AY 2021/2022

Politecnico di Milano

DD: Design Document

Ottavia Belotti Alessio Braccini Riccardo Izzo

Professor
Elisabetta Di Nitto

Version 1.0
December 12, 2021

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to provide a full technical description of the system described in the RASD document. In this design document we discuss about both hardware and software architectures in terms of interaction among the components that represent the system. Moreover, there are mentions about the implementation, testing and integration process. This document will include technical language so it is primarily addressed to programmers, but stakeholders are also invited to read it in order to understand the characteristics of the project.

1.2 Scope

The scope of this design document lays in the definition of the system behavior, in both general and critical cases, and in the design of the system architecture by describing the logical allocation of the components and the interaction between them. This document also extends in part to the implementation and testing plan, where one possible course of action is explained, user interface design of user applications and requirements traceability relating to the RASD.

1.3 Definitions, acronyms, abbreviations

Acronyms

- **DREAM**: *Data-driven predictive farming*
- **RASD**: Requirement Analysis and Specification Document
- **DD**: Design Document
- **API**: Application Programming Interface
- **DBMS**: Database Management System
- **UML**: Unified Modeling Language
- **GPS**: Global Positioning System

- **IT**: Information Technology
- **GUI**: Graphic User Interface
- **UI**: User Interface

1.4 Revision history

1.5 Reference documents

- Specification document: "Assignment RDD AY 2021-2022"
- Requirements Analysis Specification Document (RASD)
- UML documentation: https://www.uml-diagrams.org/
- ArchiMate documentation: https://pubs.opengroup.org/architecture/archimate3-doc/
- Slides of the lectures

1.6 Document structure

- **Section 1** gives a brief description of the design document, it describes the purpose and the scope of it including all the definitions, acronyms and abbreviations used.
- **Section 2** delves deeply into the system architecture by providing a detailed description of the components, the interfaces and all the technical choices made for the development of the application. It also includes detailed sequence, component and ArchiMate diagrams that describe the system in depth .
- **Section 3** contains a complete description of the user interface (UI), it includes all the client-side mockups with some graphs useful to understand the correct execution flow.
- **Section 4** links the RASD to the DD, it maps the goals and the requirements described in the RASD to the actual functionalities presented in this DD.

- **Section 5** presents a description of the implementation, testing and integration phases of the system components that are going to be carried out during the technical development of the application.

2 Architectural Design

2.1 Overview

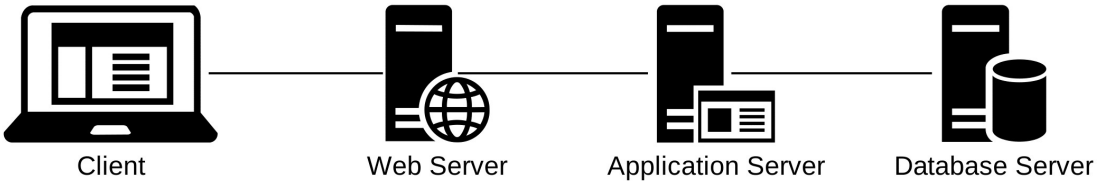

Figure 1: Four-Tier Architecture Scheme

The system is a distributed application that follows the common client-server paradigm. The architecture of the application is structured in three logic layers:

- **Presentation Layer (P)**: it manages the presentation logic and handles the user actions. It is characterized by a GUI (Graphic User Interface) that allows the user to interact with the application in a simple and effective way.
- **Logic or Application Layer (A)**: it manages all the functionalities that has to be provided to the users, it is also responsible of data exchange between the client and the data sources.
- **Data Layer (D)**: it manages the access to data sources, it gets data from the database and moves it through the other layers. It is essential to guarantee a high level of abstraction from the database in order to provide an easy to use model.

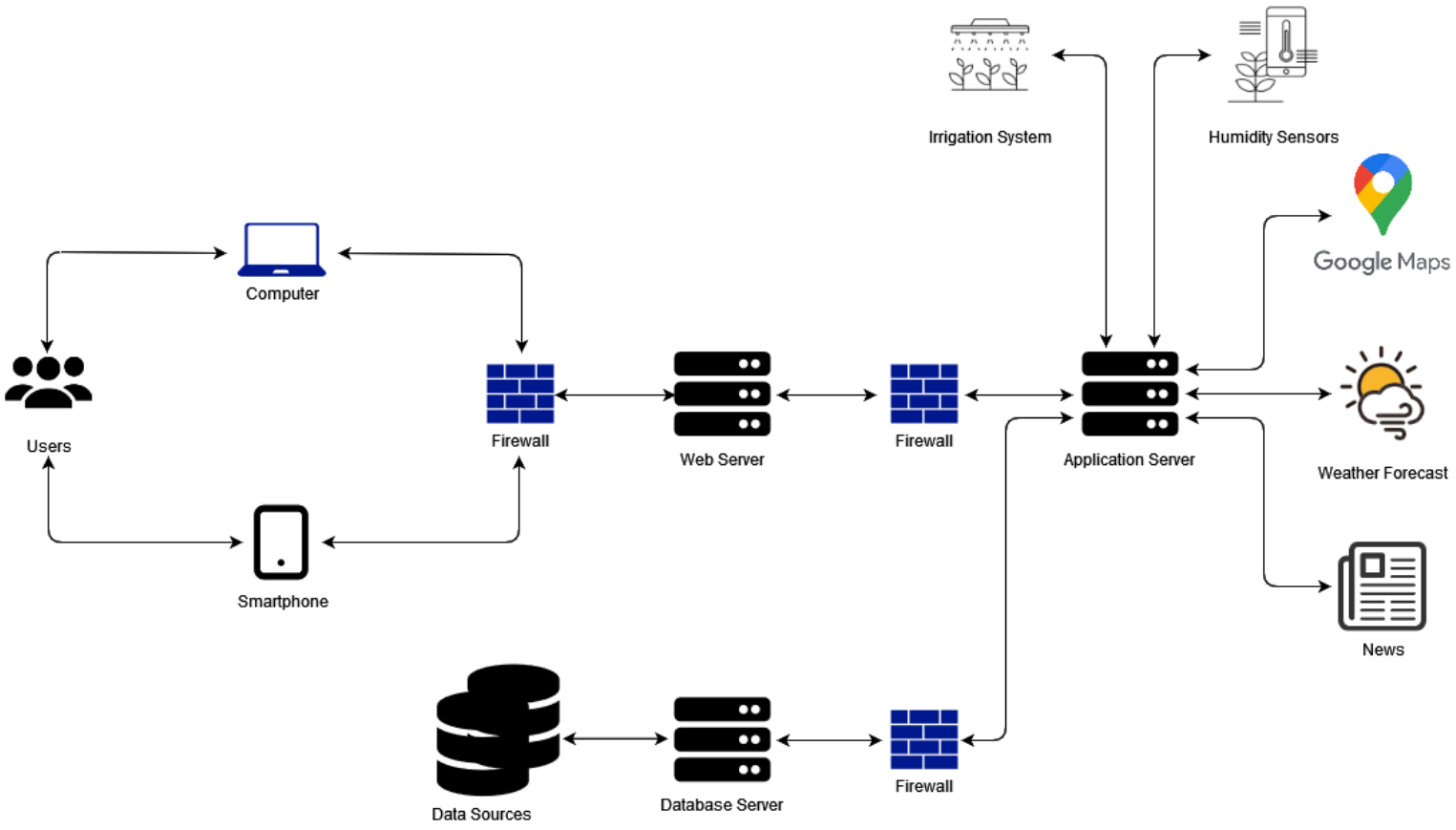

Figure 2: High Level Architecture

The system, as shown in *Figure (1)*, is based on a four-tier architecture (Client, Web Server, Application Server and Database Server), this ensures more flexibility and high scalability. The tiers are separated by firewalls in order to guarantee a higher level of security to the whole system. A thin client is used to prevent heavy computation load client side, in this way all the heavy operations are executed at server side. The client's devices can be a personal computer, a mobile device or any kind of IT device able to connect to the Web Server through a web browser. In fact, the four-tiers architecture has been specifically chosen to allow the widest variety of commercial devices

to access the application, without the need of developing an OS-specific one for each of them. The Web Server manages the HTTP requests by the users and forwards them to the Application Server. The Application Server communicates with the Database Server and transform data following its business logic. It also manages all the external APIs including: Google Maps API, the news, the weather forecast, the water irrigation system and the humidity sensors. This allows to retrieve data like weather information directly from it in order to show them to the user. To enable the geolocalization, the Application Server uses the API provided by Google Maps. The Database Server manages all the internal data sources and communicates directly with the Application Server. All the components will be described in depth in the following sections.

2.2 Component view

General Component View

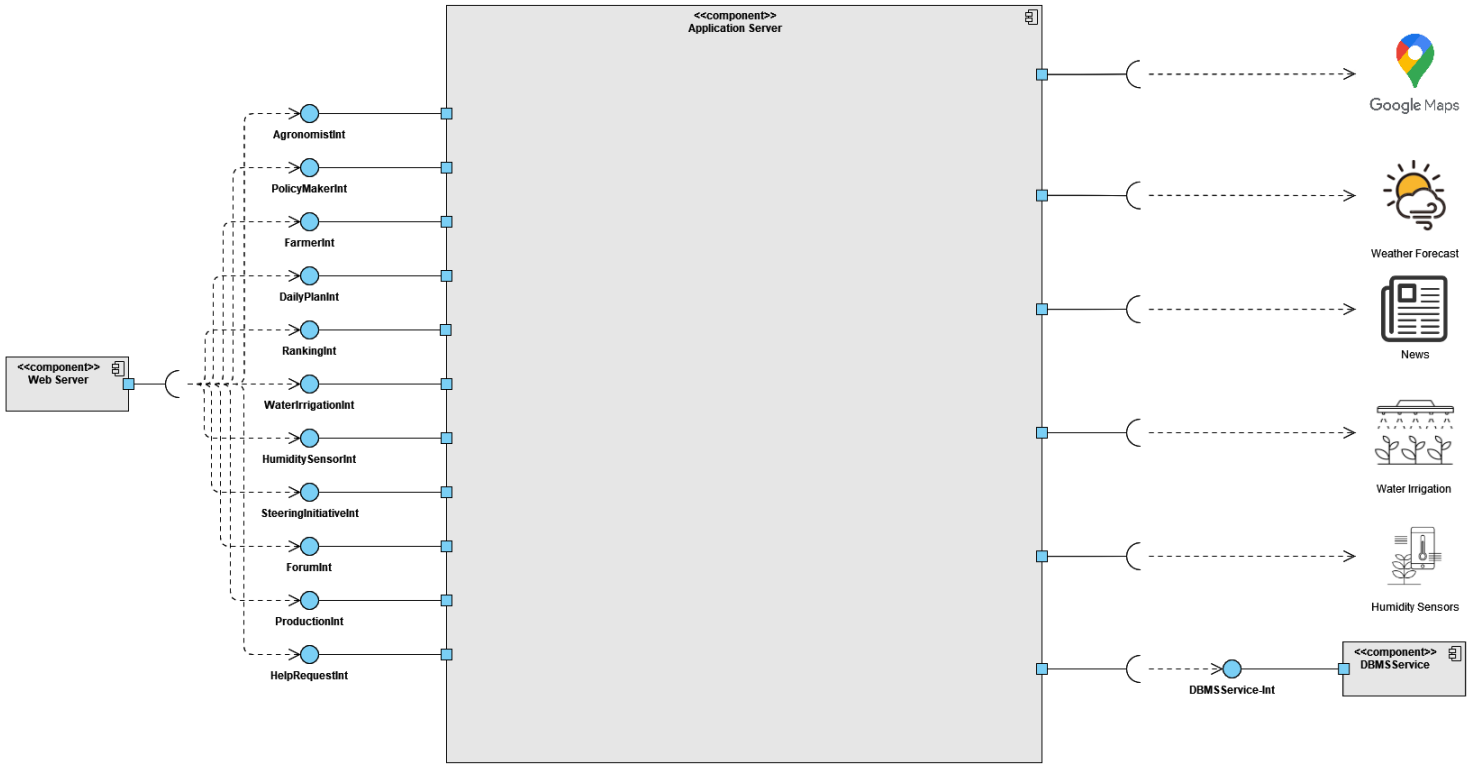

Figure 3: General Component Diagram

This image gives a high level representation of the components of the system. On the left are shown the provided interfaces between the Web Server and the Application Server that in this scheme is represented as a "black box"; a complete description of it is provided in the next section. All the interfaces basically represent the main functionalities requested by the client application. On the right there are the requested interfaces. Among these, one is responsible of the geolocalization and is provided by the Google

Maps API, the others manage the remaining external services such as the weather forecast, the news, the water irrigation system and the humidity sensors. Finally, the DBMS interface manages the DBMS service and it's responsible of the communication between the Application Server and the Database Server.

Application Server Component View

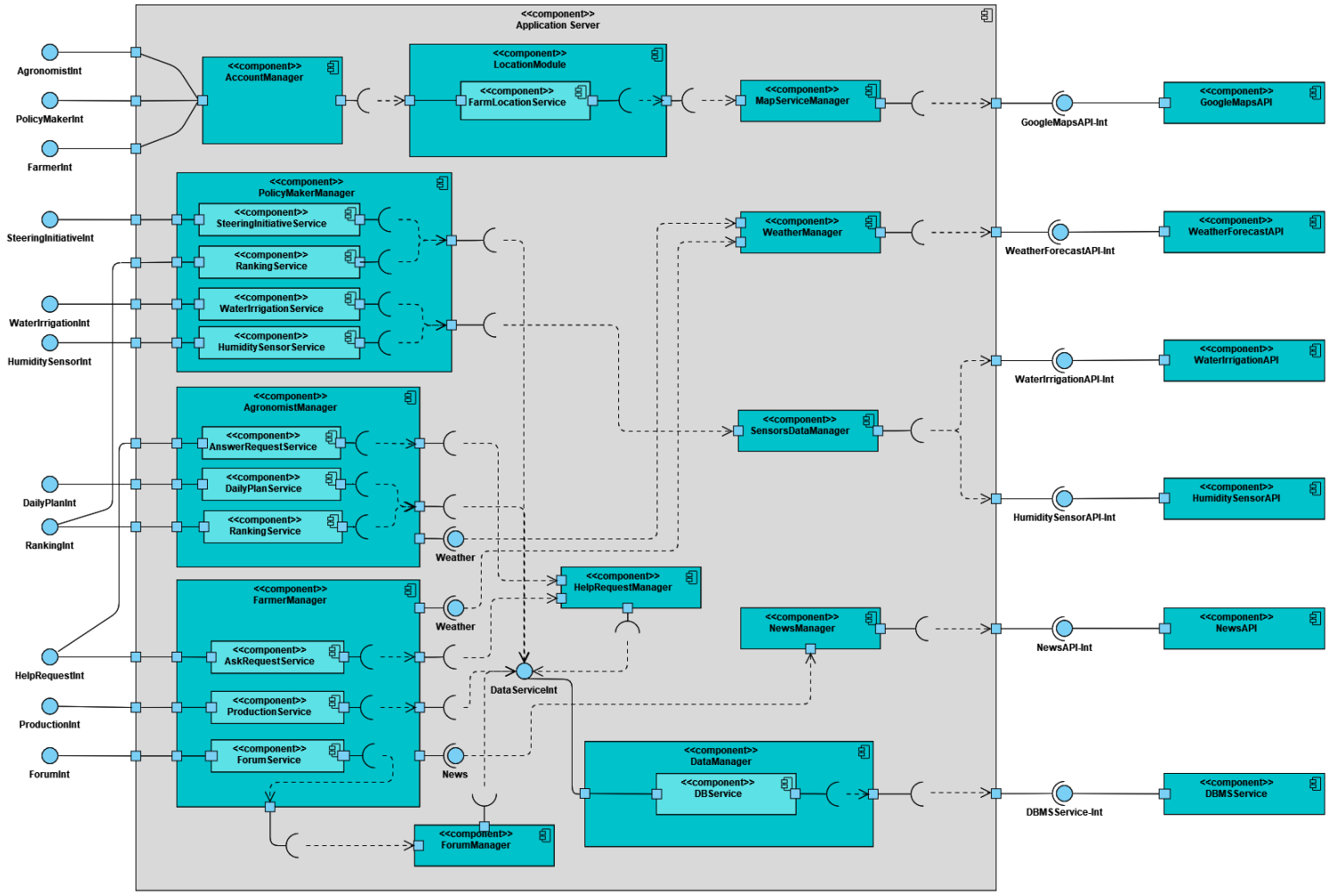

Figure 4: Application Server Component Diagram

The following component diagram gives a detailed view of the Application Server, it shows the internal structure and the interaction between the components. External elements in the diagram are represented in a simplified way.

- **AccountManager**: this component handles all the basic requests made by the client. This includes the authentication process that is composed by the log in, the log out and the sign up services. It also includes the edit profile routine invoked by an user editing the personal account. Once a user is logged in, all the specific functionalities are provided by the component that manages this type of user. Finally it communicates with the *LocationModule* that provides the geolocalization service.

- **LocationModule**: this module provides the interface that allows the geolocalization of the farm based on the GPS and on the address provided by the farmer. In order to provide this service, it communicates with the *MapServiceManager*.

- **MapServiceManager**: this component communicates directly with the external API provided by Google Maps, it provides information regarding the region and allows the user to visualize the map of a specific location. Mainly it adapts the data received by the API in a comprehensible way for the other components, it also manages the API requests.

- **PolicyMakerManager**: it manages the policy maker services, these include: steering initiatives, ranking, water irrigation system and humidity sensors. The steering initiative service and the ranking service are mapped through an interface to the internal database, on the other side the water irrigation service and the humidity sensor service are considered external services and are managed by the *SensorsDataManager*.

- **AgronomistManager**: it manages the agronomist services, these include: answer help requests, daily plan and ranking. It includes the weather external interface, it communicates with the weather component in order to provide the weather forecast service to the agronomist.

The daily plan service and the ranking service are linked to the internal database, instead the answer request service communicates directly with the *HelpRequestManager* component.

- **FarmerManager**: it manages the farmer services, these include: ask a help request, production and forum. It includes two external interfaces: the first one communicates with the *WeatherManager* component in order to provide the weather forecast service to the farmer, the other one communicates with the *NewsManager* component. In this case, only the production service is linked directly to the internal database, instead the ask request service is linked to the *HelpRequestManager* component, while the forum service exposes an interface to connect to the *ForumManager* component.

- **ForumManager**: this component manages the forum section, in particular it is responsible of the management of all the topics with the related messages between farmers.

- **DataManager**: it provides access to the external interface of the database, it manages queries and interacts with the DBMS. It includes one component that expose an interface that allow other internal components to communicate with the database. This component is connected to the *DBMSService* external component, this establishes the connection between the Application Server and the Database Server.

- **HelpRequestManager**: it manages the help requests between farmers and agronomists. It communicates with the *FarmerManager* component for the help requests to ask, on the other way it communicates with the *AgronomistManager* component for the ones to answer. Finally it provides an interface that communicate with the internal database in order to store the requests.

- **NewsManager**: this component manages the service related to the news, it provides updated suggestions about crops and fertilizers for the farmers. To do this it interacts with the external component *NewsService*.

- **WeatherManager**: this component manages the service related to the weather forecast. It interacts with the external component *WeatherForecastService* in order to get the weather data.

- **SensorsDataManager**: this component handles two external APIs, the first one provided by the water irrigation system and the other one provided by the humidity sensors. Both services are used by the *PolicyMakerManager*.

2.3 Deployment view

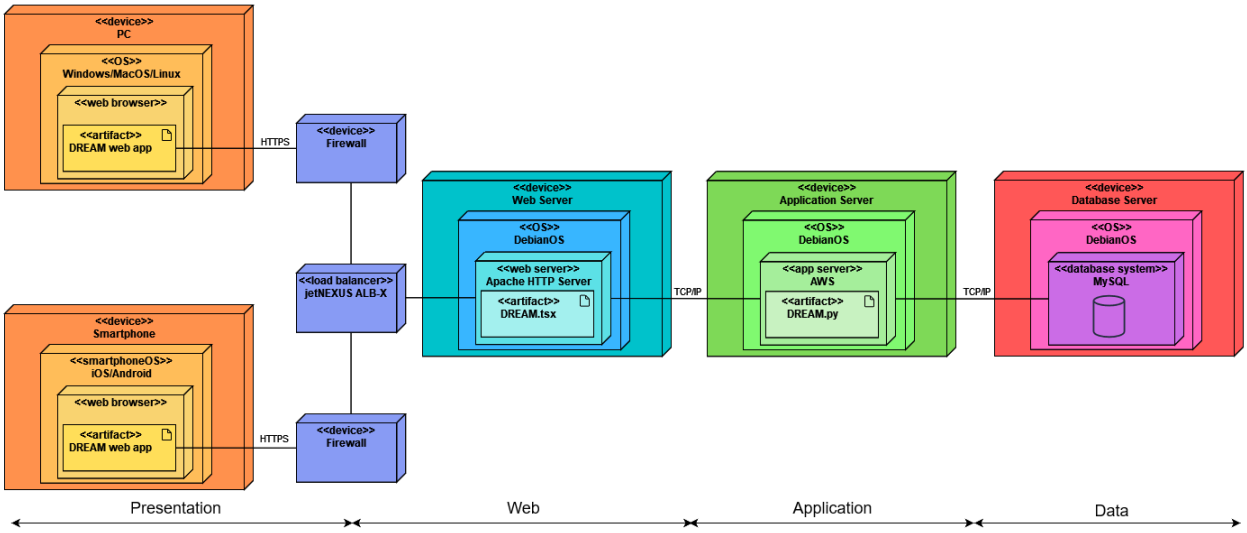

Figure 5: Deployment Diagram

The deployment diagram in *Figure (5)* shows the most important components necessary for the correct behaviour of the system. The devices shown in the diagram are:

- **PC/Smartphone**: they are the client machines, can be used by all the type of users. It allows the user to connect to the Web Server through a web browser as long as the HTTPS protocol is used.
- **Firewall**: first level of security, it guarantees safety access to the internal network of the system.
- **Load balancer**: the main goal is to prevent the overload of a server, this device distributes the incoming requests equally across multiple servers. This increase availability and reliability of the application. The one selected is the *jetNEXUS ALB-X*.

- **Web Server**: it manages the HTTPS requests from the clients and route them to the *Application Server* for processing. On the other way it is responsible of the incoming requests from the application server, they are managed by serving an HTML file to the client in addition to CSS and JS sheets. The result page is built with a client-side scripting. Finally web servers are duplicated to ensure high scalability and to guarantee good performance in case of a single point of failure.

- **Application Server**: it runs the core functionalities of the system, here there is the application logic. It also manages all the external APIs such as the Google Maps API and the weather forecast API. Just like the *Web Server*, for the same reasons, it is duplicated. The *Application Server* and the *Database Server* are connected through an internal LAN, there are no firewalls between them in order to increase the average connection speed. Finally, it establishes the connection with the data tier through the DBMS interface.

- **Database Server**: it represents the data tier, it receives the requests from the *Application Server* and manages them. MySQL has been selected as the main database infrastructure, this allow to manage easily and in a secure way all the data sources.

2.4 Runtime view

Here are represented the runtime view of some relevant use cases of the system through some sequence diagrams. In the diagrams, the part regarding the user is omitted because it has been considered superfluous to the understanding of the interaction. In later diagrams some parts, like the login phase or the come back to home page, were omitted for the above motivations.

Sign Up

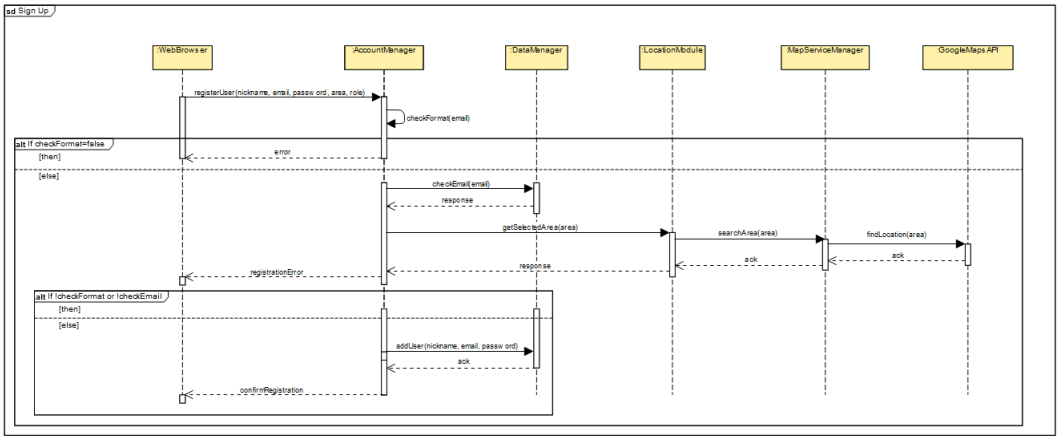

Figure 6: Sign Up

Sign up is pretty standard: user registers himself/herself by inserting first name, last name, email, password, job role and area. If the user is a farmer, he/she chooses also the crop type the farm deals with. The *LocationModule* and the *MapServiceManager* are used in this phase in order to ease the users in the selection of their location using the GPS. However users can also select manually their location without the use of GPS.

At the end of the process, if every check results succesfull, the user is registered into the database by the *DataManager* component.

Log In

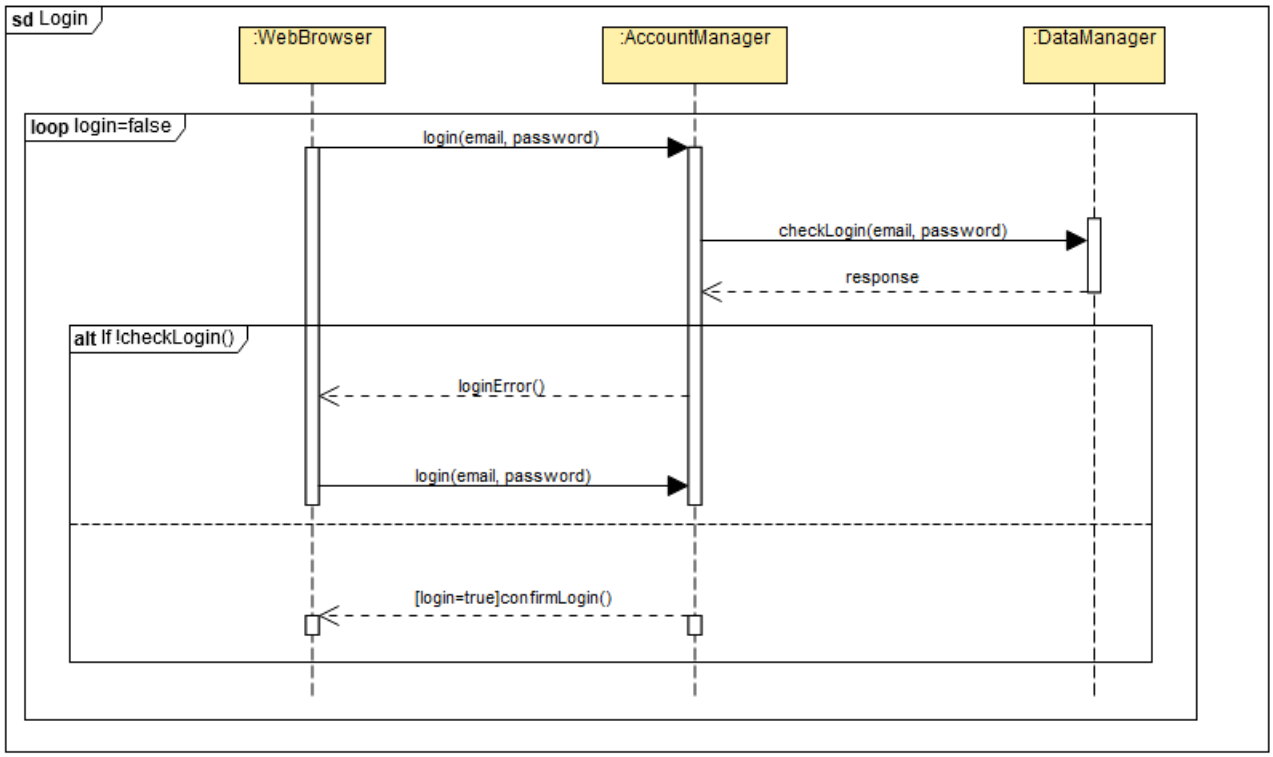

Figure 7: Log In

The Log In phase simply consists in the user action of inserting their email (primary and unique key of the database) and password in the login fields, then the system checks if the pair corresponds to an entry in the database. In case of success, the user can log in the system and use its functionalities available for that particular type of account chosen during the Sign Up phase.

View Steering Initiative Reports

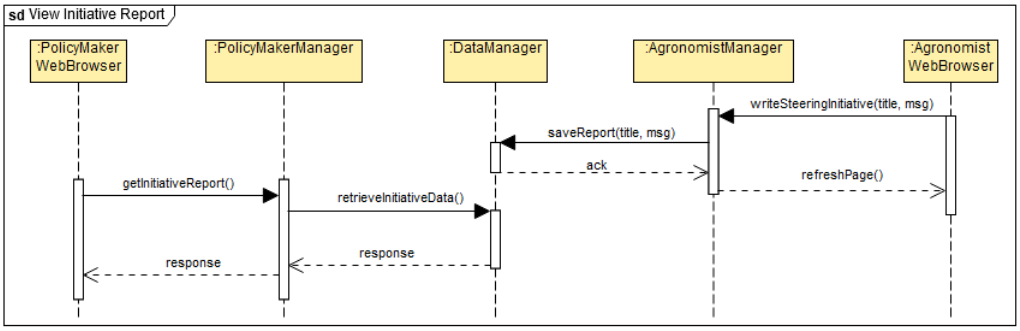

Figure 8: View Initiative Report

This sequence diagram represents the interactions between components that occur when displaying the steering initiatives (previously stored in the database) to a policy maker user.

Check Soil Humidity Data

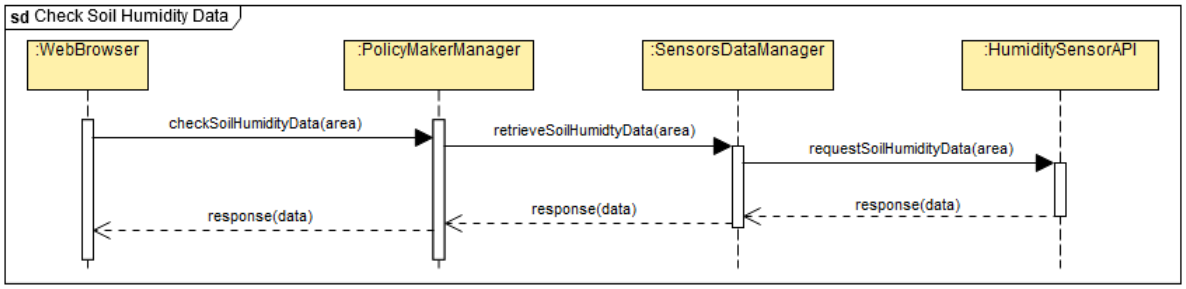

Figure 9: Check Soil Humidity Data

This sequence diagram represents the interactions between components that occur when displaying the sensors data regarding soil humidity to a policy maker user. The business logic running on the Application Server retrieves these data from an external source, furthermore it handles them just temporarely since they are not modified by our system and they're fetched on user demand. The user can also specify the area of the region by using the parameter 'area', so it's simpler to check a specific area instead of the whole region.

Check Water Irrigation Data

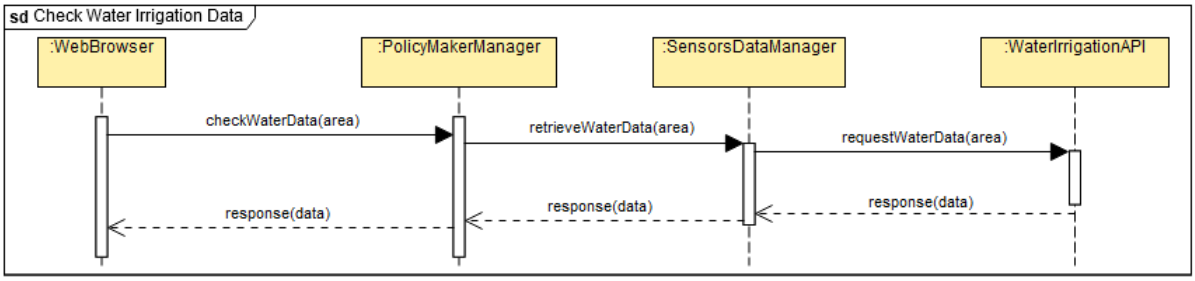

Figure 10: Check Water Irrigation Data

This sequence diagram represents the interactions between components that occur when displaying the sensors data regarding the water irrigation to a policy maker user. The function is the same of the above mentioned case.

View Farmers Ranking

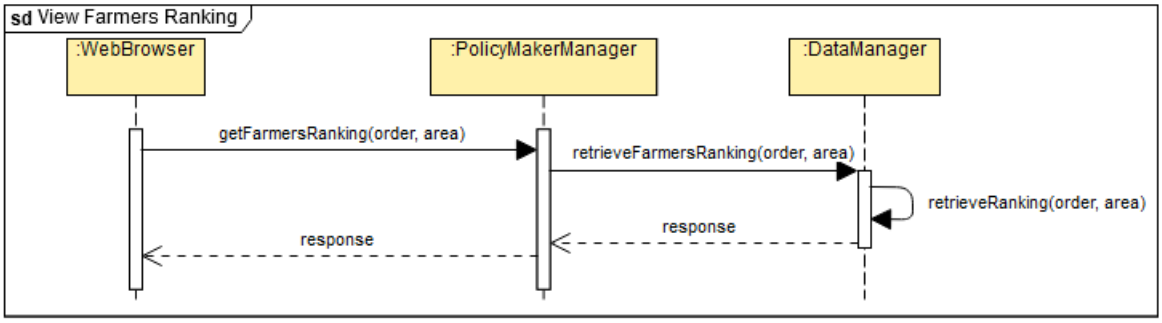

Figure 11: View Farmers Ranking

This sequence diagram represents the interactions between components that occur when displaying the farmers ranking to policy maker and agronomist users. It will be explained better later in the document. The function includes an area parameter to specify which area the user wants the ranking. In order to achieve code reusability, this function is invoked by either policy makers or agronomists. When a policy maker wants to see the ranking the parameter `area` will be set to `all` in order to obtain the ranking comprehensive of all Telangana's farmer registered in the app. Instead, when an agronomist wants to use this functionality, `area` will be automatically taken to his/her respective responsibility area. By doing this, the function becomes more modular.

View Specific Farmers Informations

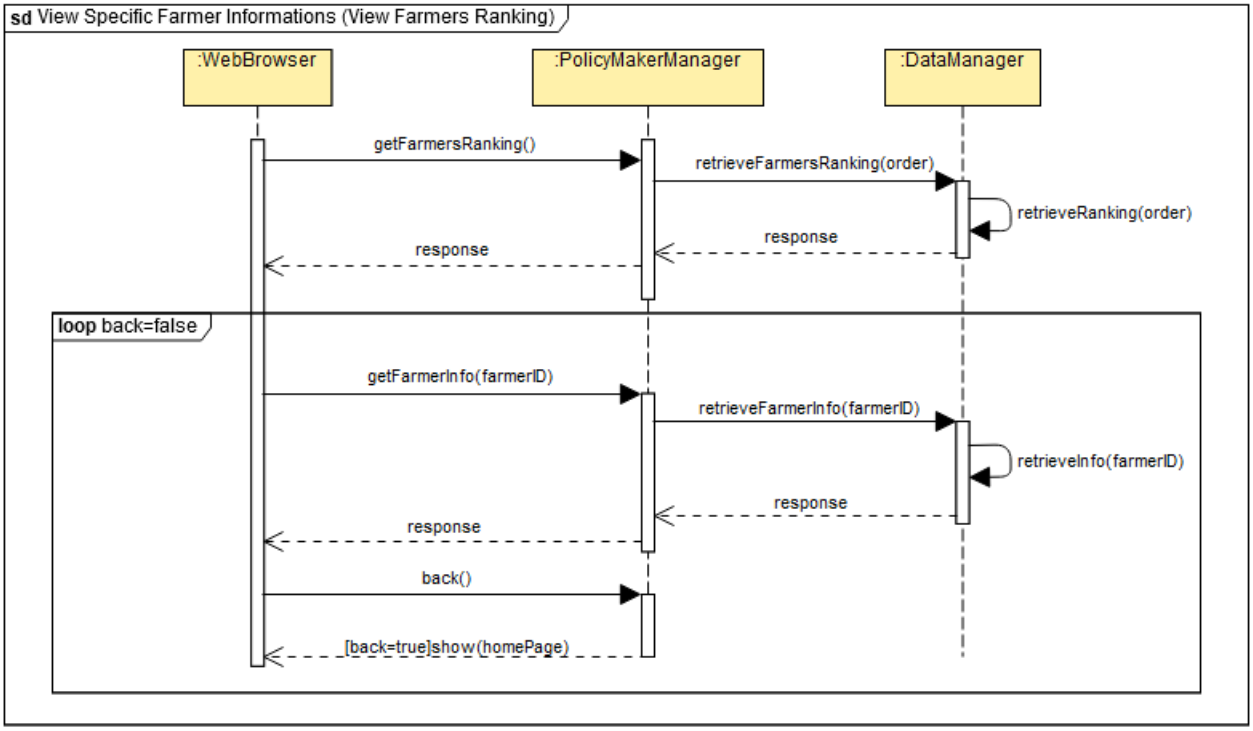

Figure 12: View Specific Farmer Informations

This sequence diagram represents the interactions between components caused by a search for a specific farmer information summary.

Profile Edit

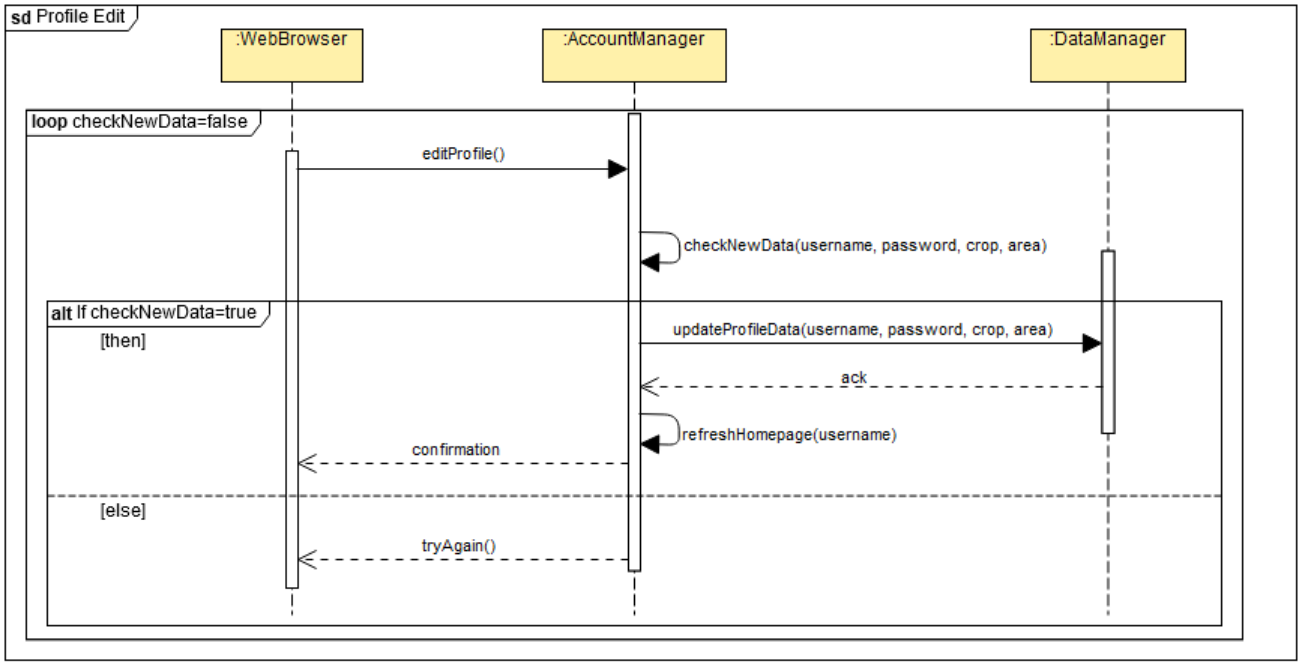

Figure 13: Profile Edit

This sequence diagram represents the interactions between components that occur when users make changes in the profile settings: edit username, password, area or crop type (if they're farmer users).

Insert Production Data

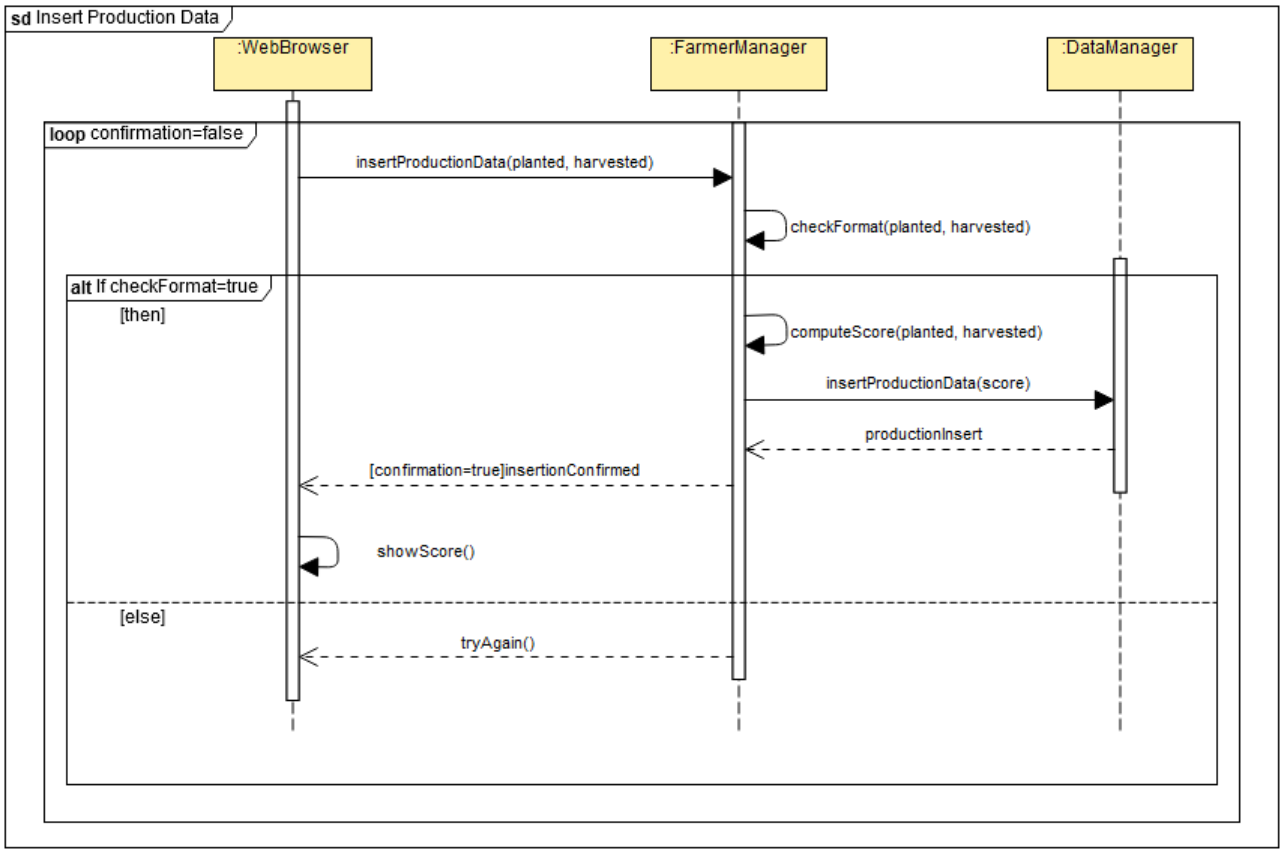

Figure 14: Insert Production Data

This sequence diagram represents the interactions between components caused by an insertion of production data from a farmer user. Each new production entry cause the score associated to the farmer to be calculated again and updated. So the score shall be an attribute of each farmer profile. Since a farmer's rank is directly proportionated to the his/her score, the recomputation can lead to a shifting in the general ranking too.

Check News

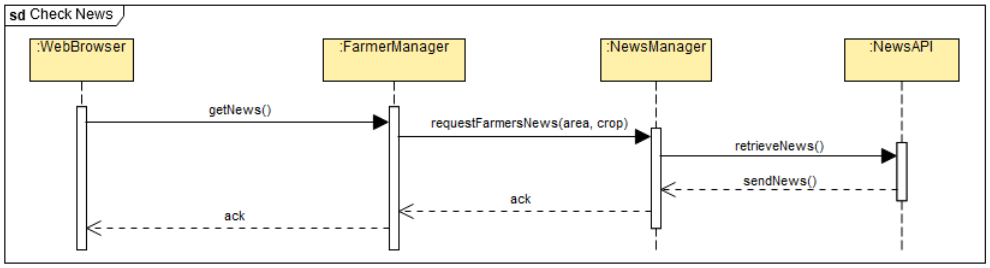

Figure 15: Check News

This sequence diagram represents the interactions between components that occur when displaying the news regarding farmers' interests to them, such as news concering the same area or crop type of the farmer. The *NewsManager* uses the *NewsAPI* component in order to get always up-to-date news.

Forum

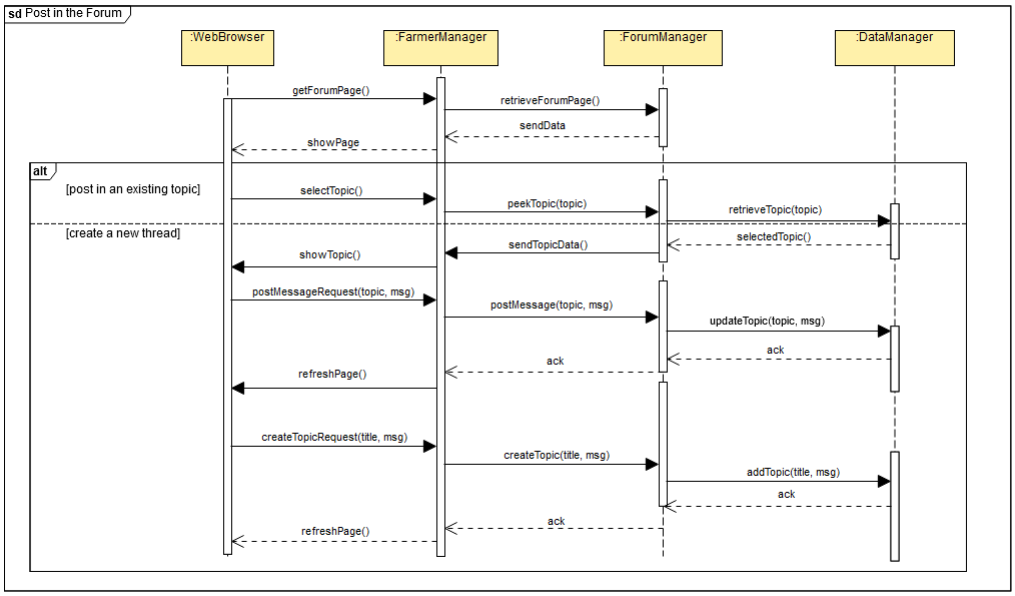

Figure 16: Forum

This sequence diagram represents the interactions between the components of the forum. It is split into two part: the one that allows the user to make a post in an already existing topic and the one that allows the user to create a new topic.

Check Weather Forecast

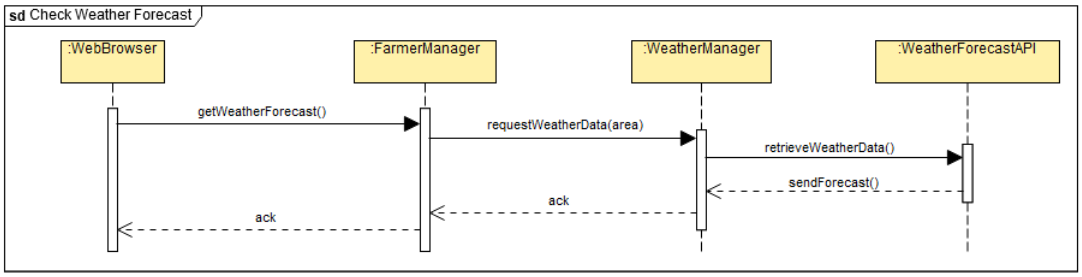

Figure 17: Check Weather Forecast

This sequence diagram represents the interactions between components that occur when displaying the weather forecast to the users. The weather report is tailored to the user area. The *WeatherManager* component uses the *WeatherForecastAPI* component in order to get always up-to-date forecast.

Help Request

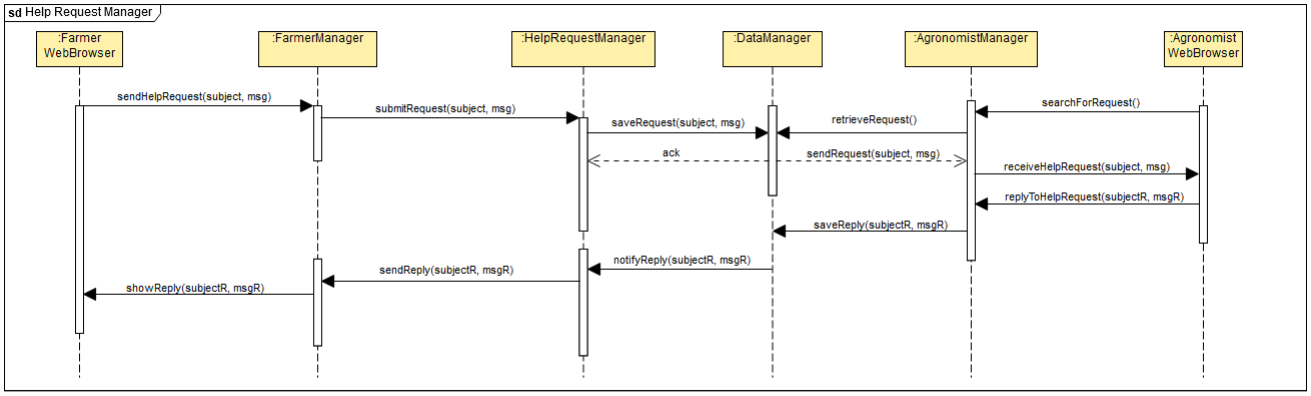

Figure 18: Help Request

This sequence diagram represents the interactions between the components needed to manage the help requests that the farmers send to the agronomists. When a farmer sends an help request the system saves it in the database. When the agronomist is online an automatic script checks if there are some unread messages in the database and sends them to the agronomist. The agronomist replies to the request and saves it into the database. When the farmer is online another automatic script checks if there are some unread replies in the database and displays them to the farmer. Since the exchange is necessarly asyncronous because of the fact that the receiver of the message might be offline in the moment of the delivery, the messages have to be stored on the database and the unread ones have to be fetched each time the user is back online.

Daily Plan Management

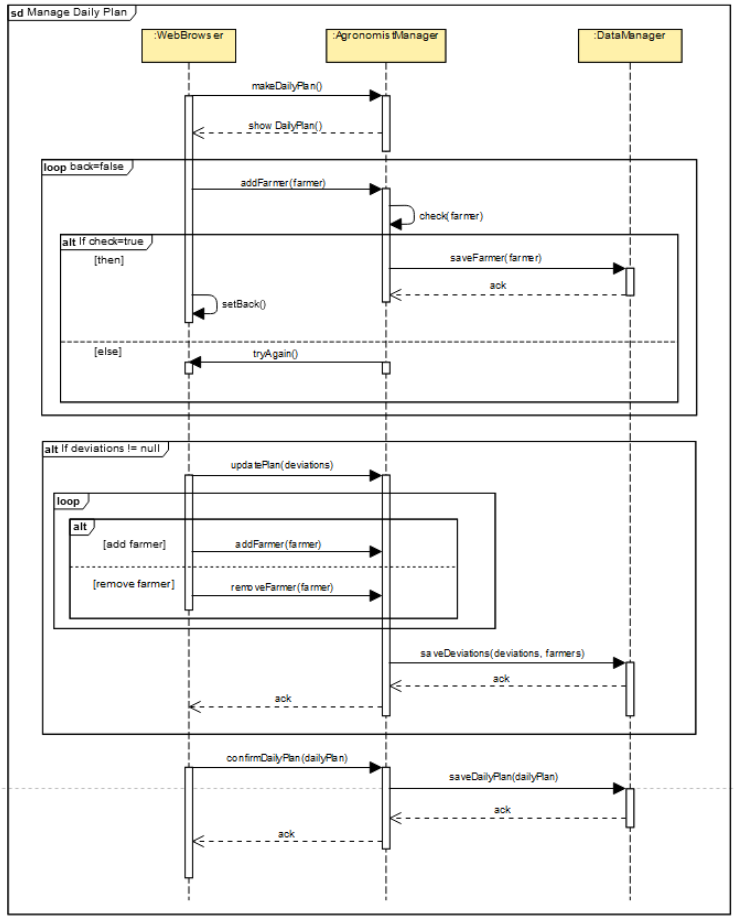

Figure 19: Daily Plan

This sequence diagram represents the interactions between components in managing the daily plan. In the diagram, there are both the creation and the updating of the daily plan.

2.5 Component interfaces

2.6 Selected architectural styles and patterns

2.7 Other design decisions

3 User Interface Design

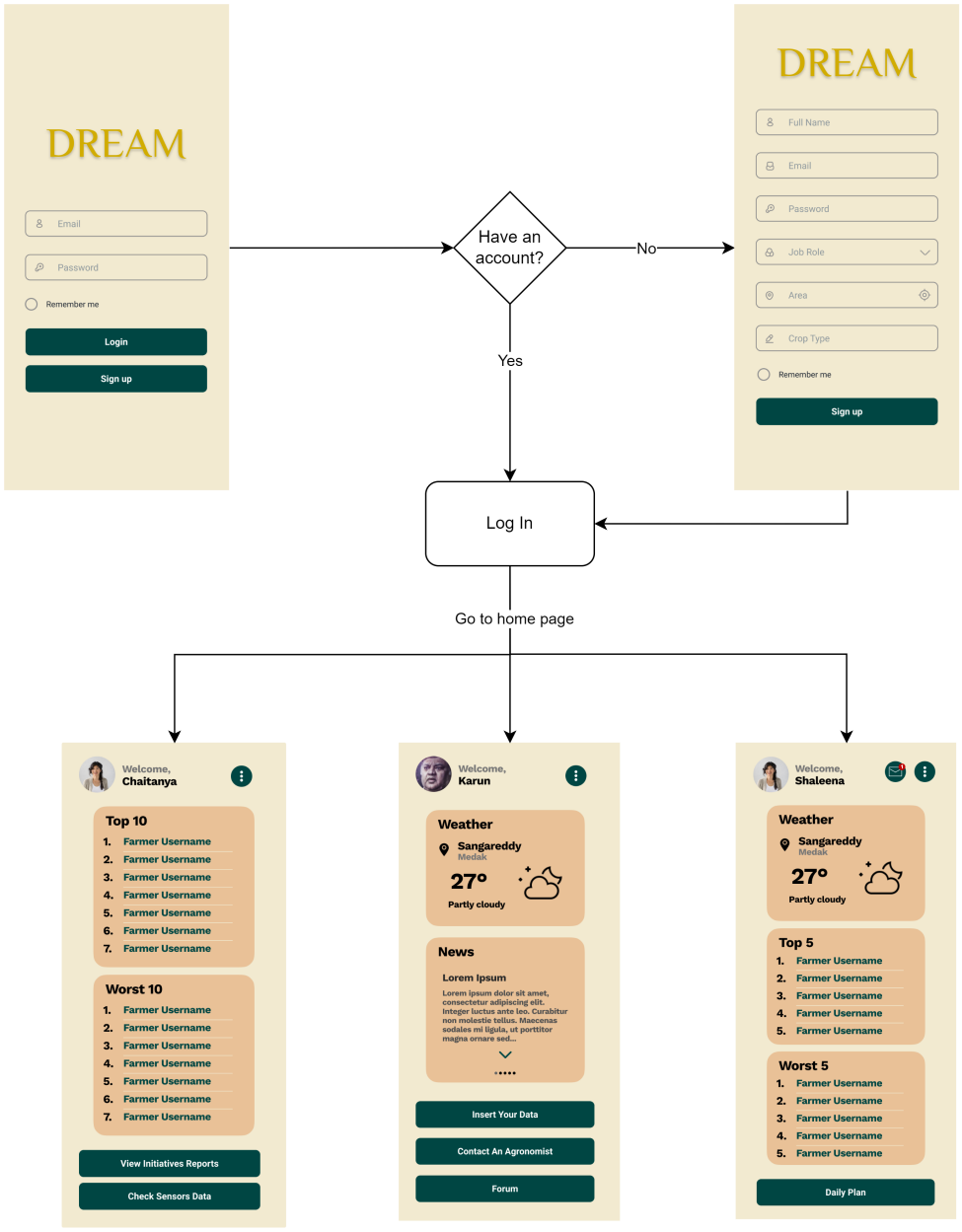

Figure 20: Sign Up and Log In

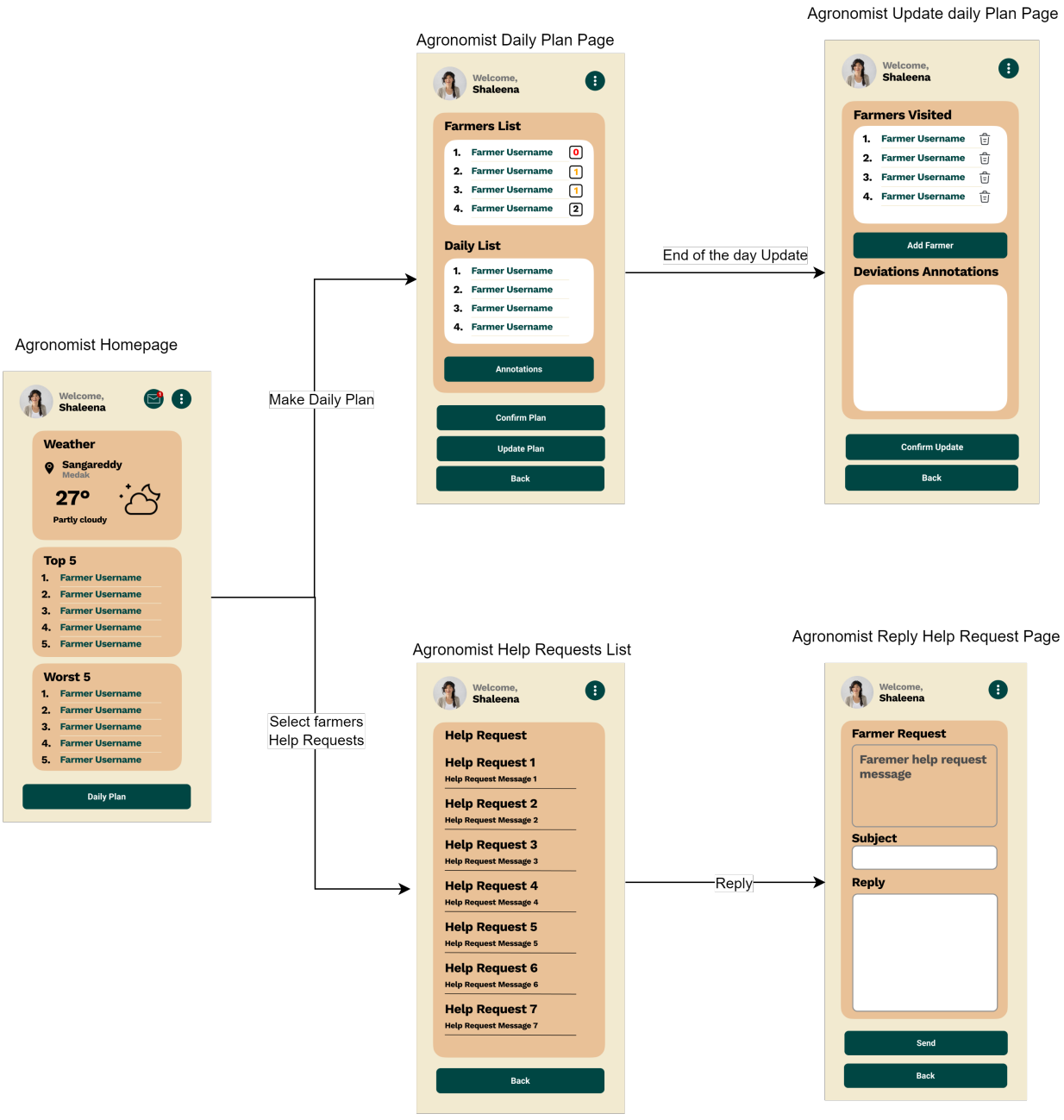

Figure 21: Agronomists Interactions

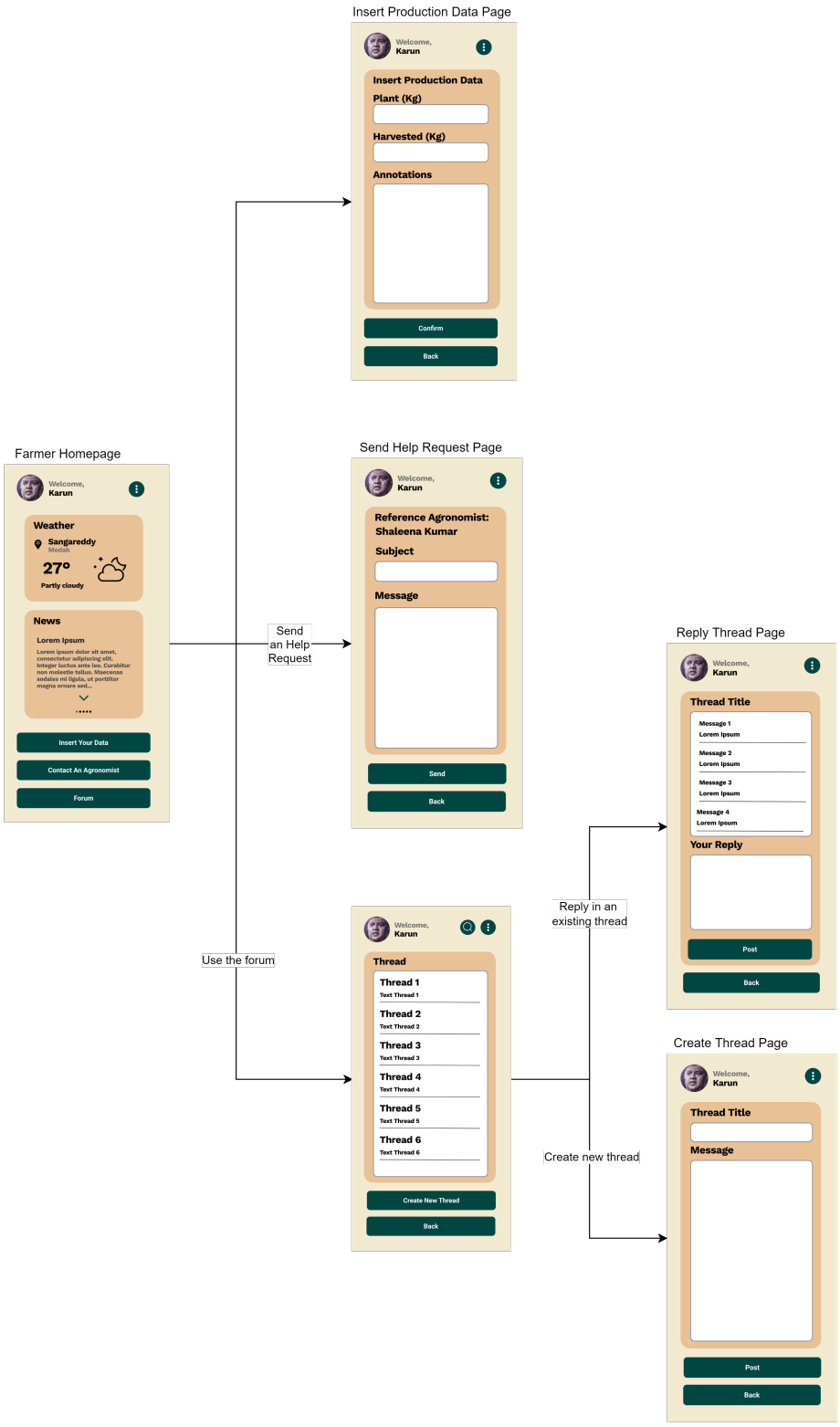

Figure 22: Farmers Interactions

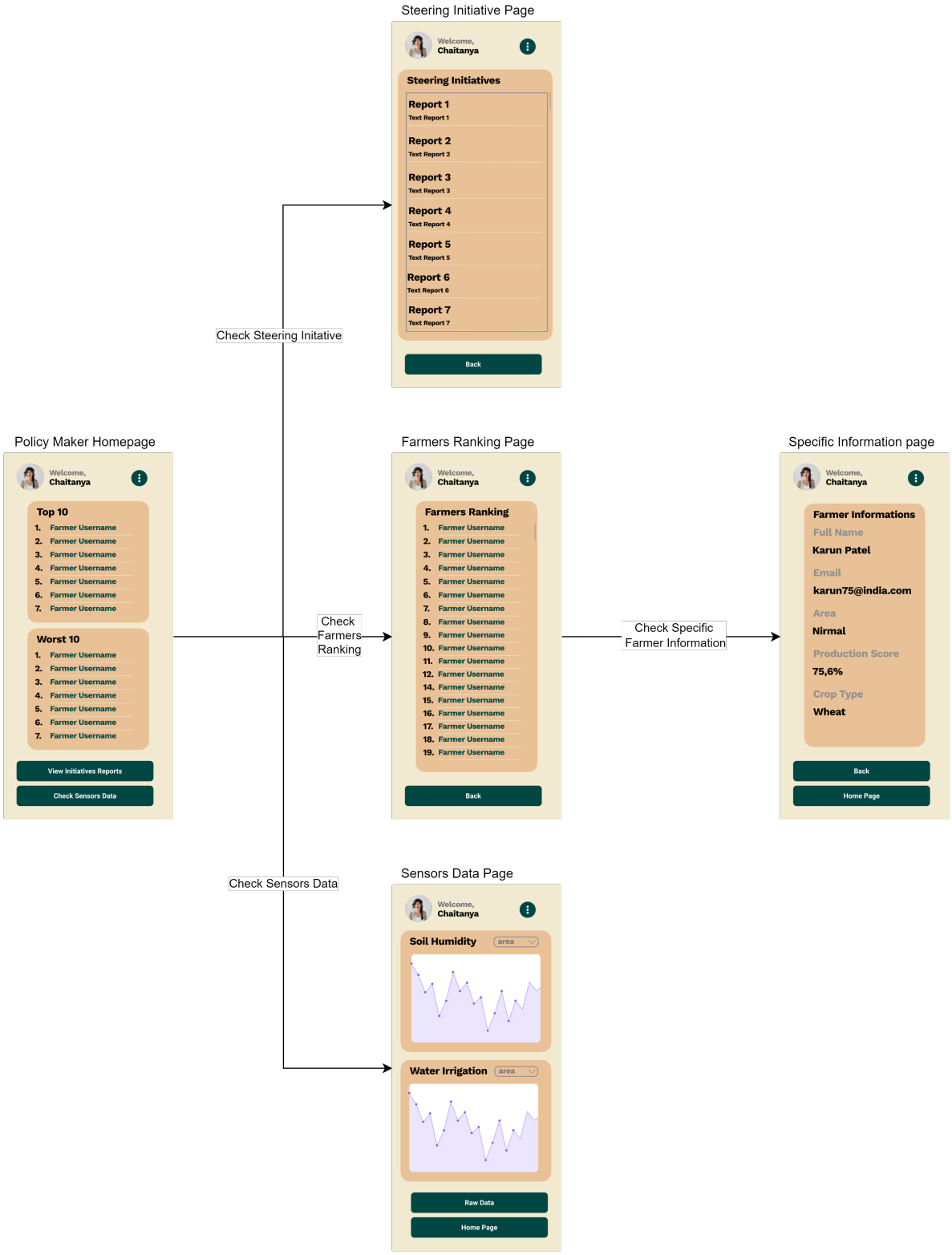

Figure 23: Policy Maker Interactions