the Farmer to delete a discussion.
  - Farmer Services (*Discussion Module)*
- **R20:** The system shall allow the Farmer to edit a discussion.
  - Farmer Services (*Discussion Module)*
- **R21:** The system shall allow the Farmer to delete a comment.
  - Farmer Services (*Discussion Module)*
- **R22:** The system shall allow the Policy maker to assign a badly performing farmer to a Steering Initiative.
  - Farmer Services (*Steering Module)*

5 Implementation, integration and test plan

This section defines the implementation order for the different components of DREAM that must be followed by the development team.

For the implementation, integration and testing of the system a *bottom-up* approach will be used, without omitting the dependencies between components within the same subsystem. This approach has the purpose of developing components that can be used in different situations and facilitates bug tracking by allowing incremental testing. External services do not need to be unit tested since it is assumed that they are reliable.

5.1 Implementation plan

What is described below is the implementation plan of the server-side, as it is the most complex part which presents the most critical parts of the system.

The first element that must be implemented is *Database Connection*, the internal component that manages the interaction between the system and the *DBMS*.

Then we can develop all the components that provide the services offered to users by the system. These elements are independent of each other, so they can be implemented in any order.

Starting from *Farmer Services*, its subcomponents are implemented following the order in which a farmer will interact with them, this defining an indirect information dependency:

1. Information Module
2. Production Module
3. Discussion Module

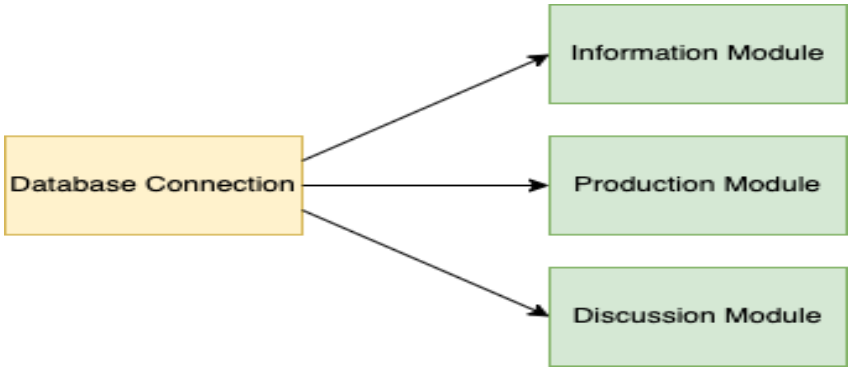

Figure 31: Implementation Farmer Services

Inside *Policy Maker Services*, its subcomponents are implemented following the order in which a farmer will interact with them, this defining an indirect information dependency:

1. Evaluate Module
2. Steering Module

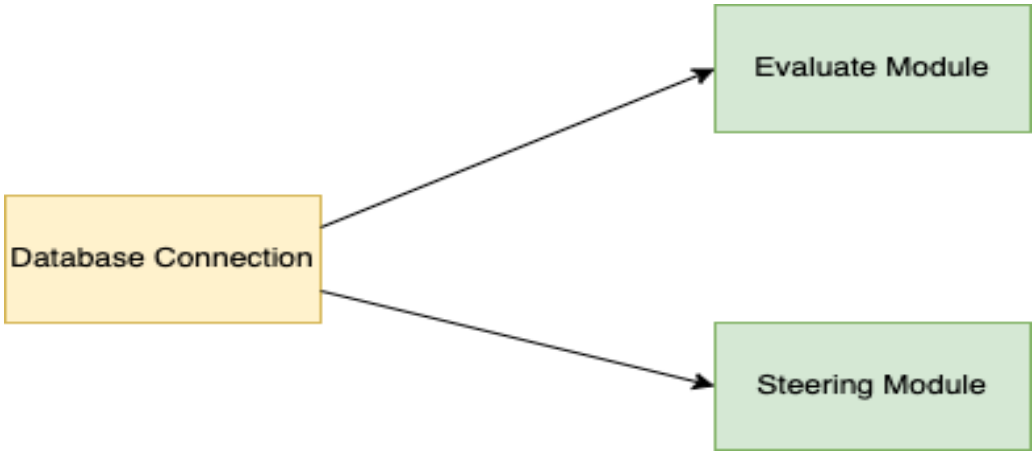

Figure 32: Implementation Policy Maker Services

Finally, the *Account Services* component, can be implemented as a single block as it provides standard functionalities used for authentication and authorization purposes.

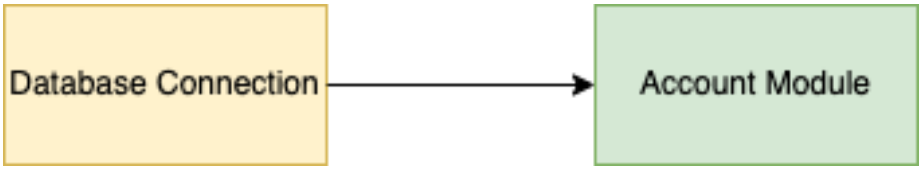

Figure 33: Implementation Account Services

The implementation of the user interfaces can be performed in parallel with the implementation of the previous components.

5.2 Integration and test plan

The implementation of the application must be accompanied by a meticulous testing phase.

In addition to unit test, a series of integration test must be performed during the development of a component with the part of system already implemented.

For this reason, the order of integration is strictly dependent by the order of implementation.

The style adopted for this system allows the development of the view components in parallel, so we can perform tests of each functionality without having to complete the whole system.

After the complete integration, the application must be tested using *system testing* to verify that the requirements are satisfied.

6 Effort spent

6.1 Ferrara Alessio

Topic	Hours
General Reasoning	10 h
Purpose & Scope	2 h
Section 2	5 h
UI mock-ups	4 h
Section 4	5 h
Section 5	3 h

6.2 Mazzola Francesco

Topic	Hours
General Reasoning	10 h
Purpose & Scope	2 h
Sequence diagrams	4 h
UX diagrams	4 h
Ui mock-ups	6 h
Section 5	3 h