POLITECNICO
MILANO 1863

Computer Science and Engineering

DD: Design Document

Software Engineering 2 Project
Academic year 2021 - 2022

22 December 2021
Version 1.0

Authors:
Lorenzo IOVINE
Nicola LANDINI
Francesco LEONE

Professor:
Matteo Giovanni ROSSI

Deliverable:	DD
Title:	Design Document
Authors:	Lorenzo Iovine, Nicola Landini, Francesco Leone
Version:	1.0
Date:	09-January-2022
Download page:	https://github.com/fraleone99/IovineLandiniLeone
Copyright:	Copyright © 2021, Lorenzo Iovine, Nicola Landini, Francesco Leone – All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

This application aims to improve the agricultural process of Telangana by allowing farmers to communicate with each other and with agronomists. Furthermore, a comprehensive view of problems and results is provided to policymakers. All users should be able to access the application's functionalities through both a mobile app and a web application.

1.2 Scope

The application will enable different actors to use the application:

- The **farmer** will insert data about his production and information about his problems. He can check the weather and personalized suggestion based on his types of crops and the weather of his zone. In addition, he can ask a question on the forum and can reply to questions of other farmers. In case of serious problems, he can send a private help request to the agronomist responsible for their zone.
- The **agronomist** can organize his visits to the farms, can check the weather and register good practices. In addition, he can answer help requests and forum questions, he can set threshold for production.
- The **policymaker** can visualize a dashboard with relevant information about farmers and the agricultural production of Telangana.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Client-side scripting: it is performed to generate a code that can run on the client end (browser) without needing the server side processing.

Code On Demand: in distributed computing, it is any technology that sends executable software code from a server computer to a client computer upon request from the client's software.

RESTful: it s a software architectural style that defines a set of constraints to be used for creating Web services.

Tier: it is a row or layer in a series of similarly arranged objects. In computer programming, the parts of a program can be distributed among several tiers, each located in a different computer in a network.

1.3.2 Acronyms

DREAM	Data-dRiven prEdictive fArMing
API	Application Programming Interface
DBMS	DataBase Management System
DD	Design Document
ER	Entity-Relationship model
HTTP	Hypertext Transfer Protocol
RASD	Requirements Analysis and Specification Document

1.3.3 Abbreviations

[Rn] n-th functional requirement.

[ID] identifier.

1.4 Revision history

v. 1.0 - 09/01/2022 Initial release

1.5 Reference documents

WeBeeP channel - Project Assignment

RASD - L. Iovine, N. Landini, F. Leone

1.6 Document Structure

- **Section 1: Introduction**
  This section offers a brief description of the document that will be presented, with all the definitions, acronyms and abbreviations that will be found reading it.
- **Section 2: Architectural Design**
  This section provides a more detailed description of the architecture of the system. The first part describes the chosen paradigm and the overall split of the system into several layers. Then, a high-level description of the system is provided, together with a presentation of the modules composing its nodes. Finally, there is a concrete description of the tiers forming the System.
- **Section 3: User Interface**
  This section contains several mockups of the application, together with some charts useful to understand the correct flow of execution of it. The presented mockups refers to the client-side experience.
- **Section 4: Requirements Traceability**
  This section acts as a bridge between the RASD and DD document, providing a complete mapping of the requirements described in the RASD to the logical modules presented in this document.
- **Section 5: Implementation, Integration and Test Plan**
  The last section describes the procedures followed for implementing, testing and integrating the components of our System. There will be a detailed description of the core functionalities of it, together with a complete report about how to implement and test them.

2 Architectural Design

2.1 Overview

The system paradigm is client-server. In particular, there is a thin client and a fat server. The client contains only a presentation layer. The server, instead, contains both the business application logic and the data management logic. The layer of the application are:

- **Presentation Layer:** it manages the presentation logic and all the interactions with the user.
- **Application Layer:** it manages all the functionalities of the system.
- **Data Layer:** it manages the access and the storage of data.

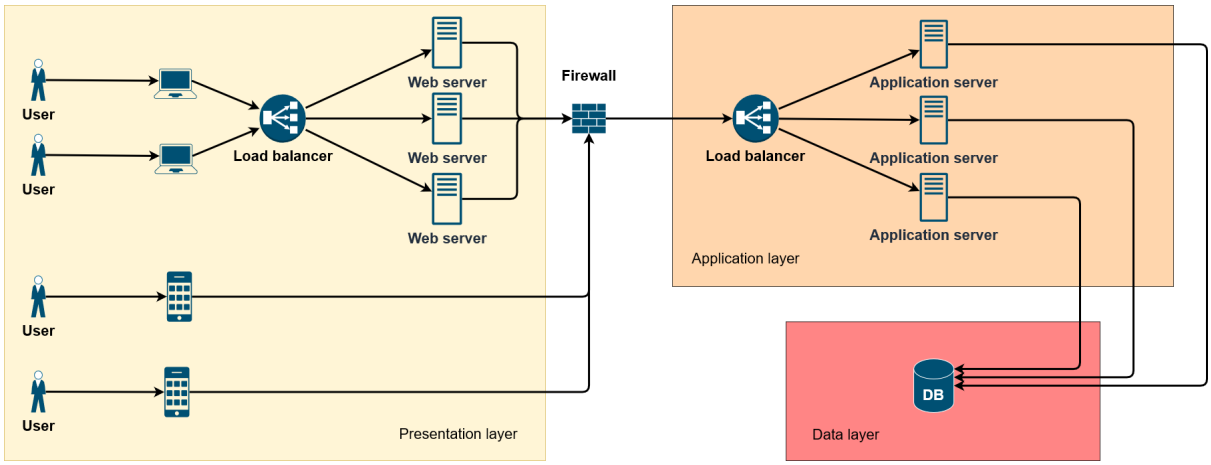


Figure 1: Architecture of the application

As shown in figure 1 the application is divided into 3 layers and 4 tiers. Users can access the service both from a mobile app or web interface. The figure shows the division of the different layers. Since the application follows the REST standard the servers are stateless. There will be a more precise description of the components in the following sections.

2.2 Component View

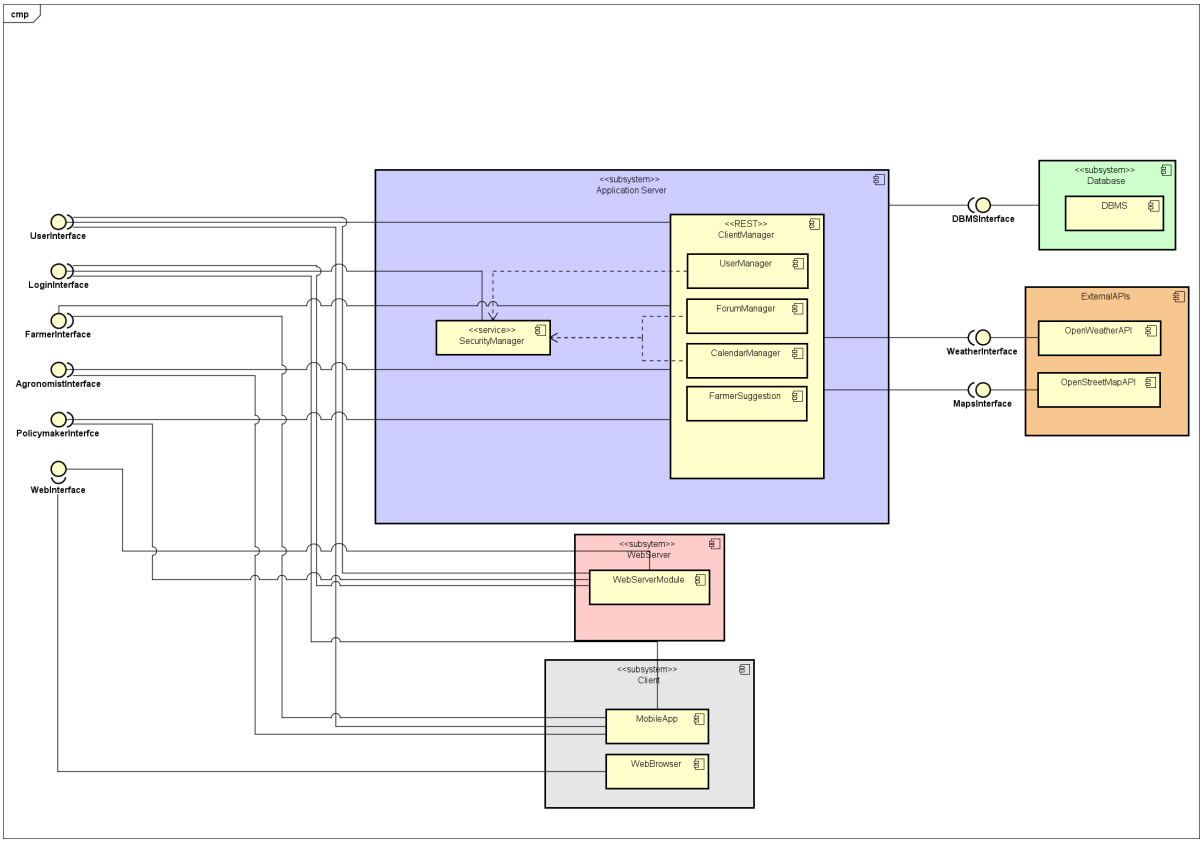


Figure 2: Component Diagram

Figure 2 represents in detail the layers described before. The web server has the function to route the browser requests to the application server and send back its responses.

- **Client Manager**
  This module handles all the requests made by the client. In the beginning, when the client is not logged in, the module offers (through the user manager) a loginInterface that allows the client to register and log in. After the login the module provide an interface based on the role of the user: FarmerInterface for the farmer, AgronomistInterface for the agronomist, PolicymakerInterface for the policymaker.

- **User Manager**
  This module provides all the functionalities needed to manage the user. It includes signup and login as well as role checking operation and information retrieving.

- **Forum Manager**
  This module provides all the functionalities through the ForumInterface for managing and using the forum. It includes operations for creating and replying to a discussion on the forum.

- **Calendar Manager**
  This Module provides all the functionalities needed for the Agronomist to manage his calendar. In particular, it provides through, the AgronomistInterface, the possibility to add or remove an appointment, to confirm the daily plan and to check that each farm has at least two visits per year.

- **Farmer Suggestion**
  This module handles the creation and delivery of personalized suggestions to the Farmer.
- **Security Manager**
  This module handles all the security issues.

2.3 Deployment diagram

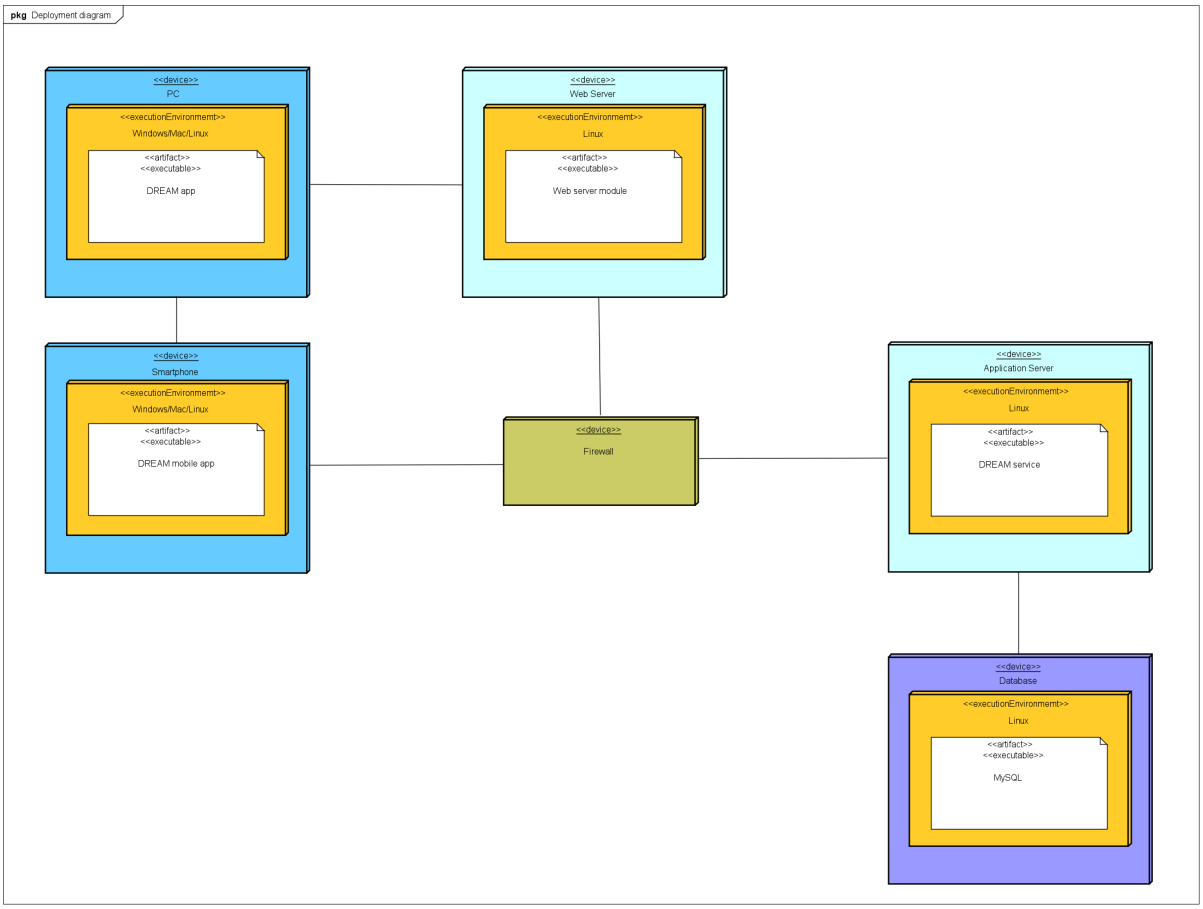


Figure 3: Deployment Diagram

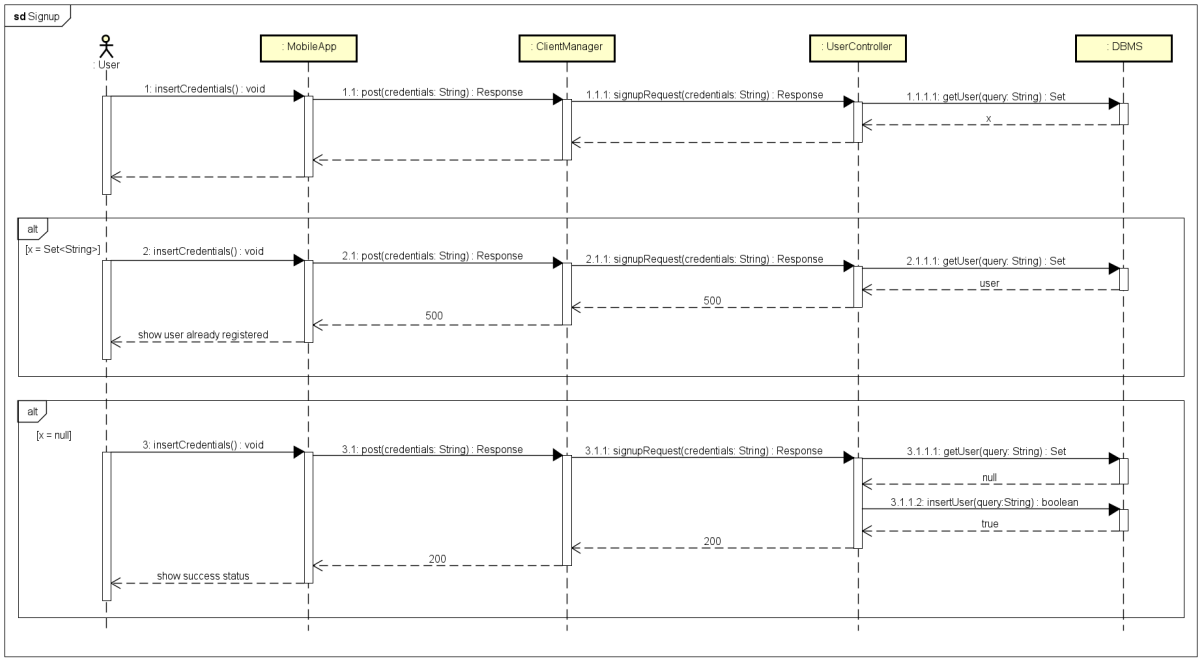
The deployment diagram in figure shows the needed components for correct system behavior. Each device owns its Operating System where the software runs. Firewall ensures application security.
The tiers in the image are the following:

- **Tier 1:** it is the client machine, which can be a computer with a web browser or the mobile application.
- **Tier 2:** includes the replicated web servers, which do not execute any business logic, but simply receive requests from the client, route them to the application servers and serve an HTML back to the client.
- **Tier 3:** it includes the application servers that contain the whole application layer and communicate to the data tier.
- **Tier 4:** it contains the DBMS servers that store and manage data, according to the instructions given by the application servers.

2.4 Runtime View

The following sequence diagrams represent the runtime view, for the most relevant operations in DREAM, from the mobile app's perspective (for agronomists and farmers) and the web browser's perspective (for policymakers).
Every interface use REST API through HTTP GET and POST.

2.4.1 Sign Up

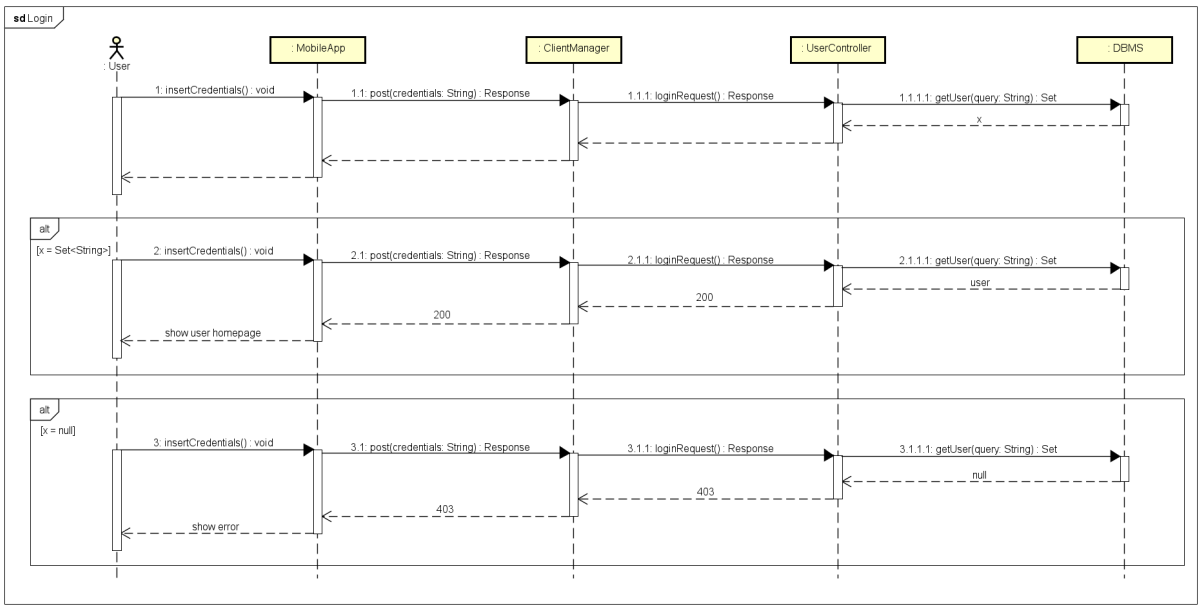


This diagram represents the process of signing up a user through two possible situations:

- User is already registered;
- User isn't already registered.

After the User inserts his credentials and the data are transmitted to the database, they are analyzed and we will have the two separate ways.
If these credentials are already present in the database, the UserController sends a 500 response's status code, warning the User that a previous registration with these data exists.
Otherwise, if the credentials aren't present in the DBMS, the UserController inserts the User into the database and sends a 200 response's status code, informing the User that the registration was successful.

2.4.2 Login

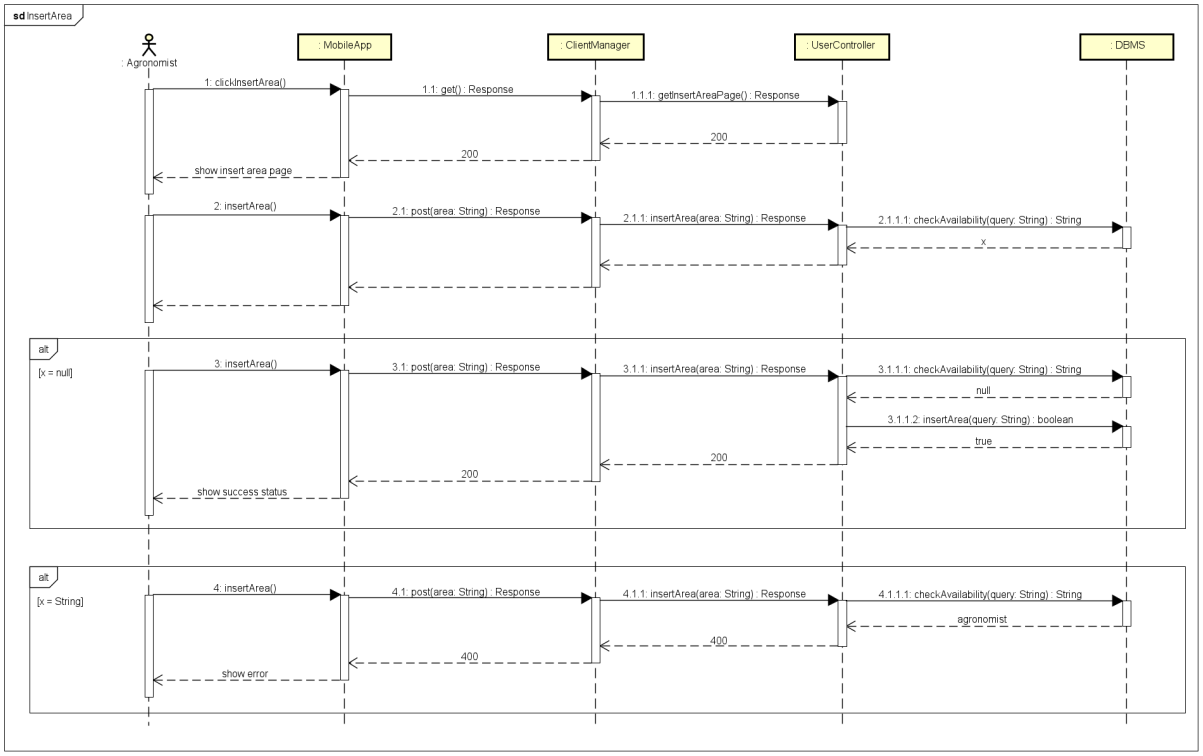


This diagram represents the process of login of a User and has the same two ways as the process of signing up:

- User is already registered;
- User isn't already registered.

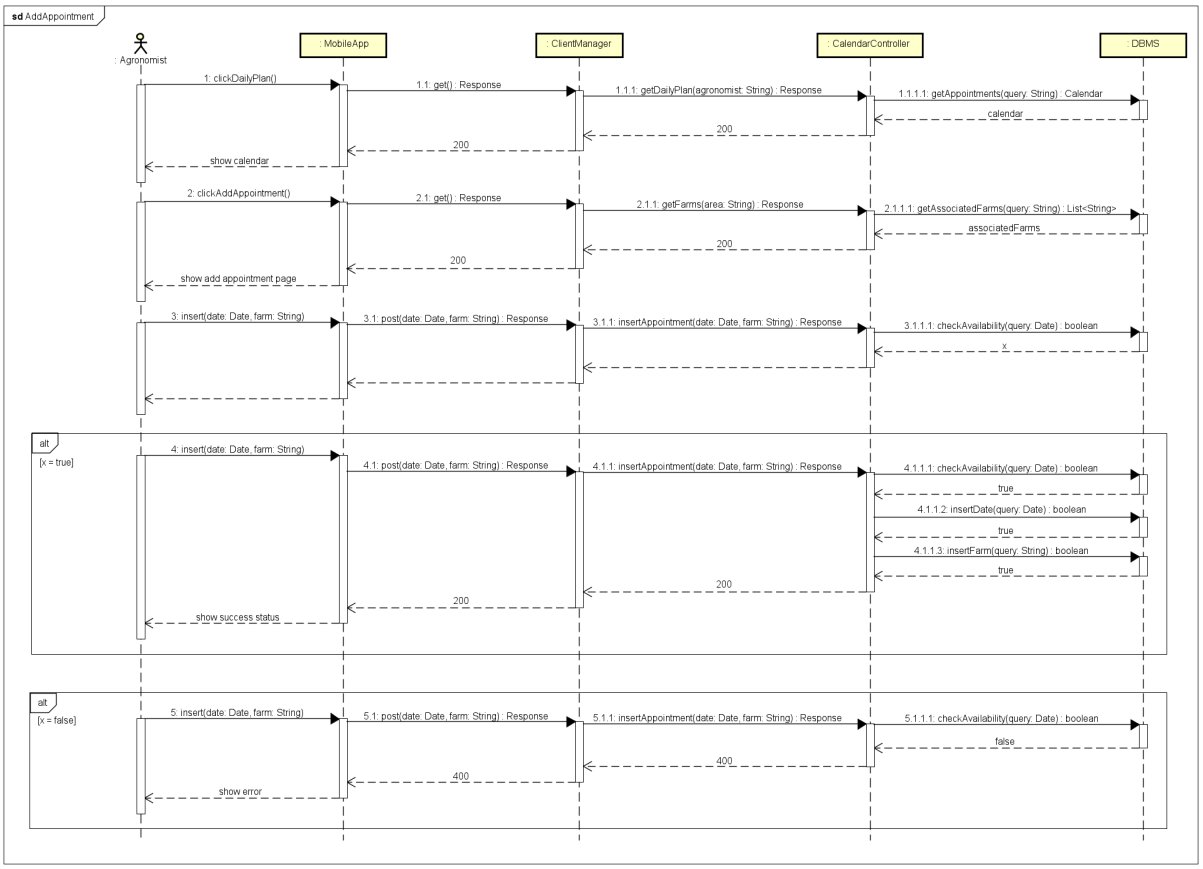
After the user enters his credentials and these are analyzed by the database, the two paths are opposite with respect to the registration case.
If the user is already present in the database, the UserController sends a 200 response's status and the User is taken to the homepage.
If the user is not already present in the database, the UserController sends 403 responses's status, warning the User that a previous registration with these data doesn't exist.

2.4.3 Insert Area



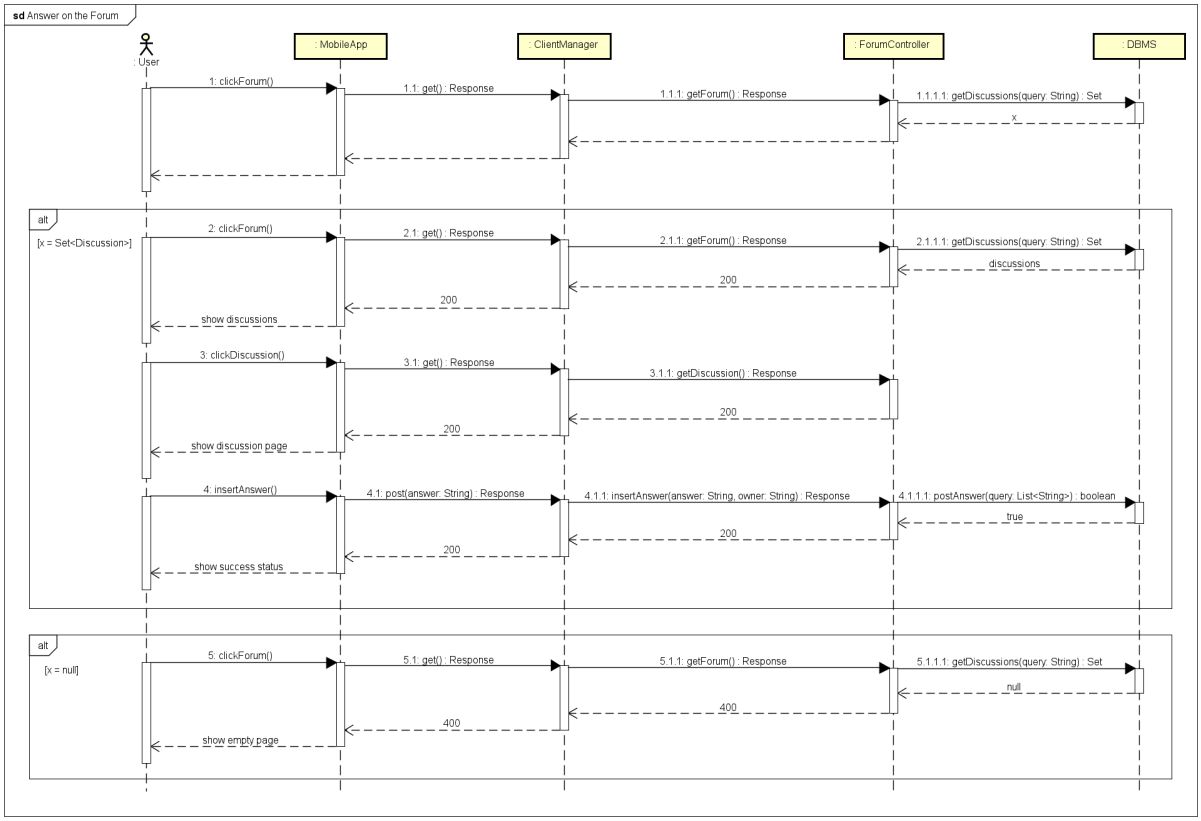
This sequence diagram represents an operation that could be done only by agronomists.
After the user inserts the area he is responsible for, the UserController checks in the DBMS if there is already a registered agronomist responsible for the same area that the User inserted.
If there isn't a registered agronomist, the UserController inserts the area in the DBMS and sends a 200 response's status code to the User; otherwise sends a 400 response's status code to the User.

2.4.4 Add appointment



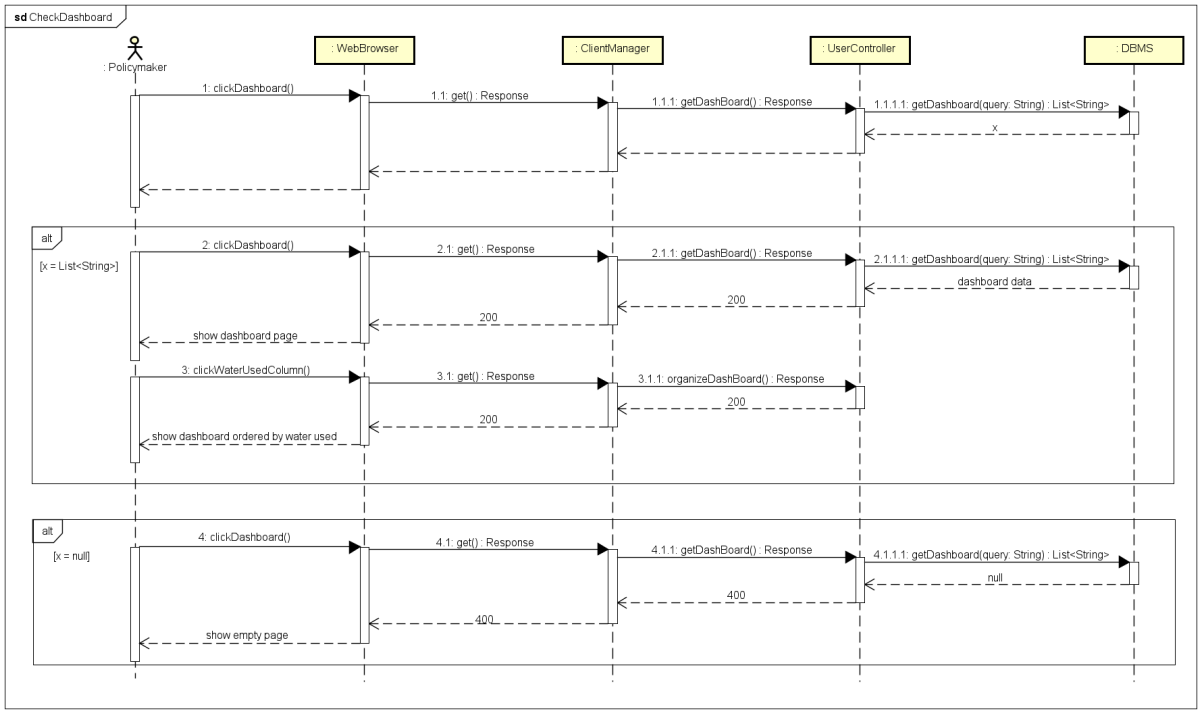
This sequence diagram shows the steps that an agronomist has to do for adding an appointment.
After checking the daily plan, the agronomist can press on "ADD" button and specify the farm and the date of the appointment that he wants to make.
The CalendarController query the DBMS to check if the selected date is available and here we have two different ways.
If the date is available, the CalendarController adds the appointment in the DBMS and sends a 200 response's status code to the User; otherwise, the CalendarController sends a 400 response's status code to the User.

2.4.5 Answer on the Forum



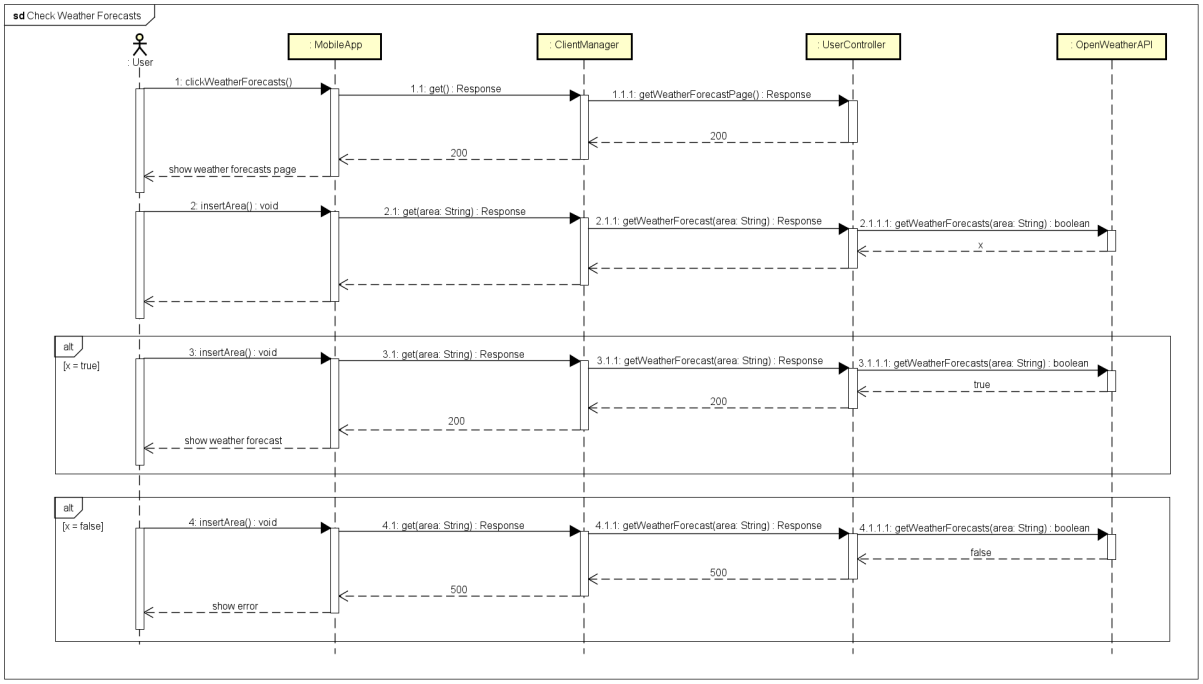
This sequence diagram represents an operation that could be done by an agronomist or a farmer.
The User has to press on a discussion and insert his answer. After that, the ForumController inserts the answer in the DBMS and sends a 200 response's status code to the User.
The ForumController sends a 400 response's status code to the User only if there isn't an open discussion thus showing an empty page.

2.4.6 Check Dashboard



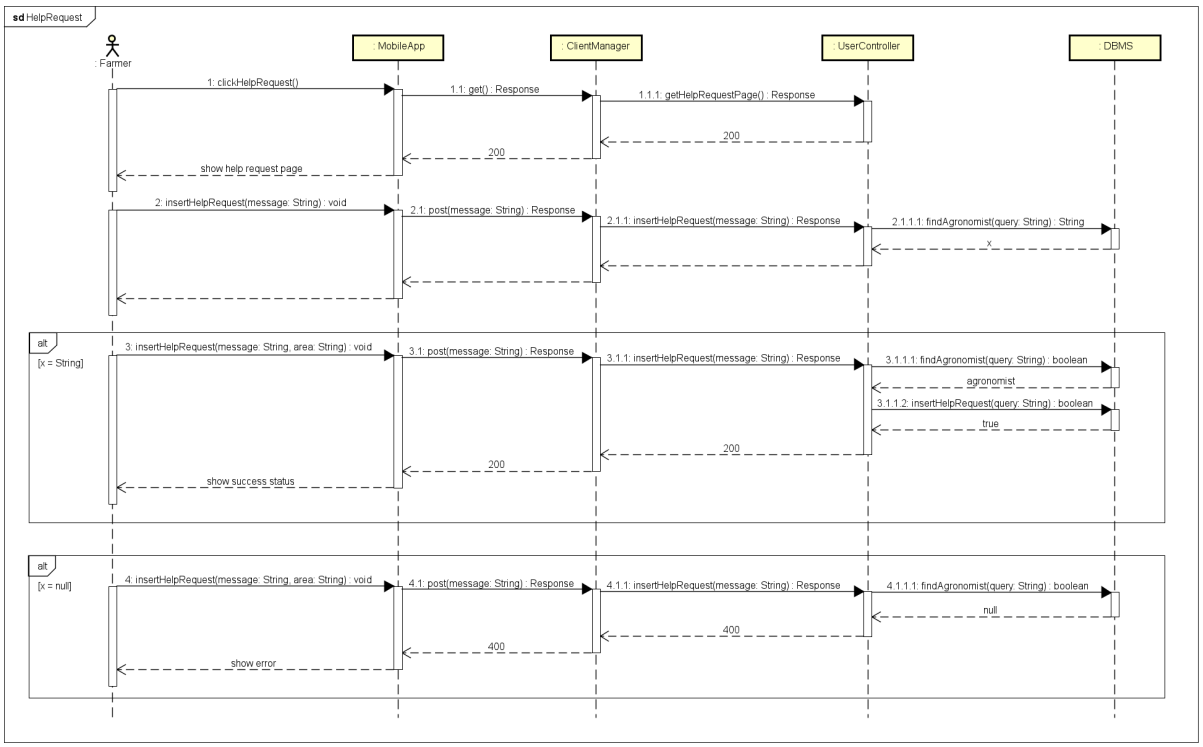
This sequence diagram represents an operation that could be done by agronomists and policymakers. The only difference between the operation done by these two different users is that the policymakers can check the dashboard of all farms, while the agronomists can check only the dashboard about the farms of the area that they are responsible for.
After retrieving the data from the DBMS, the UserController sends a 200 response's status code to the User and shows him the dashboard. The User can then reorganize the data according to one of the fields on the dashboard.
The UserController sends a 400 response's status code to the User only if there isn't data about farms in the DBMS thus showing an empty page.

2.4.7 Check Weather Forecasts



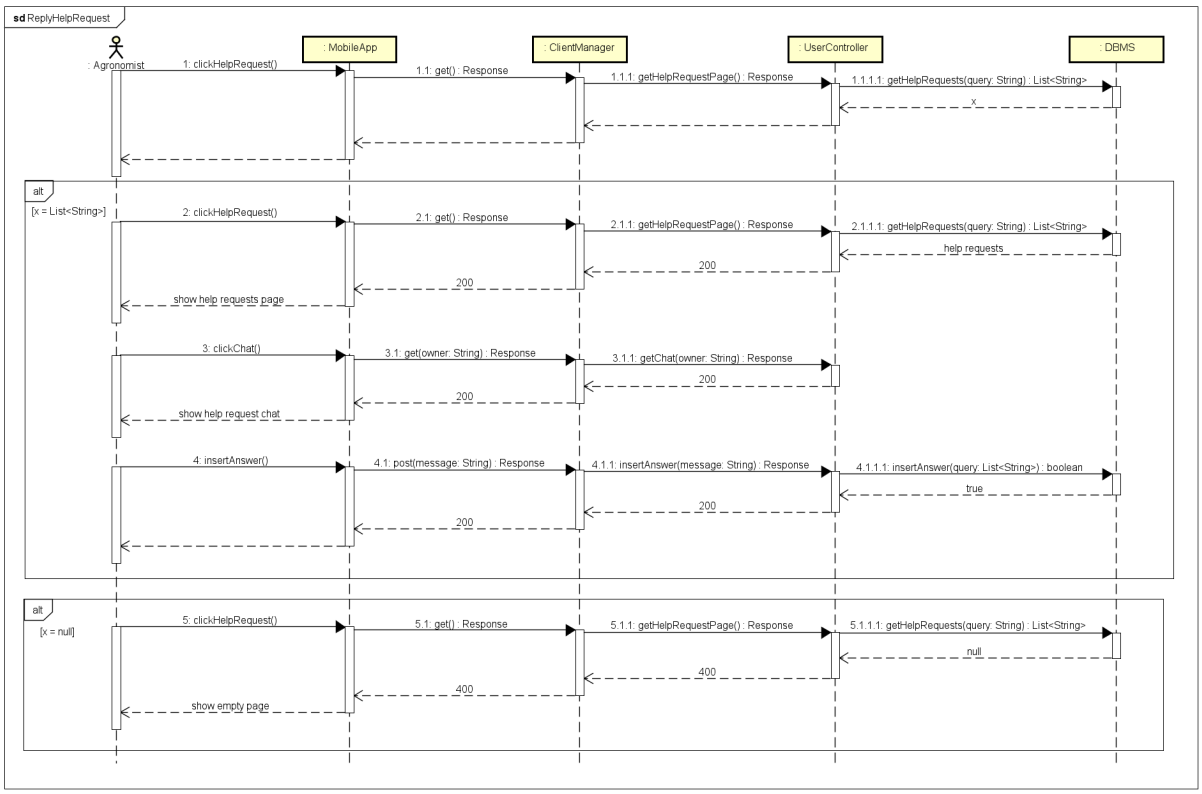
This sequence diagram represents an operation that could be done by agronomists and farmers.
On the Weather Forecasts page, the User has to specify the area for which he wants to know the weather forecast.
After that, the UserController query the OpenWeatherAPI and, if the inserted area exists, the Controller sends a 200 response's status code to the User and shows him the weather forecast.
In case of the inserted area doesn't exist, the OpenWeatherAPI responds to the request with "false" and the UserController sends a 500 response's status code to the User.

2.4.8 Help Request



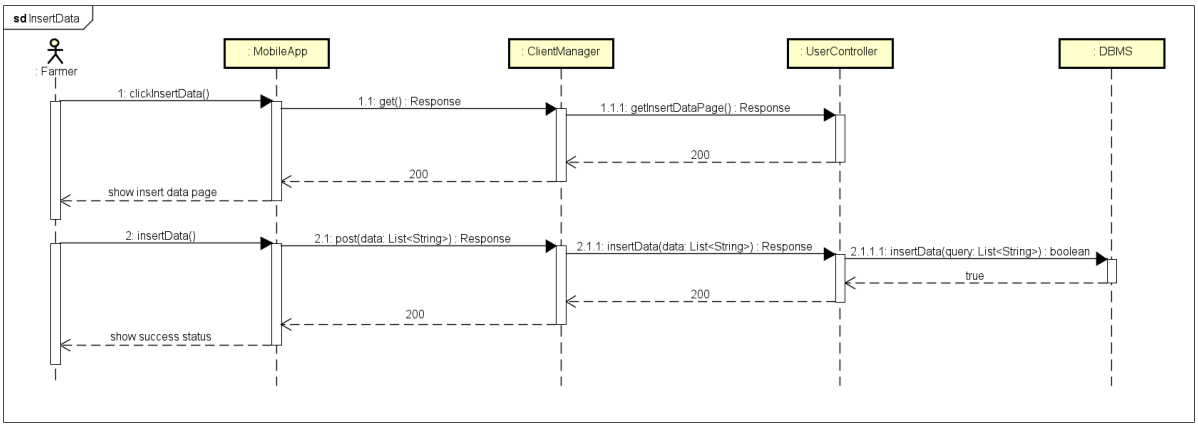
This sequence diagram represents how the private help requests by farmers occur.
After the farmer describes his problem, the UserController checks in the DBMS if there is a registered agronomist responsible for the same area where the User's farm is located.
If there is a registered agronomist, the UserController inserts the help request in the DBMS and sends a 200 response's status code to the User; otherwise sends a 400 response's status code to the User.

2.4.9 Reply Help Request



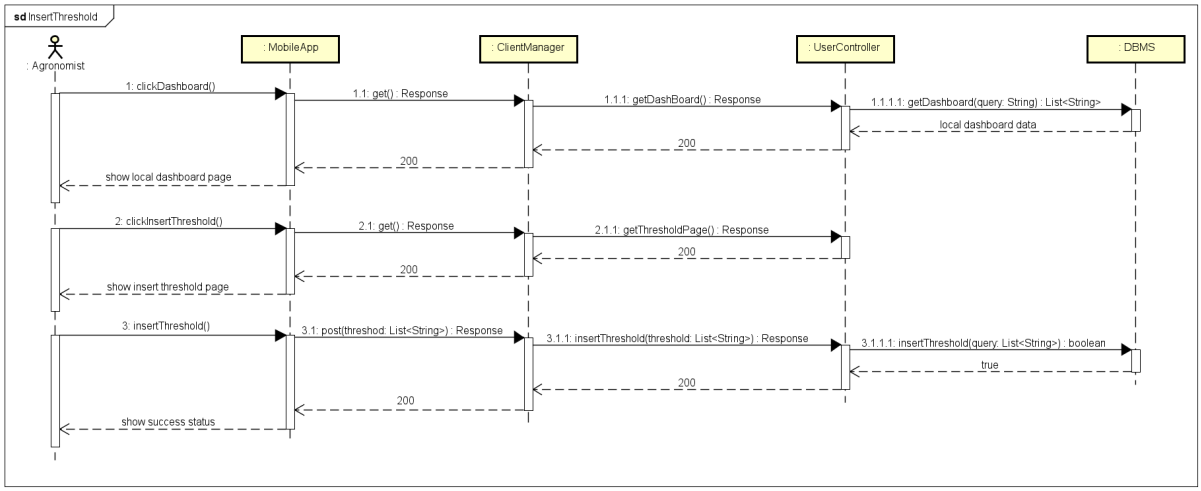
This sequence diagram represents an operation that could be done only by agronomists.
After the user has clicked on the "Help Request" icon, the UserController queries the DBMS all the help requests submitted to the agronomist and shows them to him.
The user presses on a chat and enters his answer, which the UserController then inserts in the DBMS and sends a 200 response's status code to the Agronomist.
The UserController sends a 400 response's status code to the User only if there aren't help requests in the DBMS thus showing an empty page.

2.4.10 Insert Data



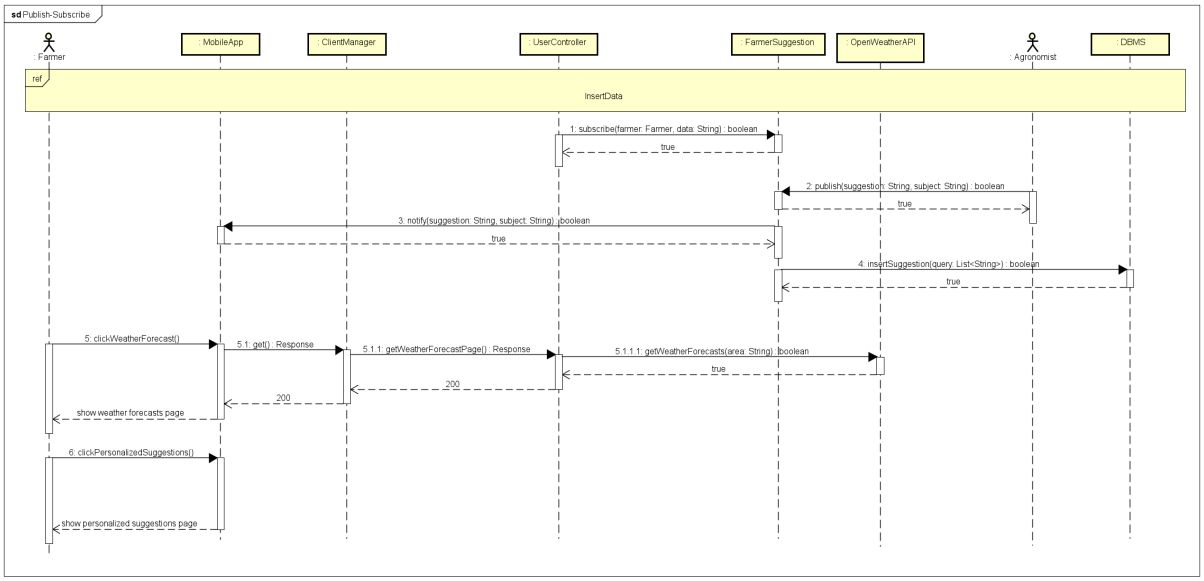
This sequence diagram represents an operation that could be done only by farmers.
After the User has entered the data concerning the crops of his farm, the UserController inserts these data in the DBMS and sends a 200 response's status code to the Farmer.

2.4.11 Insert Threshold



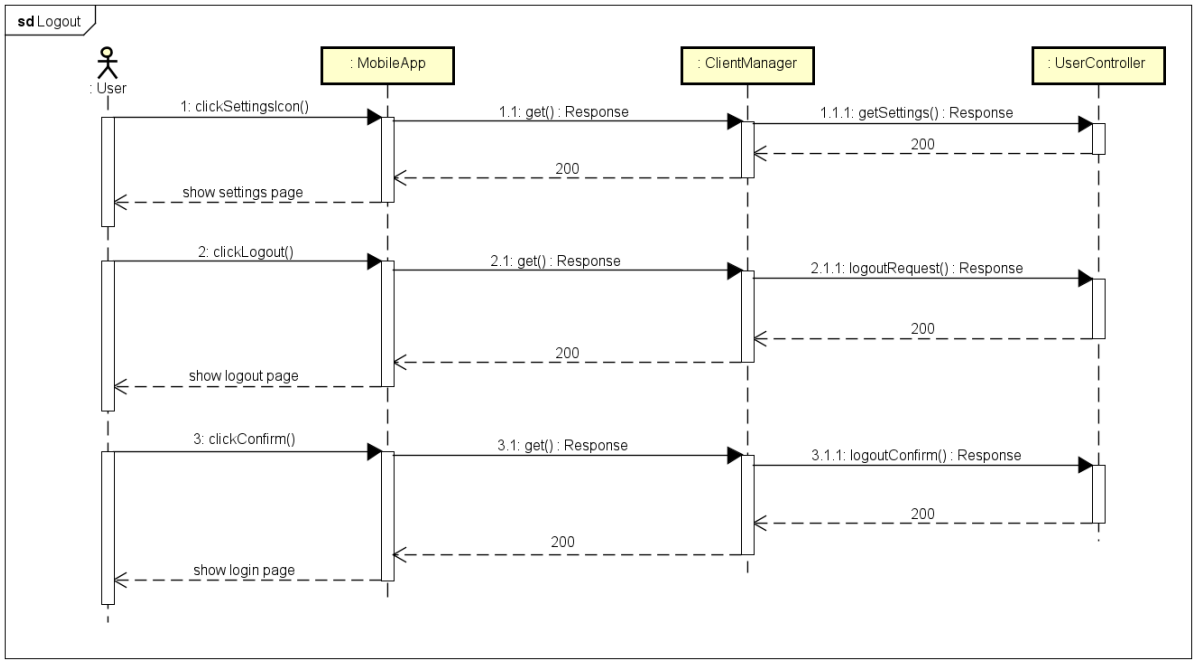
This sequence diagram represents the operation of inserting a threshold by agronomists.
From the Dashboard page, the Agronomist presses the "Insert Threshold" button and then inserts the threshold he deems appropriate.
After that, the UserController inserts the threshold in the DBMS and sends a 200 response's status code to the Agronomist.

2.4.12 Personalized Suggestions



This sequence diagram represents how the forwarding of personalized suggestions to farmers works.
After having completed the operation of entering data concerning the crops, the UserController contacts the FarmerSuggestion to automatically subscribe the User to receive suggestions on how to improve their production.
Then when an agronomist inserts a suggestion through the FarmerSuggestion, the latter will forward the suggestion to all the farmers who had been subscribed. (Some steps of this operation have been omitted from the sequence diagram to avoid repetition).
Farmers can then find all their personalized suggestions by pressing the "Personalized Suggestions" button on the Weather Forecasts Page.

2.4.13 Logout



The user clicks on the "Settings" icon and after on the "Logout" button.
The UserController asks the User if he is sure he wants to logout and after confirmation takes him to the login page.

2.5 Component Interfaces

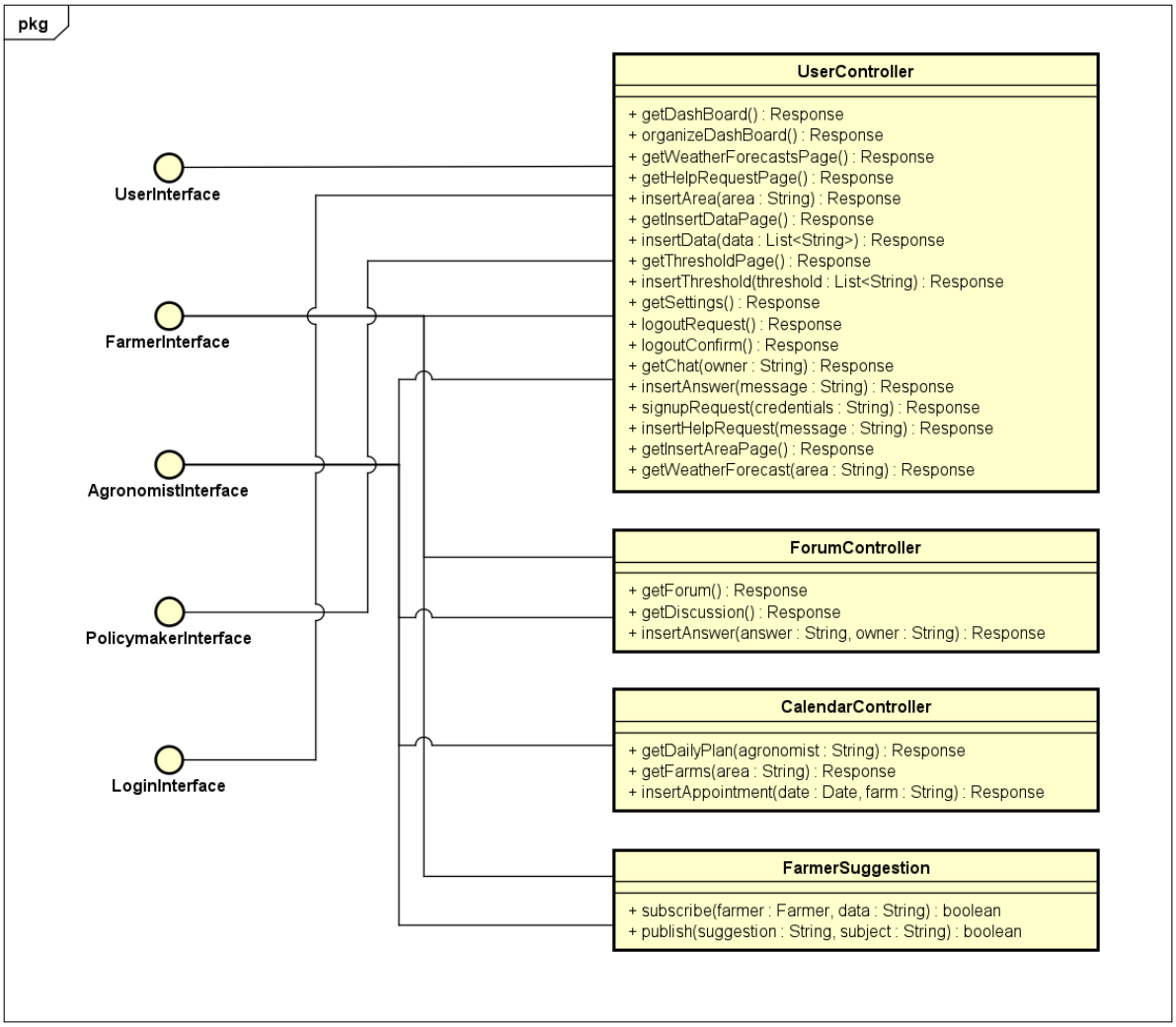


Figure 4: Interface Diagram

Figure 4 shows the main interfaces of the system.
The **UserInterface** provides the functionalities common to all users, it is realized through just one controller the **UserController**.
The **FarmerInterface** provides all the specific functionalities of the farmer and it is realized through three controllers: **UserController**, **ForumController**, **FarmerSuggestion**.
The **AgronomistInterface** provides all the specific functionalities of the agronomist and it is realized through four controllers : **UserController**, **ForumController**, **FarmerSuggestion**,**CalendarController**.
The **PolicymakerInterface** provides all the specific functionalities of the policymaker and it is realized through one controller: **UserController**.
The **LoginInterface** provides all the functionalities for login and signup and it is realized through **UserController**.

2.6 Logical Description of Data

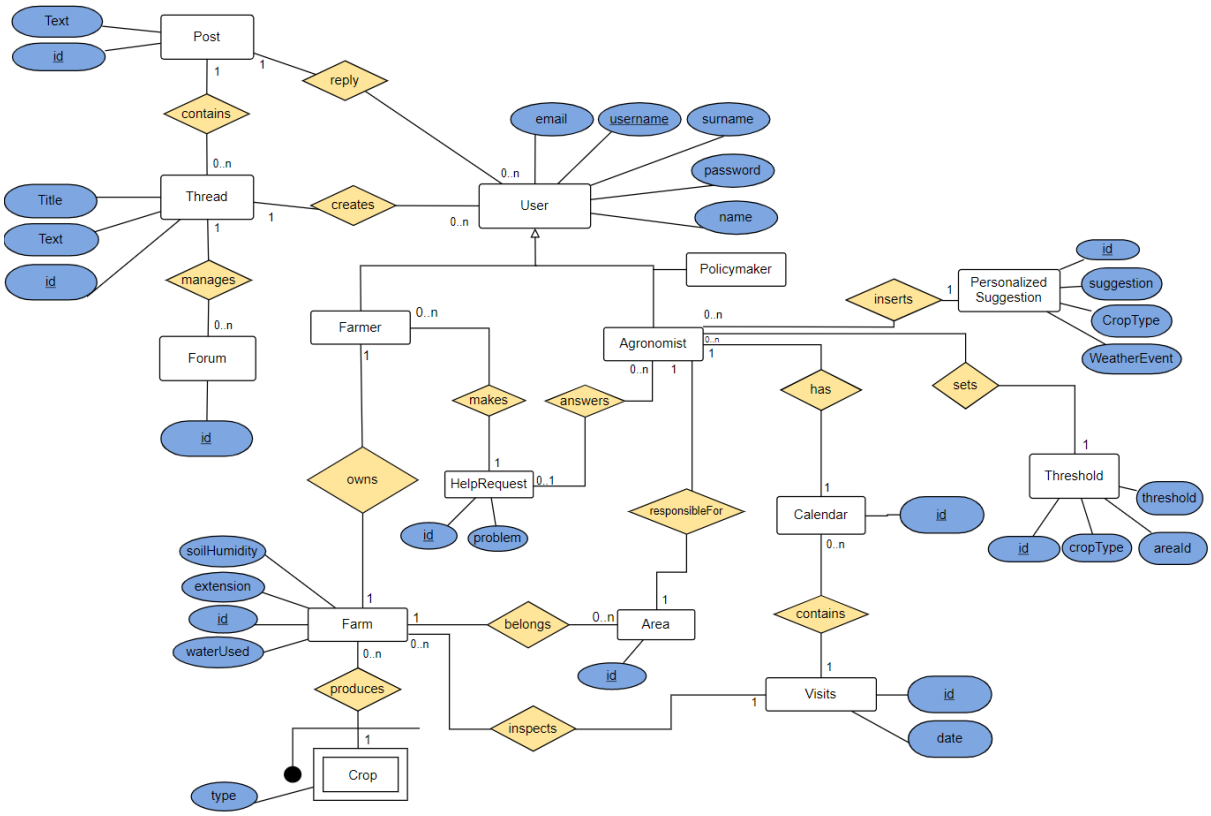


Figure 5: ER Diagram

Figure 5 shows a logical representation of the data that will be stored in the database of the system.

2.7 Architectural Style and Patterns

2.7.1 Four-tiered architecture

We chose this architecture for many reasons:

- **Flexibility:** all parts of the application can be updated independently.
- **Load Distribution:** a 4-tier application enables replication in each tier: replicated application servers, through the exploitation of a load balancer, guarantee the avoidance of requests bottlenecks.
- **Scalability:** an application divided into several tiers allows developers to scale the architecture only for the most critical components.

2.7.2 RESTful Architecture

The **RESTful architecture** is based on HTTP and adopted both on the mobile and web sides. It is stateless so the server doesn't know anything about the state of the client.
An interesting constraint of this architecture is the code on demand, which gives the possibility of sending code to the client, where it is executed locally. The application is intended to support *client-side scripting*, which means that all requests and updates of the page are made on client-side.

2.7.3 Model View Controller (MVC)

Model–view–controller (MVC) is a software design pattern commonly used for developing user interfaces that divide the related program logic into three interconnected elements:

- **Model:** it is the core component of the pattern. It is responsible for managing the data of the application. It receives user input from the controller.
- **View:** renders a presentation of the model in a particular format.
- **Controller:** accepts input and converts it to commands for the model or view.

2.7.4 Publish-subscribe Architecture

Publish–subscribe is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers, but instead categorize published messages into classes without knowledge of which subscribers if any, there may be. Similarly, subscribers express interest in one or more classes and only receive messages that are of interest, without knowledge of which publishers, if any, there are. The application is intended to exploit the *topic-based* message filtering in which messages are published to "topics" or named logical channels: subscribers will receive all messages published to the topics to which they subscribe, the publisher is responsible for defining the topics to which subscribers can subscribe.

2.8 Other Design Decisions

2.8.1 Scale-Out

As said before, the application is intended to be scaled out through the replication of nodes that are expected to generate bottlenecks. As shown in figure 1, this approach requires load balancers to redirect the requests to nodes with the lowest workload.

2.8.2 Thin Client and Fat Server

The web application will be the *thin client*: that's because the application is intended to keep as low information as possible on the client-side. This means that all the business logic resides on the server, so a stable connection is required between the parts. The advantage of that is the low computational power required on the client-side.

3 User Interface Design

This section aims to describe the flow of the main functionalities of the application.
We used some mockups already presented in the RASD, but more have been created and added to this part to cover all the main functionalities of the application. In addition, we chose to show only the mobile application because, in our opinion, Users would interact mostly with it. We want to remember that the application will work also on web browsers for all types of users, but the design and the interfaces of the web app will derive from these.

3.1 User Mobile Interface

The following images represent the flow of the functionalities common to all the users of the application.

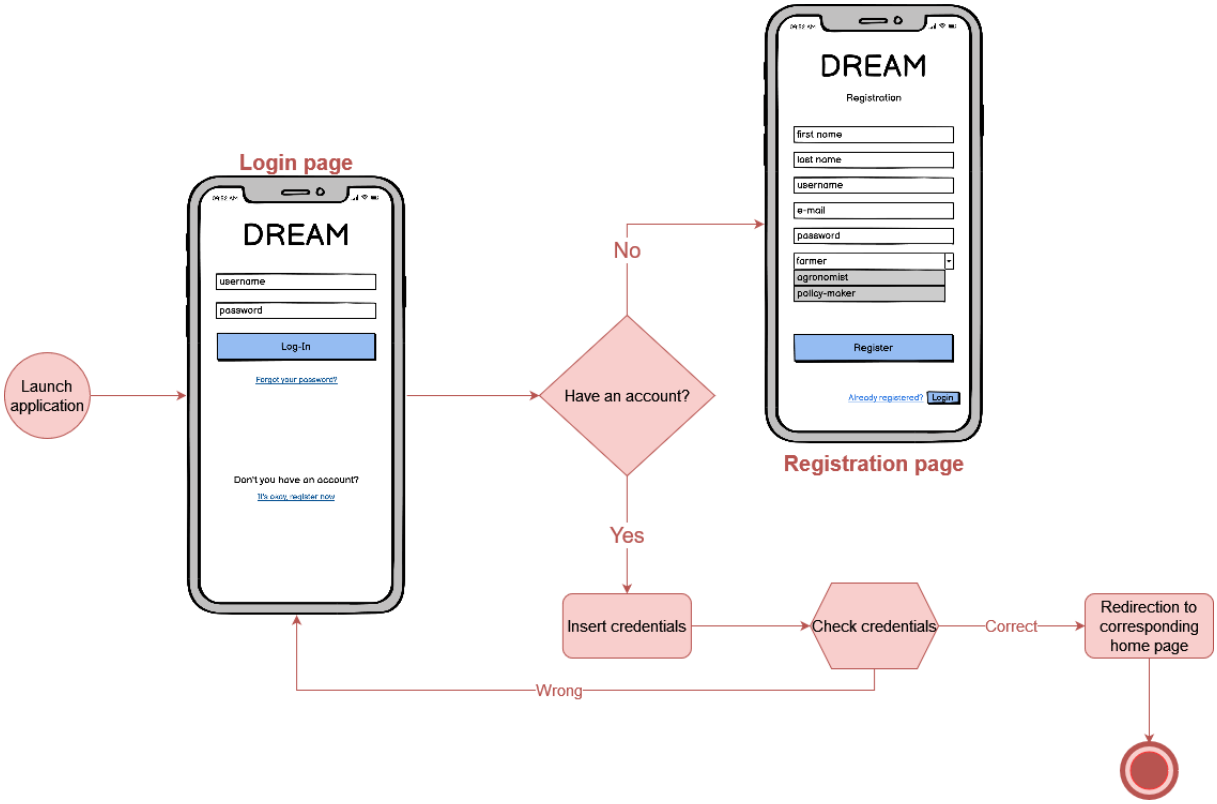


Figure 6: Sign Up & Sign In

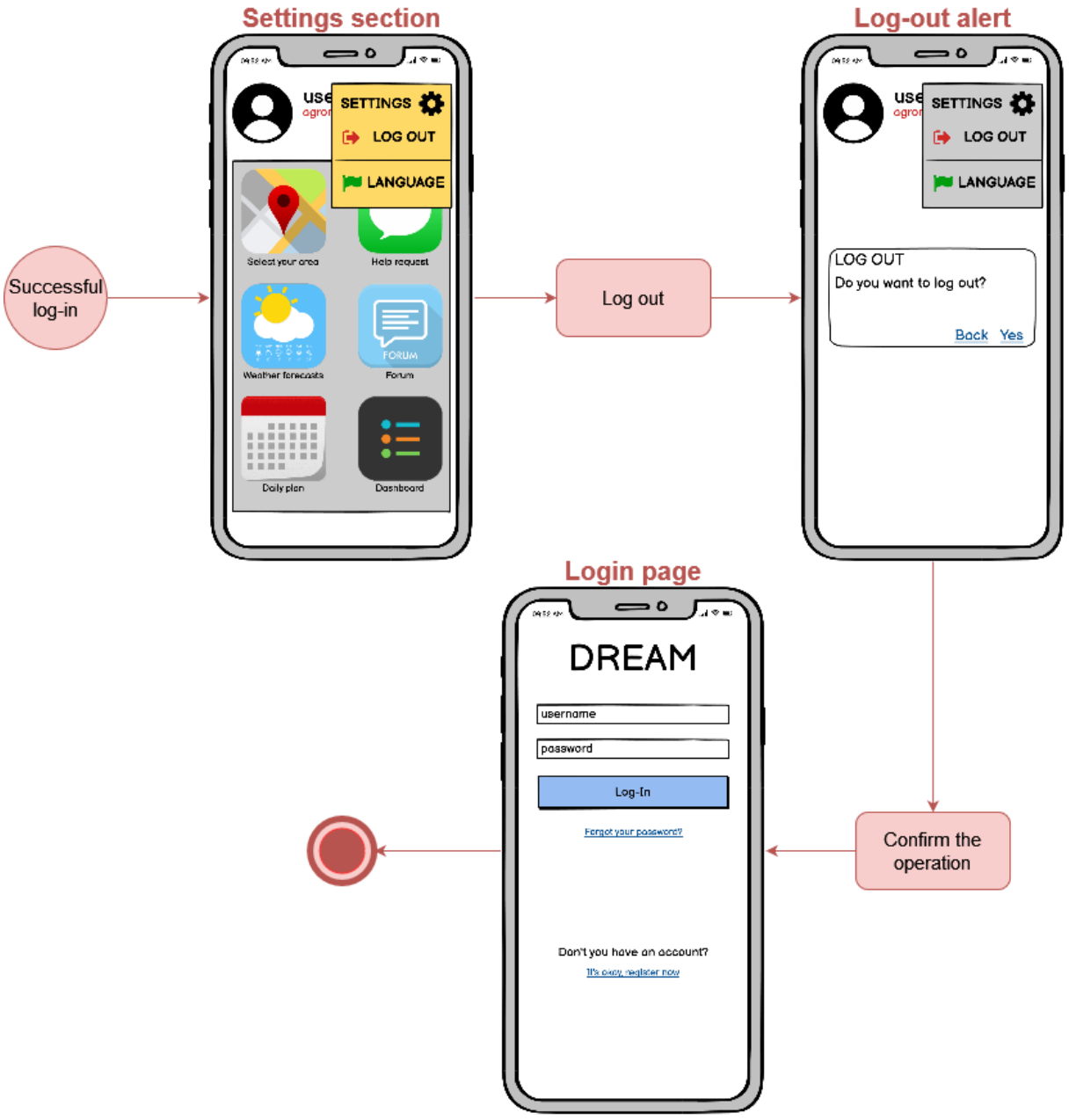


Figure 7: Logout

3.2 Farmer Mobile Interface

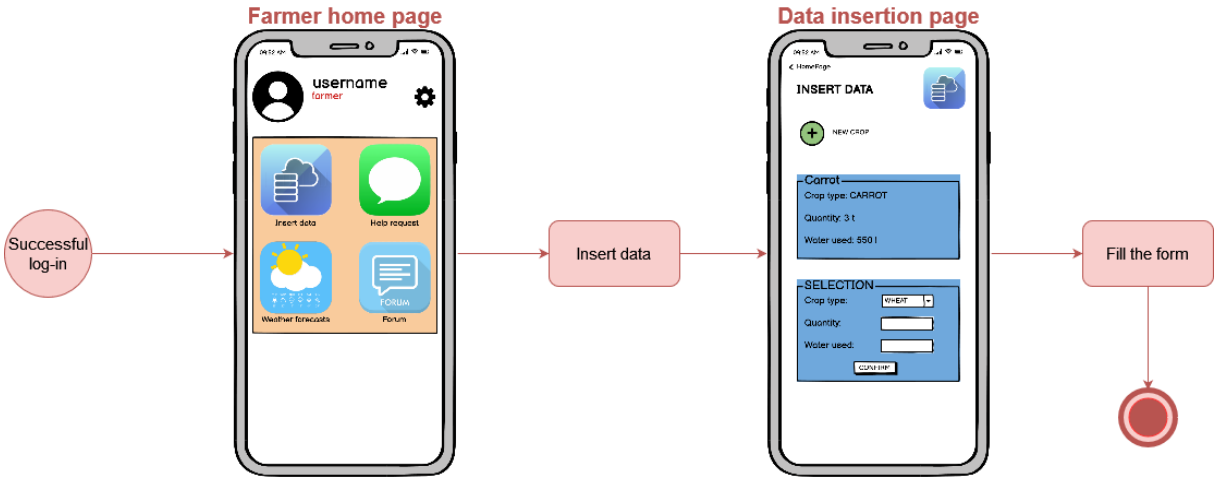


Figure 8: Data insertion

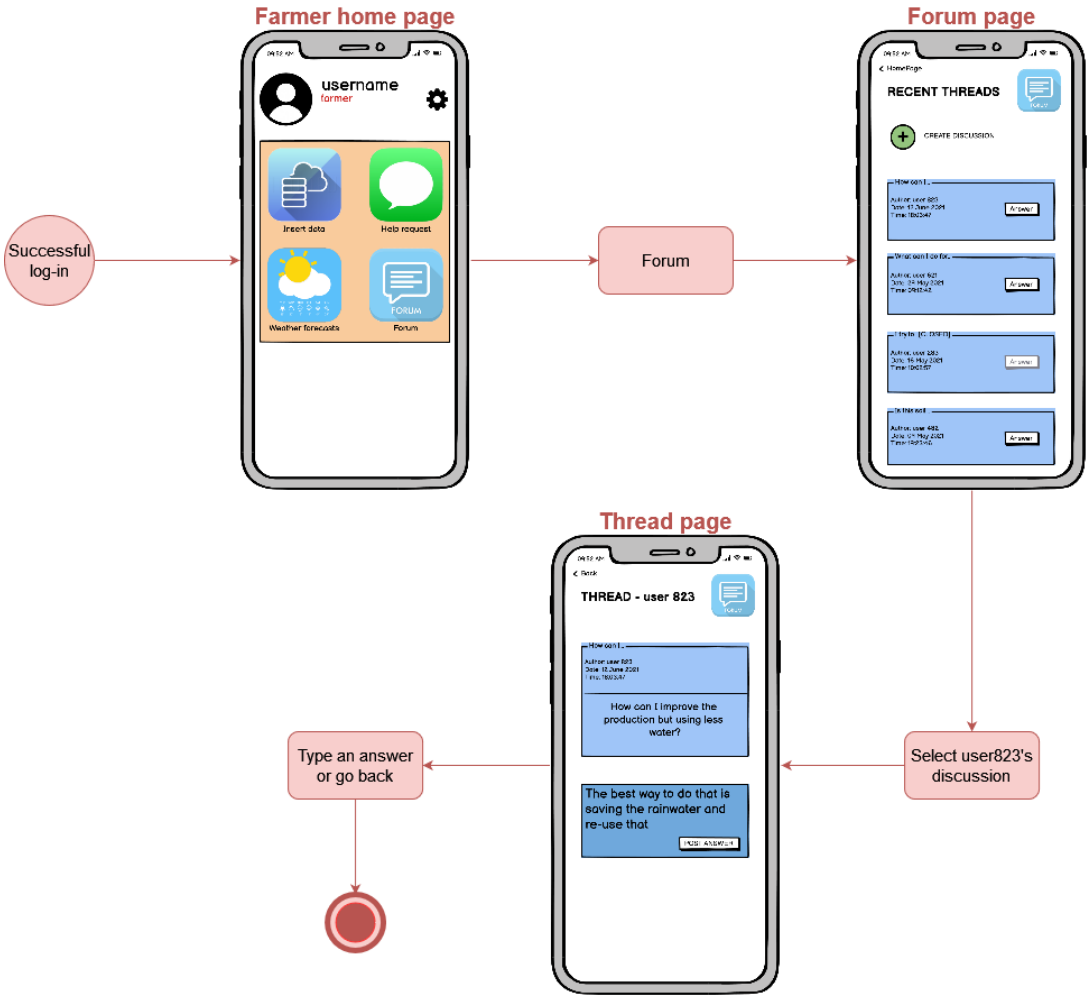


Figure 9: Forum functionalities

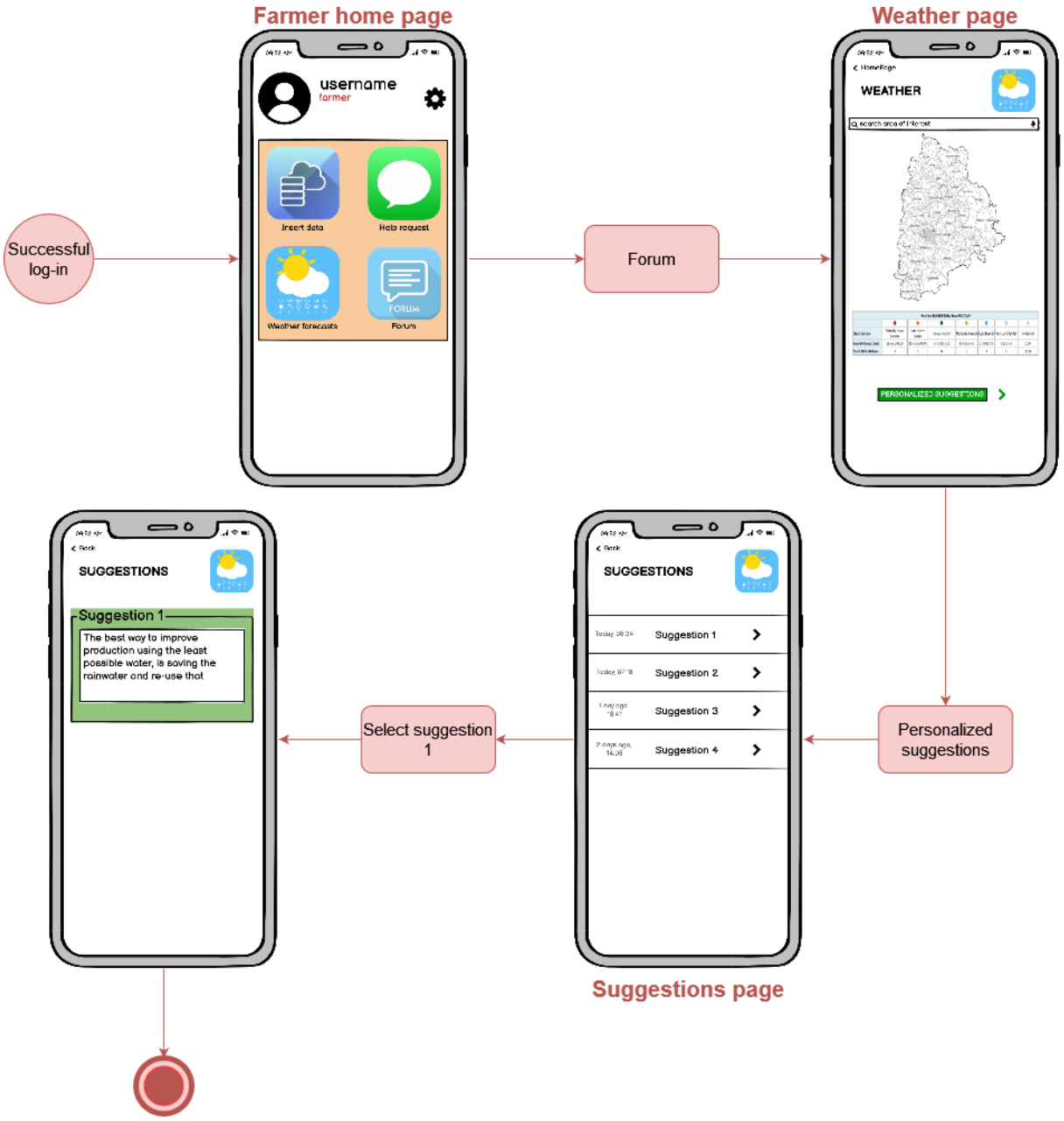


Figure 10: Weather forecasts and personalized suggestions

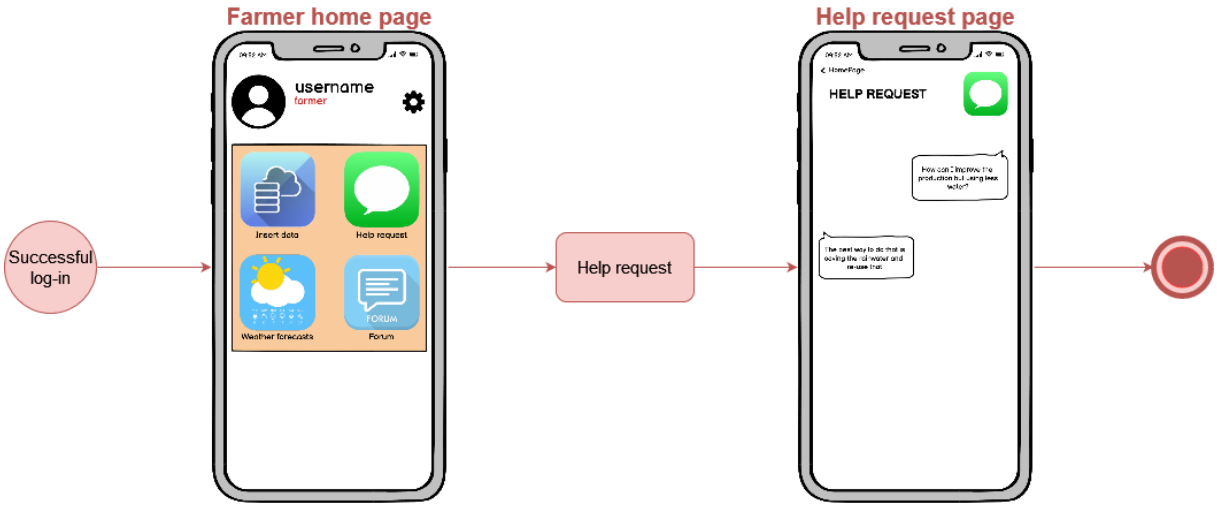


Figure 11: Help request

3.3 Agronomist Mobile Interface

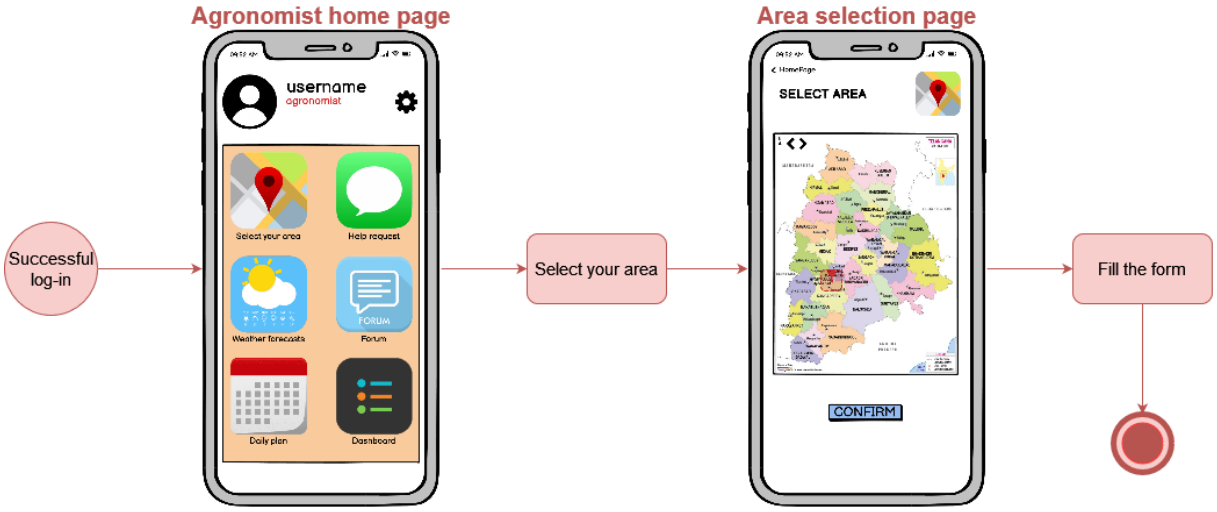


Figure 12: Area of responsibility selection page

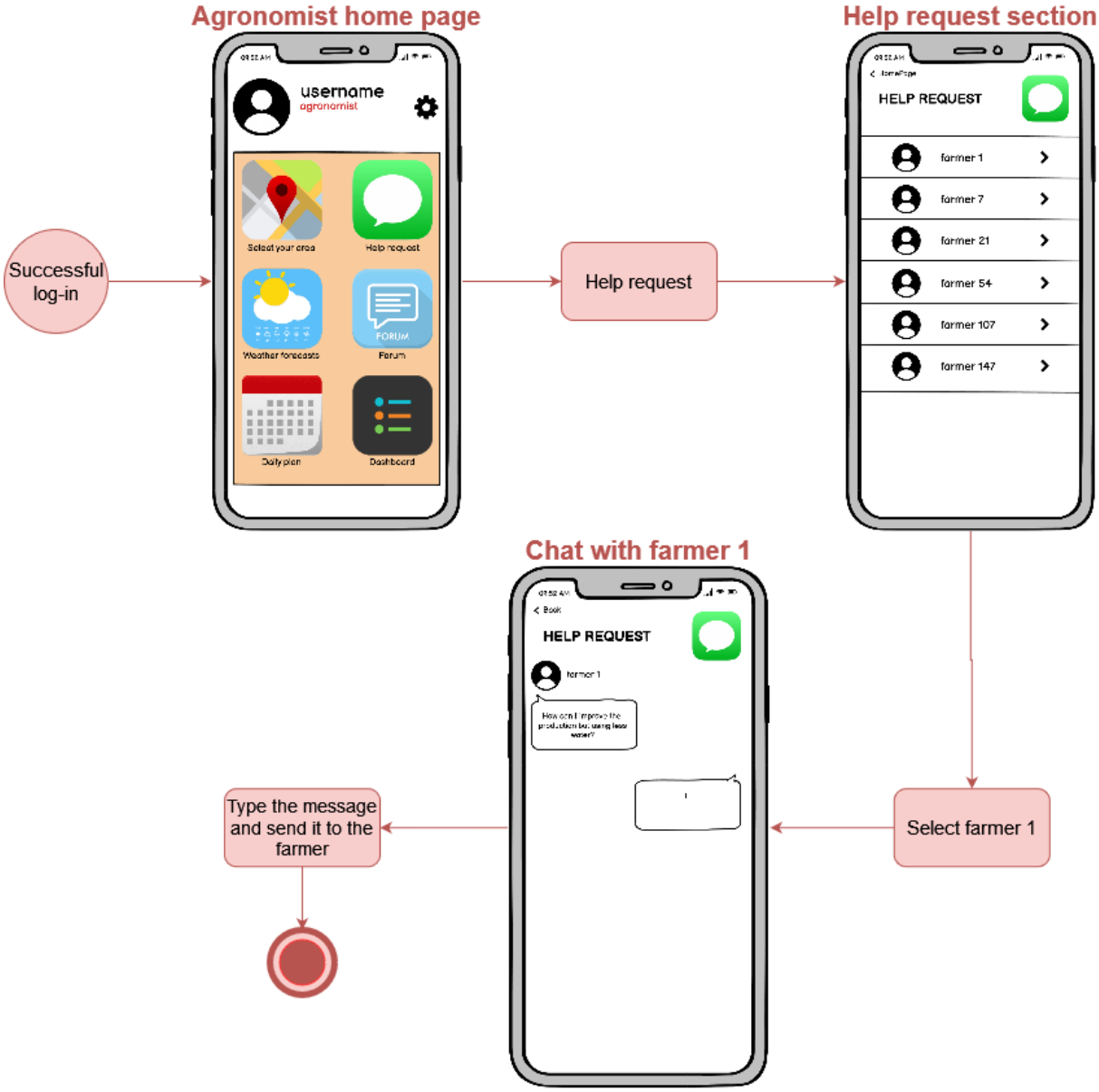


Figure 13: Help request

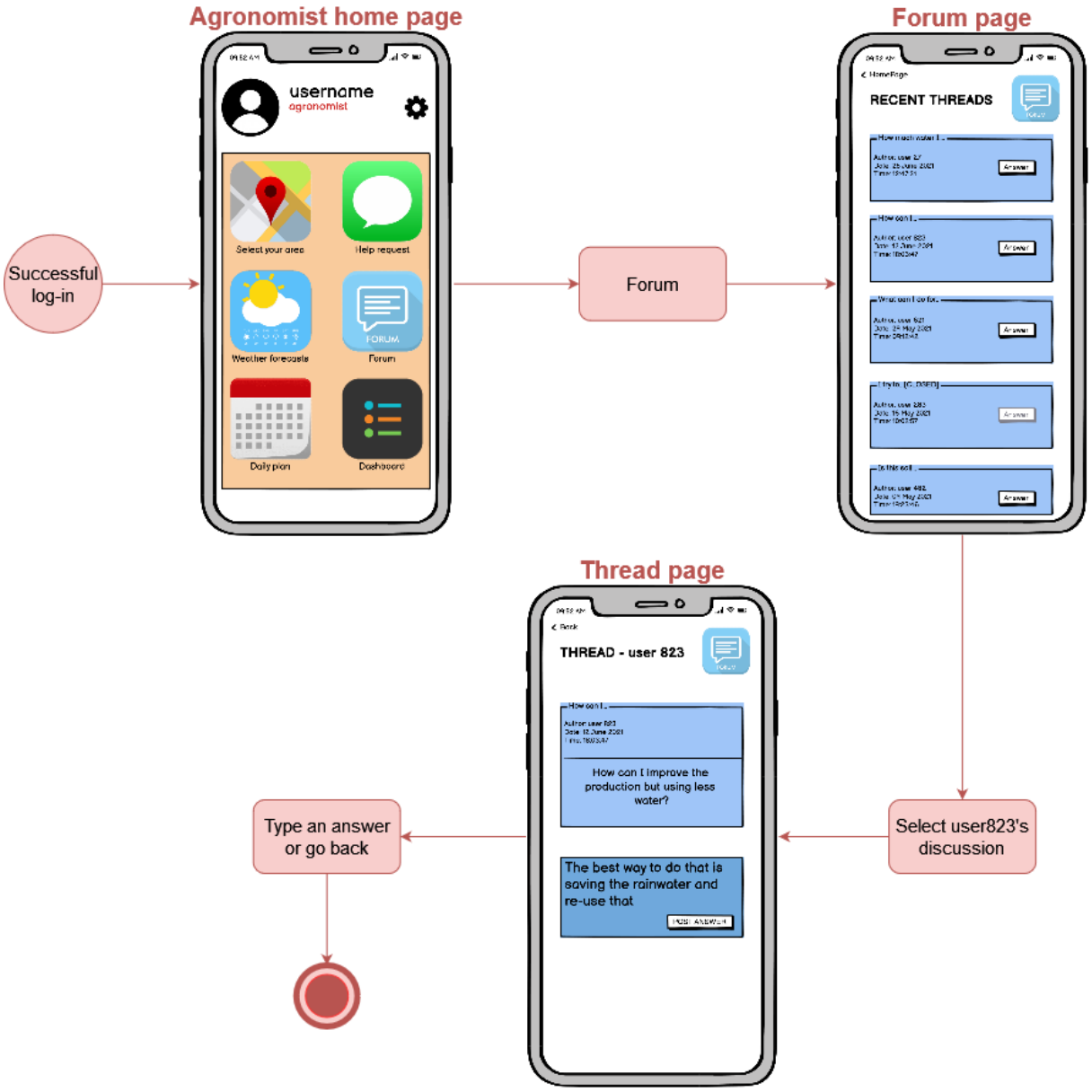


Figure 14: Forum functionalities

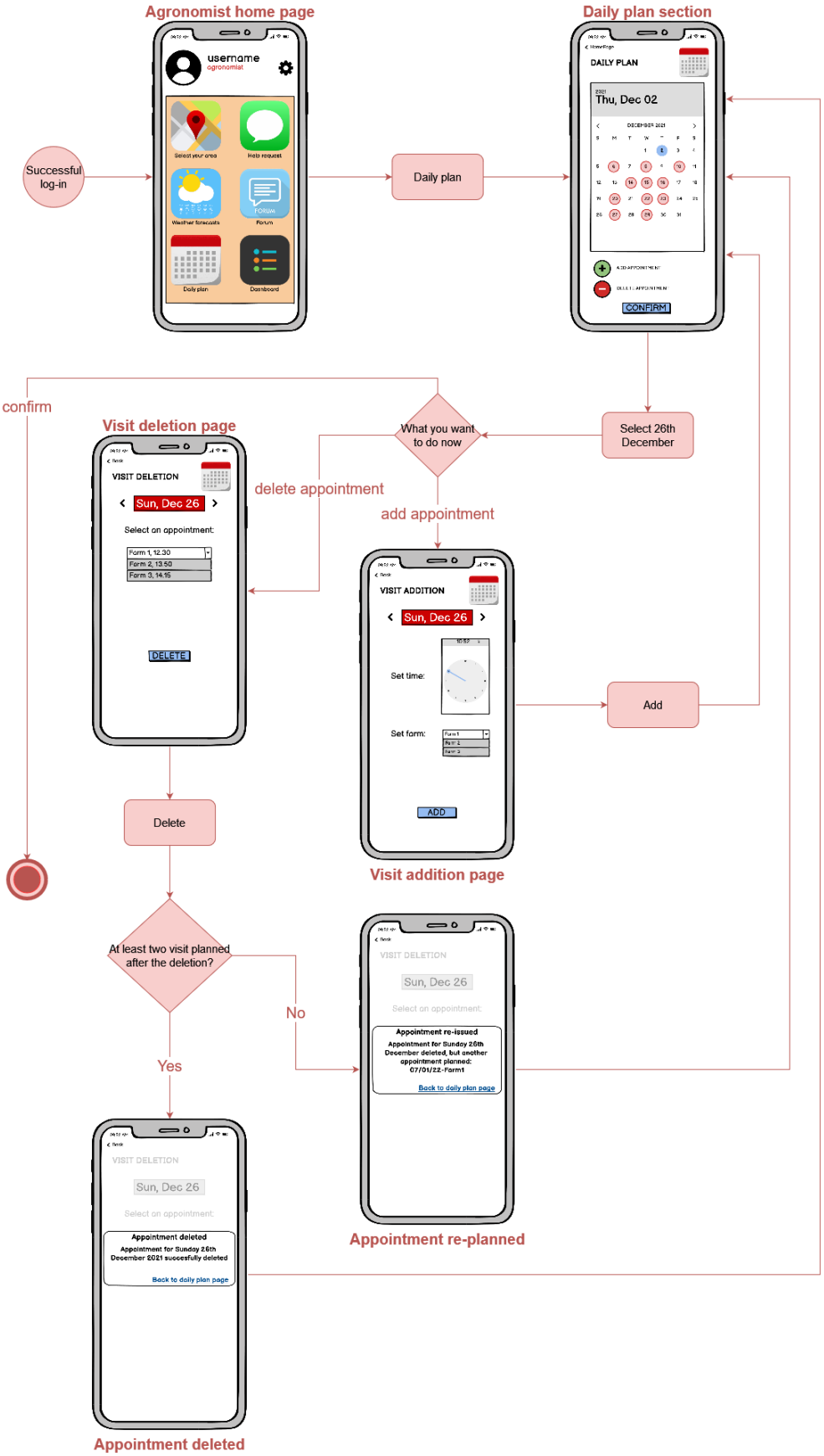


Figure 15: Daily plan functionalities

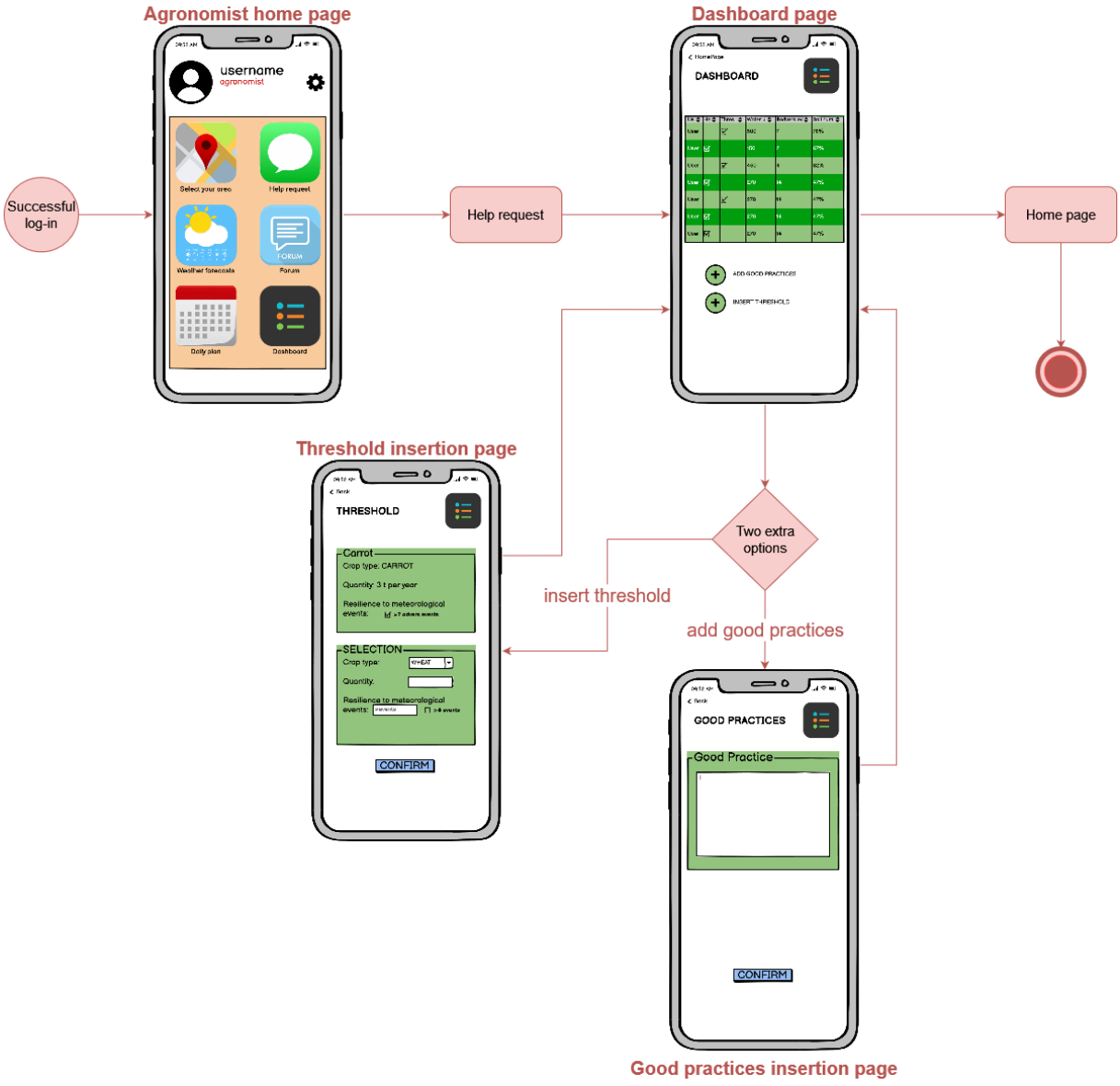


Figure 16: Dashboard

4 Requirements Traceability

This section will show the traceability between requirements and modules described in component diagram.
In the table some abbreviation have been used:

- UM: UserManager
- FM: ForumManager
- CM: CalendarManager
- FS: FarmerSuggestion
- MS: MeteoService

This is the list of requirements:

R1 The system shall keep track of the farmer who requested help and of the suggestion that they received

R2 The system shall keep track of historical data for each zone

R3 The system shall keep track of meteorological events

R4 Farmers shall be able to create a help request for the agronomist specifying the type of problem

R5 The system shall be able to record data about production inserted by the farmer

R6 Agronomist shall be able to insert significant prodution thresholds for each zone

R7 Farmers shall be able to create discussion threads on the forum

R8 Farmers shall be able to reply to discussion on the forum

R9 The system shall send a notification to the Agronomists when a farmer has requested help

R10 After an agronomist has registered a best practice or a suggestion to a farmer, the system should infer the farmers with similar characteristics and send them the same suggestion

R11 The systems shall be able to connect to an external service to retrieve meteorological forecasts

R12 The system shall give Farmers the possibility to see weather forecasts

R13 The system shall be able to manage a forum

R14 The system shall be able to record data about visits of agronomists

R15 The system shall be able to manage a calendar

R16 The system shall organize Agronomists' calendar in such a way that they visit each farm at least twice a year

R17 The system shall give Farmers the possibility to register

R18 The system shall give Farmers the possibility to login

R19 The system shall give the Farmers the possibilty to visualize personalized suggestion

R20 Agronomists shall be able to reply to discussion on the forum

R21 The system shall give Agronomists the possibility to see weather forecasts

R22 Agronomists shall be able to add or remove an appointment from daily plan

R23 At the end of each day Agronomists shall be able to confirm or communicate deviations from the daily plan

R24 The system shall give Agronomists the possibility to register

R25 The system shall give Agronomists the possibility to login

R26 Agronomists shall be able to visualize farmers' performance through the dashboard

R27 Agronomists shall be able to insert best practices discovered during inspections

R28 Agronomists shall be able to visualize disussion on the forum

R29 The system shall be able to keep track of all the areas and the agronomist responsible for them

R30 The system shall give Agronomists the possibility to know every farmer for which he is responsible

R31 The system shall be able to localize and keep track of each farm

R32 The system shall give Policymakers the possibility to register

R33 The system shall give Policymakers the possibility to login

R34 Policymaker shall be able to visualize farmers' performance through the dashboard

R35 Policymaker shall be able to visulize data inserted by farmers and agronomists

R36 Policymaker shall be able to visualize comparative performance of the farmers who received help

R37 The system shall give policymakers the ability to identify well-performing and underperforming farmers

	UM	FM	CM	FS	MS	DBMS
R1	X			X		X
R2					X	X
R3					X	X
R4	X					X
R5	X					X
R6	X					X
R7	X	X				X
R8	X	X				X
R9	X					X
R10	X			X		X
R11	X				X	
R12	X				X	
R13		X				X
R14	X		X			X
R15	X		X			X
R16			X			X
R17	X					X
R18	X					X
R19	X			X		X
R20	X	X				X
R21	X				X	
R22	X		X			X
R23	X		X			X
R24	X					X
R25	X					X
R26	X					X
R27	X			X		X
R28	X	X				X
R29						X
R30						X
R31						X
R32	X					X
R33	X					X
R34	X					X
R35	X					X
R36	X					X
R37	X					X

5 Implementation, Integration and Test Plan

5.1 Recommended Implementation

The system will be composed by:

- Client : it can be a mobile or a web application
- Web Server
- Application Server
- Database
- External APIs (such as OpenWeather and OpenStreetMap)

5.2 Implementation Plan

The above components should be developed following a bottom-up logic, therefore, avoiding difficulties in integration and testing.

Firstly some basic functionalities should be completely developed so that the other related functionalities can then be added and tested.
The first module that should be written is the one that provides authentication and account creation interfacing with the DBMS. Because of that, the system can soon reach partial functionality.
Then can be developed the web application using the REST API which includes all the main functionalities of the users. The next steps to complete the system should be:

- The client interfaces
- The coupling with external APIs

5.3 Testing Plan

Testing should be done throughout all stages of development, following a bottom up approach.

Unit tests should be written starting from the basic component so that tests for other components that use them can be written relying on the fact that the subcomponents are already tested. This speeds up debugging since it's easy to locate at which level of the component hierarchy there is a problem.

5.4 Integration Plan

In this section there is be a description about how the components are integrated and communicate. The first component to build is the DBMS, followed by the main application components that exploit it. After this, we have to integrate the API communication between the System and the external services that will be used. t this point it is possible to integrate the User Manager, which permits to Users to exploit all the functionalities. Finally, it is possible to integrate the web server module with the mobile application module and the browser with the web server module.

6 Effort Spent

Lorenzo Iovine

Hours	Description
2h	Section 1
8h	Section 2
3h	Section 4
2h	Section 5
3h	Review
18h	

Nicola Landini

Hours	Description
2h	Section 1
8h	Section 3
3h	Section 4
2h	Section 5
3h	Review
18h	

Francesco Leone

Hours	Description
2h	Section 1
8h	Section 2
3h	Section 4
2h	Section 5
3h	Review
18h	