POLITECNICO MILANO 1863

Computer Science and Engineering

Software Engineering 2

Academic year 2021-2022

Design Document

DREAM

Data-dRiven PrEdictive FArMing in Telangana

Authors:

Arslan Ali	971503
Elisa Servidio	996387
Federica Suriano	953085

January -, 2022

Version 1.0

Contents

List of Tables **2**

List of Figures **3**

1 Introduction **4**
- 1.1 Purpose . . . 4
- 1.2 Scope . . . 4
- 1.3 Definitions, Acronyms, Abbreviations . . . 4
  - 1.3.1 Definitions . . . 5
  - 1.3.2 Acronyms . . . 6
  - 1.3.3 Abbreviations . . . 6
- 1.4 Revision History . . . 7
- 1.5 Reference Documents . . . 7
- 1.6 Document Structure . . . 8

2 Architectural Design **9**
- 2.1 Overview . . . 9
- 2.2 Component View . . . 12
- 2.3 Deployment View . . . 16
- 2.4 Runtime View . . . 16
- 2.5 Component Interfaces . . . 21
- 2.6 Architectural Design and Patterns . . . 21
- 2.7 Other Design Decisions . . . 21

3 User Interface Design **22**

4 Requirement Traceability **23**

5 Implementation, Integration and Test Plan **24**

6 Effort Spent **25**
  - 6.0.1 Ali Arslan . . . 25
  - 6.0.2 Servidio Elisa . . . 25
  - 6.0.3 Suriano Federica . . . 26

List of Tables

1.1 Definitions 5
1.2 Acronyms 6
1.3 Abbreviations 7
1.4 Revision History 7

6.1 Effort spent - Ali Arslan 25
6.2 Effort spent - Servidio Elisa 25
6.3 Effort spent - Suriano Federica 26

List of Figures

2.1 System Architecture 10
2.2 Component legend 12
2.3 Component diagram 16
2.4 Farmer Registration Sequence Diagram 18
2.5 Farmer Login Sequence Diagram 20

Chapter 1

Introduction

1.1 Purpose

The goal of DD (Design Document) is to provide a more extensive overview of the architectural decisions, their communication interfaces, made to implement all the functionalities described in the RASD document. In the last chapter (5) it will also present implementation, integration and test plan.

1.2 Scope

As stated in the RASD document, the aim of the DREAM software product is to develop and adopt anticipatory governance models for food systems to strengthen data-driven statepolicy. It takes care of the acquisition and management of all data collected in order to support the work of farmers, agronomists and policy makers. The system aims to collect data not only from sensors located throughout the territory, but also from farmers. The analysis of the acquired data aims to improve the production of farmers. Low-performing farmers are identified by policy makers and helped by the best-performing ones. Everything is supervised by agronomists who take care of their own geographical areas of competence.

1.3 Definitions, Acronyms, Abbreviations

In the following section is clarified the meaning of some definitions, acronyms and abbreviations which will be use in the DD, in order to help the general understanding of the document.

1.3.1 Definitions

Table 1.1: Definitions

The system	The whole system to be developed
User	A farmer/agronomist/policy maker who uses the application
Unregistered user	A farmer/agronomist/policy maker who is not yet registered into the application
Application service	Functionality offered by the system for certain users
Policy maker	The user of the application who decides about new policies for Telangana
Farmer	The user of the application who owns or manages a farm
Address	The address inserted by the farmer which corresponds to his/her farm's location. It is composed by city, zip code, street and number
Performance	Indicator of the progress of a farmer's activity up to a certain date. Its value is calculated as numerical score
Score	Performance rating computed by a function that depends on: type and quantity of harvested product, weather conditions, quantity of water consumed, soil moisture
Well performing farmer	A farmer who has a score higher than a certain threshold
Bad performing farmer	A farmer who has a score below a certain threshold
Agronomist	The user of the application dealing with the management of a certain mandal

Mandal	A local government area and administrative division. Telangana is subdivided into districts which are themselves subdivided into mandals
Daily plan	Application service that allows the agronomist to manage his/her daily work schedule. Specifically, it allows him/her to track and organize visits to farmers
Discussion forum	Application service which a farmer can use to exchange ideas and opinions about a topic
Post	Contributions of the participants to the discussion placed one below the other in sequence
Thread	It is composed of the topic followed by the posts left by the various participants in the discussion
Help request	Application service which a farmer can use to request for help to the agronomist or/and other well performing farmers

1.3.2 Acronyms

Table 1.2: Acronyms

DREAM	Data-dRiven PrEdictive FArMing in Telangana
RASD	Requirement Analysis and Specification Document
UML	Unified Modelling Language
API	Application Programming Interface

1.3.3 Abbreviations

Table 1.3: Abbreviations

G.i	i-th goal
R.i	i-th requirement
D.i	i-th domain assumption
UC.i	i-th use case

1.4 Revision History

Table 1.4: Revision History

Version	Date	Authors	Summary
1.0	0-/01/2022	Arslan Ali Elisa Servidio Federica Suriano	First release

1.5 Reference Documents

- Specification document: "R&DD Assignment A.Y. 2021/2022"
- Software Engineering 2 course slides A.Y. 2021/2022 - Politecnico di Milano
- EEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications
- ETH Zürich (2009), Requirements Specification: `se.inf.ethz.ch/courses/2011a_spring/soft_arch/exercises/02/Requirements_Specification.pdf`
- UML offcial specification `https://www.omg.org/spec/UML/`

1.6 Document Structure

The DD is structured in the following five chapters:

- **Chapter 1 - *Introduction*:**
- **Chapter 2 - *Overview*:**
- **Chapter 3 - *User Interface Design*:**
- **Chapter 4 - *Requirement Traceability*:**
- **Chapter 5 - *Implementation, Integration and Test Plan*:**
- **Chapter 6 - *Effort Spent*:**

Chapter 2

Architectural Design

In this chapter are reported the main architectural design choices from a high level point of view. In the following sections are presented the chosen paradigm of the system and its components, followed by the description of how the system will be deployed. Moreover, components' sequence and interface diagrams are listed to describe the way components interact to accomplish specific tasks of the application. Eventually, the last two sections provide specifications regarding the selected architectural style and other design choices.

2.1 Overview

The software to be developed is a distributed application with a client-server multi-tiered architecture.
The general architecture of the system can be subdivided into the three logic layers:

- *Presentation Layer*: it handles the interaction with all kinds of user. It provides an interface to users accessing both the mobile application and the web application allowing them to exploit DREAM's services in the most efficient and intuitive way.
- *Business logic or Application Layer*: it handles the functions to be provided for the users. It also manages communication between the end user interface and the database.
- *Data Layer*: it manages the persistent storage and retrieval of data accessing to the database.

The three logic software layers are built on top of a four tier architecture, which consists in four different hardware tiers that represent a machine (or a group of machines):

- *Client tier*: it consists of the client programs that are used to exploit system's services and the devices where those programs are installed, which in this case are pc for the web application and smartphone for the mobile applications.
- *Web tier*: it consists of components that handle the interaction between client tier and the business tier. It contains Web Servers. It dynamically generates content in various formats for the client tier from which it collects inputs and return appropriate results from the components in the business tier.
- *Business tier*: it processes all dynamic content and the interactions between the Client/Web Tier and the Data tier. It contains Application Servers.
- *Data tier*: runs the DBMS and holds all of the sensitive application data. It contains database servers.

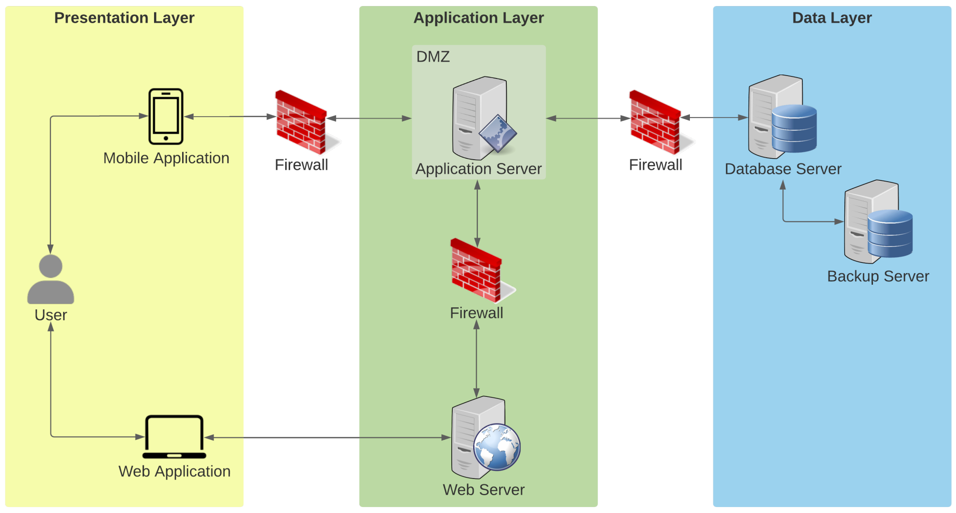

Figure 2.1: System Architecture

The Application Server is the core of the system's architecture as it is the central point between all the other components. It holds the business logic of the system with Web Server. Web and Application servers communicate to each other. It has persistent connection with Data tier and by communicating with its DBMS it can handle all the operations of reading, writing and uploading of data.

The service is supposed to be accessed through both a web application and a mobile application, and this is valid for all kinds of users. Users who access through DREAM web application can communicate with the Web Server while users logged into the mobile application communicate directly with the Application Server.

To guarantee high level of security due to the sensitivity of treated data, firewalls are installed before and after the Application tier to create a Demilitarized Zone (DMZ) for the application servers so that the external network can access only to the resources exposed in the DMZ.
Moreover, the Database Server periodically stores critical data in the Backup Server.

The described architecture has been chosen because tiering ensures the ability to scale applications, guaranteeing the system scalability and flexibility to future integrations and modifications. The servers that constitute each tier in fact may differ both in capability and number. If needed, it can be possible to add more servers (horizontal scaling or scaling out) or more powerful servers (vertical scaling or scaling up) to each singular tier.

2.2 Component View

The purpose of this section is to show the component diagram which is intended to represent the internal structure of the modeled software system in terms of its main components and the relationships among them.

The following legend is used:

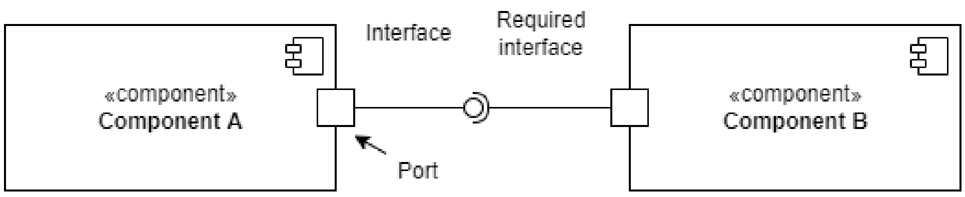

Figure 2.2: Component legend

All components within the diagram are described in detail below.

- **WebApplication**: it manages all user interactions with the web application and associates them with the required functions.
- **MobileApplication**: it manages all user interactions with the mobile application and associates them with the required functions.
- **AgronomistService**: This component includes the following subcomponents.
  - **HelpResponseService**: it allows an agronomist to manage the response to a help request received
  - **MandalMapService**: it retrieves and displays on a mandal map relevant personal data of farmers and their performance score to the agronomist responsible of the mandal. This component is closely linked to the TableService component, which is called up as soon as the agronomist selects a farmer of interest on the map. Obviously, in order to collect all the data of interest, the component must be connected to the DatabaseAccess component. Although each farmer's location is saved in the local database, external APIs must again be used to place the farmers anywhere on the map. This is why this component is connected to GoogleMapsAPI component.
  - **TableService**: it retrieves and displays the table of farmers' performance in ascending order based on their performance score. Obviously, in order to collect all the data of interest, the component must be connected to the DatabaseAccess component.

  - **DailyPlanService**: it deals with the creation and updating of the daily plan of each agronomist. Specifically, it guarantees that for each farm there are at least two visits per year by the responsible agronomist, also after changes made by the agronomist to the plan. To retrieve the scheduled plan, this component must be connected with DatabaseAccess component.
  - **WeatherForecastsService**: it collects the data related to the weather forecasts in the mandal of a certain agronomist starting from the current day and up to the following seven days. To do this, it needs to be connected to the external API TelanganaGovernmentService as well as to the DatabaseAccess component. These data are placed on the appropriate mandal map thanks to an external API component: GoogleMapsAPI.
  - **NotificationAgronomistService**: it deals with managing all notifications addressed to a specific agronomist, in particular whenever a help request is created in the mandal for which he/she is responsible, a notification is sent to the agronomist. In order to retrieve the notification of interest, this component is connected with the DatabaseAccess component.

- **FarmerService**: This component includes the following subcomponents.

  - **HelpRequestService**: it is necessary for the creation and subsequent management of help requests by the farmer. The connection to DatabaseAccess component allows it to retrieve the recipients to whom the request can be sent (mandal agronomist and well performing farmers).
    In addition, if a farmer who is well performing has received a request for help from another farmer, this component takes care of managing the function that allows the farmer to respond to the request.
  - **DiscussionForumService**: it manages the creation of posts by a farmer on the discussion forum, the creation of threads in response to a specific post (always by other farmers) and the search by topic of posts already created. It also performs these functions thanks to the connection to the DatabaseAccess component.
  - **PerformanceService**: it takes care of updating a farmer's performance data every time he/she enters new data about his/her crop. Not only is it necessary to connect to the DatabaseAccess, but also to external APIs to retrieve the following data: soil moisture (retrieved from CopernicusClimateDataStoreService) and quantity of water consumed (retrieved from TelanganaWaterIrrigationService).

  - **VisitService**: it retrieve the visits that each farmer has from the agronomist in charge of his/her mandal. To retrieve them from the database, this component must be connected to the DatabaseAccess component.
  - **WeatherConditionService**: it collects the data related to the weather conditions in the position of a certain farmer's farm for the current day. To do this, it needs to be connected to the external API TelanganaGovernmentService as well as to the DatabaseAccess component.
  - **SuggestionsService**: it takes care of calculating the suggestions of a farmer based on his/her relevant information regarding his/her production and the position of his/her farm, data obtained thanks to the connection to the DatabaseAccess component.
  - **NotificationFarmerService**: it deals with managing all notifications addressed to a specific farmer, in particular he/she receives notifications based on personalized suggestions, scheduled visits, help requests, replies to posts created by him/her. In order to retrieve the notification of interest, this component is connected with the DatabaseAccess component.

- **PolicyMakerService**: This component includes the following subcomponents.

  - **TimeChartService**: it computes and displays Telangana's farmers data in a graph so that the policy maker can analyze the various parameters of interest over time.
  - **MapService**: it computes and displays Telangana's farmers performance map using some filter. In order to collect all the data of interest, the component must be connected to the DatabaseAccess component.

- **AuthenticationService**: it is necessary for every type of user who wants to access the application (both from the mobile app and the web app) and use it according to the permissions he/she possesses, therefore according to the role with which he/she is registered.

- **EmailManager**: it is necessary when a user who has not yet registered wants to register. To verify his/her email during registration, the system automatically sends a confirmation email to the user who wants to register and activates a 10-minute timer within which the user must confirm his/her registration.

- **DatabaseAccess**: it is responsible to get data from a data source. In fact, it manages the access to the database.

- **Database**: This component includes the following subcomponent.
  - **DBMS**: Database Management System refers to a software that allow users to access databases and manipulate, maintain, report, and relate data. It is used to reduce data redundancy, share data in a controlled manner, and reduce data integrity issues.
- **ExternalAPIs**: This component includes the following subcomponents.
  - **GoogleMapsAPI**: it allows the connection of the application to Google Maps APIs in order to retrieve the position (latitude and longitude) of farms' address inserted by farmers and show it on the mandal map of the agronomist. More generally, all functions that require access to geographic data not yet saved in the internal database, will rely on this component.
  - **CopernicusClimateDataStoreService**: it is responsible for retrieving soil moisture data in the Telangana region, which is needed when a farmer wants to enter data on his/her crop. Based on the position of the farmer's farm, the system takes care of adding the soil data.
  - **TelanganaGovernmentService**: it takes care of retrieving data relating to weather conditions relating to the position of the farmer's farm and it also has the purpose of obtaining the weather forecasts related to each mandal for which an agronomist is responsible. It also accesses ID codes' DB managed by Telangana government to check validity of ID code inserted by policy makers and agronomist while registering into DREAM.
  - **TelanganaWaterIrrigationService**: it is responsible for retrieving the amount of water consumed by each farmer in the Telangana region, which is needed when a farmer wants to enter data on his/her crop. Based on the position of the farmer's farm, the system takes care of adding the amount of water consumed by each farmer.

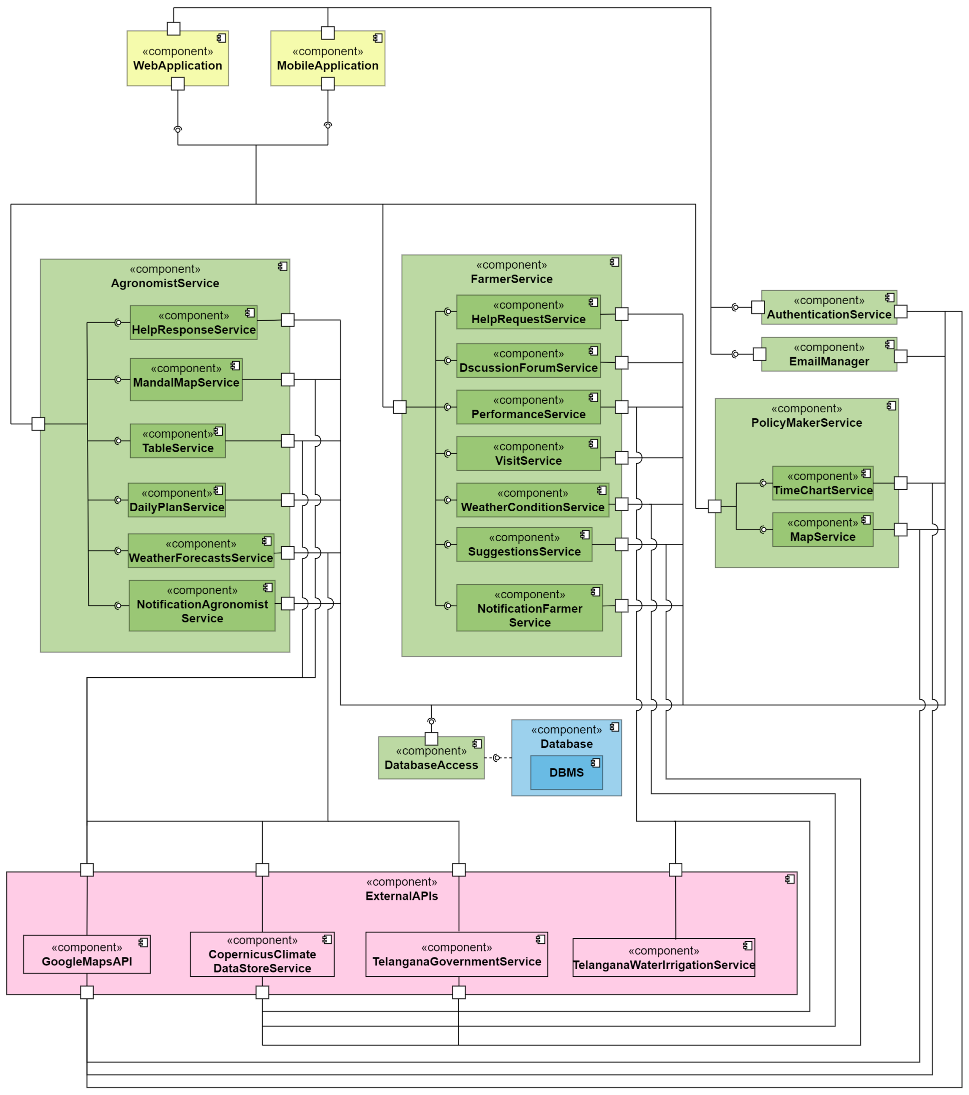
«component» WebApplication
«component» MobileApplication
«component» AgronomistService
«component» HelpResponseService
«component» MandalMapService
«component» TableService
«component» DailyPlanService
«component» WeatherForecastsService
«component» NotificationAgronomistService
«component» FarmerService
«component» HelpRequestService
«component» DscussionForumService
«component» PerformanceService
«component» VisitService
«component» WeatherConditionService
«component» SuggestionsService
«component» NotificationFarmerService
«component» AuthenticationService
«component» EmailManager
«component» PolicyMakerService
«component» TimeChartService
«component» MapService
«component» DatabaseAccess
«component» Database
DBMS
«component» ExternalAPIs
«component» GoogleMapsAPI
«component» CopernicusClimateDataStoreService
«component» TelanganaGovernmentService
«component» TelanganaWaterIrrigationService

Figure 2.3: Component diagram

2.3 Deployment View

2.4 Runtime View

TO DO LIST (the green ones can go in the same sequence. the orange ones are done)

1. farmer registers SERVICE: authentication - email API: Google

2. farmer logs in SERVICE: authentication + farmer visualizes relevant information (weather conditions and visits) SERVICE: weather conditions - visit API: weather, moisture
3. farmer make a help request (+notification of new help request received to agronomist) SERVICE: help request (farmer) - notification
4. agronomist replies to help request (+notification of new help response received to farmer) SERVICE: help request (agronomist) - notification
5. farmer solves a help request SERVICE: help Request (farmer)
6. farmer insert data (+retreive data and computation performance score) SERVICE: harvest service API: moisture, irrigation
7. agronomist visualizes maps and table in the homepage SERVICE: map - table API: google maps
8. policy maker visualizes maps and table in the homepage SERVICE: map - time chart API: google maps
9. agronomist updates daily plan (+ notification to farmer) SERVICE: Daily Plan + notification
10. agronomist confirms daily plan SERVICE: Daily Plan
11. agronomist visualizes weather forecasts SERVICE: weather forecasts API: google maps, weather, soil moisture
12. farmer opens a thread on discussion forum SERVICE: discussion forum
13. farmer replies to a thread on the discussion forum (+ notification of new post received to farmer who made the thread) SERVICE: discussion forum
14. system sends suggestions to farmer (they depend on performance, weather and moisture) SERVICE: suggestion (for the computation) + notification API: weather - moisture

Farmer Registration

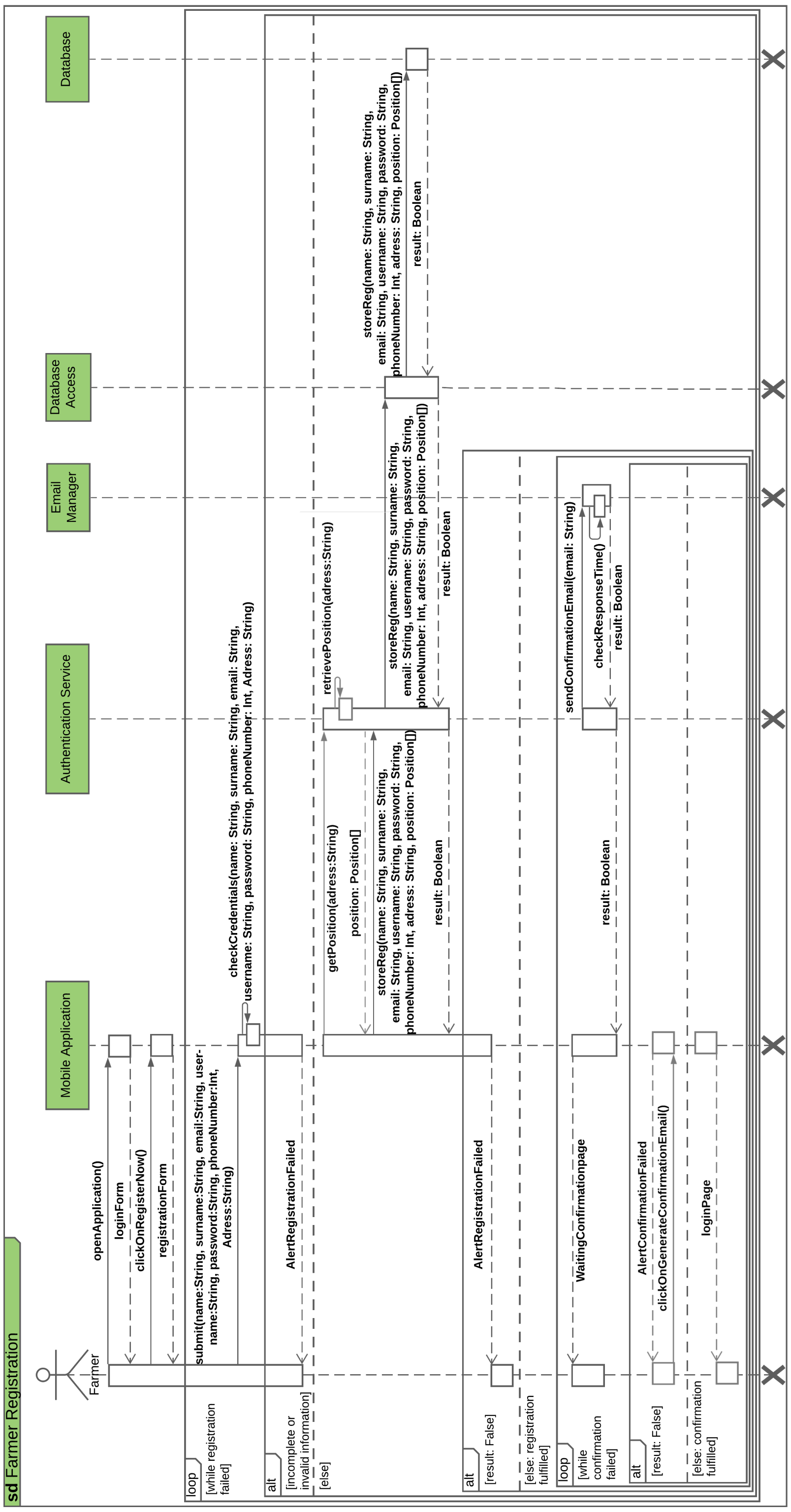
sd Farmer Registration
Farmer
Mobile Application
Authentication Service
Email Manager
Database Access
Database
openApplication()
loginForm
clickOnRegisterNow()
registrationForm
submit(name:String, surname:String, email:String, username:String, password:String, phoneNumber:Int, Adress:String)
checkCredentials(name: String, surname: String, email: String, username: String, password: String, phoneNumber: Int, Adress: String)
AlertRegistrationFailed
getPosition(adress:String)
retrievePosition(adress:String)
position: Position[]
storeReg(name: String, surname: String, email: String, username: String, password: String, phoneNumber: Int, adress: String, position: Position[])
result: Boolean
AlertRegistrationFailed
sendConfirmationEmail(email: String)
checkResponseTime()
WaitingConfirmationpage
AlertConfirmationFailed
clickOnGenerateConfirmationEmail()
loginPage
loop
[while registration failed]
alt
[incomplete or invalid information]
[else]
[result: False]
[else: registration fulfilled]
[while confirmation failed]
[else: confirmation fulfilled]

Figure 2.4: Farmer Registration Sequence Diagram

Farmer Login

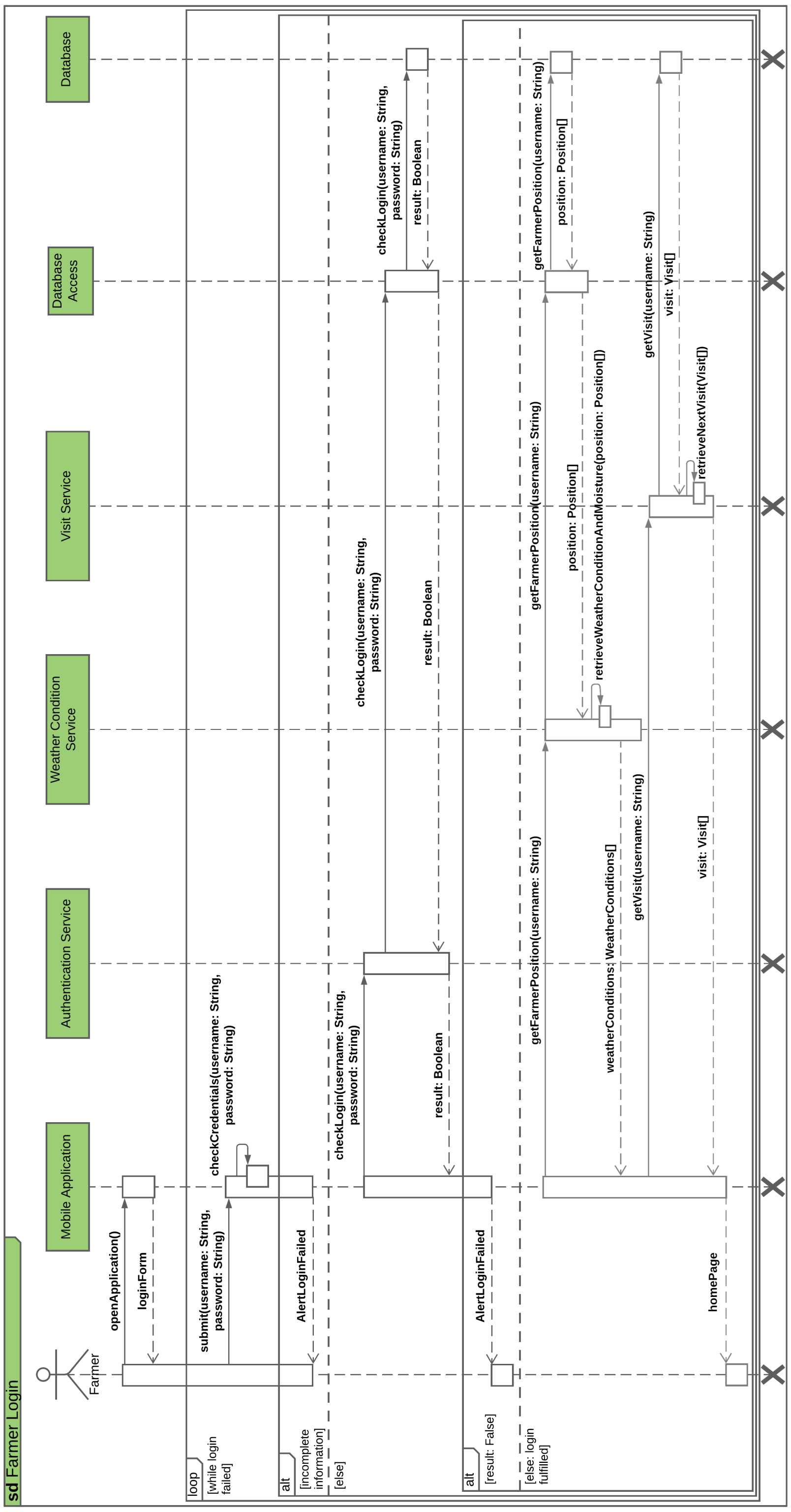
sd Farmer Login
Farmer
Mobile Application
Authentication Service
Weather Condition Service
Visit Service
Database Access
Database
openApplication()
loginForm
loop
[while login failed]
submit(username: String, password: String)
checkCredentials(username: String, password: String)
alt
[incomplete information]
AlertLoginFailed
[else]
checkLogin(username: String, password: String)
checkLogin(username: String, password: String)
checkLogin(username: String, password: String)
result: Boolean
result: Boolean
result: Boolean
alt
[result: False]
AlertLoginFailed
[else: login fulfilled]
getFarmerPosition(username: String)
getFarmerPosition(username: String)
getFarmerPosition(username: String)
position: Position[]
position: Position[]
retrieveWeatherConditionAndMoisture(position: Position[])
weatherConditions: WeatherConditions[]
getVisit(username: String)
getVisit(username: String)
visit: Visit[]
retrieveNextVisit(Visit[])
visit: Visit[]
homePage

Figure 2.5: Farmer Login Sequence Diagram

2.5 Component Interfaces

2.6 Architectural Design and Patterns

2.7 Other Design Decisions

Chapter 3

User Interface Design

Chapter 4

Requirement Traceability

Chapter 5

Implementation, Integration and Test Plan

Chapter 6

Effort Spent

6.0.1 Ali Arslan

Table 6.1: Effort spent - Ali Arslan

Task	Hours
Introduction	
User Interface Design	
Implementation	
Integration	
Test Plan	
Document revision	

6.0.2 Servidio Elisa

Table 6.2: Effort spent - Servidio Elisa

Task	Hours
Overview	

Runtime View	
Requirement Traceability	
Document revision	

6.0.3 Suriano Federica

Table 6.3: Effort spent - Suriano Federica

Task	**Hours**
Component View	5.00
Deployment View	
Requirement Traceability	
Document revision	