POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

DREAM - Data-dRiven PrEdictive FArMing in Telangana

DD
Software Engineering 2

TEAM MEMBERS:

Simone Brunello - 10831750
Nicholas Nicolis - 10867841

Academic Year: 2021-22

Contents

Contents **i**

1 INTRODUCTION **1**
1.1 Purpose . . . 1
1.2 Scope . . . 1
1.3 Definitions . . . 2
1.4 Document Structure . . . 2

2 ARCHITECTURAL DESIGN **4**
2.1 Overview . . . 4
2.2 Component view . . . 5
2.2.1 High level component diagram . . . 5
2.2.2 Farmers services . . . 6
2.2.3 Policy maker services . . . 7
2.2.4 Main Logic . . . 8
2.3 Deployment view . . . 9
2.3.1 Recommended Deployment . . . 10
2.4 Runtime view . . . 10
2.4.1 Sequence diagrams . . . 10
2.5 Component interfaces . . . 13
2.6 Selected architectural styles and patterns . . . 15
2.6.1 Architectural Style . . . 15
2.6.2 Patterns . . . 15
2.7 Other design decisions . . . 16

3 USER INTERFACE DESIGN **17**

4 REQUIREMENTS TRACEABILITY **18**

5 IMPLEMENTATION, INTEGRATION AND TEST PLAN 19

6 EFFORT SPENT 20

7 REFERENCES 21

1 | INTRODUCTION

1.1. Purpose

This document is called Design Document, its purpose is to describe an overall view to the architecture of the software product already discussed in the RASD.
It contains a general description of the structure chosen to design the DREAM system. In addiction to this there are a representation of the UI and a list of the architectural components present in this document. It also contains a set of design characteristics required for the implementation by introducing constraints, quality attributes and it gives a presentation of the implementation, integration and testing plan.
The DD will be given to the development team indeed it has the purpose to guide them through the entire production process.

1.2. Scope

In order to manage more efficiently the communication between farmers and policy makers our DREAM application will provide an easy way to access the system which will make available to all different users a dedicate set of tools and information.

Farmers will be able to monitor weather conditions, crops and fertilized suggestions. They will have the possibility to send direct requests to expert or other farmers in order to receive advice. The ease of communicating their production data and problems will be a key point.

Telangana's policy makers will be able to monitor farmers performances and decide if current policies are providing good results. They will also be supported in the visualization of critical situations in order to intervene in advance.

1.3. Definitions

Definition	Description

Table 1.1: Table of Definitions

Acronyms	Description
JEE	
RASD	
DD	
API	
EIS	
EJB	
JSP	
DBMS	

Table 1.2: Table of Acronyms

Abbreviations	Description

Table 1.3: Table of Abbreviations

1.4. Document Structure

This document is composed of six chapters:

1. **Chapter 1: Introduction.** This part contains the scope the purpose of the DD document. This chapter also includes the structure of the document and a set of abbreviations , acronyms definitions used.

2. **Chapter 2: Architectural Design.** This part contains the architectural design choices , there are an overview of the designed architecture , all the components ,

the interfaces , and the technologies used for the design of the application. This chapter also includes the functions of the interfaces and the process in which they are utilized. At the end there is an explanation of the design pattern chosen to develop the application.

3. **Chapter 3: User Interface Design.** This part contains a representation of how the UI should look like. Tt completes the User Interfaces subsection present inside the RASD.

4. **Chapter 4: Requirements Traccability.** This part contains a description of how the requirements defined in the RASD map to the design elements described in this document.

5. **Chapter 5: Implementation, Integration and Test Plan.**This part contains the plan which describes how to implement the subcomponents of the system and the order of implementation. In addiction to this there is also the test planning for the system.

6. **Chapter 6: Effort Spent.** In this part there are information about how many hours each member of the group spent for every part of this document.

7. **Chapter 1: References.** Here there is just a list of the references used to complete this document.

2 | ARCHITECTURAL DESIGN

2.1. Overview

The application which is going to be implemented is a distributed one and it will has three logic software levels: presentation, application and data.

1. **Presentation:** It includes the components which have the aim to provide the final user interfaces. If the user logs in through the mobile application the interfaces are provided internally by the application. If they logs in through the web browser the interfaces are requested and then provided by the DREAM web server.

2. **Application:** It includes all the components used in order to define the business logic of the DREAM application. All the computations are located here.

3. **Data:** It includes all the parts which implement the storage and the management of data. So it is composed by DBMS and the database. In order to avoid losses of data there is a backup database. Part of the data are not stored internally, but retrieved from external APIs.

In order to to guarantee the security of sensible data, firewalls and an external authentification server have been designed.
Since this system will be used by a huge audience, cache servers will help to reduce the traffic through the main server. It's possible to add also a load balancer in case of necessity.
The application level also interacts with external APIs, which are represented as external servers like wheather server, map server, land registry server, news server and archive server.
This architecture is called three-tier which allows to implement specific characteristics as maintainability, scalability and reliability. Different hardware machines will be used for each tier.

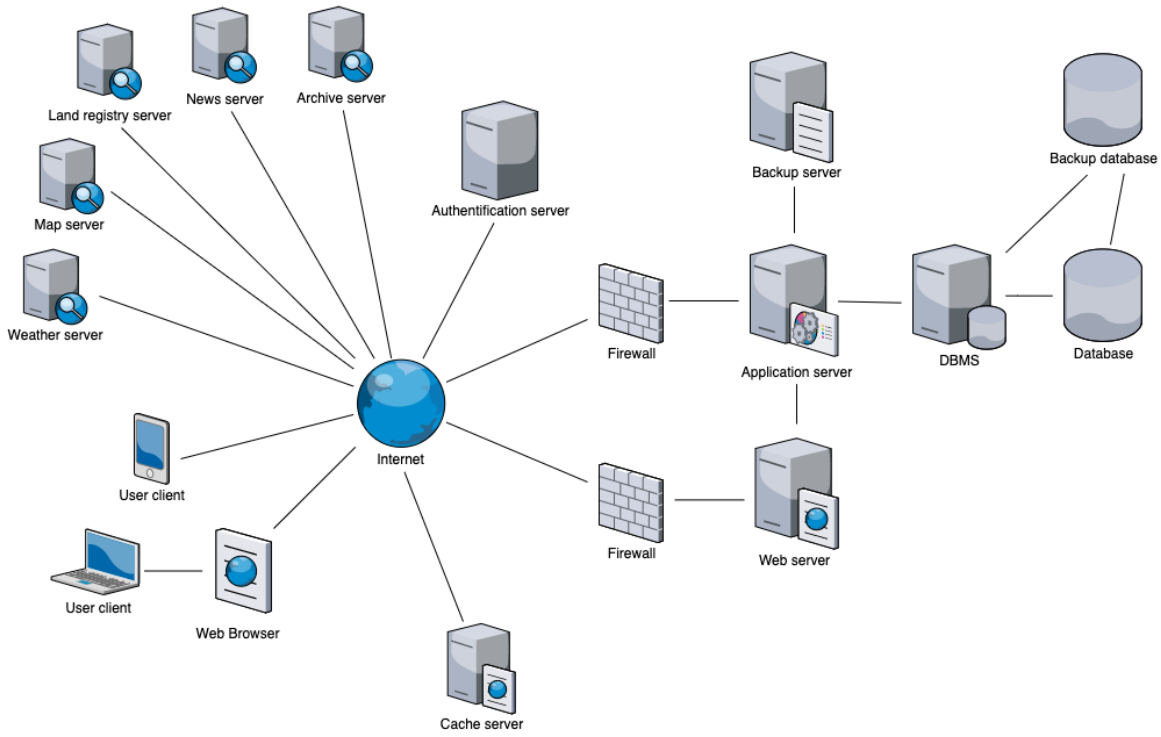

Figure 2.1: high-level architectural overview

2.2. Component view

2.2.1. High level component diagram

The following diagram explain how the main components of the system interact with the interfaces.

2.2.2. Farmers services

The following diagram shows how the subsystem Farmer Mobile Services is built and how its modules communicate with the external ones.
The subsystem has 4 modules, and each one provides to the application some interfaces.

1. **Ticket Module:** send_ticket, read_ticket_answer
2. **Report Module:** send_report, read_answer
3. **Forum Module:** post_on_thread, read_thread, close_thread
4. **Archive Module:** find_in_archive

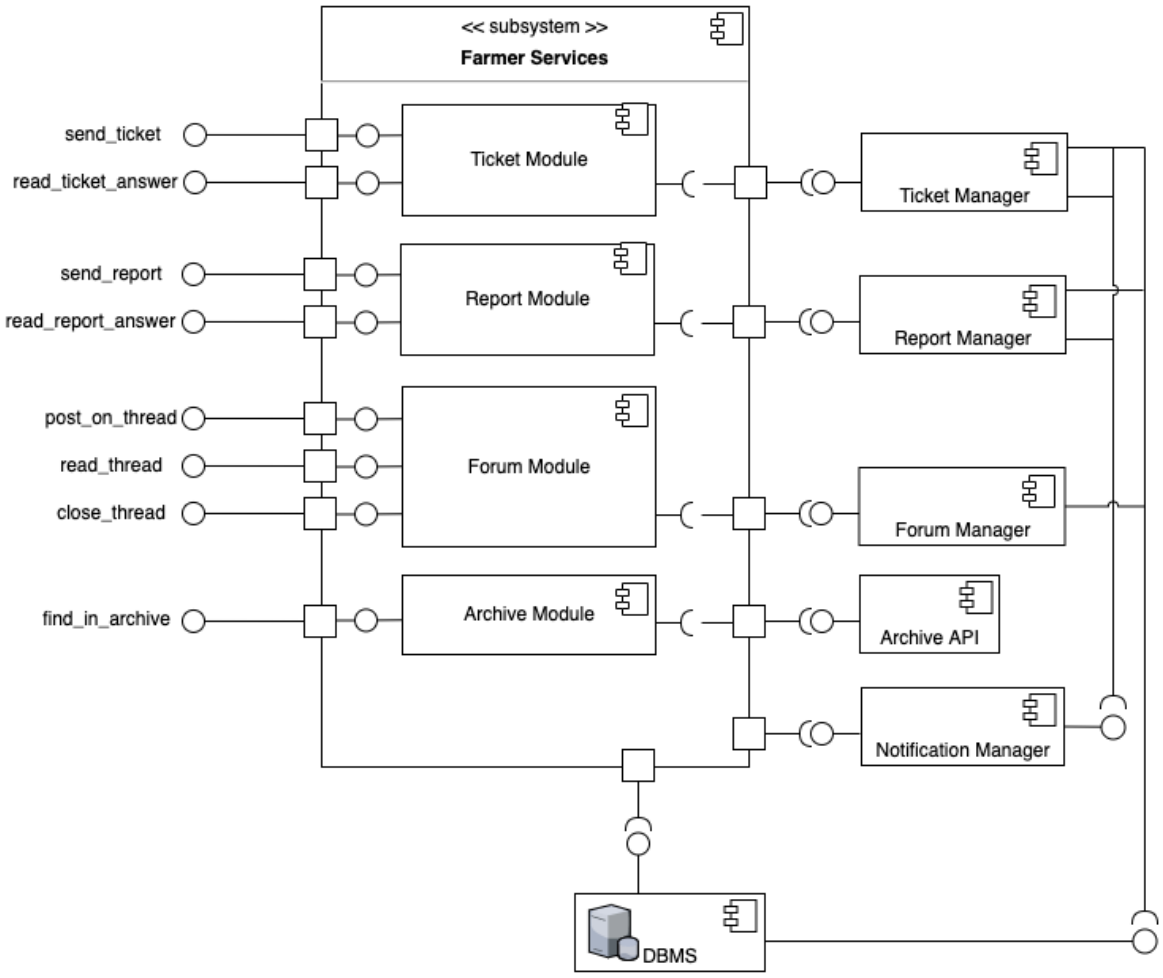

Figure 2.2: farmer services - component diagram

2.2.3. Policy maker services

The following diagram shows how the subsystem Policy Maker Mobile Services is built and how its modules communicate with the external ones.
The subsystem has 5 modules, and each one provides to the application some interfaces.

1. **Ticket Module:** answer_ticket, read_ticket
2. **Report Module:** answer_report, read_report
3. **Contacts Module:** read_contacts
4. **Forum Module:** read_thread
5. **Statistics Module:** aggregate_data, show_data

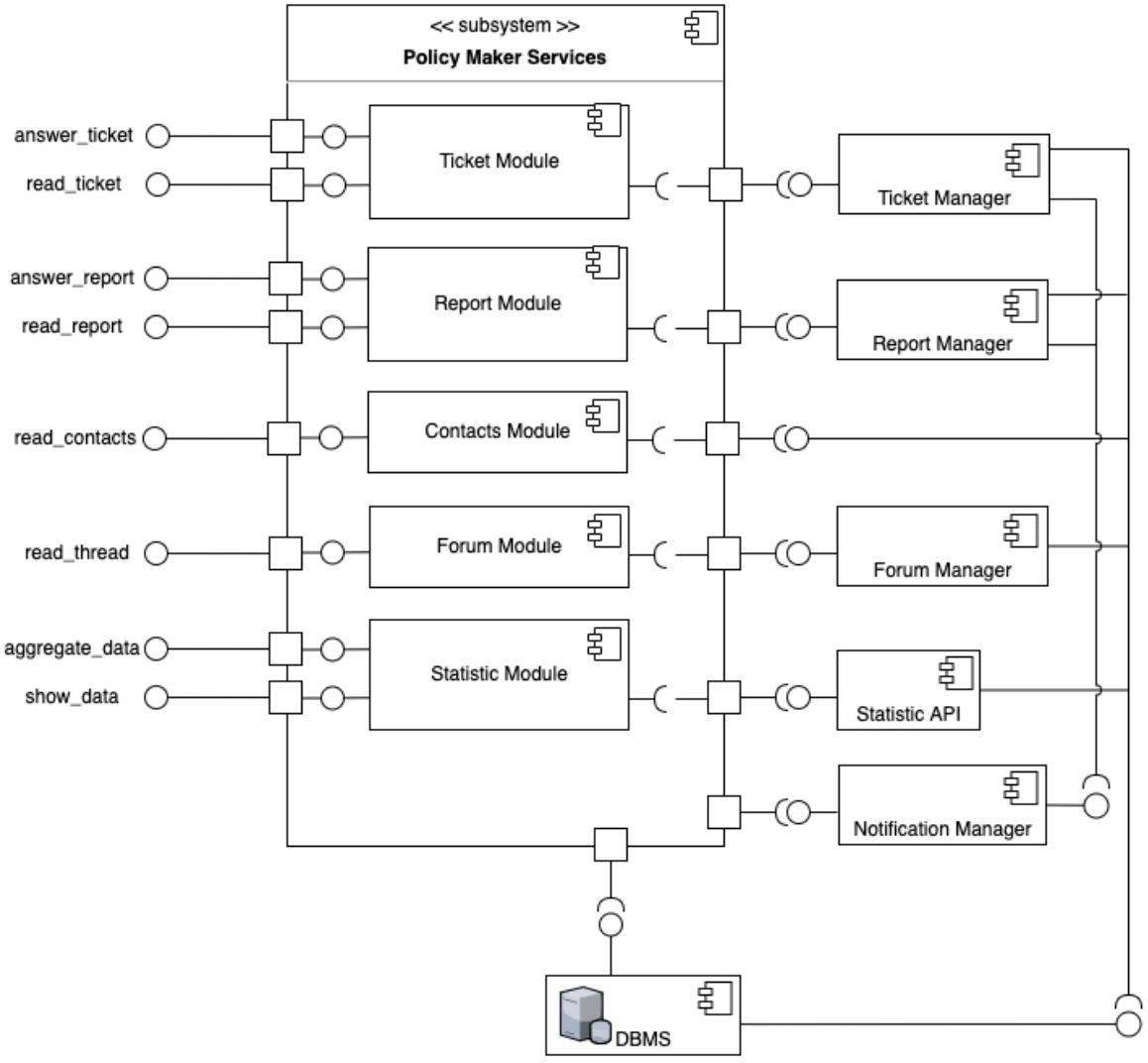

Figure 2.3: policy maker services - component diagram

2.2.4. Main Logic

The following diagram shows how the subsystem Main Logic is built and how its modules communicate with the external ones.
The subsystem has 2 modules and each one provides to the application some interfaces.

1. **WeatherMap Module:** show_wheather, show_forecast, show_map
2. **News Module:** show_news
3. **Account Module:** access_manager
4. **Notification Module:** send_notification

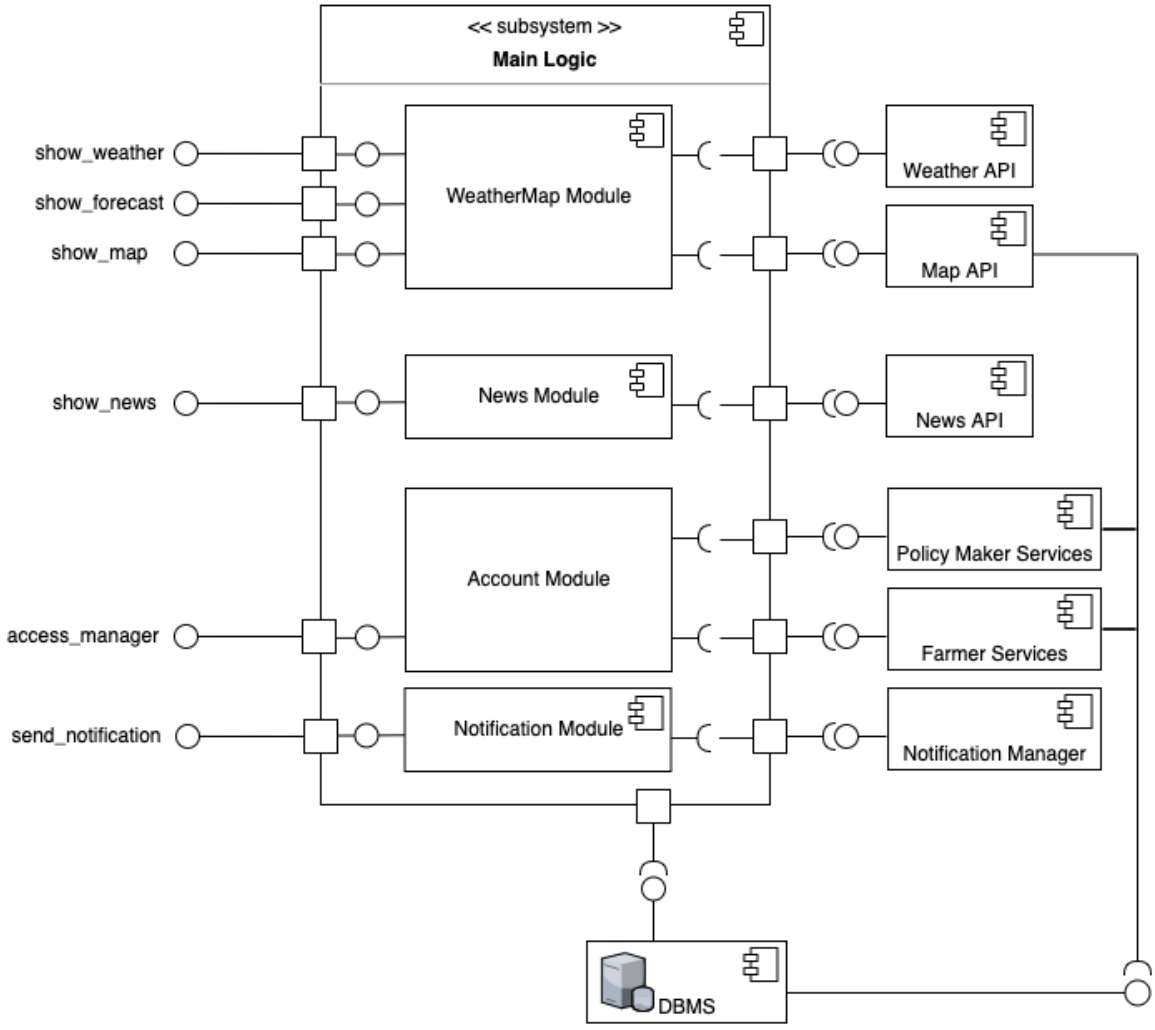

Figure 2.4: main logic - component diagram

2.3. Deployment view

The system architecture is divided into 4 tiers using the standard JEE:

1. **EIS Tier:** In this label are collected all the functions which are connected with the database. All the cryptography components are organised here.
   This part is connected with the database and the business tier.
2. **Business Tier:** In this part are collected all the EJB which manage the server-side logic of the dream application.
   This part is connected with the business tier and the web tier.
3. **Web Tier:** In this label are collected all the servlet and the JSP which provide all the interfaces that clients can use.
   This tier is connected with the business tier and with Internet to serves users requests.
4. **Client Tier:** In this tier we find the DREAM application for mobile user and a browser connection point for web users.

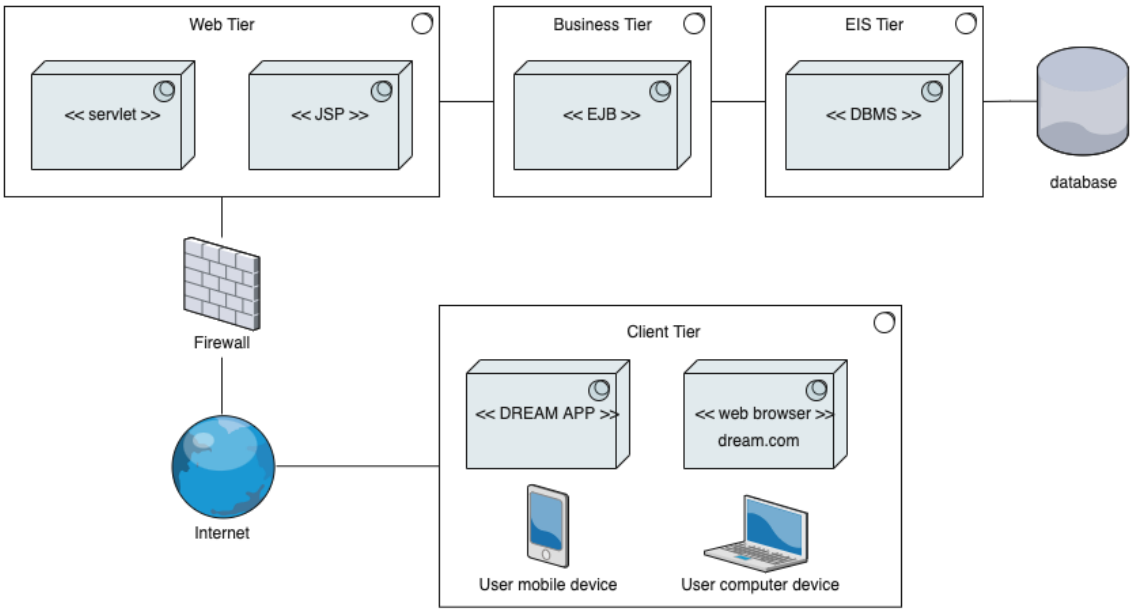

Figure 2.5: high-level deployment overview

2.3.1. Recommended Deployment

All the tiers must be developed as separate elements. This organisation will produce various advantages as easy maintainability because it is possible to work on a single module without interfering with the others. In this way the deployment can be done faster in separate teams. It will be also possible to upgrade the system in the future by adding single elements without modify the general structure, furthermore this approach promote the scalability.
Given that all the tiers are implemented with the JEE all the components are easy and already structured to be integrated with each others.
We recommend to follow the enumeration at the beginning of this section as the order of priority to implement the components of the system. More information about the implementation will be given in the next chapters.

2.4. Runtime view

In the following sequence diagrams we are going to represent the behaviour of the most important components of the DREAM application. The diagrams show just an high level representation of the interaction between the various components. All the other details will be described and implemented during the development process.

2.4.1. Sequence diagrams

In this section some sequence diagrams are presented as an extension of the sequence diagrams already showed in the RASD document. Their purpose is to highlight the internal behaviour of the system, showing the interactions through components and DBMS.

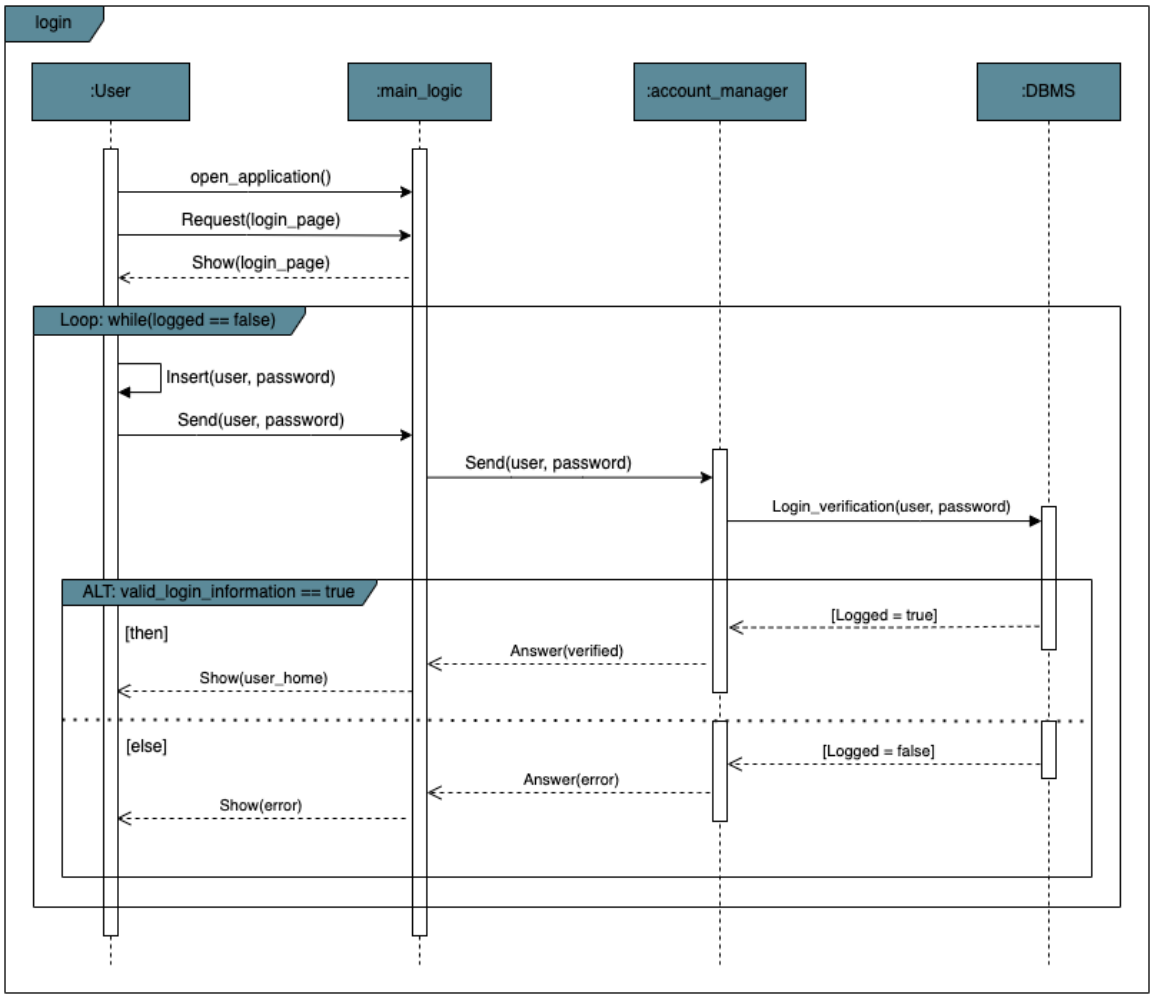

Figure 2.6: login - sequence diagram

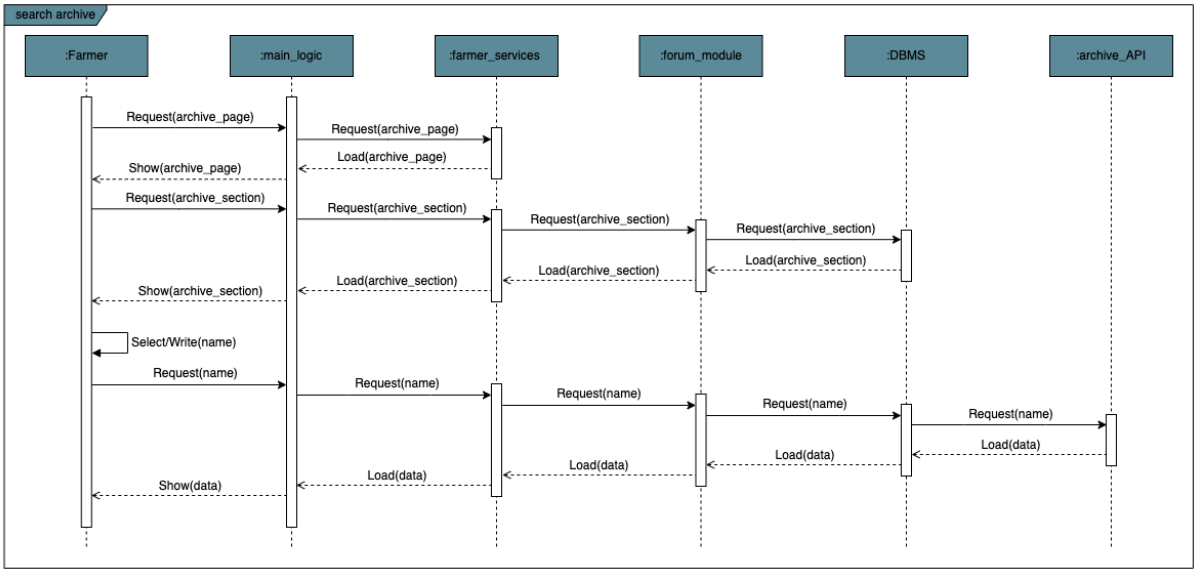

Figure 2.7: search archive - sequence diagram

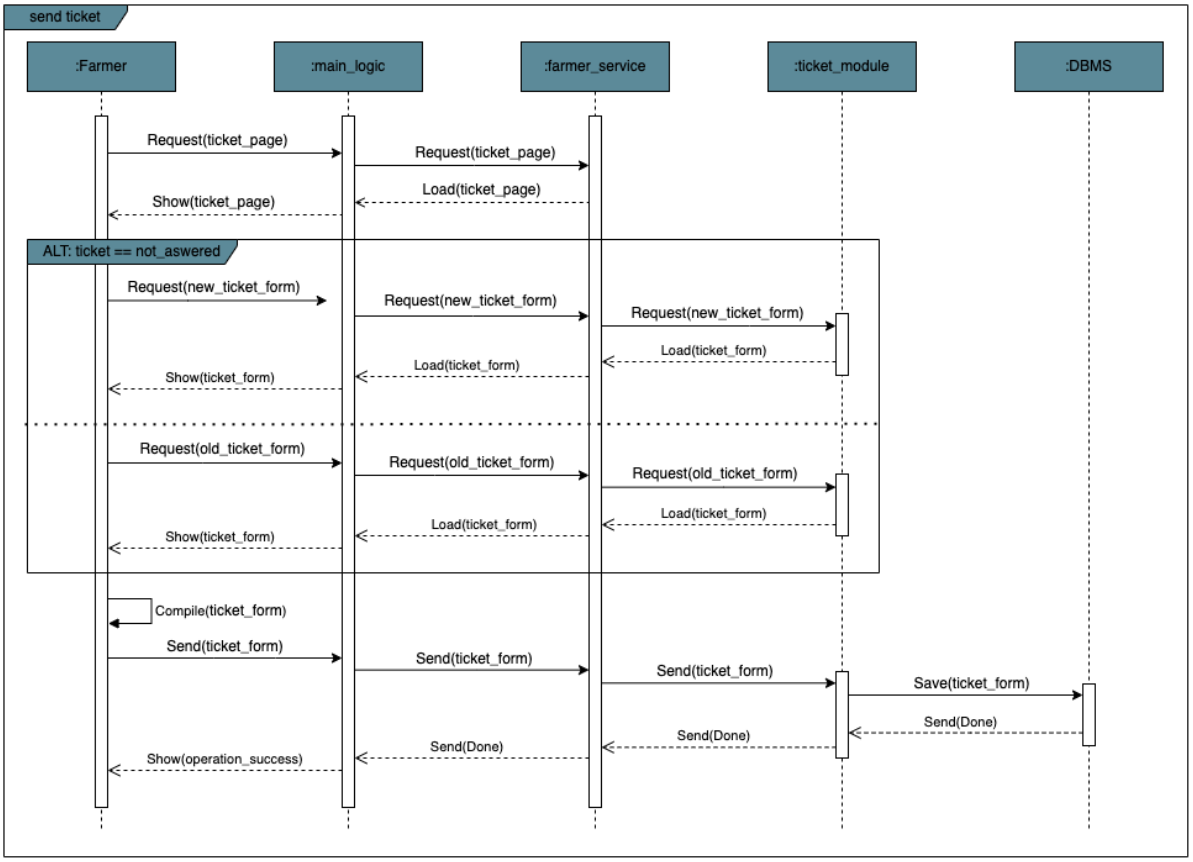

Figure 2.8: send ticket - sequence diagram

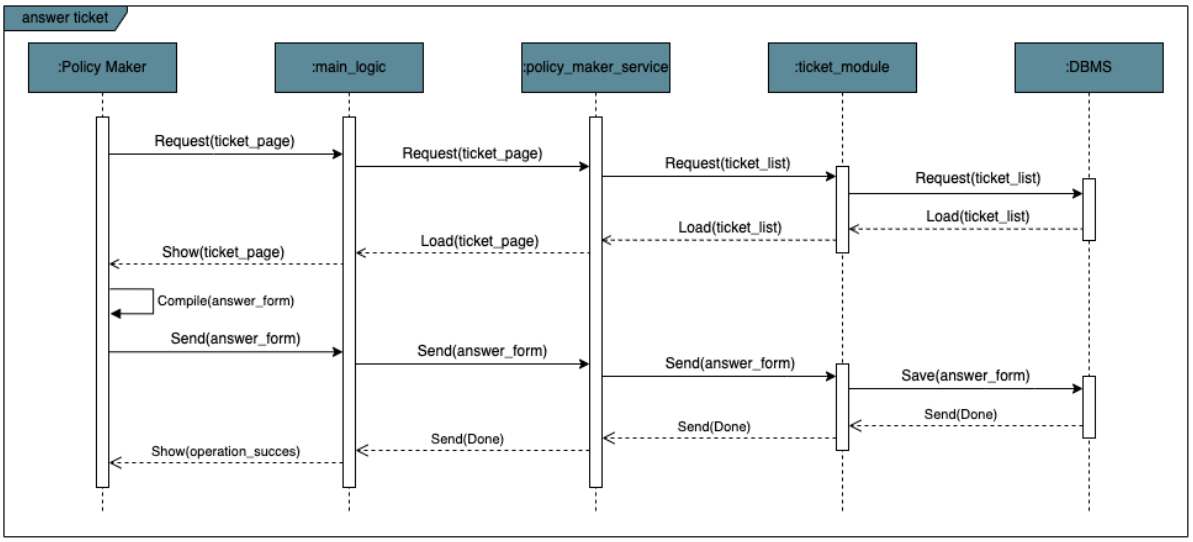

Figure 2.9: answer ticket - sequence diagram

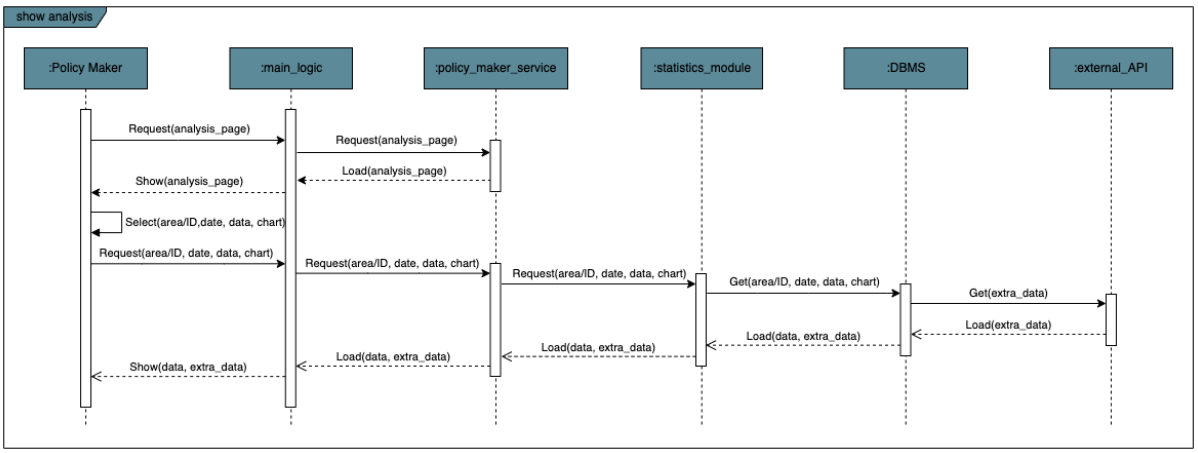

Figure 2.10: show analysis - sequence diagram

2.5. Component interfaces

In this section the interfaces of the various components will be analyzed and also the way to access to them. There are one diagram for each subsystem of the server-side of the system.

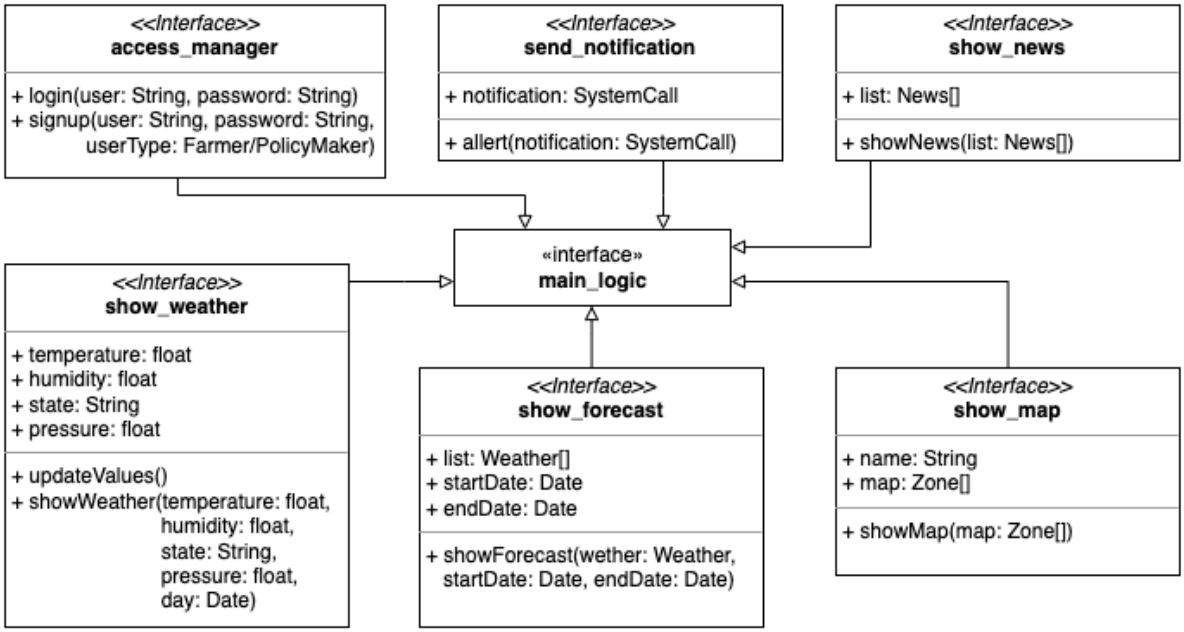

Figure 2.11: main logic - interface diagram

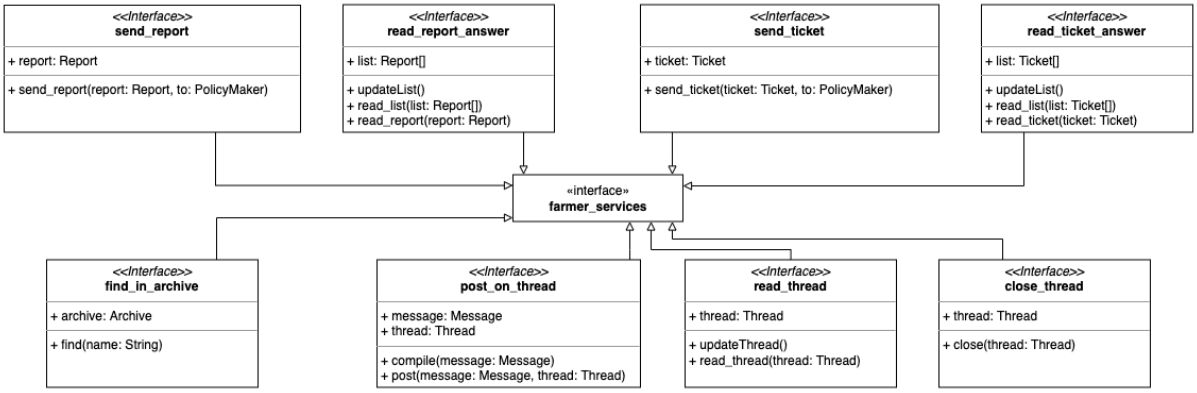

Figure 2.12: farmer services - interface diagram

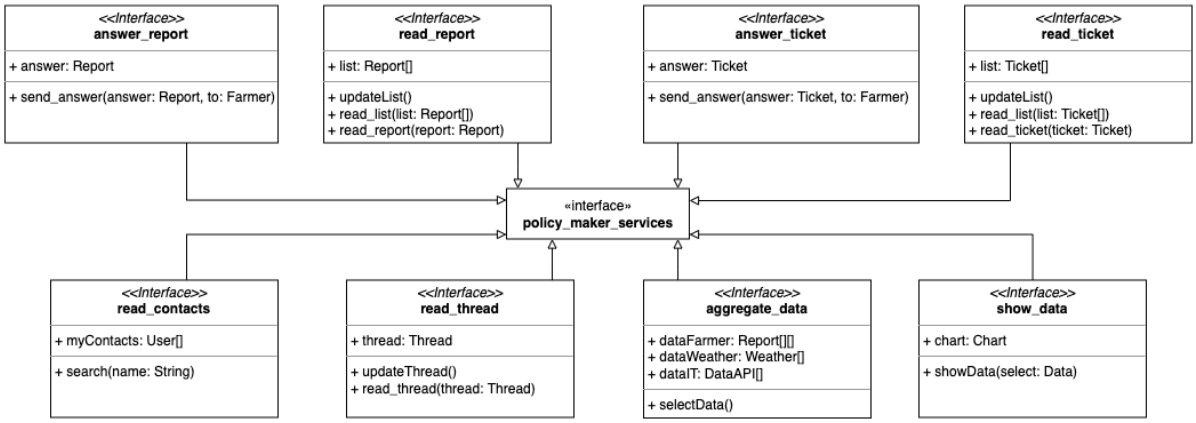

Figure 2.13: policy maker services - interface diagram

2.6. Selected architectural styles and patterns

2.6.1. Architectural Style

In the overview section there is a description of a three-tier client server architecture has been used to design the system. This choice guarantees to the system an increasing reusability, scalability and flexibility.
For the communication between the application and and the main logic the choice taken is REST architecture. This possibility let us to update the server side with all the needed information or modification without touching the client side of the system. In order to keep this architecture the system will be:

- Layered Architecture
- Client - Server

2.6.2. Patterns

Recommended pattern to implement the application:

- **Model-View-Controller Pattern:** It is an architectural pattern which divides the application into three interconnected parts. It is used to separate the how the information are represented inside the system from what the user actually see. It is a really good and easy pattern to use when a system is based on a three tier architecture presented in the previous sections.
- **Facade Pattern:** It simplifies a complicated a complex system by introducing a single interface to a set of interfaces within a subsystem.

- **Factory Pattern:** It simplifies a lot the logic of system because it let to create farmers and policy makers object by using the User class and to avoid useless redundant parts.

2.7. Other design decisions

- **Geo-localization API:** In order to keep track of the position of farms and zones the system needs a geo-localization API, in addiction to this it helps to localize the position where it is necessary to make a forecast.
- **Land Registry API:** It is necessary in order to obtain sensible data of every farmer and their territory.
- **Weather API:** Thanks to this API it is possible to obtain weather information, in addiction to the geo-localization API it is possible to understand with enough precision and forewarning when and where a weather event happens.
- **Chart API:** In order to better organise the data gathered by the policy maker the system use a chart API which let to create different graphs and table to analyze in different ways the data.
- **Archive API:** This API let the archive and its inside elements to stay up to date, with the best descriptions possible.
- **News API:** This API let the system to stay up to date with the last information about the weather, the politics and agriculture changes inside the country.

3 | USER INTERFACE DESIGN

In addiction to the view presented in the RASD, in this part there are other representations of additional mockups which will give a complete overview of how the application looks like

4 | REQUIREMENTS TRACEABILITY

Explain how the requirements you have defined in the RASD map to the design elements that you have defined in this document.

5 | IMPLEMENTATION, INTEGRATION AND TEST PLAN

Identify here the order in which you plan to implement the subcomponents of your system and the order in which you plan to integrate such subcomponents and test the integration.

6 | EFFORT SPENT

In this section you will include information about the number of hours each group member has worked for this document.

Student 1: Brunello Simone

Topic	Hours

Student 2: Nicolis Nicholas

Topic	Hours

7 | REFERENCES