Politecnico di Milano

DREAM
Design Document

Jiayan Cui
10756377
Yudong Wang
10829382

January 9, 2022

Contents

1. Introduction

1.1 Purpose

This document provides the functional description of DREAM, an application helped not only farmer, but also agronomists and policy makers to work together so that they can stabilize their food supply. Starting from DREAM Requirements And Specification Document, the system architecture is described, together with the design of the user interfaces. Each system component is mapped to one or more requirements specified in the RASD document. Finally, adequate plans to adopt when implementing, integrating and testing the system are provided.

1.2 Scope

DREAM is an organic system of policy producers, agronomists and farmers to solve farmers' land yield problems, report problems to policy makers, agronomists and get feedback, and DREAM is also a forum among farmers. To reach such an objective, the scalability and the usability of the system are taken into account when defining:

- The system architecture, which should scale well with the rate of adoption of the system.
- The user experience, which should be effective for people of all ages and conditions.
- The testing of all the system's components and of the system as a whole, to guarantee an adequate level of quality.

1.3 Definitions, acronyms, abbreviations

1.3.1 Definitions

Backend The part of a computer system or application that is not directly accessed by the user, typically responsible for storing and manipulating data.

CI Pipeline A series of steps that introduces automation to improve the process of application development, particularly at the integration and testing phases.

Elasticity The degree to which a system is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomic manner.

Layer A logical structuring mechanism for the elements that make up a software solution.

Pilot project An initial small-scale implementation that is used to prove the viability of a project idea.

Tier A physical structuring mechanism for a system infrastructure.

1.3.2 Acronyms

CI Continuous Integration.

DD Design Document.

RASD Requirements Analysis and Specification Document.

UML Unified Modeling Language.

DREAM Data-Driven Predictive Farming in Telangana.

G Goal.

IT device Information Technology device.

R Requirement.

SP Shared Phenomenon.

WP World Phenomenon.

1.3.3 Abbreviations

1.4 Revision history

Version	Date	Notes
V1.0	January 09, 2022	Initial release.

1.5 Reference documents

- **R&DD Assignment AY 2021-2022**
- **DREAM - Requirements Analysis and Specification Document**
- **UML documentation**

1.6 Document structure

This document is structured in the following way:

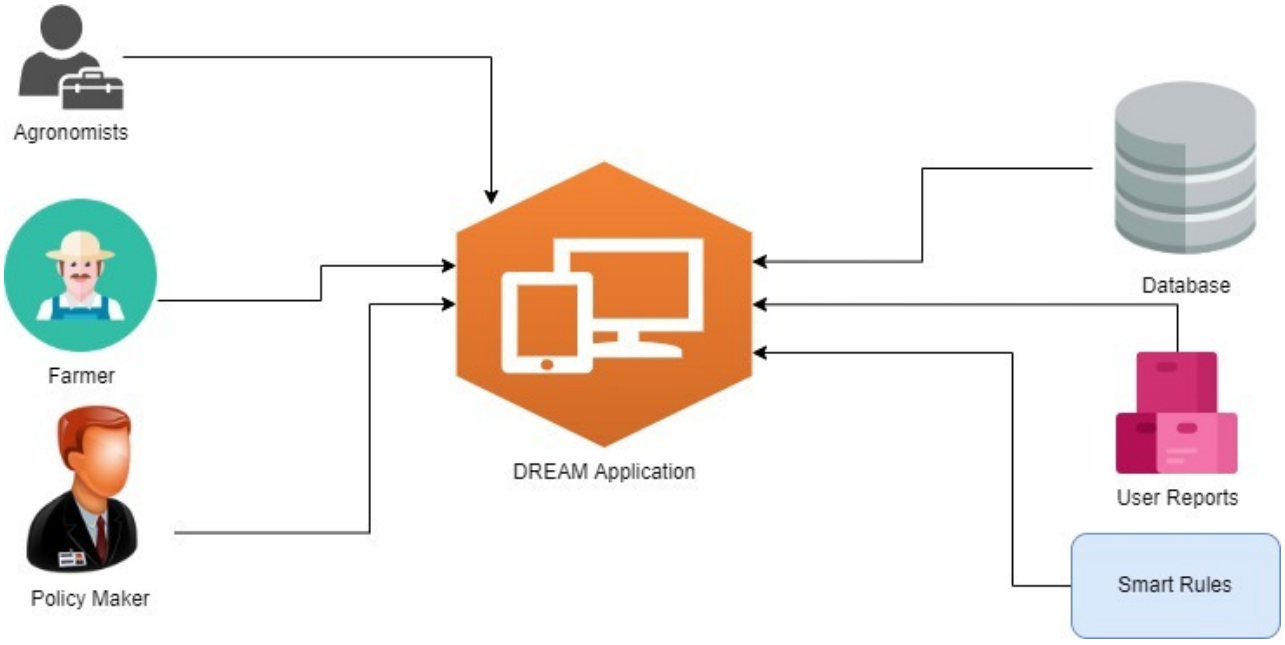
1. The first chapter is an introduction and an overview of the project, setting the context leading to the design choices made in this document.
2. The second chapter describes the system architecture, providing at first a high level description of the components and their interactions, then describing in detail each component both through static and dynamic views. Moreover, this chapter describes how the components are to be deployed using well known architectural styles and patterns and detailing the motivations for each choice.

3. The third chapter extends the "User interfaces" section described in the RASD by adding further details on how all the interactions with the users will be handled.
4. The fourth chapter provides traceability information that allows to map the components described in this document with the requirements enumerated in the RASD.
5. The fifth and last chapter provides the plans that should be adopted when implementing, integrating and testing the system.

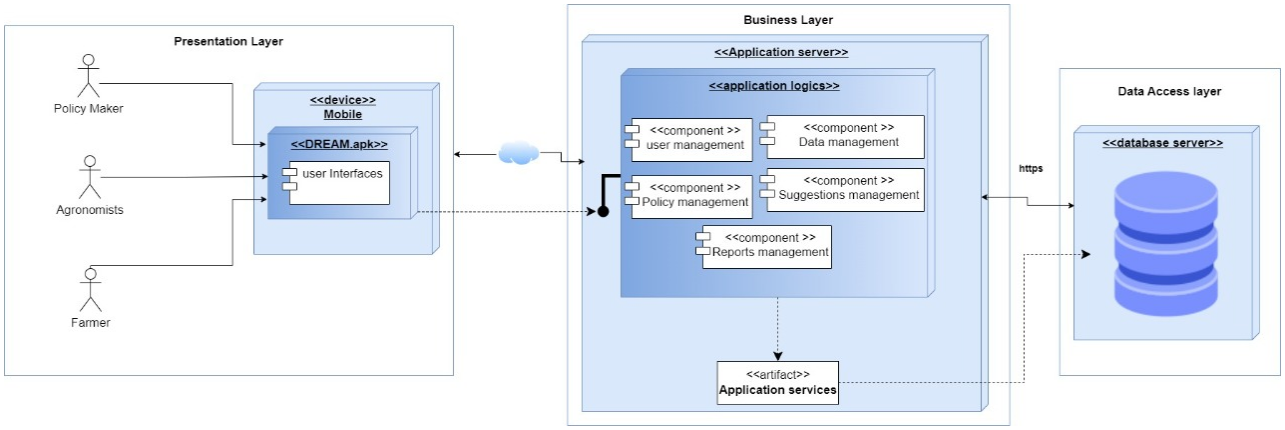
2. Architectural design

2.1 Overview

A system architecture is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system. The purpose of system architecture activities is to define a comprehensive solution based on principles, concepts, and properties logically related to and consistent with each other.



The below architecture of DREAM application shows three layers of the system the first is the presentation layer which contains the interaction between system users and the interface of the system. The second layer of the architecture is business layer which represents the web server containing application features system is offering. The third layer is the data access layer for the system database to store all the system information. The three main users of the system are policy makers, farmers and agronomists.



2.2 Component view

The component diagram below presents a general view of the application. The image describes the application components with the highest level of detail, as developers will have to implement the business logic themselves. The other nodes in the system, as well as the services, are depicted more abstractly. The main components are the database management, model, controller and view.

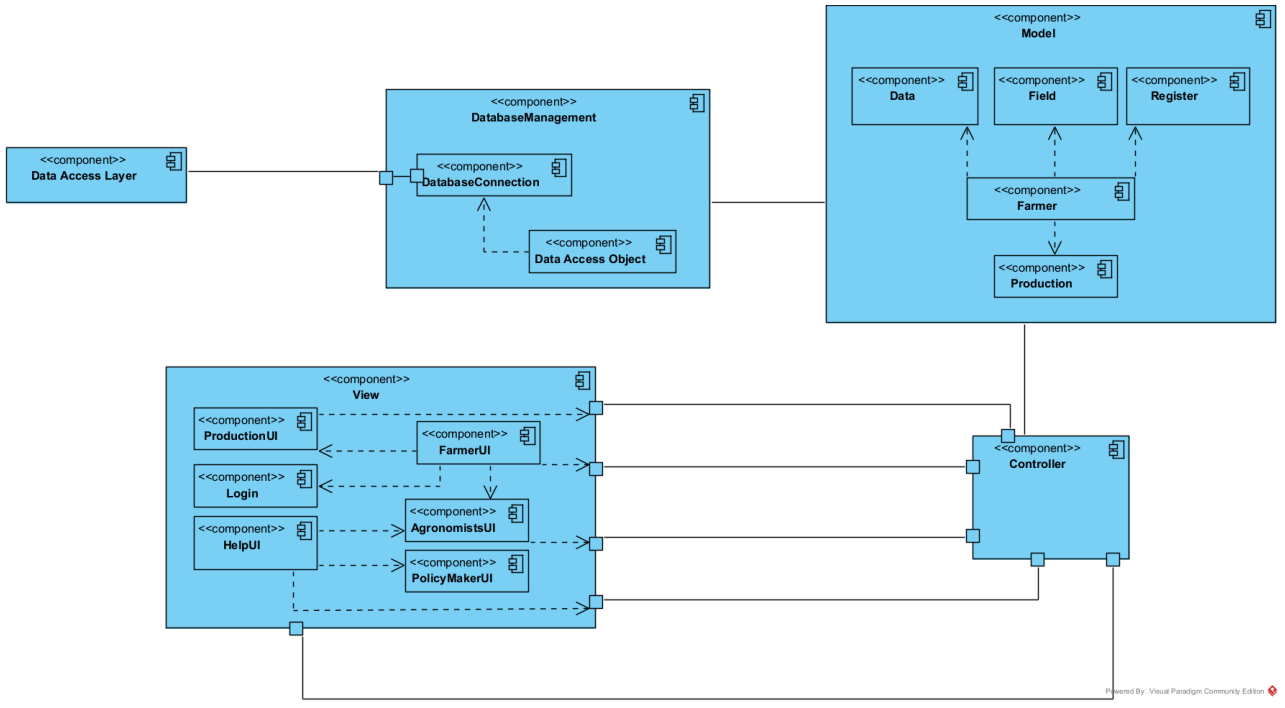


Figure 2.1: Component diagram

Farmers Register with personal data, upload produce information and production, get help in the app, and post questions in the forum. Farmers use the system to get weather information for their current location

and to get advice on planting crops. Farmers can also create discussion forums like a forum where they can upload their current problems in the system.

Agronomists Register with areas of responsibility and help farmers who are facing problems. Agronomists can answer farmers' questions and get the current weather data of the region and the best performing farmers in the system. They can also access local farmers according to the daily plan and update the daily plan daily.

Telengana's policy makers Register as an administrator and have access to view detailed data on farmer production and farmers who have been helped. Telengana's policy makers have access to the farmer performance rankings in the DREAM system to distinguish between farmers who are performing well and those who are not. The effectiveness of the policy is determined by comparing the ranking of the badly performing farmers over time.

2.3 Deployment view

The development view is primarily for developers who will be building the modules and the subsystems. It should show dependencies and relationships between modules, how modules are organized, reuse, and portability. The deployment diagram below represents the users interaction with the device and the application servers infrastructure, including application, web and database server. It also represents the GPS service using by DREAM application.

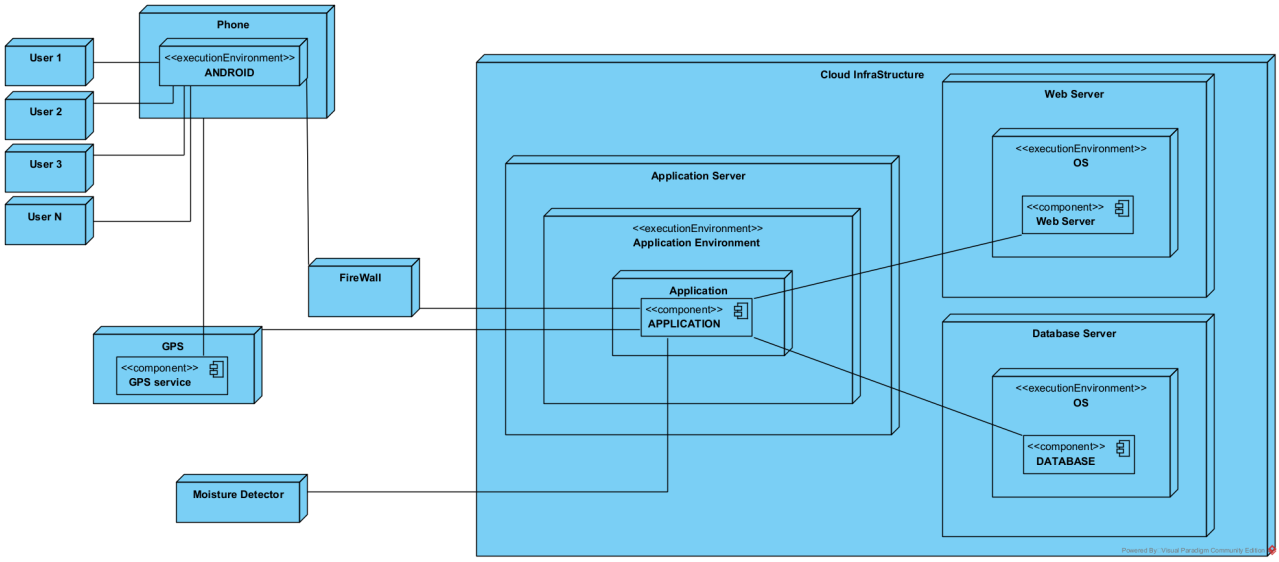


Figure 2.2: Deployment diagram

2.4 Runtime view

This view describes the behavior and interaction of the system's building blocks as runtime elements.

2.4.1 Runtime Scenario 1 - Policy Maker Sign Up

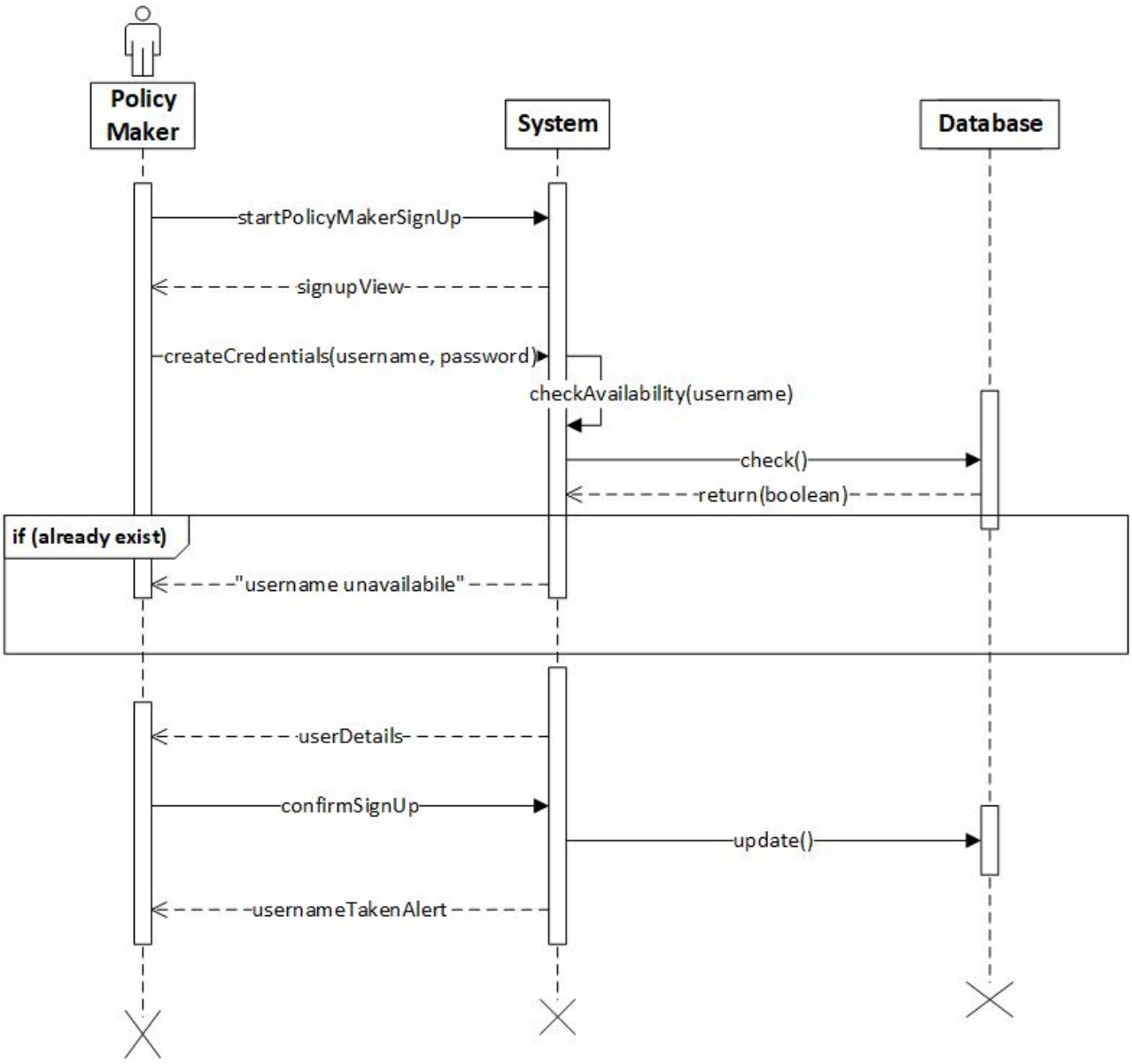


Figure 2.3: Policy maker sign up sequence diagram.

2.4.2 Runtime Scenario 2 - Farmer Sign Up

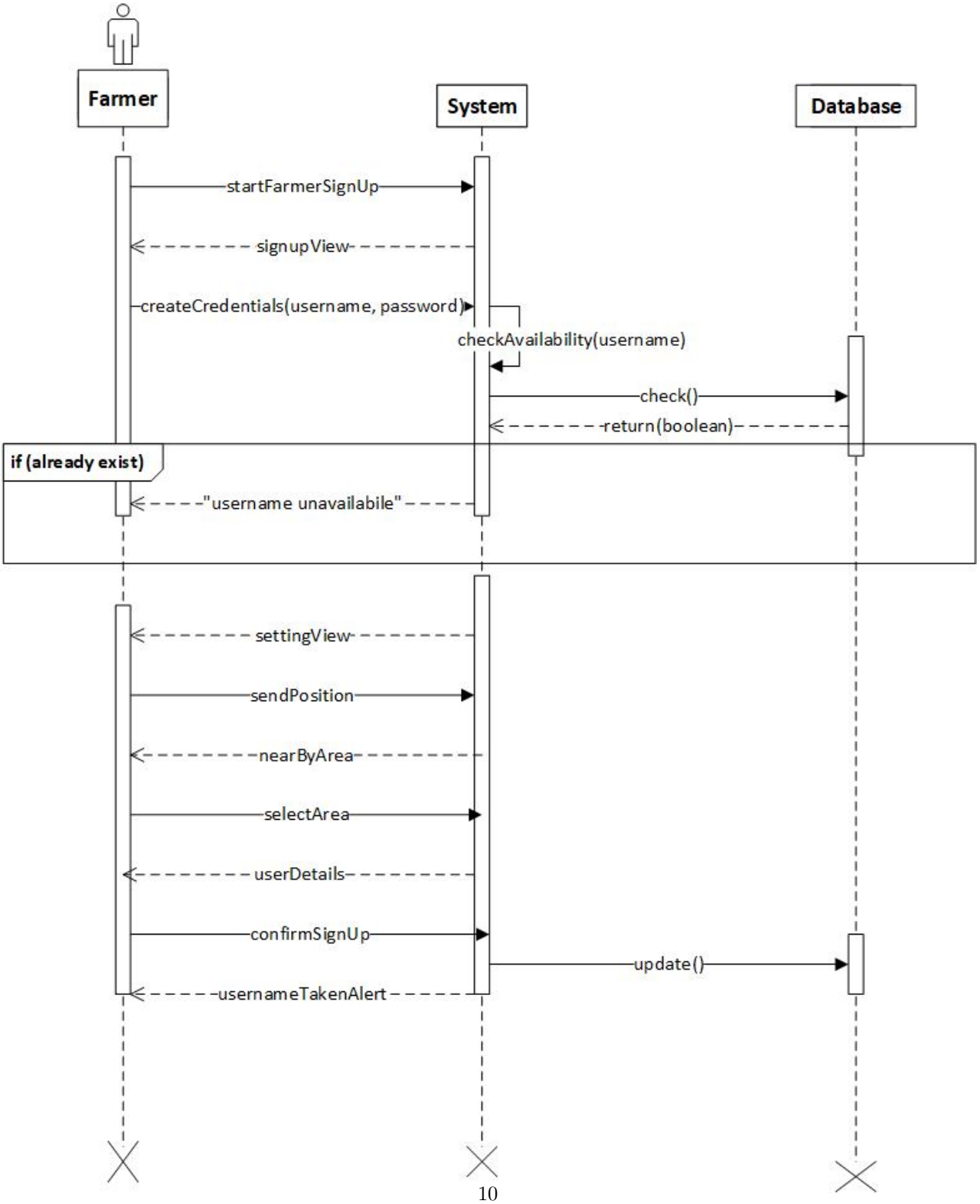


Figure 2.4: Farmer sign up sequence diagram.

2.4.3 Runtime Scenario 3 - Agronomists Sign Up

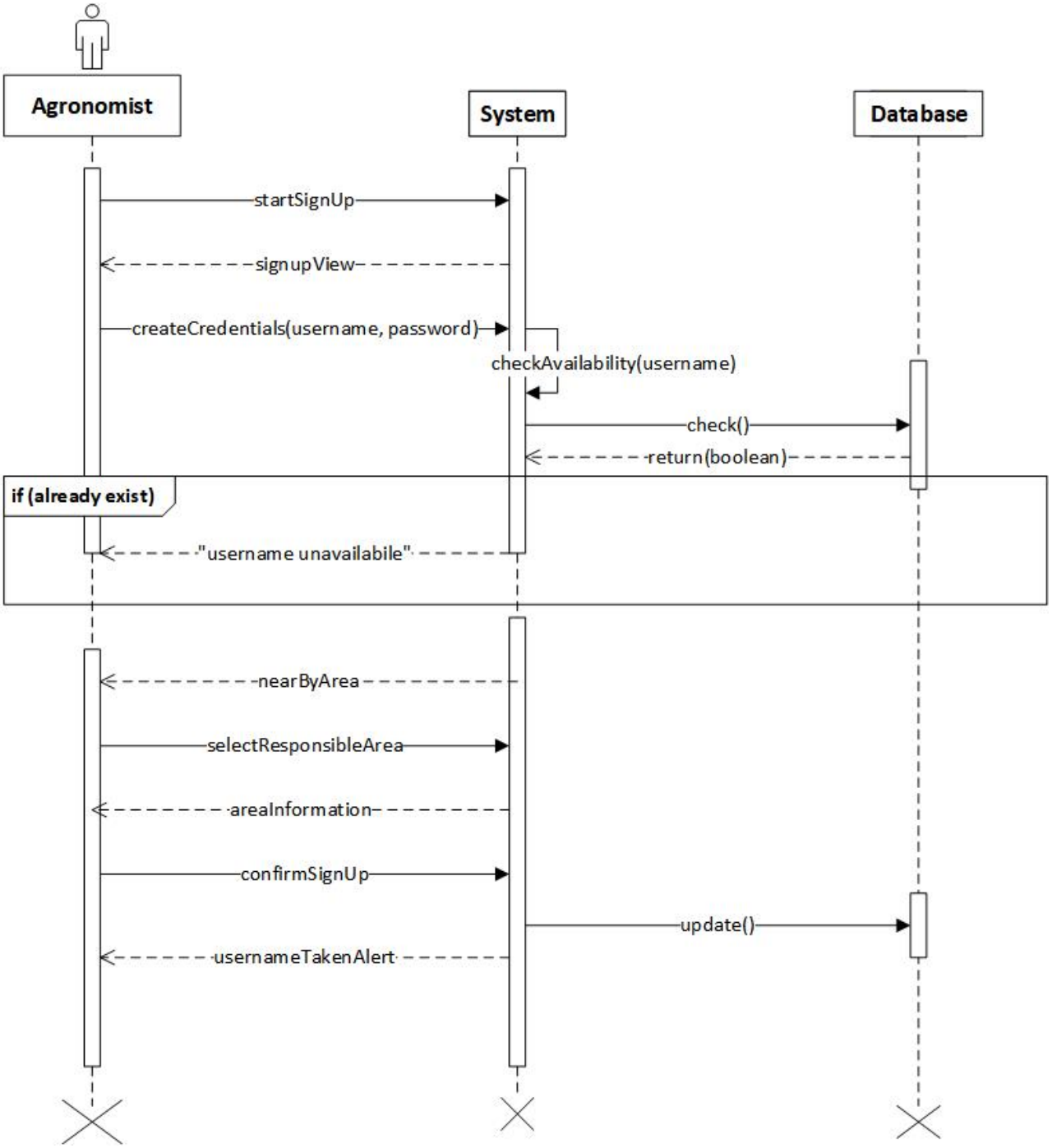


Figure 2.5: Agronomists Sign Up sequence diagram.

2.4.4 Runtime Scenario 4 - Identify performing Well Former

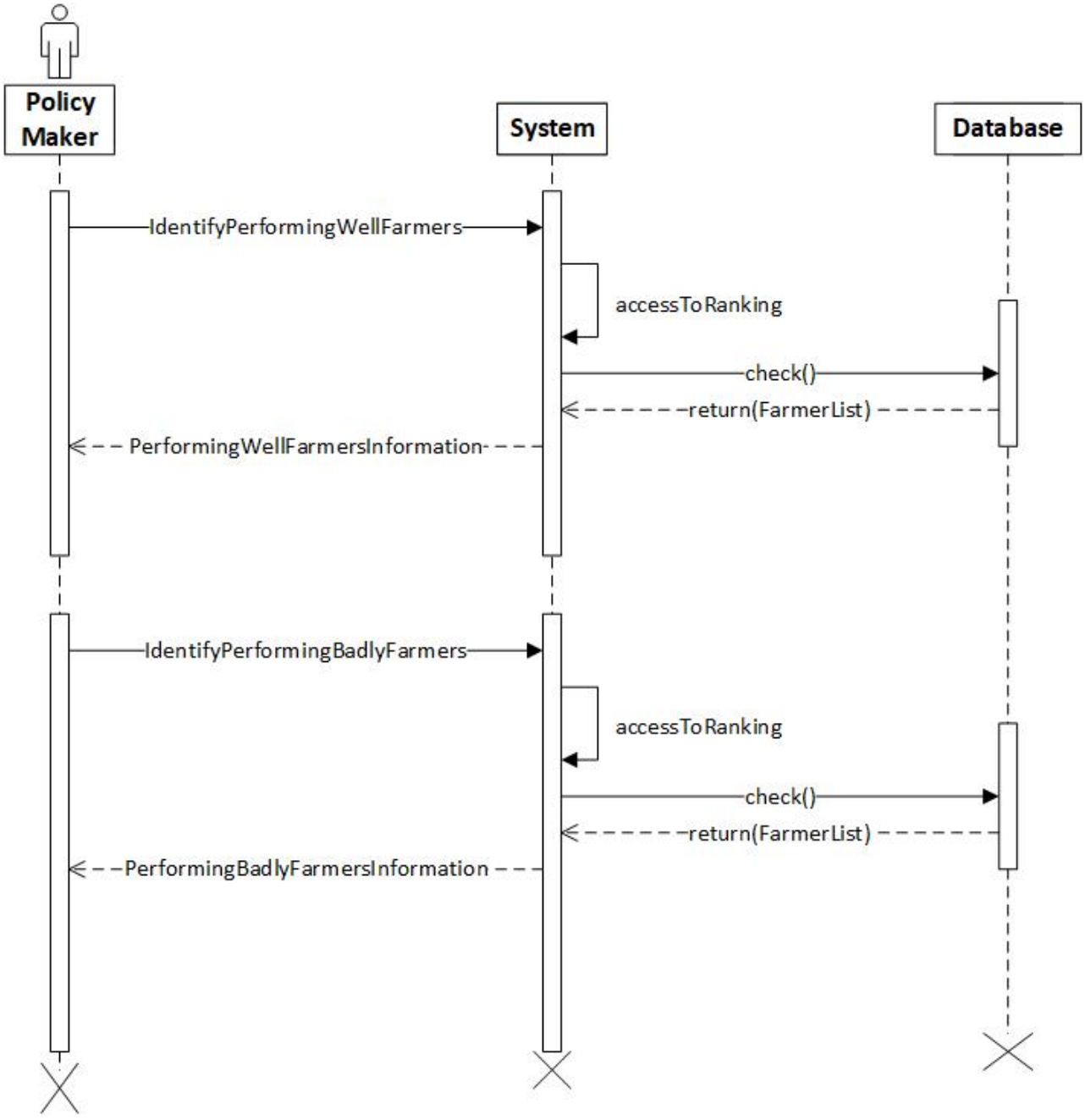


Figure 2.6: Identify performing well Farmer sequence diagram.

2.4.5 Runtime Scenario 5 - Check Results

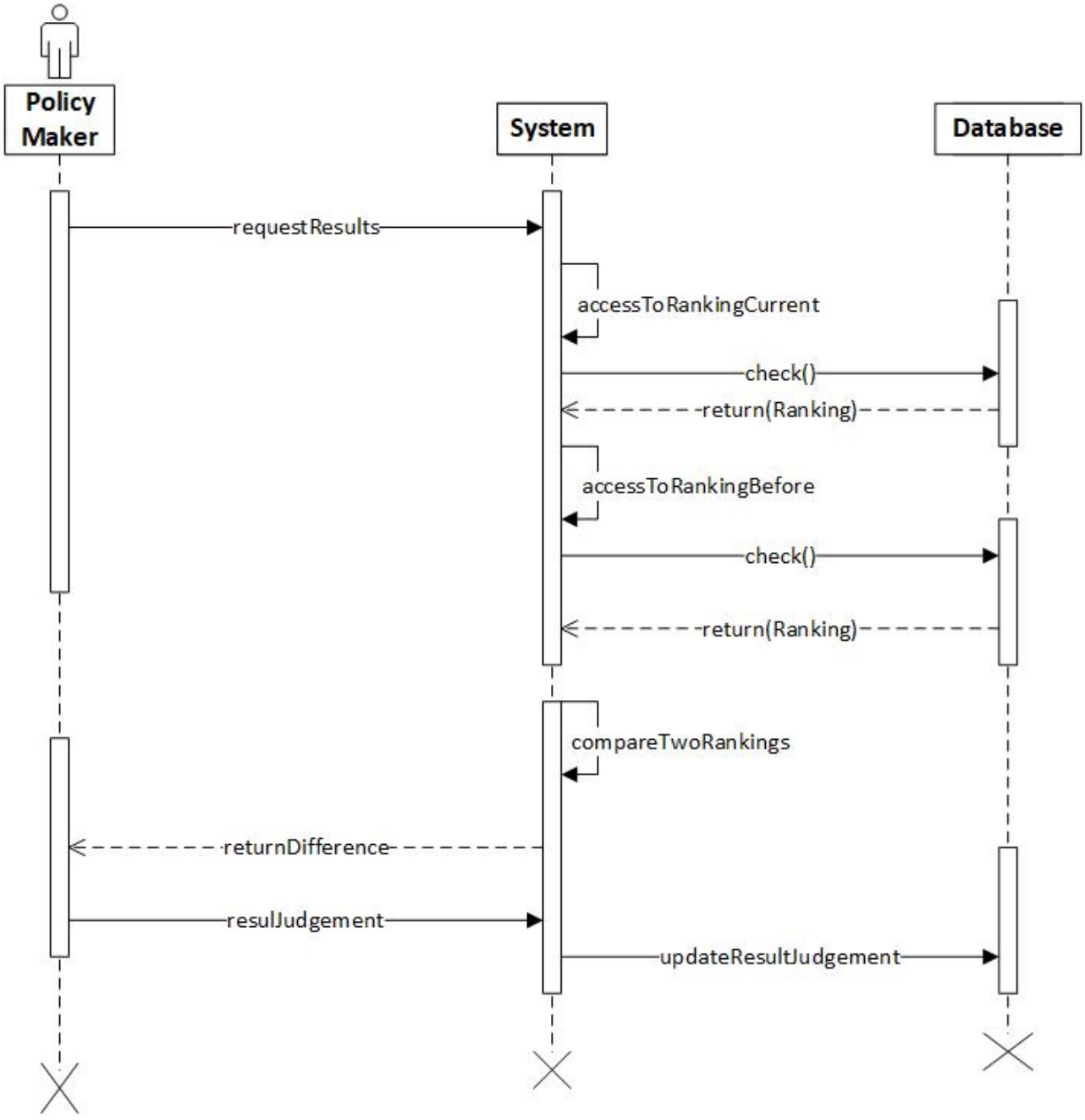


Figure 2.7: Check results sequence diagram.

2.4.6 Runtime Scenario 6 - Suggestion for problems

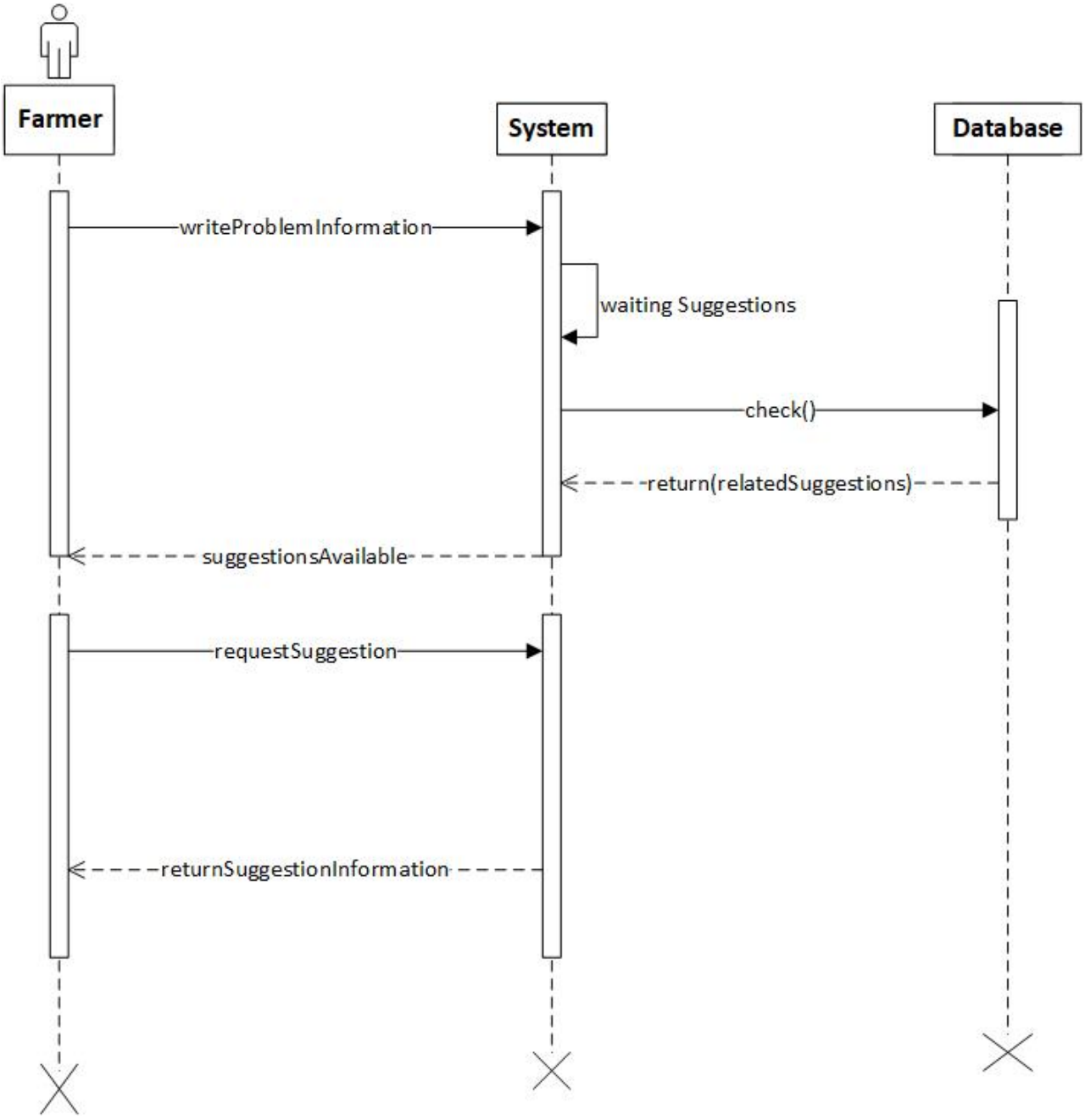


Figure 2.8: Suggestion for problems sequence diagram.

2.4.7 Runtime Scenario 7 - Discussion Forums

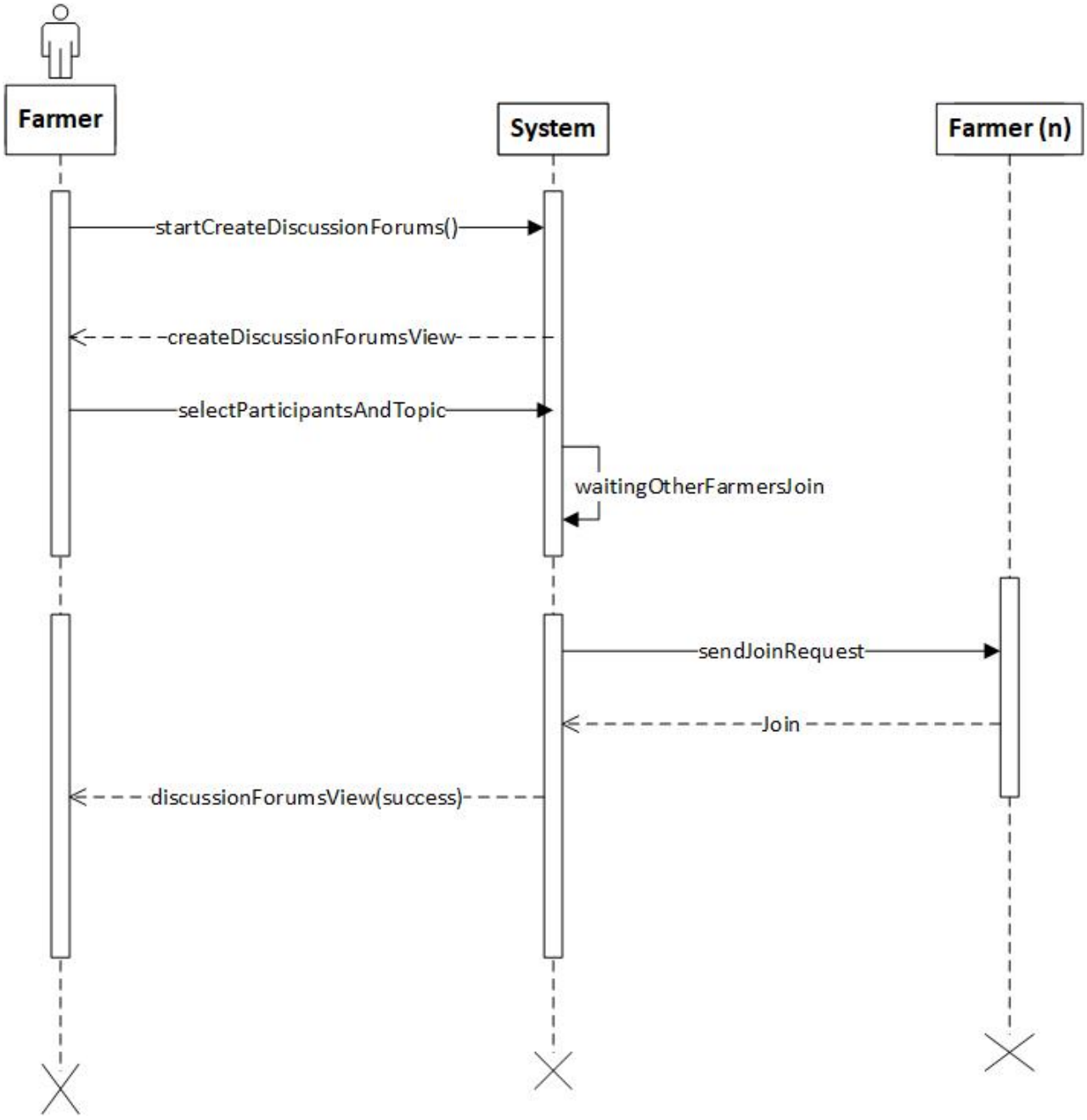


Figure 2.9: Discussion forums sequence diagram.

2.4.8 Runtime Scenario 8 - Access Weather Data

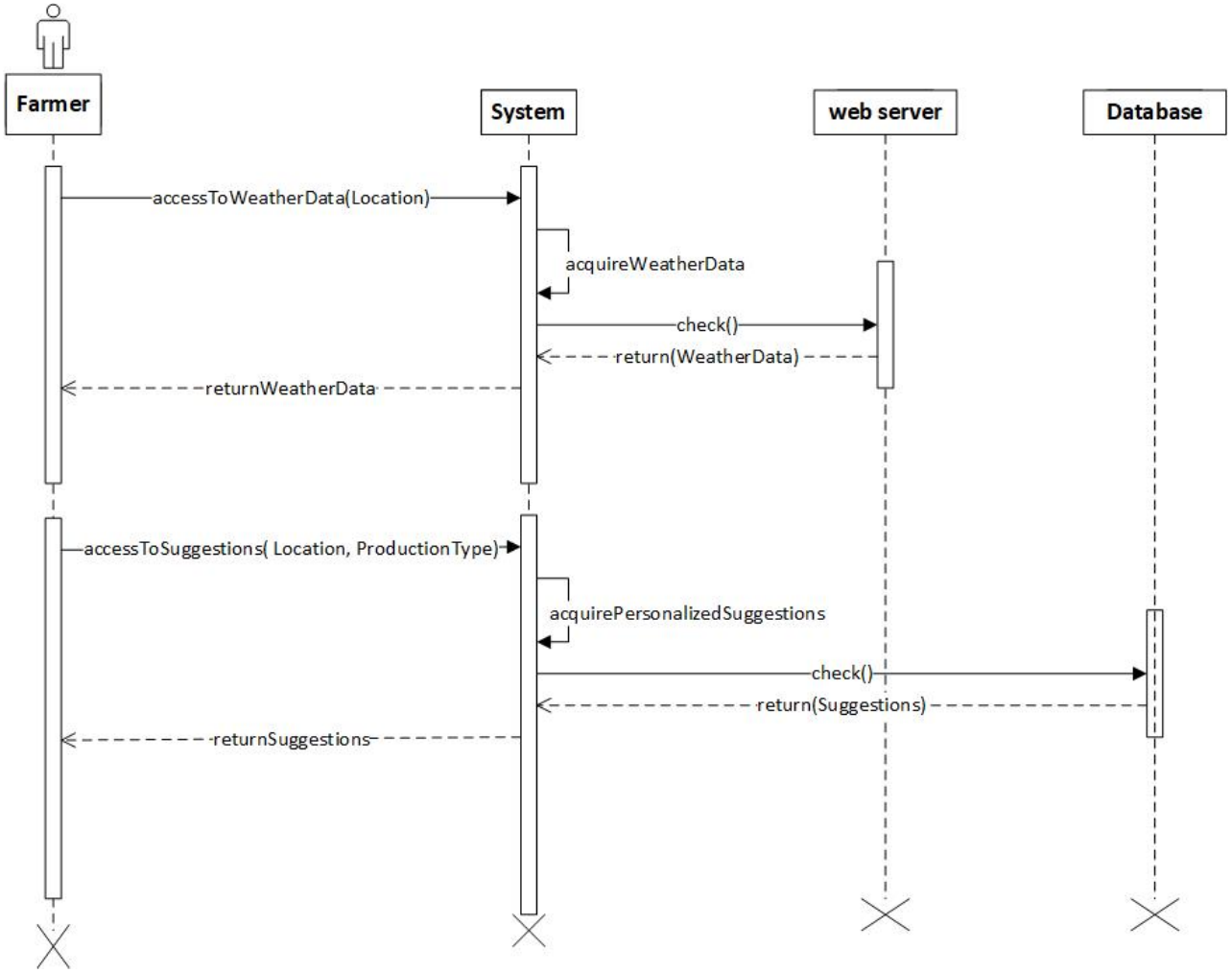


Figure 2.10: Access weather data sequence diagram.

2.4.9 Runtime Scenario 9 - View Daily Plan

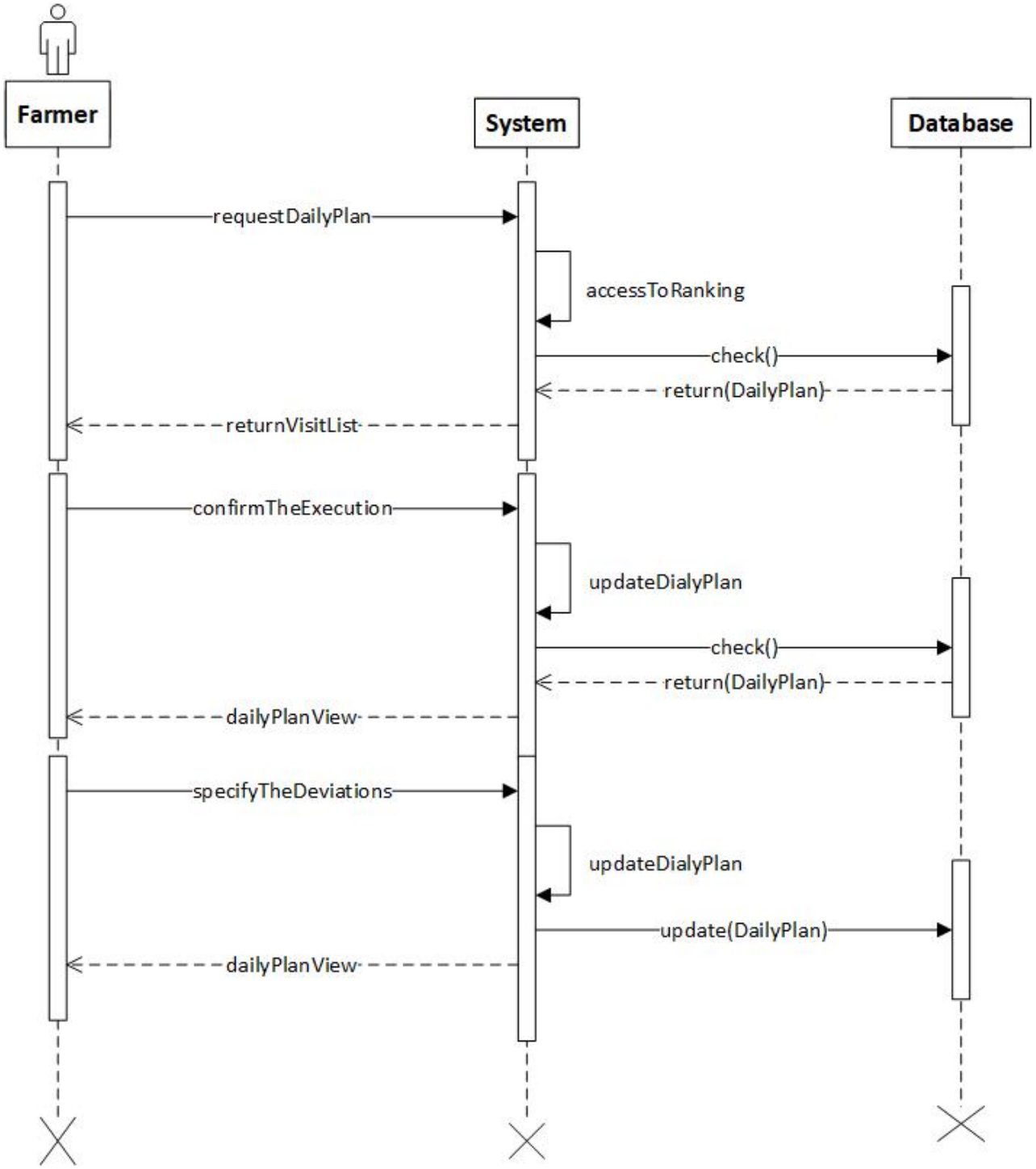


Figure 2.11: View daily plan sequence diagram.

2.5 Component interfaces

```
// Components

public interface Registeration {
  void uploadInformation(Object personalInformation);
}

public interface Discussion {
  void createDiscussion(int personalId, int numberOfPeople);
}

public interface AskQuestion {
  void createDiscussion(int personalId, Object question);
}

public interface requestRanking {
  Map<Object, id> selectRankOfFarmer(int personalId, int cropProduciton);
}

public interface searchQuestion {
  Collection<Object> selectQuestion(int questionId);
}

public interface requestWeather {
  void selectWeather(string location);
}

// DBMS

public interface DBMSAPI {
  void deleteInformation(int personalId);
  void deleteQuestion(int questiontId);
  void insertQuestion(int personalId, int quesiton);
  void uploadInformation(Object personalInformation);
  void createDiscussion(int personalId, int numberOfPeople);
  void createDiscussion(int personalId, Object question);
  void selectWeather(string location);
  Map<Object, id> selectRankOfFarmer(int personalId, int cropProduciton);
  Collection<Object> selectQuestion(int questionId);
  Collection<Object> selectVisits(int areaId);
  void updateList(int personalId);
}

// External services

public interface VerificationAPI {
  boolean checkToken(String token);
}
```

```
public interface WeatherAPI {
  String getWeather(Object Area);
}

public interface SensorAPI {
  double humidityInformation(Location devicePosition);
}
```

2.6 Selected architectural styles and patterns

As indicated in the description, the overall design is based on the conventional client-server architectural style, but in a multi-tiered arrangement. Because of the separation of concerns generated by the mapping of tiers into distinct layers, this pattern has been field-tested and provides adequate room for future revisions.

3. User interface design

3.1 Overview

The user interfaces mock-ups represented in section 3.1.1 of the RASD document are referenced here to describe the user interactions with the interface.

3.1.1 DREAM Application

1.Login

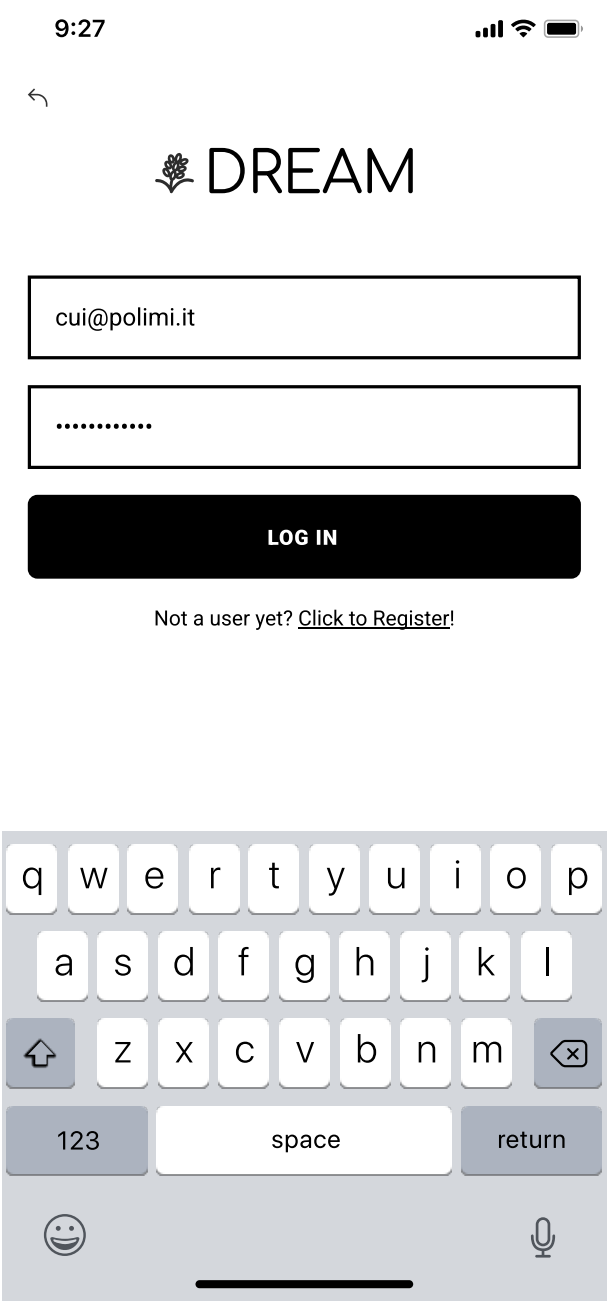


Figure 3.1: Login interface

2.Sign up

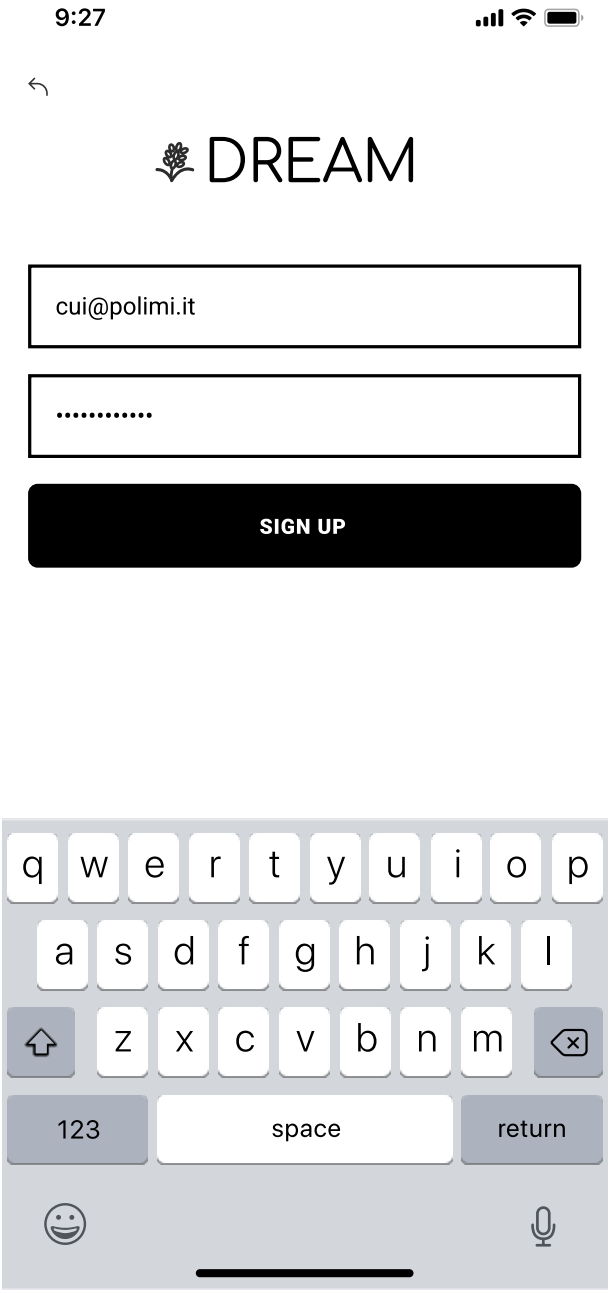


Figure 3.2: Sign up interface

3.Homepage

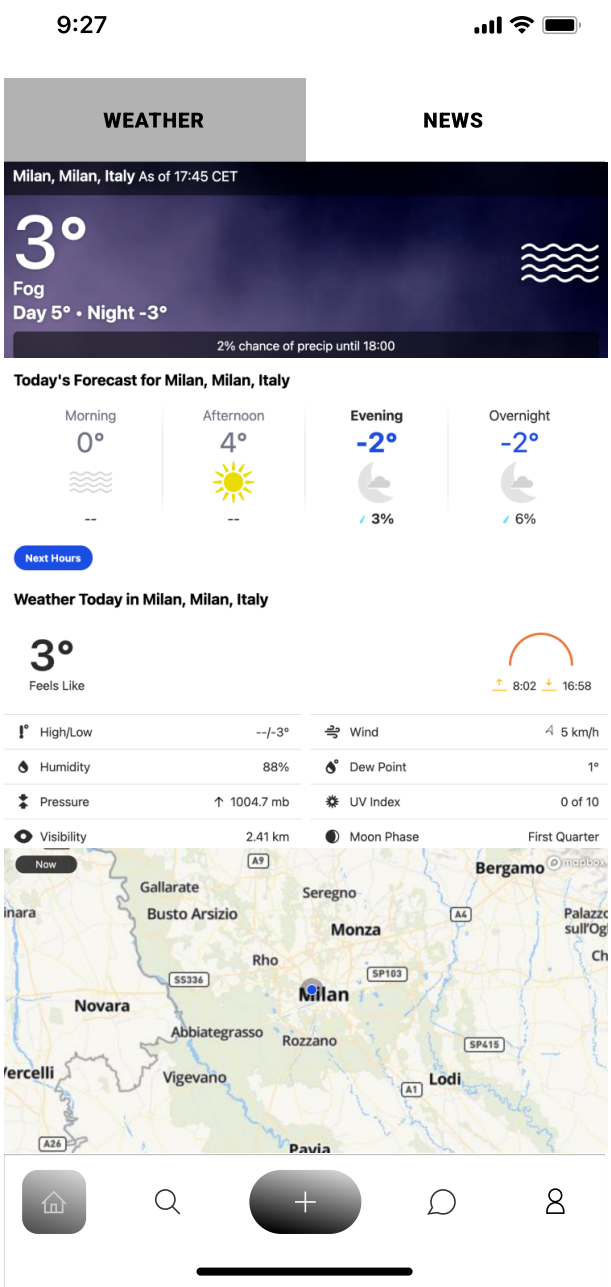


Figure 3.3: Homepage interface

4.Logout

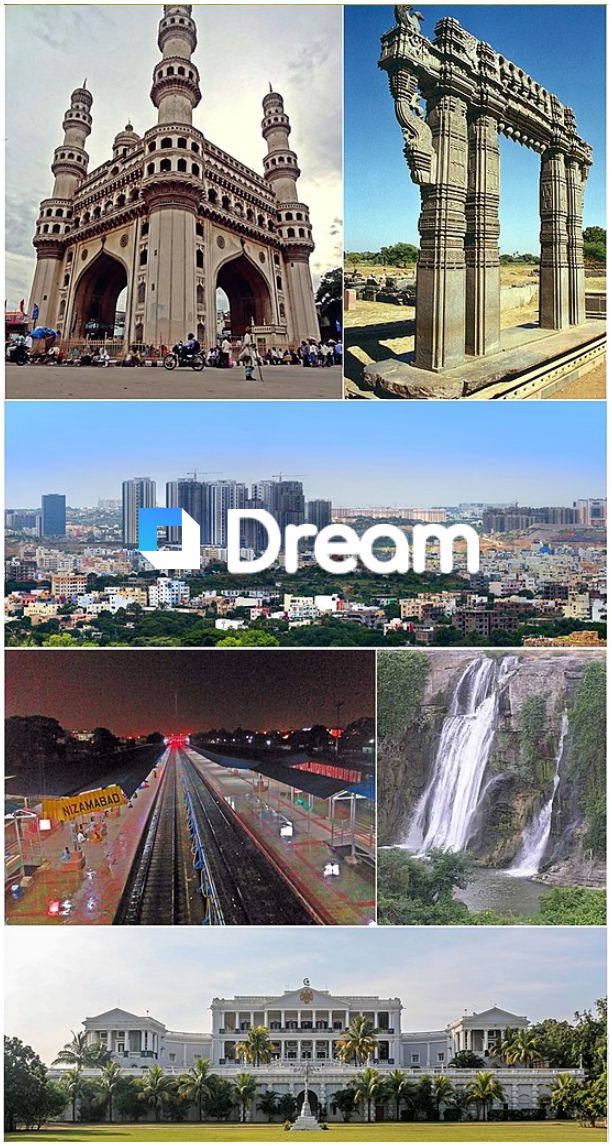


Figure 3.4: Logout interface

5.Individual Chat

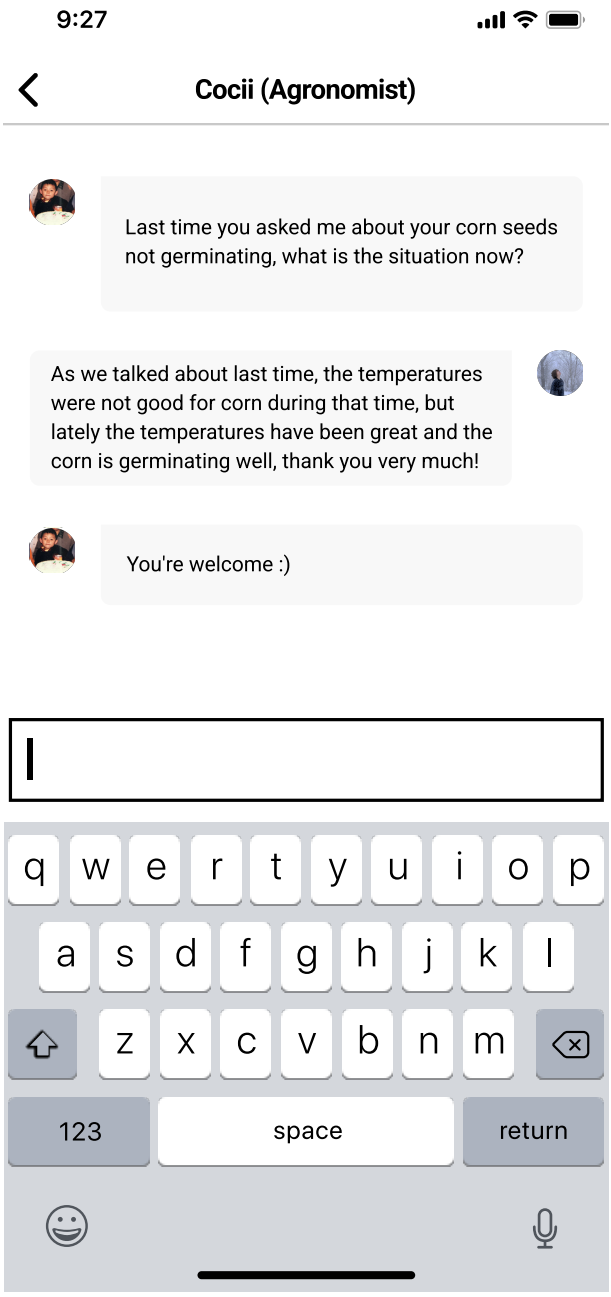


Figure 3.5: Individual Chat interface

6.Chat

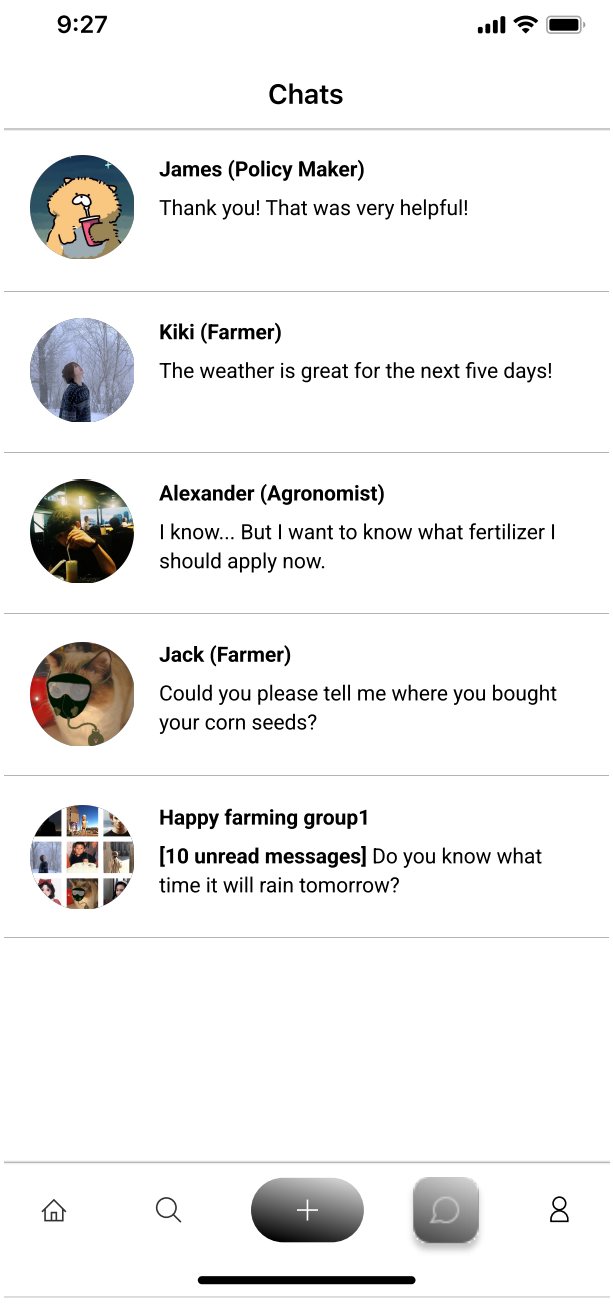


Figure 3.6: Chat interface

7.Profile

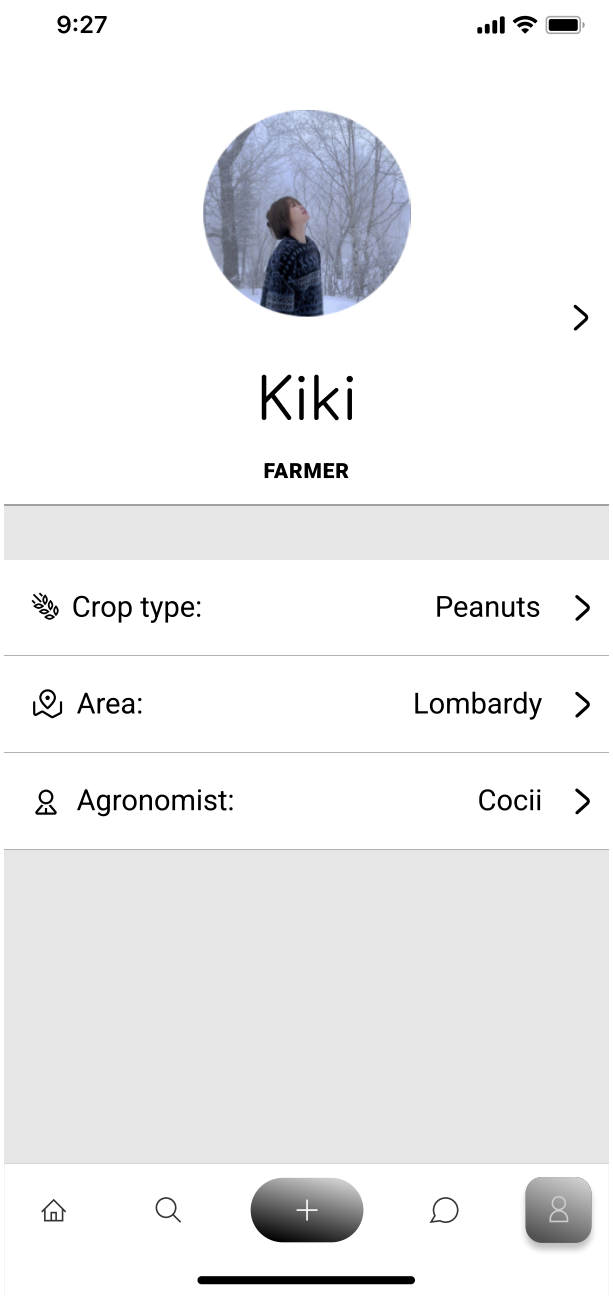


Figure 3.7: Profile interface

8.Search

Figure 3.8: Search interface

4. Requirements traceability

4.1 Overview

The following table shows the mappings between the Requirements presented in the Requirements And Specification Document and the components described in this document that implement their functionalities.

Component	Requirement IDs																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Farmer				x			x			x	x		x				
Policy maker			x		x	x											
Agronomists															x	x	x
Information		x	x	x					x	x		x					
Requests														x			x
Registration	x						x	x									
Suggestion														x			
Daily plan			x														
Results									x						x		

5. Implementation, integration and test plan

5.1 Overview

This part addresses the implementation, integration, and testing strategy for the components defined in the component view section. For each section, a strategy is defined that should be followed by the various teams developing the DREAM application in order to deliver the product in the most efficient way possible, with the shortest time to market possible, while also conforming to the software system attributes defined in the RASD section.

5.2 Implementation Plan

The complete system, as well as its related sub-systems, must be implemented, tested, and integrated from the ground up. With this method, the system may be built incrementally, allowing testing to begin concurrently with construction. Because the components are built, tested, and validated in a hierarchical manner, from bottom to top level, this technique provides more robustness. The implementation is done in a hierarchical order, starting with the bottom-most components, which have a closer interface with the database, and progressing to the upper-most components, which are farther away from the back end components and may rely on others. This ensures that the system may be constructed incrementally, and that the testing and integration process can begin concurrently with the implementation, allowing for better bug tracking, which leads to higher quality. Consequently, the components should be implemented following this order:

1. Farmer
2. Policy maker
3. Agronomists
4. Information
5. Registration
6. Requests
7. Results
8. Daily plan
9. Suggestion

The client-side components of the application may be implemented concurrently with the rest of the system by developers. The user interface, in particular, is independent of the web server APIs, and designers and developers may create prototypes and mocks and implement them as quickly as feasible, validating and testing them. When the back end is ready, it will be feasible to connect its API calls to the clients.

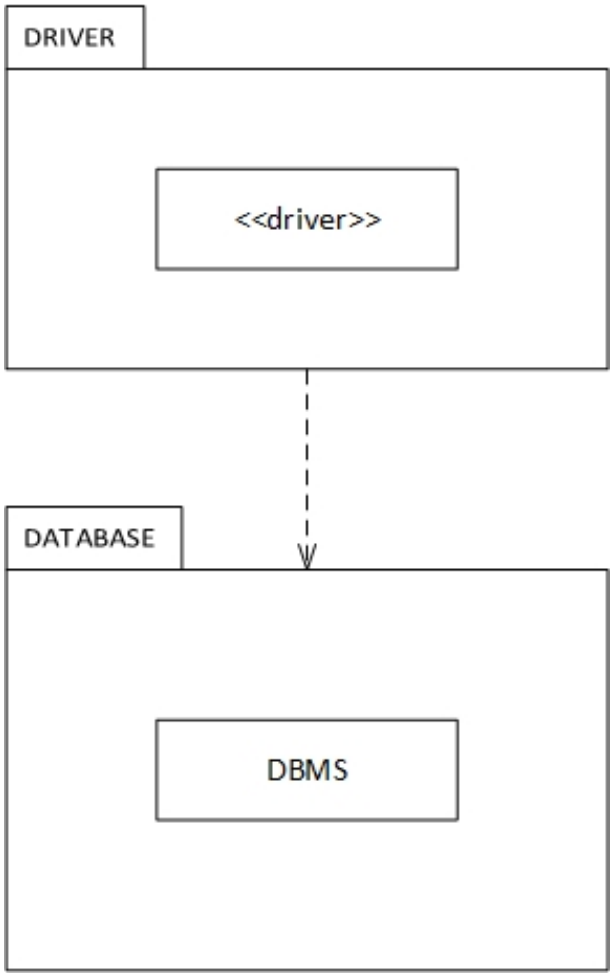
5.3 Integration and Testing Plan

5.3.1 Overview

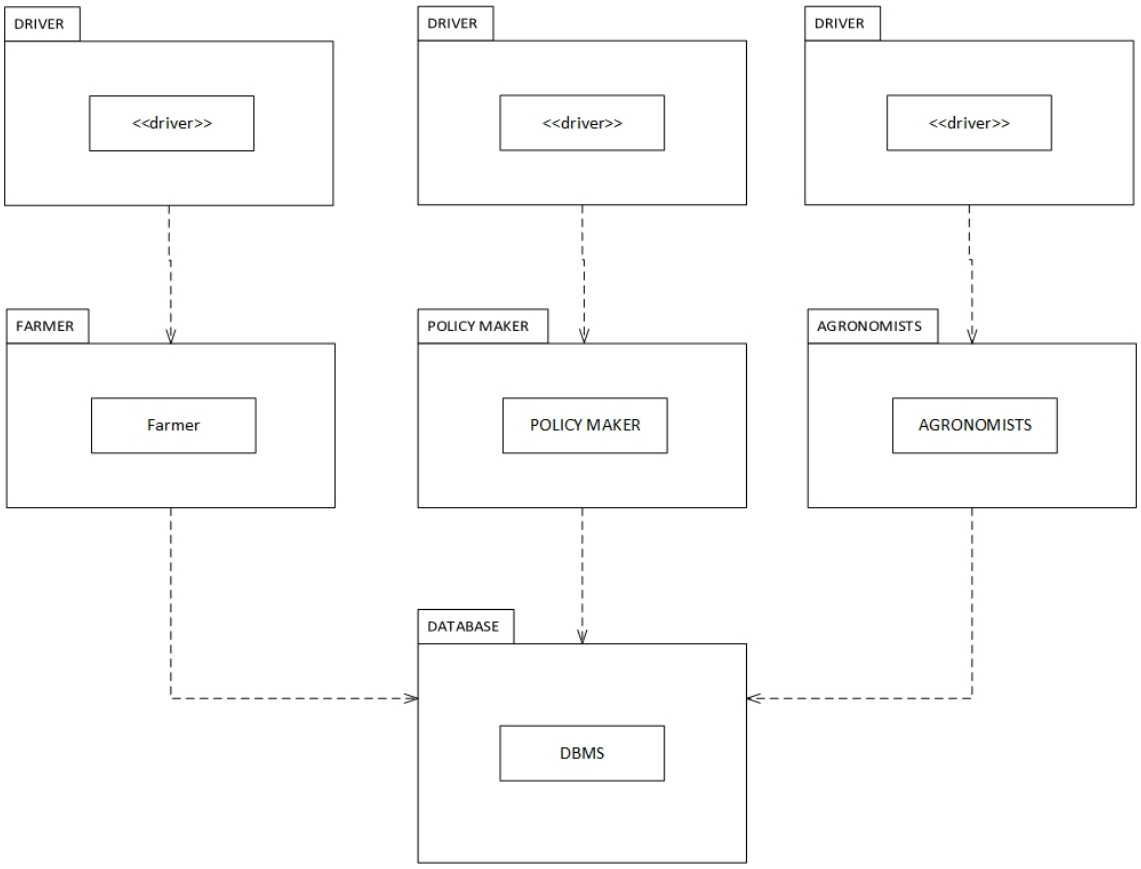
A component should be unit tested as soon as it is implemented; if necessary, a driver can be used in place of components that are not yet completely built. As a result of selecting a bottom-up method in the implementation phase, we should also select a bottom-up strategy for integration and testing. After a component's unit tests pass, it is integrated into the system and integration testing is performed: every interface that the component utilizes is tested. Due to the usage of a bottom-up method, this process concludes when the components at the top of the hierarchy are incorporated into the system. At that time, system tests may be used to test the entire system.

5.3.2 Plan

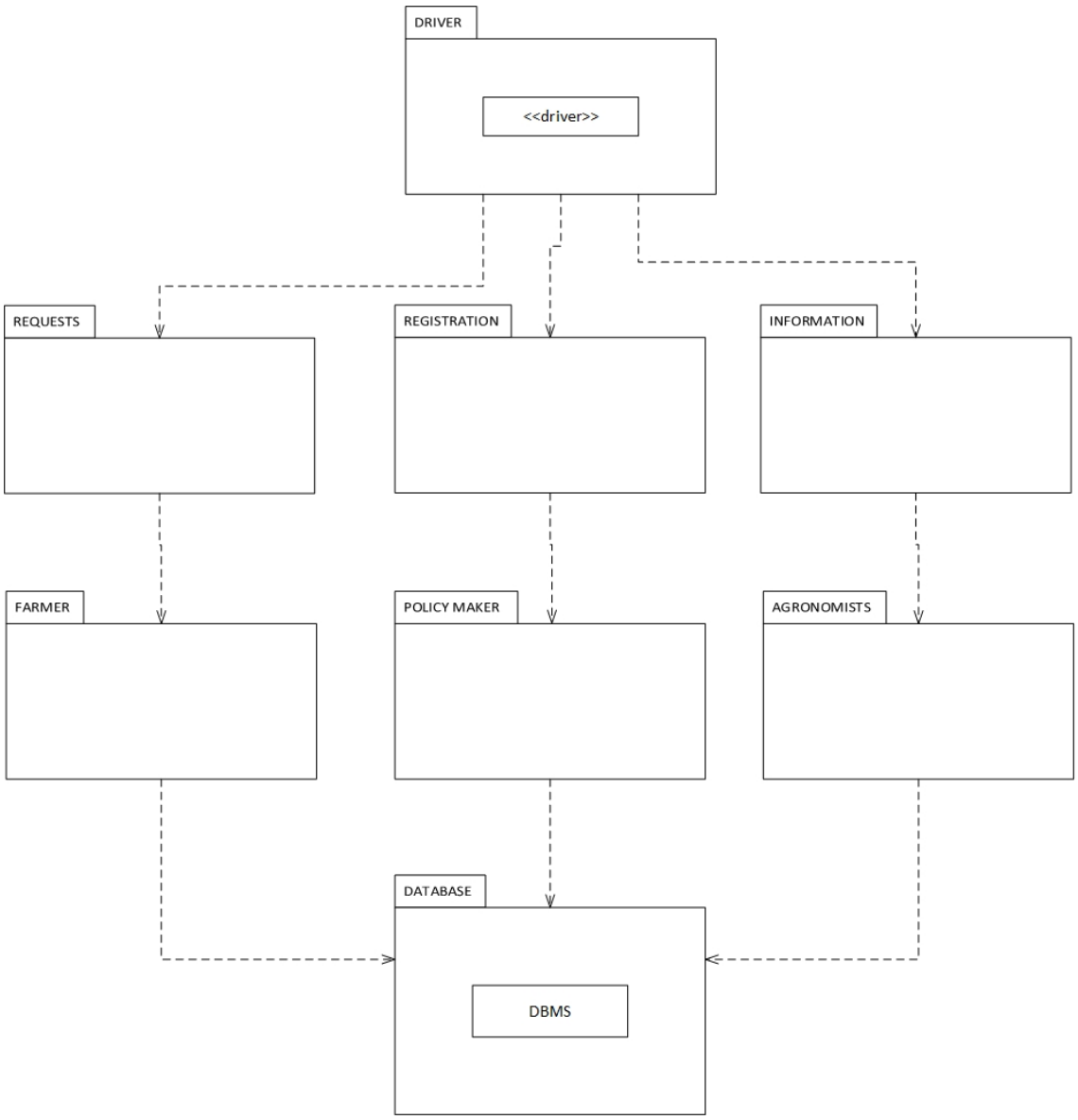
1. At first, the DBMS API has to be tested, in order to check for misconfiguration and correctness of the interface.



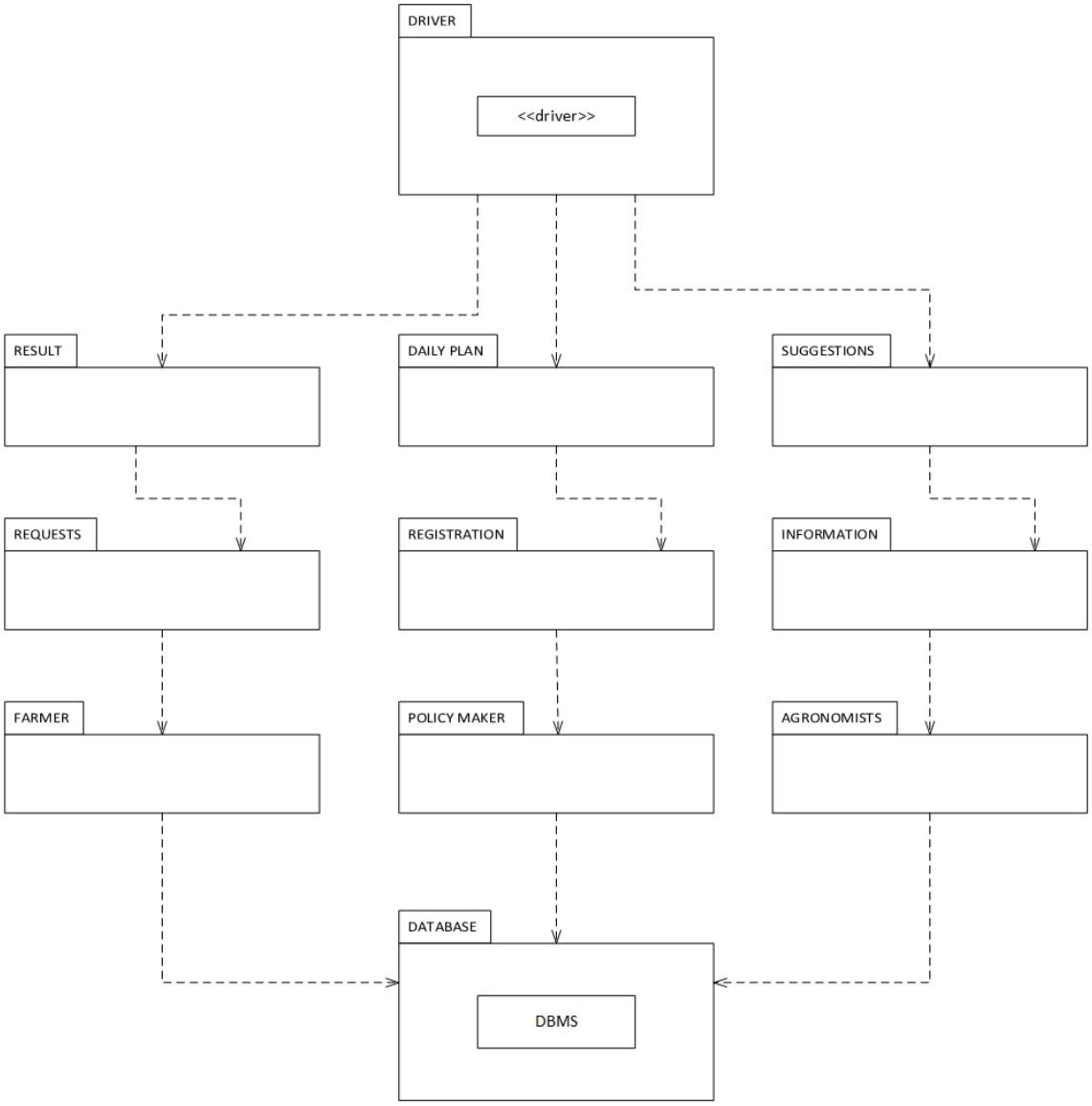
2. Then, the implementation goes on for Farmer, Policy Maker and Agronomists:



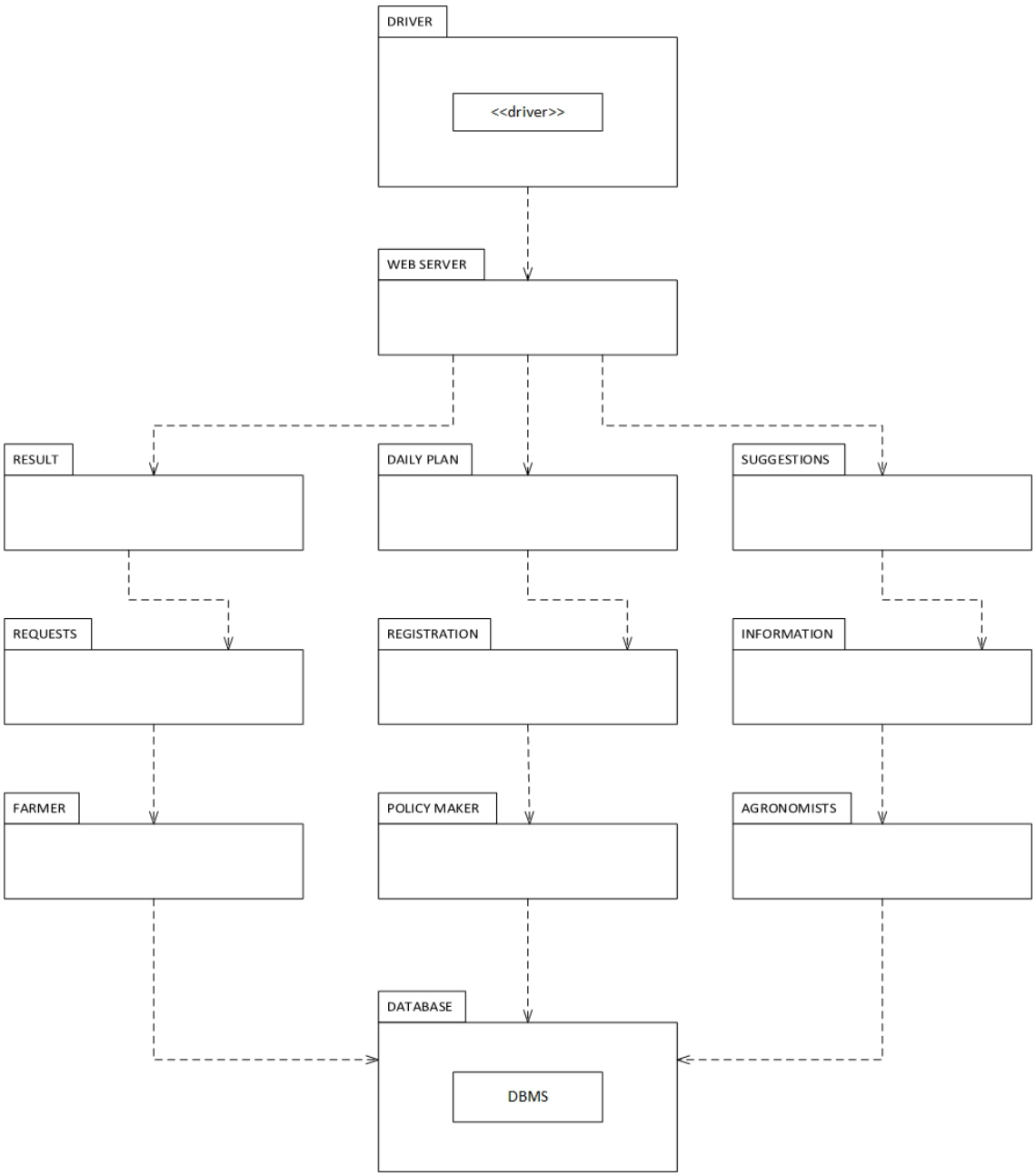
3. Then, Requests, Registration and Information components are integrated and tested.



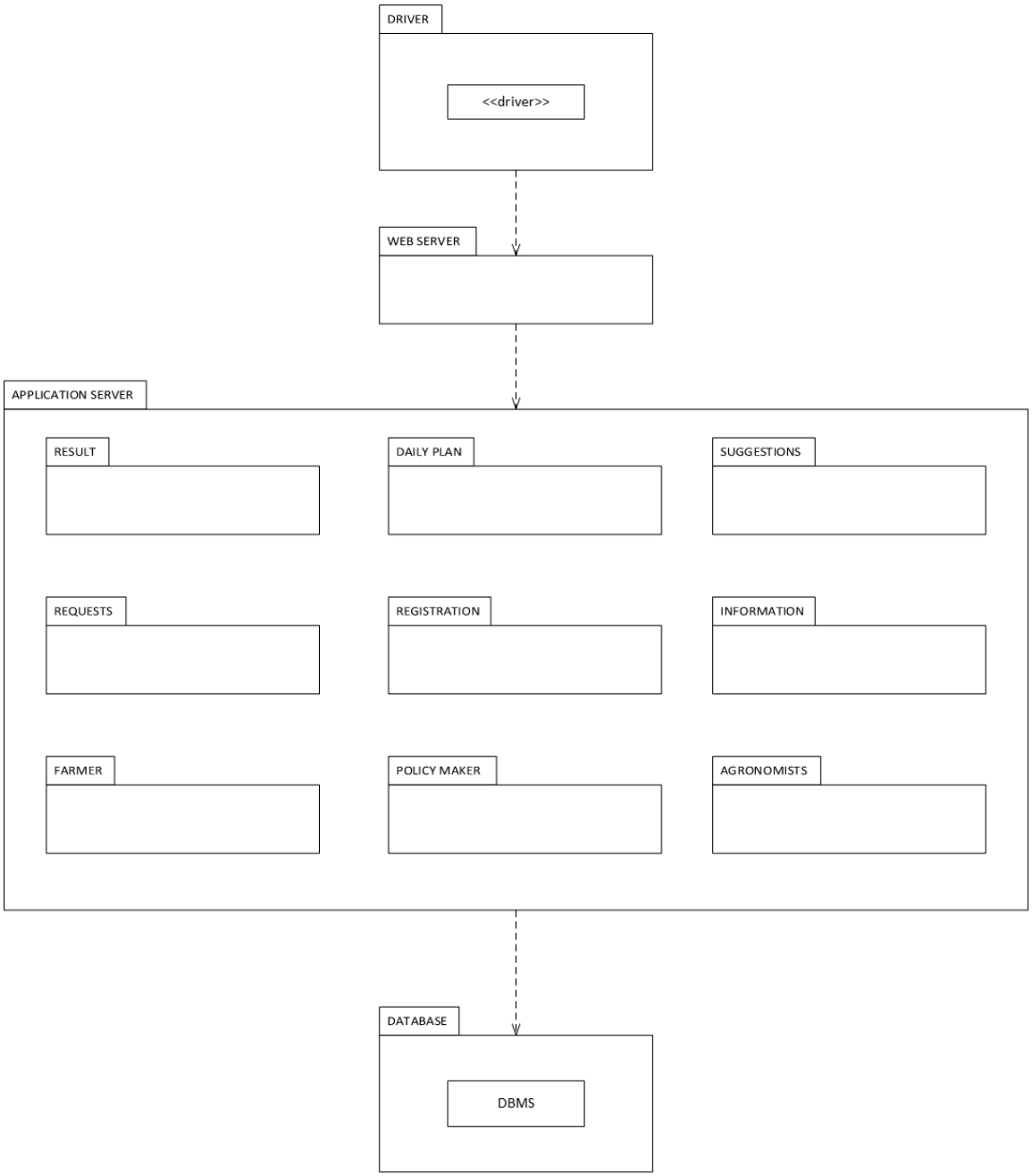
4. Later, the implementation of Results, Daily plan and its integration with Suggestion Service can begin.



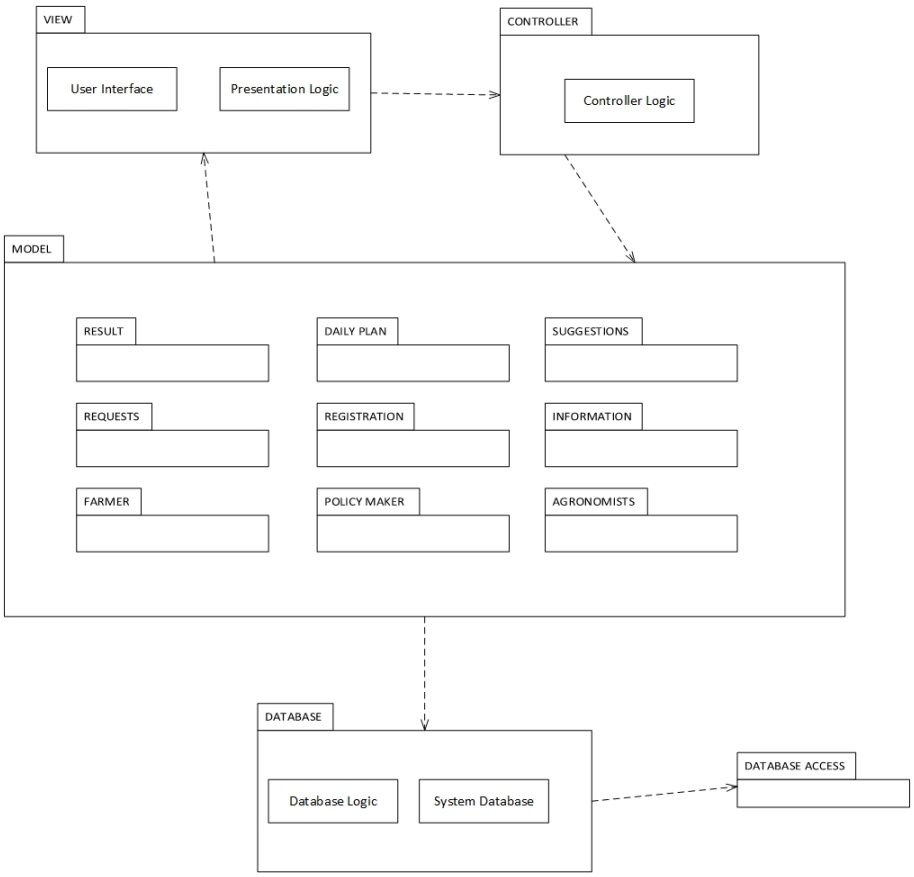
5. Later, the implementation of the connection with web server will begin.



6. Then, after the deployment of the Web Server component, its integration in the whole system is tested.



7. 7. Finally the interface view and controller components services will integrated and tested.



5.4 System Testing Plan

When the system is finished, it must be tested as a whole to ensure that all functional and non-functional criteria, as well as software system characteristics and performance requirements, are met:

- **Load testing**: This test enables you to learn how a system responds under predicted demand. In this scenario, we anticipate that the system will operate without any performance issues until it reaches its full capacity. The purpose of this test is to see if DREAM's responsiveness can be maintained even at the given maximum capacity settings.
- **Stress testing**: These tests aid in understanding the top limits of a system's capacity by simulating a load that is greater than the projected maximum. The purpose of this test is to determine whether the top limits of DREAM's system exceed the maximum capacity specified in RASD.
- **Performance testing**: This test is used to examine how a system performs in terms of responsiveness and stability under a specific workload. The purpose of performance testing in our context is to check and verify the system's quality qualities, such as scalability, dependability, and resource utilisation. This test demonstrates that it is also useful for evaluating cloud platform expenses, as they are closely tied to resource utilisation. As a result of the data obtained during the testing process, we can evaluate and even anticipate the costs associated with cloud platforms.

It is useful to have monitoring implemented during runtime to ensure that the system is performing as planned. Monitoring should consider both infrastructure-scoped statistics and application-scoped information. Because DREAM relies on cloud platforms to host its services, infrastructure-scoped statistics may make use of data supplied by the cloud platforms in use. Before releasing any feature or modification to the production environment, all updates are deployed in a staging environment that is as close to the production environment as feasible. After the core DREAM functionalities and those impacted by the upgrade have been validated to work properly, they are propagated to the production environment.

6. Effort spent

6.1 Cui Jiayan

Date	Effort spent (h)	Notes
06/12/2021	1.5	Introduction briefing
08/12/2021	1.0	Document setup
28/12/2021	1.5	Introduction
01/01/2022	6.0	User interface design
01/01/2022	1.0	Meeting
03/01/2022	1.5	Mappings between components and requirements
04/01/2022	2.0	Runtime view
07/01/2022	1.0	Alignment meeting
08/01/2022	1.0	Improvements to runtime view
08/01/2022	1.5	Component interfaces
09/01/2022	3.0	Alignment meeting
09/01/2022	1.0	Refactoring and general improvements
09/01/2022	3.0	DD finalization

6.2 Wang Yudong

Date	Effort spent (h)	Notes
06/12/2021	1.5	Introduction briefing
07/12/2021	0.5	Harmonization
01/01/2022	2.0	Overview
01/01/2022	1.0	Overview completion
01/01/2022	1.0	Meeting
02/01/2022	3.0	Deployment view
03/01/2022	2.0	Design decisions
04/01/2022	1.0	Briefing
06/01/2022	1.0	Harmonization
07/01/2022	4.0	Component view
08/01/2022	1.0	Component view
09/01/2022	1.0	Alignment meeting
09/01/2022	1.0	Improvements to deployment and component view
09/01/2022	3.0	DD finalization