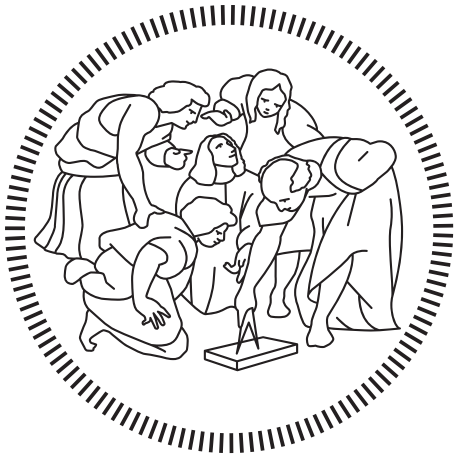

POLITECNICO DI MILANO

Computer Science and Engineering

Requirements Analysis and Specifications Document

DREAM - Data-dRiven PrEdictive FArMing in Telangana

Software Engineering 2 Project

Academic year 2021 - 2022

December 22, 2021

Version 1.0

Authors:
Kinga Marek
Józef Piechaczek
Mariusz Wiśniewski

Professor:
Damian Andrew Tamburri

Contents

Chapter 1

Introduction

1.1 Purpose

Over 58 percent of rural Indian households rely on agriculture as their primary source of income, with 80 percent of them being smallholder farmers with less than 2 hectares of land. Furthermore, more than 20 percent of smallholder agricultural households live in poverty. Food consumption is anticipated to climb by 59 percent to 98 percent by 2050, according to the Harvard Business Review [1].

With climate change posing a serious danger to agriculture, a complete overhaul of the system that transports food from farmers to our tables is required. On the top of that, the coronavirus pandemic created a huge disruption in food supply chains, exposing the weak parts of small-scale farmers as well as the significance of constructing resilient food systems.

Telangana is India's 11th biggest state, having a land area of 122,077km^2 and a population of 35,193,978 people (data from 2011) [4]. The project's goal is to help Telangana's government promote data-driven policy-making in the state by designing, developing, and demonstrating anticipatory governance models for food systems utilizing digital public goods and community-centric approaches [2].

1.1.1 Goals

G1. Improve farmers performance by providing them with personalized suggestions.

G2. Acquire, combine, and visualize data from external systems.

G3. Facilitate performance assessment of the farmers.

G4. Promote regular farms' visits by agronomists, depending on the type of problems they face.

G5. Enable agronomists to exchange information with farmers.

G6. Enable farmers to exchange their knowledge.

1.2 Scope

Data dRiven PrEdictive FArMing (DREAM) is a project that intends to redesign the food production process in the Telangana area in order to create more resilient agricultural systems that will be required to meet the region's rising food demand. To address such a broad issue, it is critical to bring together individuals from many professions and backgrounds; hence, the participation of groups such as stakeholders, policy makers, farmers, analysts, and agronomists is desirable.

Data acquisition and combination can be viewed as the first feasible step in achieving the goal. This information might be gathered from farmers, specialized sensors installed on the ground that detect soil humidity, and government agronomists who visit fields on a regular basis.

The system aims to enable policy makers to identify farmers who perform well, especially under tough weather conditions, farmers who perform particularly badly and require assistance, and understand if agronomist-led steering initiatives yield substantial outcomes. This is going to be achieved by continuous assessment based on the visualizations of the acquired data.

Another group that would benefit from the system are the farmers, since it will allow them to exchange good practices between each other and request help when needed. All of that will be supplemented with visualizations of relevant data, including suggestions and weather forecasts.

The project also intends to facilitate the work of agronomists by creating a daily plan for field visits, giving statistics on farmers' performance, and informing them about incoming assistance requests.

The system attempts to address a highly difficult topic, which necessitates the coordination of numerous individuals. All of these criteria requires the system to be very easy to use, intuitive, and straightforward, so that everyone can get the most out of it.

1.2.1 World Phenomena

ID	Phenomena
WP1.	Telangana's government collects information concerning short-term and long-term meteorological forecasts [3].
WP2.	Water irrigation systems collect information about the amount of water used by each farm.
WP3.	Sensors collect information about the humidity of the soil at a given farm.
WP4.	An agronomist visits a farm.
WP5.	A farmer produces crops on his farm.
WP6.	A farmer experiences issues on his farm.
WP7.	A user forgets his password.
WP8.	An external company provides a farm with a water irrigation system.
WP9.	An external company provides a farm with a sensor system.

1.2.2 Shared Phenomena

ID	Content	Controlled by
SP1.	A user logs into the system.	World
SP2.	An unregistered user creates an account in the system.	World
SP3.	The system creates an instance of a farm during a registration process of a farmer.	Machine
SP4.	The system reads and stores information about short-term and long-term meteorological forecasts provided by the API of Telangana's government.	Machine
SP5.	The system reads and stores information about the amount of water used by each farm, provided by the water irrigation system's API.	Machine
SP6.	The system reads and stores the data collected by the sensors at a given farm, provided by the sensor system's API.	Machine
SP7.	The system sends a request for help to well performing farmers and agronomists in a given mandal.	Machine
SP8.	The system shows a filterable list of farmers.	Machine
SP9.	The system shows farmer's summary.	Machine
SP10.	A user requests password reset.	World
SP11.	The system sends an email with password reset link.	Machine
SP12.	A user resets the password using a link obtained in an email.	World
SP13.	A registered user deletes his account.	World
SP14.	A farmer manages his production data in the system.	World
SP15.	A farmer requests for help in the system.	World
SP16.	A farmer deletes his help request in the system.	World
SP17.	A farmer with a positive note responds to a request for help in the system.	World
SP18.	A farmer with a positive note deletes his help response in the system.	World
SP19.	A farmer sees a list of all the forum threads.	World
SP20.	A farmer reads a forum thread.	World
SP21.	A farmer creates a thread on the forum.	World
SP22.	A farmer removes his thread from the forum.	World
SP23.	A farmer comments in a given thread.	World
SP24.	A farmer removes his comment from a given thread.	World
SP25.	An agronomist saves comments collected during a farm visit in the system.	World

SP26.	The agronomist inserts into the system personalized suggestions for the farmers inside his area of responsibility.	World
SP27.	An agronomist responds to a request for help in the system.	World
SP28.	An agronomist deletes a help response in the system.	World
SP29.	An agronomist manages his area of responsibility.	World
SP30.	The system visualizes data about farmers with a specific note in the agronomist's area of responsibility.	Machine
SP31.	The system shows the daily plan of an agronomist.	Machine
SP32.	An agronomist manages the daily plan.	World
SP33.	An agronomist submits the execution state of the daily plan.	World
SP34.	The system generates farm visits for an agronomist in a given area of responsibility.	Machine
SP35.	A policy maker assesses a farmer's performance, giving him a negative, neutral or positive note and saves it in the system.	World
SP36.	A policy maker understands whether activities performed by agronomists and farmers helped a farmer with a negative note based on the data from the system.	World

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Expression	Definition
Water irrigation system	Collection of external devices which gather information about the amount of water used on a given farm. Collected data are available via an API.
Sensor/Humidity sensor	External device which collects information about the humidity of soil on a given farm.
Sensor system	Collection of all the sensors on a farm. Collected data are available via an API.
Farmer's summary	Summary of farmer and farm data. Contains: information about farmer and his farm, note history, short-term weather forecasts, production data, farm visits, weather forecast history, soil humidity data, water usage, and help requests.
Farmer's note	Evaluation note given by a governmental policy maker based on a farmer's performance. Can include a type of problem, that the farmer is facing. Can be negative, neutral, or positive.

Farm data	Data concerning a farm, inserted by a farmer during account creation. Consist of water irrigation system's ID, sensor system's ID and farm's address (address line 1, address line 2, postal code, city and mandal).
Personalized suggestion	Suggestions proposed to the farmers, based on a mandal and types of production of a given farm. Suggestions are inserted into the system by agronomists.
Production type	Kind of plants being produced on Telangana's farms.
Production data	Information provided by a farmer about his production. It lists production types and quantities (in kg) generated per type.
Mandal	A local government area in India. Part of a district, which in turn is a part of a state.
Dashboard	The first screen that appears to the user after logging in to the system.
Area of responsibility	Set of mandals that an agronomist is responsible for.
Daily plan	A daily schedule for visiting farms in a certain area of responsibility.
Casual visit	A visit automatically arranged by the system. There are two casual visits to each farm every year.
Request for help, Help request	An issue created by a farmer. It contains a topic, description and a problem. The system automatically forwards it to well-performing farmers and agronomists in the same mandal.
Peak hours	6 a.m. to 9 a.m., and 5 p.m. to 8 p.m. in the GMT+5:30 time zone.
External systems	Systems that are not a part of DREAM and collect various farm-specific data and share them via an API. Following external systems are used: Weather Forecast System, Water Irrigation System, Sensor System.
Water irrigation system's ID	Number which uniquely identifies a water irrigation system. It is necessary for definition of the external system's API endpoint. The number is provided during installation of the water irrigation system.
Sensor system's ID	Number which uniquely identifies a sensor system. It is necessary for definition of the external system's API endpoint. The number is provided during installation of the sensor system.

1.3.2 Acronyms

Acronyms	Expression
DREAM	Data-dRiven PrEdictive FArMing in Telangana
RASD	Requirements Analysis and Specifications Document
G	Goal
WP	World Phenomena
SP	Shared Phenomena
R	Requirement
A	Domain assumption
API	Application Programming Interface
PBKDF2	Password-Based Key Derivation Function 2
HMACSHA1	Hash-based Message Authentication Code (HMAC) using the SHA1 hash function

1.3.3 Abbreviations

Abbreviations	Expression
a.m.	ante meridiem
p.m.	post meridiem
e.g.	exempli gratia

1.4 Revision History

Date	Revision	Notes
22.12.2021	v.1.0	First release.

1.5 Reference Documents

[1] Maarten Elferink and Florian Schierhorn. *Global Demand for Food Is Rising. Can We Meet It?* https://hbr.org/2016/04/global-demand-for-food-is-rising-can-we-meet-it. Apr. 2016. (Visited on 11/21/2021).

[2] Elisabetta Di Nitto, Matteo Rossi, and Damian Tamburri. *A.Y. 2021-2022 Software Engineering 2 Requirement Engineering and Design Project: goal, schedule, and rules*. Oct. 2021.

[3] Telangana State Development Planning Society. *Automatic Weather Stations*. https://tsdps.telangana.gov.in/aws.jsp. (Visited on 11/21/2021).

[4] *Telangana — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Telangana&oldid=1055881688. 2021. (Visited on 11/21/2021).

1.6 Document Structure

1. **Introduction**: describes the general purpose of the project, as well as its scope. Furthermore, it defines specific definitions, acronyms, and abbreviations used throughout the document.
2. **Overall Description**: provides a general description of the system, with the focus on its perspective, functions, characteristics, and assumptions.
3. **Specific Requirements**: focuses on external interface, functional, and performance requirements. Design constraints and software system attributes are defined in this chapter. Specific use cases are depicted on *UML* diagrams.
4. **Formal Analysis using Alloy**: describes the formal modeling activity, and the model itself. Additionally, it provides verification of critical aspects of the system and an example of the generated world.
5. **Effort Spent**: depicts information about the project contribution of each group member.
6. **References**

Chapter 2

Overall Description

2.1 Product Perspective

This section contains scenarios and further information on the shared phenomena, as well as a domain model consisting of a class diagram and state charts.

2.1.1 Class diagram

The class diagram presented in figure 2.1 provides a high-level overview of the system. Below are brief descriptions of the purpose of each class.

- User classes:
  - **User:** an abstraction of a registered user, being a base of all the other user classes.
  - **Agronomist:** a class modeling an agronomist described in section 2.3.1.
  - **Farmer:** identifies a farmer described in section 2.3.2.
  - **PolicyMaker:** a class modeling a policy maker whose description is to be found in section 2.3.3.
- Enumerators:
  - **VisitState:** describes the status of agronomist's visit to a farm.
  - **Note:** contains possible notes given to a farmer for his performance by a policy maker.
  - **WeatherType:** contains all weather condition types handled by the system.
  - **VisitReason:** introduces possible reasons for a visit.

  Responses from external systems:
  - **SensorSystemResponse:** models a response from the sensors available on a farm.

  - **WaterIrrigationSystemResponse:** models a response from the water irrigation system on a farm.
  - **WeatherSystemResponse:** models a response from the system providing meteorological forecasts.

- Others:

  - **Farm:** identifies a farm.
  - **Production:** a class modelling production on a given farm. It represents the amount of each type of crops produced.
  - **ProductionType:** identifies a type of produced crops.
  - **Mandal:** identifies a local government area in India.
  - **Visit:** a class modelling agronomist's visit on a farm.
  - **HelpRequest:** a class representing a farmer's request for help.
  - **HelpResponse:** a class representing a response to a request for help.
  - **ForumThread:** identifies a discussion forum created by a farmer.
  - **ForumComment:** identifies comments posted on a discussion forum.
  - **ProblemType:** a class representing a mapping of a problem type to the number of additional visits necessary in case of its occurrence.
  - **FarmerNote:** a class modelling a note given by a policy maker to a farmer together with the date of its assignment.
  - **Suggestion:** identifies suggestions based on a production type and farmer's location.

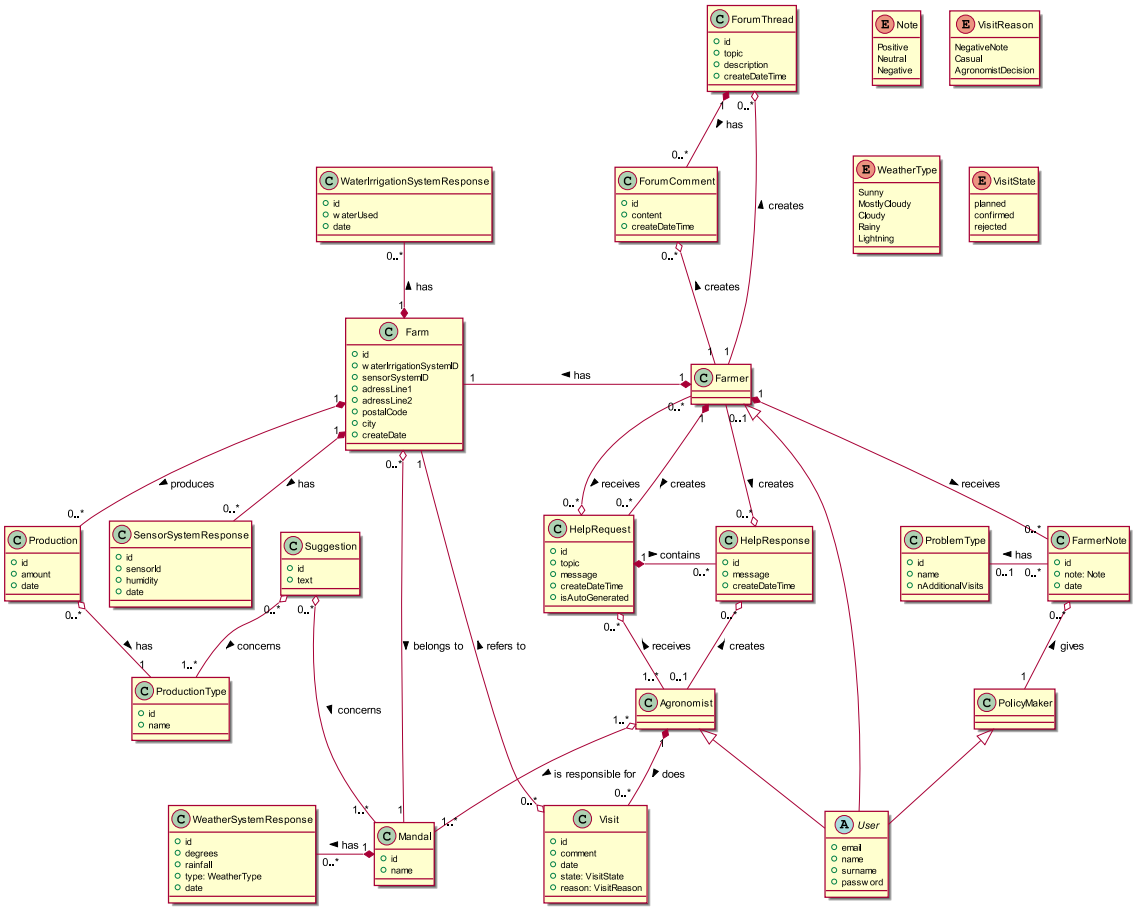

Figure 2.1: Class diagram

2.1.2 State diagrams

The event-ordered behavior of an object of the *VisitState* class is presented in the figure 2.2.

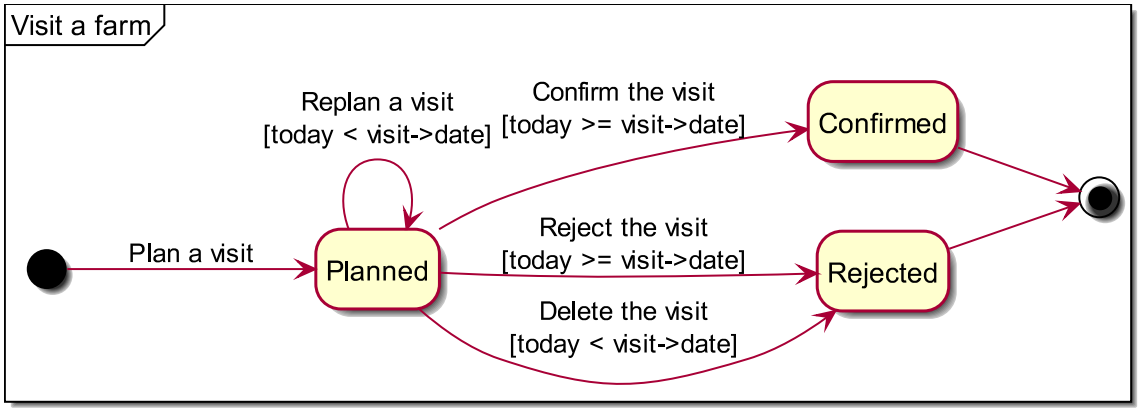

Figure 2.2: State diagram presenting an agronomist's visit on a farm

There are some constraints to be read from the diagram:

- Rejecting a visit means changing its state to *Rejected* and is possible only on the day or after its planned date. It can happen only during plan submission.
- Deleting a visit means changing its state to *Rejected* and is possible only before its date.
- A visit can be confirmed only on the day or after its planned date. It can happen only during plan submission.
- An update of a visit is available only for future visits. Furthermore, it is only possible to change its date. In case an agronomist wants to change other parameters, e.g. the farm to be visited, the visit has to be deleted, and then he needs to create a new one.

2.1.3 Scenarios

Farmer's performance assessment by a policy maker

John, a policy maker, logs in to the application to see how farmers are performing. He goes through the list of farmers and notices that despite very dry weather conditions in the region, the Joe Sharma's production of carrots has significantly increased. John gives Sharma's a positive note, and then he continues to look at the list of farmers. He spots that the production of Tamia Anand and Amar Khatri has plunged. He thinks it may be an effect of droughts that were reported in the system. He gives both farmers a negative note, hoping that other farmers and agronomists will be able to help them.

An agronomist and a farmer help a farmer who obtained a negative note

Dream automatically creates and delegates the requests for help about poorly performing farmers to both the agronomists in the given area of responsibility and farmers with positive notes. As a result, an agronomist, Rajesh Khanna, receives requests that indicate the farmers with negative note. Rajesh looks through the farmers' summaries, to determine any possible reasons for the farmers' bad performance. In the case of Tamia Anand's, he notices that her irrigation system entries from the last month show understated values. Rajesh uses a response option to advise Tamia to check if the system is working properly.

The same requests are also obtained by a farmer, Trisha Gupta, who recently obtained a positive note. She shares some general advices about dealing with adverse weather conditions in the mandal using the response option.

Policy maker checks whether a farmer obtained proper help

Shaila, a policy maker, is already logged in to the application. She wants to see if farmers with negative notes are obtaining proper help. Therefore, she looks at the farmer's summary belonging to Amar Khatri, who was marked with a negative note. She reads the advice provided by both agronomists working in the area and Joe Sharma, who is a farmer with a positive note. Joe Sharma shared some best practices on how to deal with droughts in the farm. Shaila wants to wait to see if the production of Amar Khatri will improve in the following month, but is happy to see that the farmer obtained meaningful advices.

A farmer reads individualized recommendations

Ishaan Patel grows potatoes and wants to prepare a planting plan of his farm. For that reason, he opens the application and looks for a weather forecast in his region. Ishaan finds out that it is going to rain next weekend, but afterwards the weather will be perfect for planting. Hence, he decides to start planting next Monday. Next to the weather forecasts, among farmer's relevant data, Ishaan reads a recommendation about potassium salt that can be used to increase the potato crops. As a result, Ishaan thinks about including potassium salt fertilization in his planting plan.

A farmer inserts production data

Shalene Reddy assessed that she produced 40 tons of rice on her farm in the last month. She logs in to the application and updates the production data. She looks through the history of her production entries to verify whether she is on the right track.

A farmer requests for help

Nalin Laghari experiences difficulties on his cotton farm. He is striving to fight off bollworms. Thus, he wants to create a request for help in the application. He opens the application and describes the details of his problems with bollworms.

The application automatically delegates a request for help to both the agronomists in the given area of responsibility and well-performing farmers nearby. As a result, an agronomist, Devraj Jain, receives a request for help. Devraj knows some theory as well as several practices that farmers used before to deal with bollworms, so he shares some hints in a response to the request. The same request is also obtained by a farmer, Joe Sharma. Joe recommends one of the insecticides in the response, since he successfully fought off bollworms a year ago.

A farmer creates a discussion forum

Paidi Jairaj has a large carrot farm in Jagtial. Lately, he heard some rumors about potato farming becoming more and more profitable. He is wondering if it is worth switching his type of production to another type of crops in the next season. He wants to reach other farmers in his mandal to get to know their opinion. To do this, Paidi logs into DREAM application and chooses the option to create a discussion thread, in which he specifies the topic and description of his dilemma.

Gajam Anjaiah, a potato farmer from Jagtial, logs in to DREAM application to check, if there are any interesting discussions on the discussion forum. He sees Paidi's thread, and knowing a lot about potatoes production, creates a comment in the thread, containing various advices and remarks concerning potato's production.

An agronomist manages daily plan

Zakir Hussain, a governmental agronomist, wants to visualize and check his daily plan. He logs in to DREAM app, chooses the option to see his plan, and then clicks on the following day. After reading the generated plan, he realizes that he will be driving near Syed Nayeemuddin's farm, who recently asked for help. He decides to alter his plan to visit Syed's farm. He chooses an option to add a new visit to his the plan, types Syed's name in a search bar and chooses him from the list which contains suggestions of farmers in Zakir's area of responsibility. Lastly, Zakir looks at his updated daily plan.

An agronomist sets daily plan's execution state

Zakir Hussain, a governmental agronomist, has finished today's farm visits. During visits, he collected some comments about conditions of the farms. Unfortunately, he could not visit one of the farms that was supposed to be visited in terms of a casual visit. Zakir logs into DREAM app, chooses the option to see daily plan, and then clicks on the current day. In the daily plan view, he clicks on a button to submit the daily plan's execution state. A pop-up appears which tells him to specify deviations from the plan by indicating which farms he has and has not visited. For every farm he has visited, Zakir fills a short comment. After completing the form, he confirms the daily plan's execution.

The application automatically plans a new casual visit in approximately 5 days to the farm that was not visited, and in approximately half of a year for the farms he paid a casual visit to.

An agronomist responds to a farmer's request for help

Sachindranath Mitra, a governmental agronomist, wants to check if any farmer in his area of responsibility needs help. He logs into DREAM app and sees a new help request in the dashboard from Paidi Jairaj. He chooses the option to manage help requests and clicks on Paidi's request. A pop-up appears, which allows him to provide some advice. After filling the form, the response is sent to the farmer.

An agronomist looks at the list of well performing farmers

Jarnail Singh, a governmental agronomist, wants to check if there is any new well-performing farmer in his area of responsibility. He logs into DREAM app, and on the dashboard he sees a quick summary, containing the weather forecast for the day and some best performing farmers. He clicks on the option to see full summary, which allows him to see a complete list of all the well-performing farmers in the area of responsibility.

2.2 Product Functions

This section provides a summary of the major functions that the system will introduce to achieve the goals outlined in the section 1.1.1.

2.2.1 Registration

The system allows for registration of three different types of users: agronomists, farmers, and policy makers. Characteristics of each user are described in the section 2.3.

Agronomist

- Inserts name, surname, email and password.
- Inserts his area of responsibility.

Farmer

- Inserts name, surname, email, and password.
- Inserts water irrigation system's ID and sensor system's ID.
- Inserts farm address, containing address line 1, address line 2, postal code, city, and mandal.

Policy maker

- Inserts name, surname, email, and password.

Additionally, the system introduces a functionality to reset a forgotten password. The user needs to press on a dedicated button and provide his email address. He will later receive a link to a web page where he will be able to introduce a new password.

2.2.2 Performance assessment of the farmers

DREAM system utilizes external services such as water irrigation systems and sensors measuring the humidity of the soil to extract valuable data. Based on that, together with production data which is updated by a farmer, weather conditions, note history, visit history and help requests, a policy maker is able to assess the performance of a farmer by assigning him a positive, negative, or neutral note.

2.2.3 Update of the production data

The system allows a farmer to update his production data. It is required to specify the type of crops as well as their quantity produced during the entire month. This data is further used for the performance assessment of each farmer.

2.2.4 Request for help

Using DREAM enables farmers to request assistance and advice from agronomists and other farmers in a convenient way. The system asks them to create an issue by describing their problems, and subsequently sends requests to all the agronomists and well-performing farmers from the same mandal. Then each of the recipients can provide help. The track of the issue remains visible only to the people involved as well as the policy makers.

2.2.5 Coordinate agronomist's daily plan

Each farm should be inspected at least twice a year, but those that are underperforming should be visited more frequently, depending on the nature of the problem. The system automates these visits by developing a daily plan for each agronomist, which is accessible via the dashboard. Furthermore, at the end of each day, an agronomist can validate the completion of the daily plan or identify deviations from it.

2.2.6 Create discussion forum

The system provides also a possibility for a farmer to freely create discussion forums with other farmers. Such threads do not have any restrictions on their topic. They can include suggestions, queries, and even off-topic material. Each farmer can participate in a certain thread by leaving a comment. All forum threads are visible to all the registered farmers.

2.2.7 Visualize farmer's summary

The system enables farmers to display data relevant to them, such as weather forecasts and tailored recommendations for certain crops to plant or fertilizers to utilize, based on their region and kind of production. This information will be visible on the main dashboard after logging in to the system.

2.3 User Characteristics

DREAM is a project whose key component is data acquisition and integration. All of this is used to support the work of three distinct types of registered actors: agronomists, farmers, and policy makers. Additionally, there is a description of an unregistered user provided.

2.3.1 Agronomist

In the realm of agriculture, an agronomist serves as an intermediary between farmers and policy makers. He advises farmers on soil management and crop production.

Governmental agronomists in the Telangana region visit farms in their areas of responsibility on a frequent basis to acquire information regarding farmers' performance.

The system provides many functionalities in order to make agronomists' job more convenient. Firstly, it allows an agronomist to insert his area of responsibility. Then, it provides them with visualizations depicting best performing farmers in the region as well as weather forecasts. To help manage their schedule, it creates an interactive daily plan to visit farms for them, where they can confirm its execution and also specify any deviations from it. Additionally, the system significantly facilitates their contact with the farmers by showing them incoming help requests and providing an easy way to respond to them directly. Lastly, it allows them to manage the list of suggestions for mandals in their area of responsibility that are later displayed to the farmers.

2.3.2 Farmer

This actor is directly involved in Telangana's food production. He owns a farm where he raises crops that are later distributed around the region.

During the production process, a farmer faces many difficulties caused by external factors such as weather conditions.

To assist a farmer in accomplishing his objective, the system allows him to manage his production data. Moreover, it provides information important to him, such as tailored advice about which crops to plant or which fertilizers to use depending on his location and kind of production, as well as weather forecasts. Furthermore, it enables him to create discussion forums, where he could share his difficulties and solicit recommendations or assistance from agronomists and other farmers.

2.3.3 Policy maker

A member of Telangana's governmental department who is in charge of developing new rules and legislation pertaining to food production.

His primary responsibility is to ensure that new policies help to increase food production by monitoring farmers' productivity data.

The system's purpose is to help them identify which farmers are performing well and which are not. With this kind of help, a policymaker can more correctly assign notes to farmer's performance and understand if the steering initiatives carried out by agronomists with the support of excellent farmers create major outcomes thanks to the data obtained by the system.

2.3.4 Unregistered user

An individual belonging to one of the aforementioned groups who is not yet registered in the system.

2.4 Assumptions, Dependencies, and Constraints

2.4.1 Assumptions

ID	Assumption
A1.	Each farmer possess exactly one farm.
A2.	Each farm belongs to exactly one farmer.
A3.	Each farm has at least one humidity sensor.
A4.	Each farm has a water irrigation system.
A5.	Each farm is inside exactly one mandal.
A6.	Each farmer reliably updates his production data each month.

A7. Weather is consistent in a given mandal.

A8. Agronomist's area of responsibility contains at least one mandal.

A9. Each registered user can be only one of the actors: an agronomist, a farmer, or a policy maker.

A10. The information provided by a user during registration process is valid.

A11. Well-performing farmers and agronomists are eager to help farmers with negative notes.

A12. Suggestions created by agronomists are relevant and helpful for the farmers.

A13. Each mandal has at least one agronomist who is responsible for the farms inside that area.

A14. Based on the farmer's summary, a policy maker is able to understand whether the help given by farmers and agronomists produces significant results.

A15. Farmers create meaningful and not offensive threads and comments on the forum.

A16. Farmers and agronomists create meaningful and not offensive help requests and responses.

A17. External systems are reliable and highly available.

A18. Every sensor system has a unique ID assigned by the sensor system provider, which allow accessing the water usage data via an API.

A19. Every water irrigation system has a unique ID assigned by the water irrigation system provider, which allow accessing the soil humidity data via an API.

A20. Production types are predefined.

A21. Problem types are predefined.

Chapter 3

Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

DREAM is a web application. It might be used on various devices, regardless of their form factor. However, the application is predicted to be primarily used on desktops. As such, the following mock-ups show visualizations of the desktop version of the application. Due to the large number of possible views and combinations, only the most important ones are included. The goal of the mock-ups is to visualize functionality, as such the final views may differ in details such as colors or the element placement.

Unregistered user

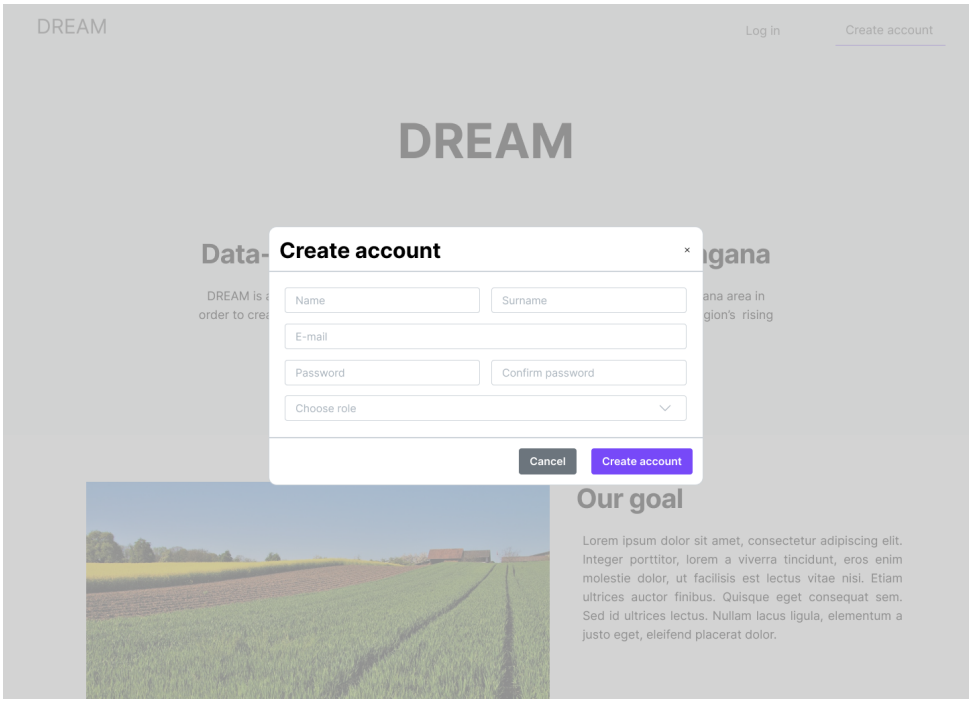

Figure 3.1: User registration

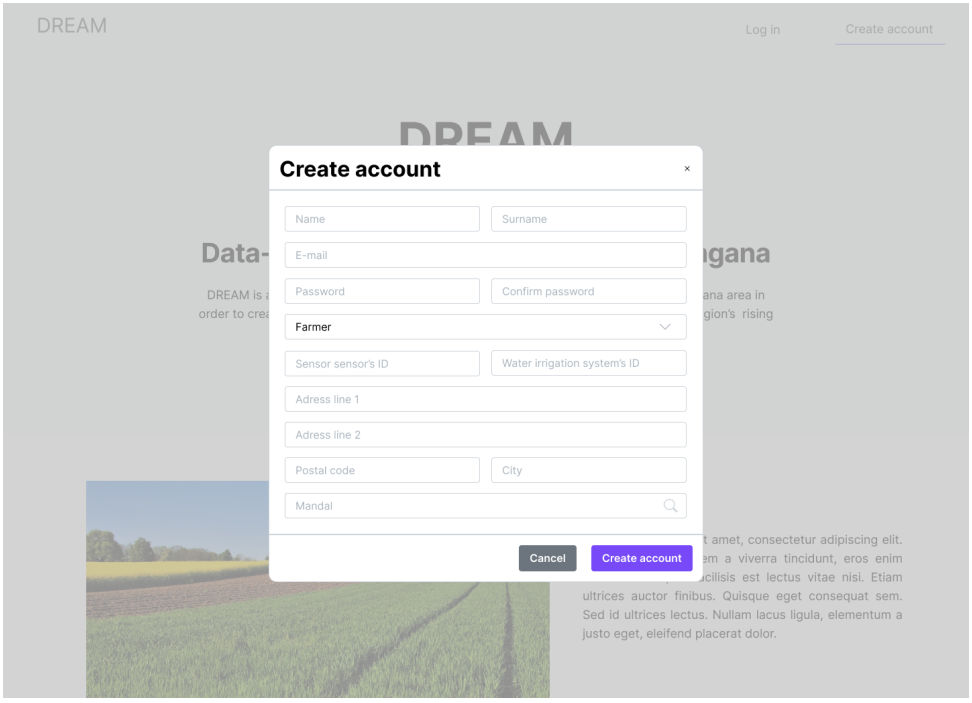

Figure 3.2: Farmer registration

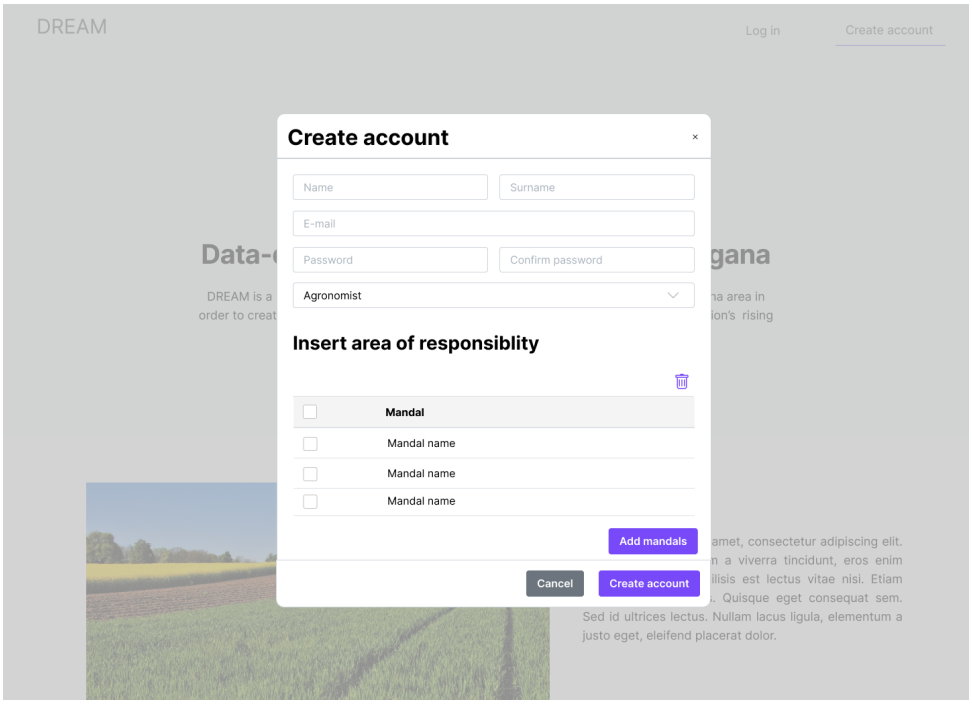

Figure 3.3: Agronomist registration

Farmer

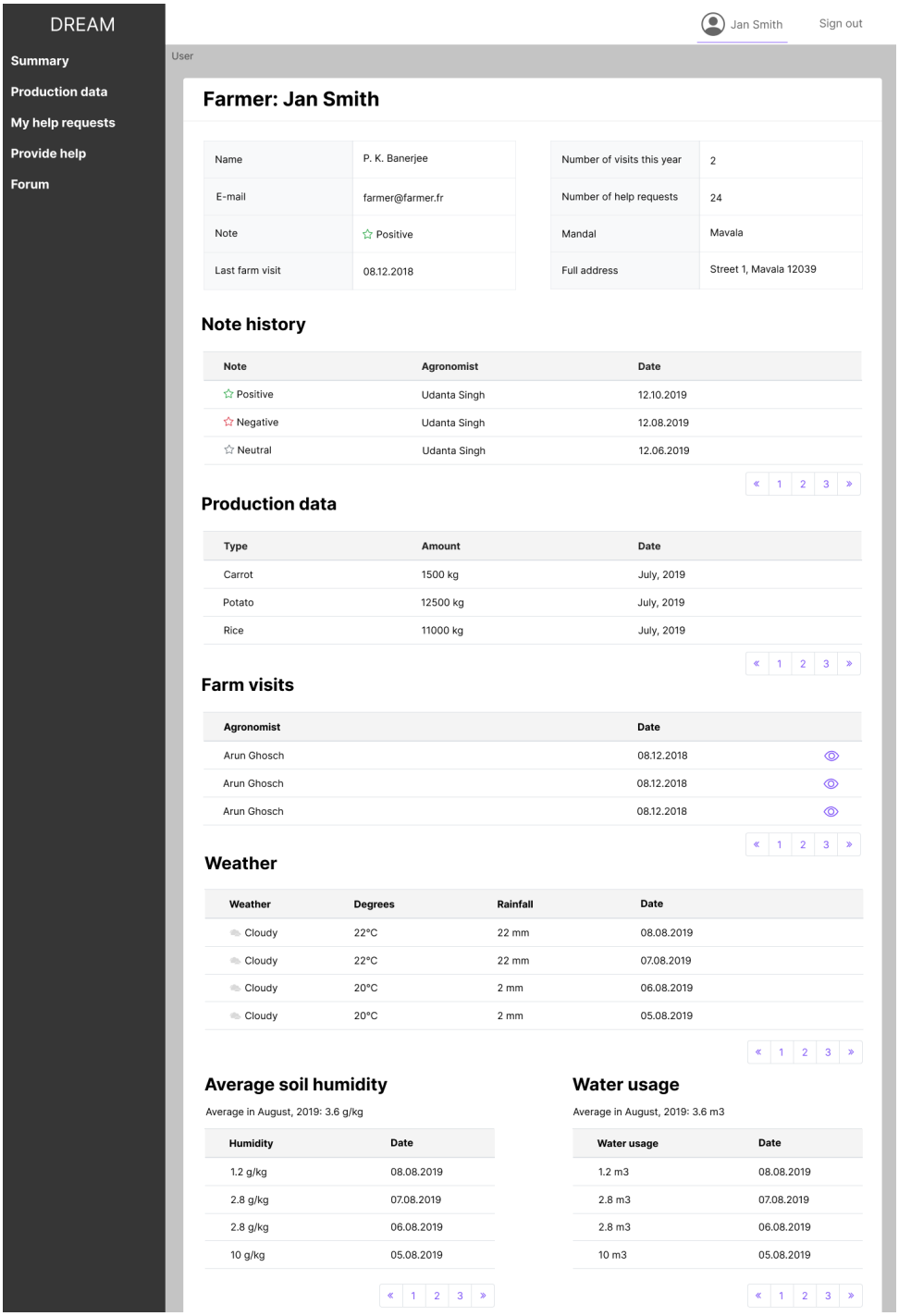

Figure 3.4: Farmer's user view - part 1

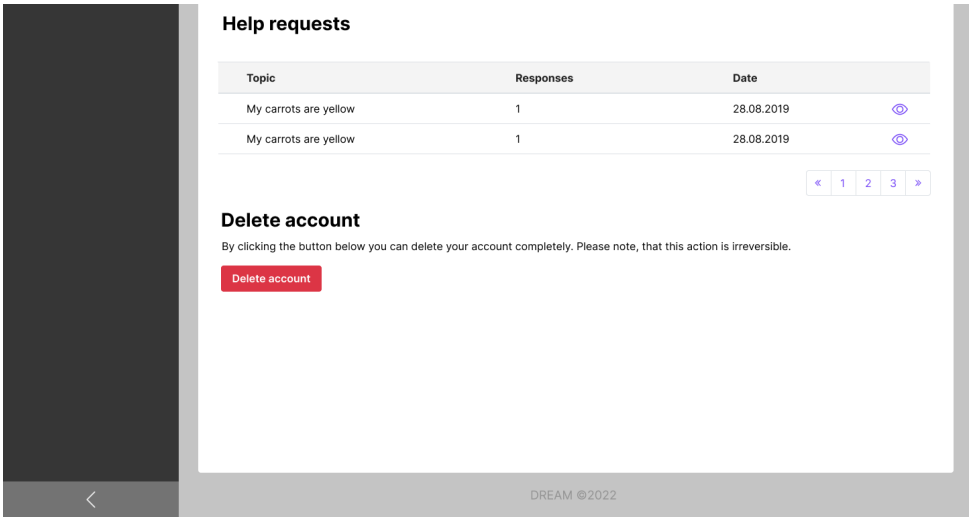

Figure 3.5: Farmer's user view - part 2

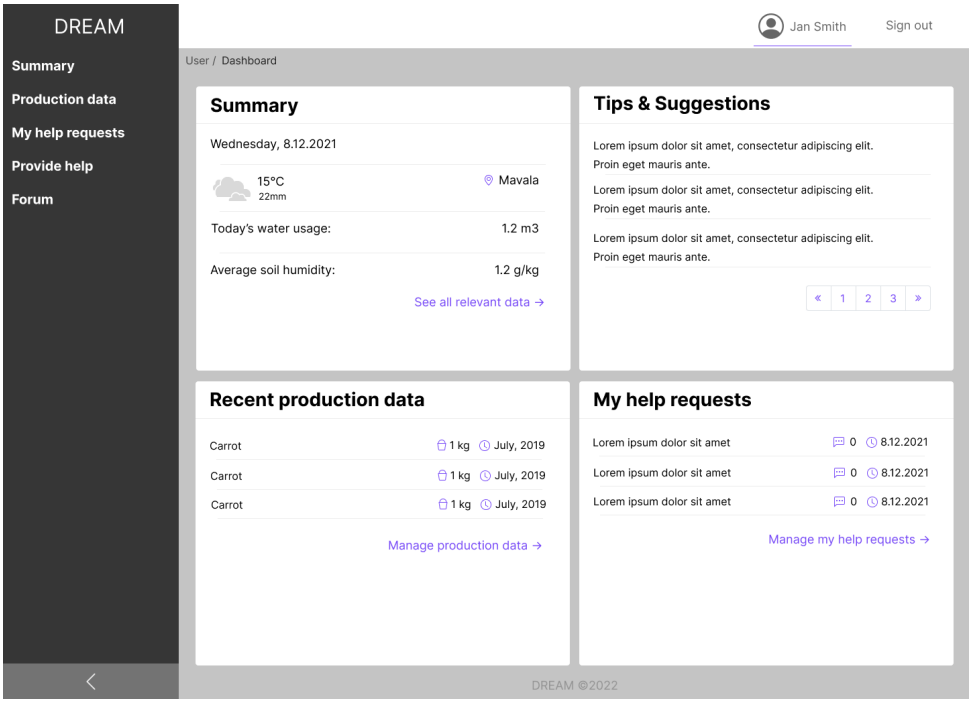

Figure 3.6: Farmer's dashboard

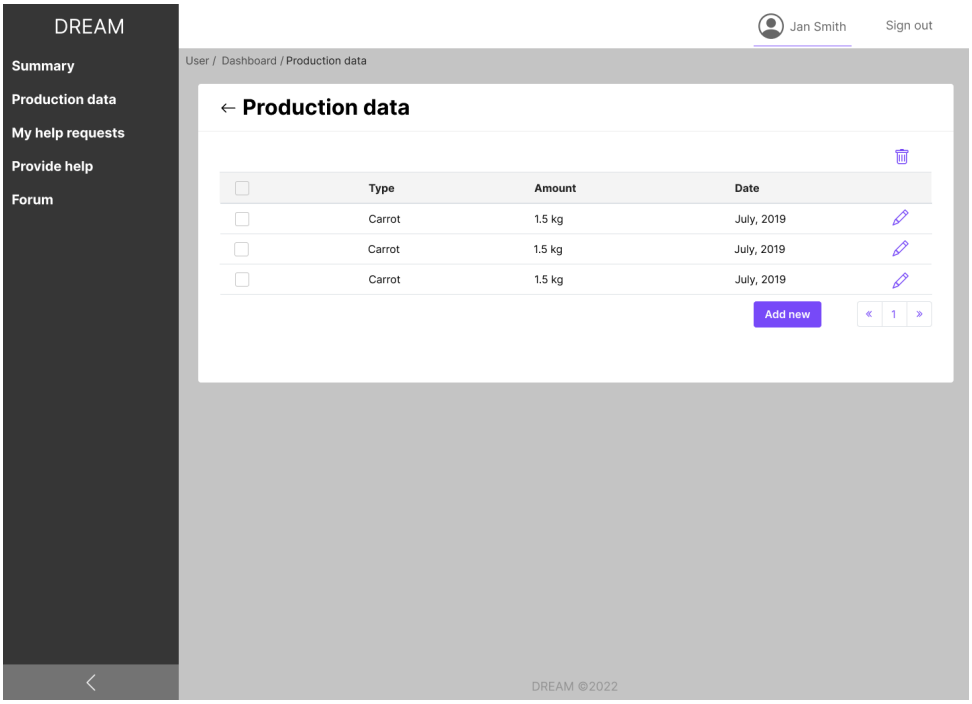

Figure 3.7: Farmer's production data

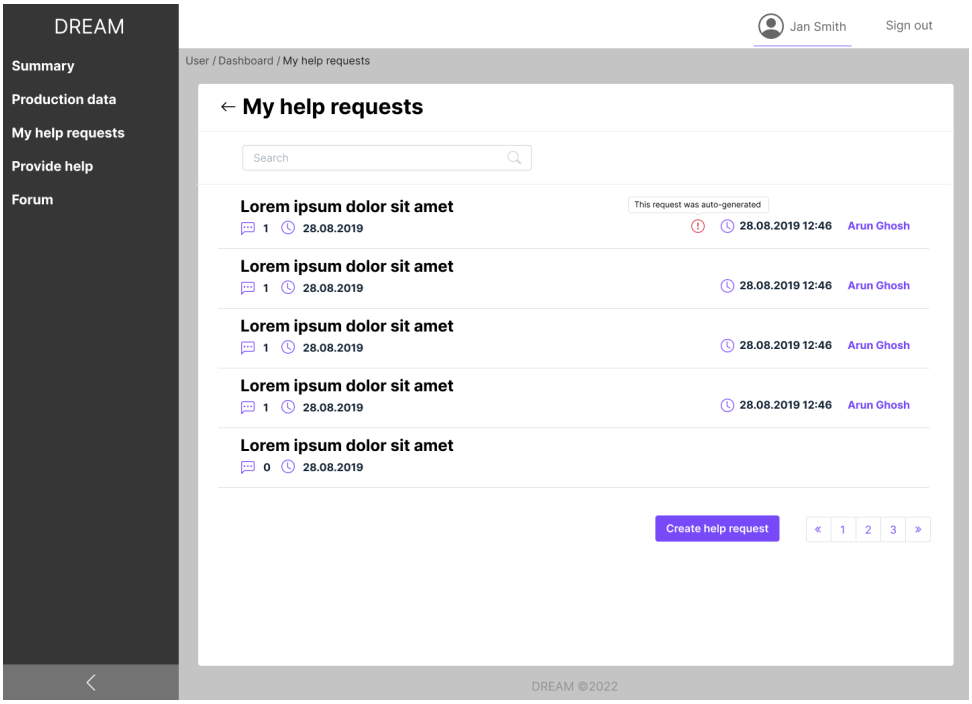

Figure 3.8: Help requests created by farmer

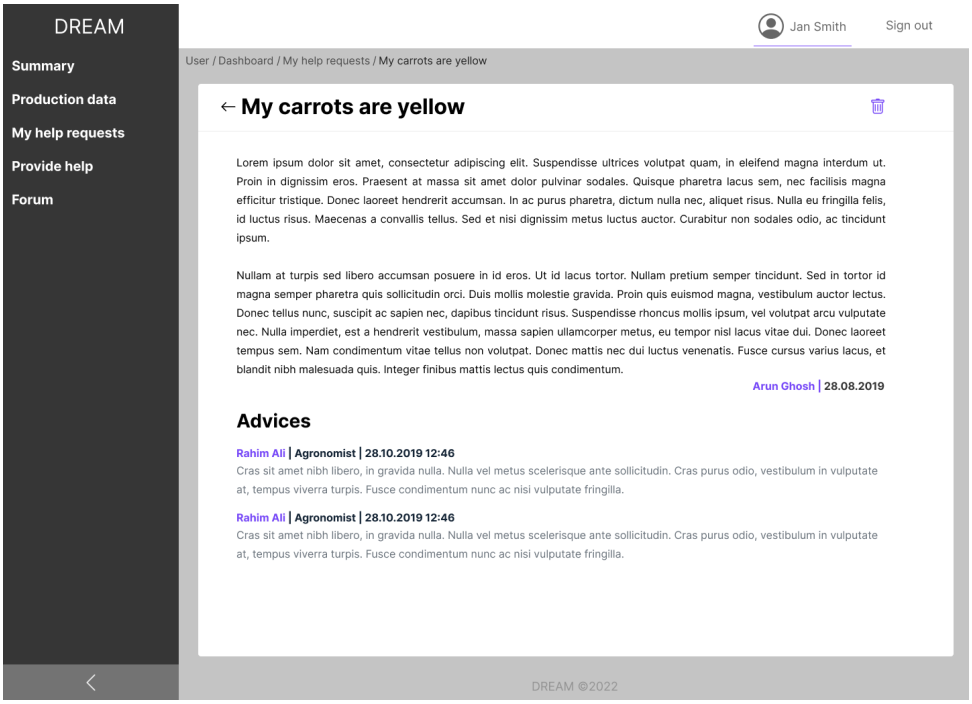

Figure 3.9: Specific help request created by farmer

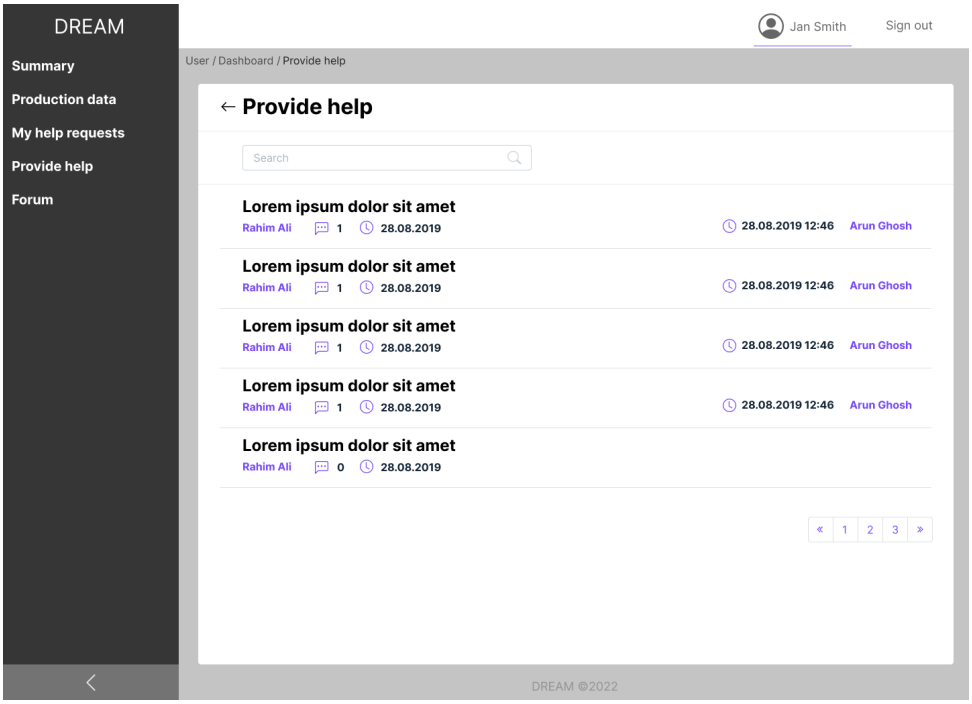

Figure 3.10: Help requests received by farmer with positive note

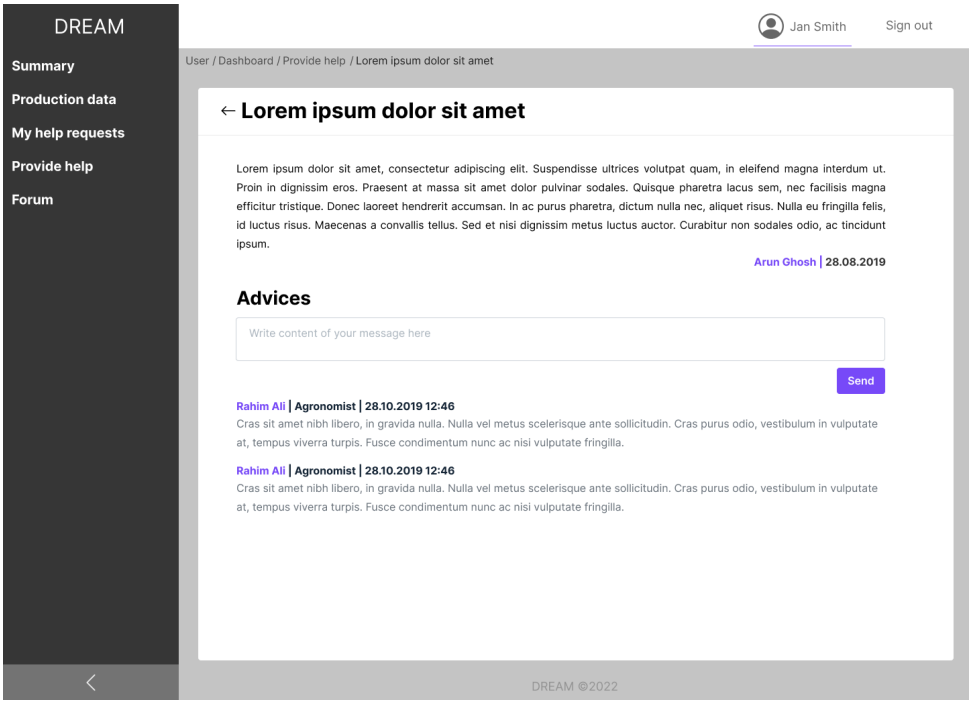

Figure 3.11: Specific help request received by farmer with positive note

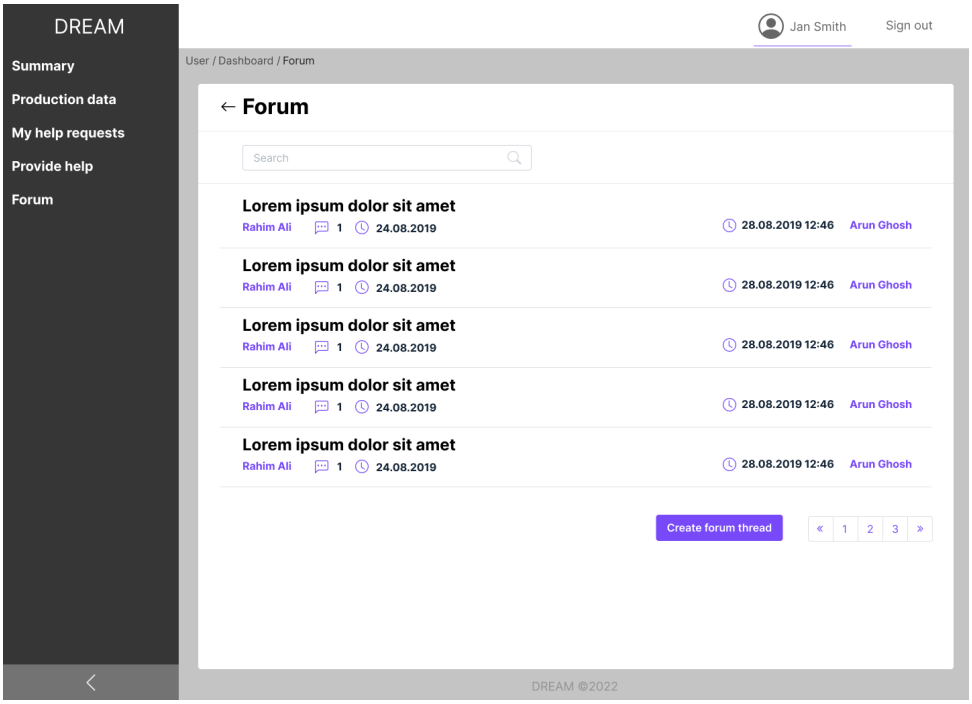

Figure 3.12: Forum view

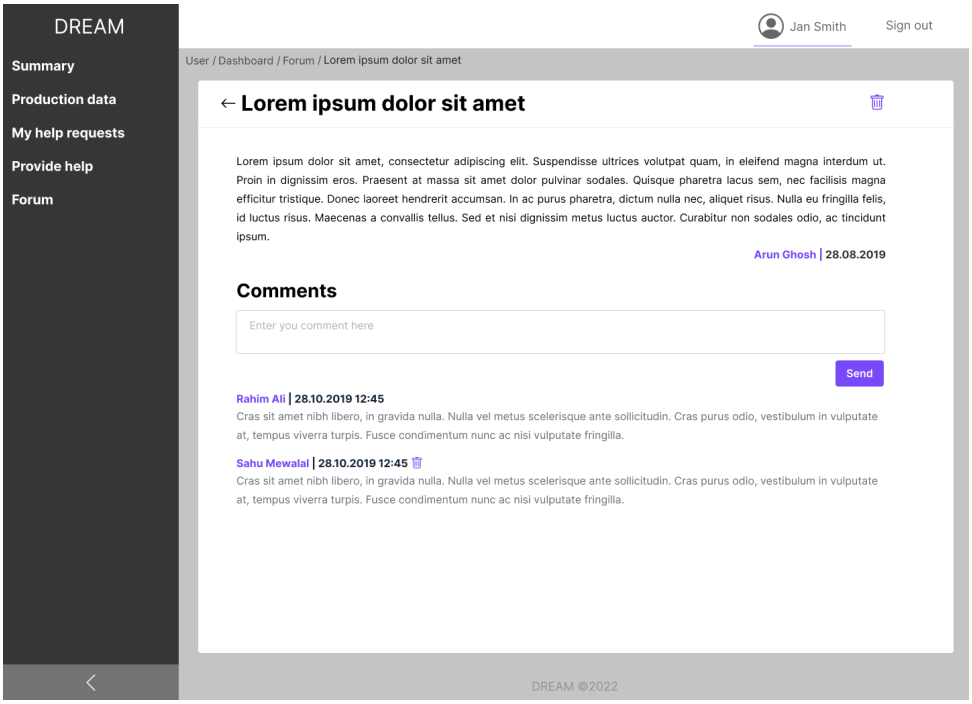

Figure 3.13: Specific forum thread

Agronomist

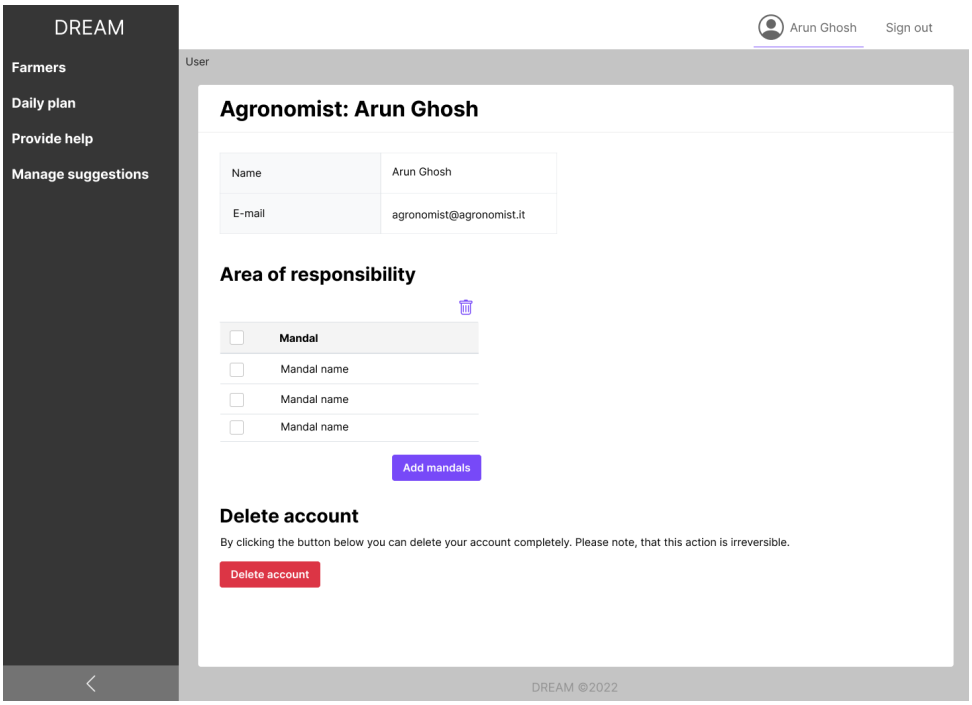

Figure 3.14: Agronomist's user view

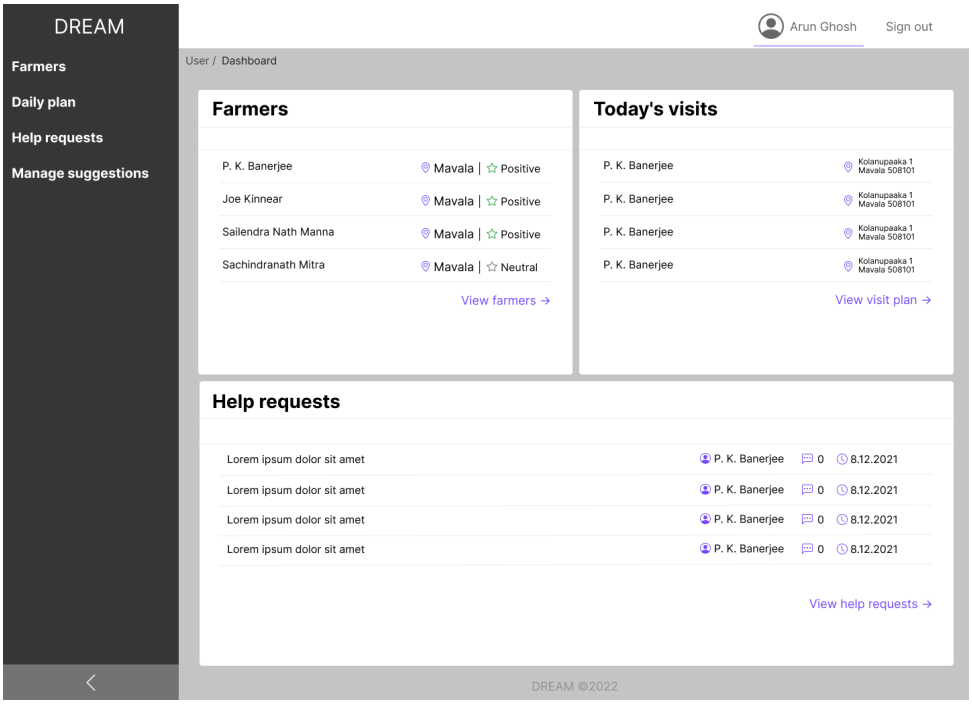

Figure 3.15: Agronomist's dashboard

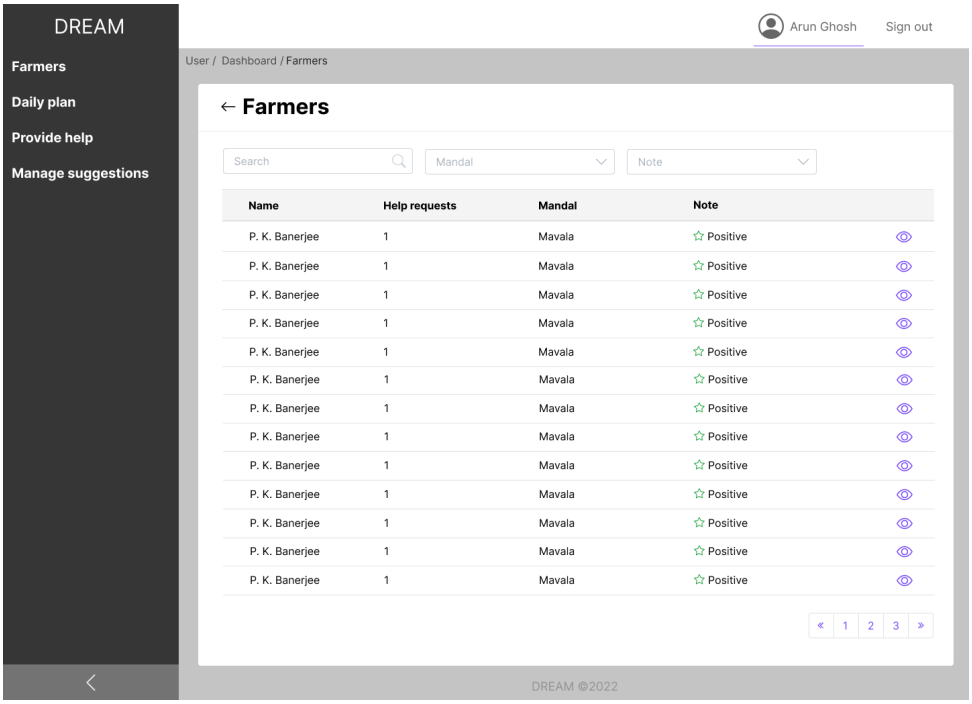

Figure 3.16: List of farmers

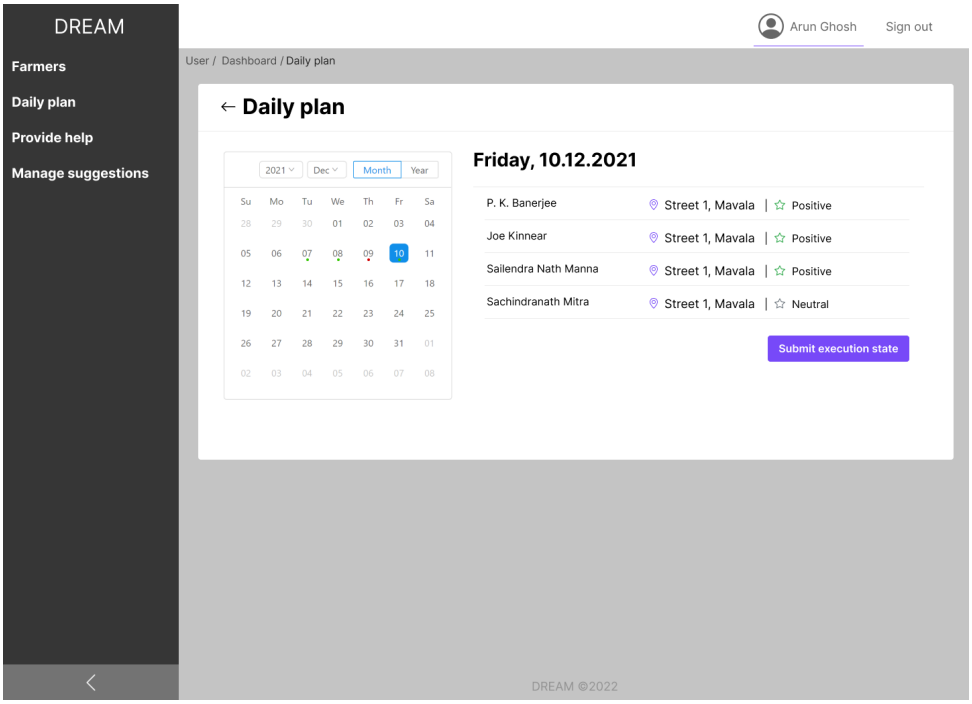

Figure 3.17: Calendar of daily plans

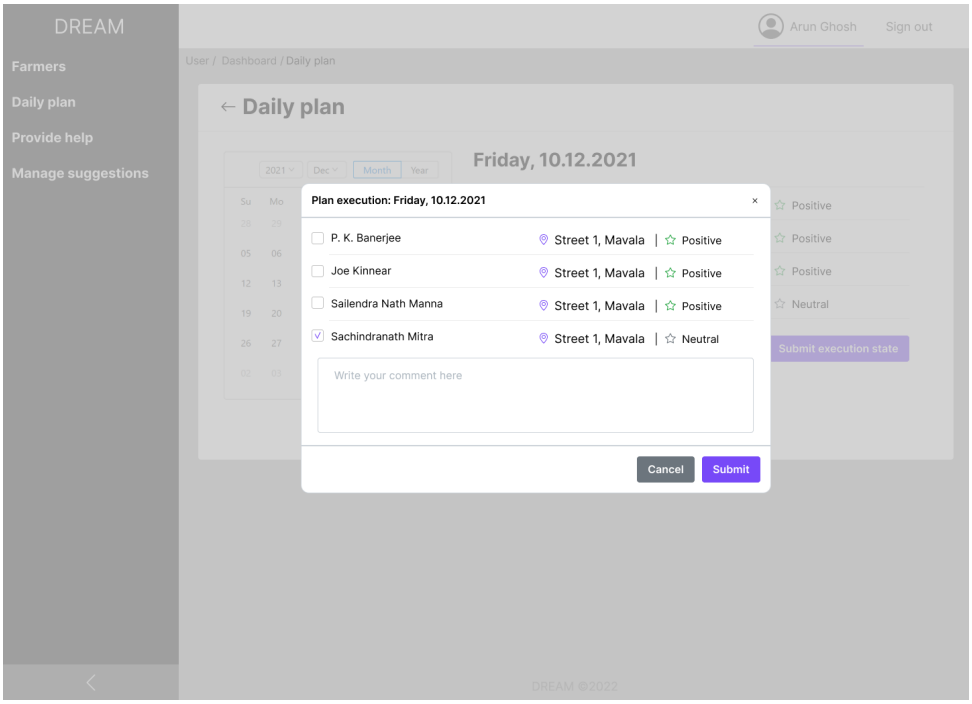

Figure 3.18: Submitting executions state of a daily plan

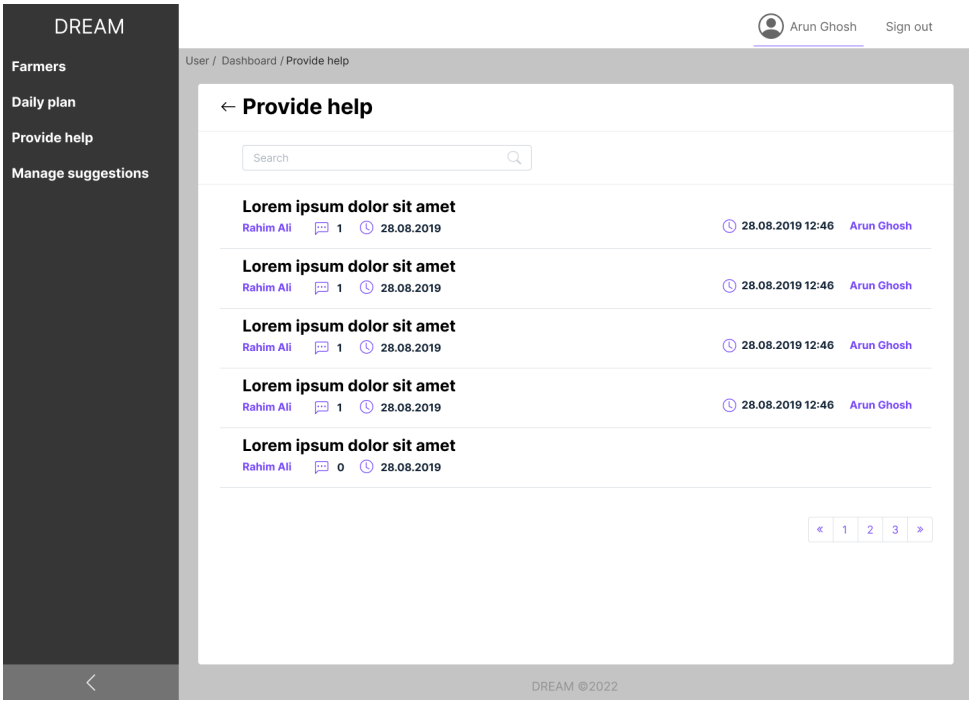

Figure 3.19: Help requests received by agronomist

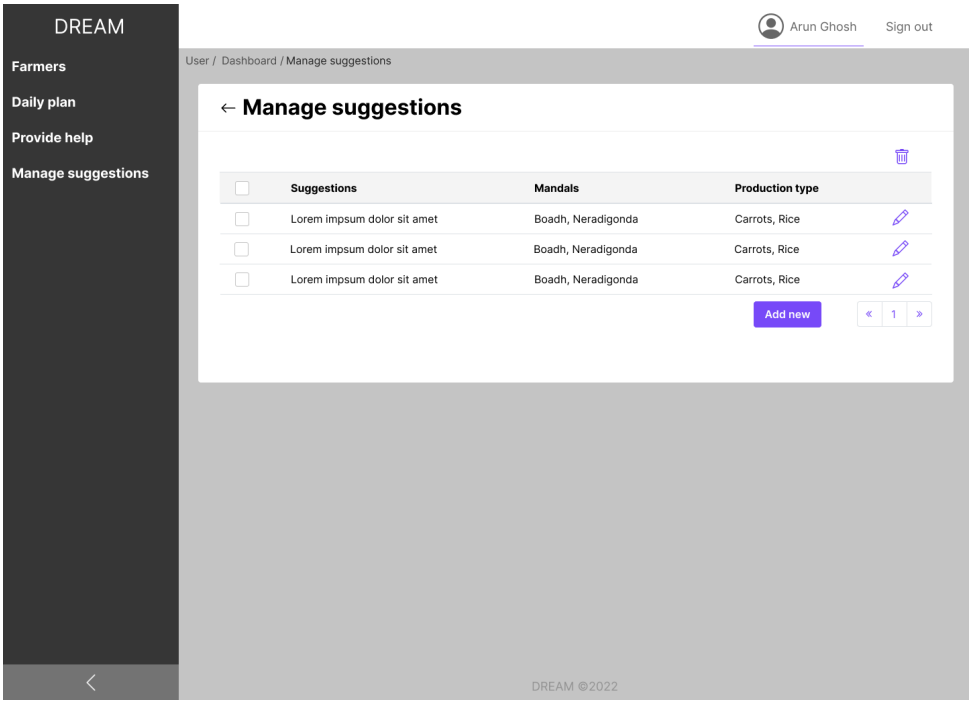

Figure 3.20: Managing suggestions

Policy maker

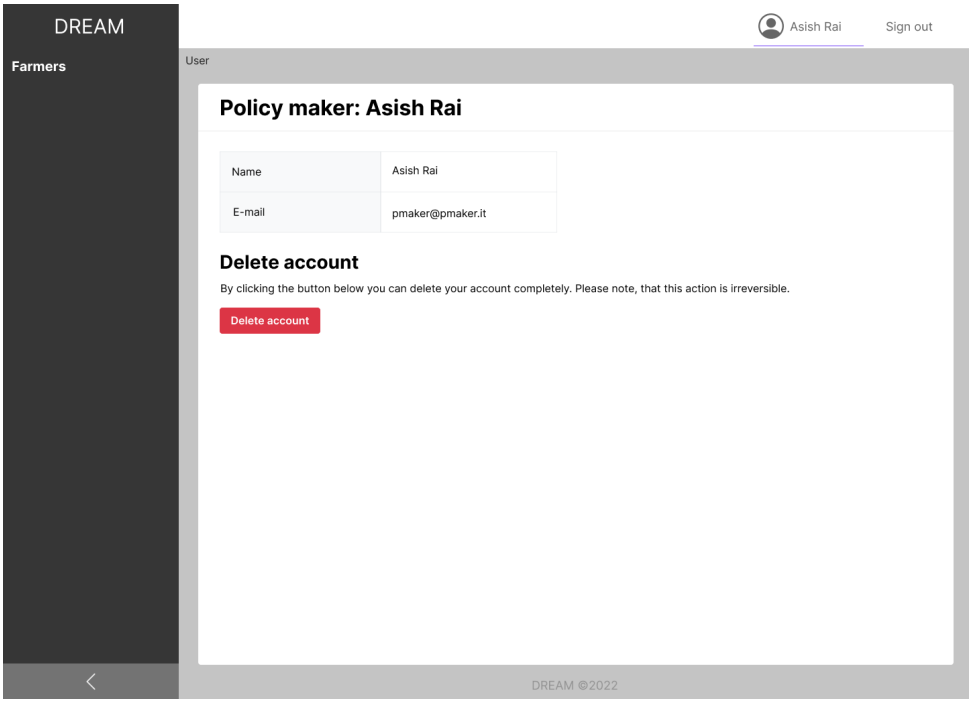

Figure 3.21: Policy maker's user view

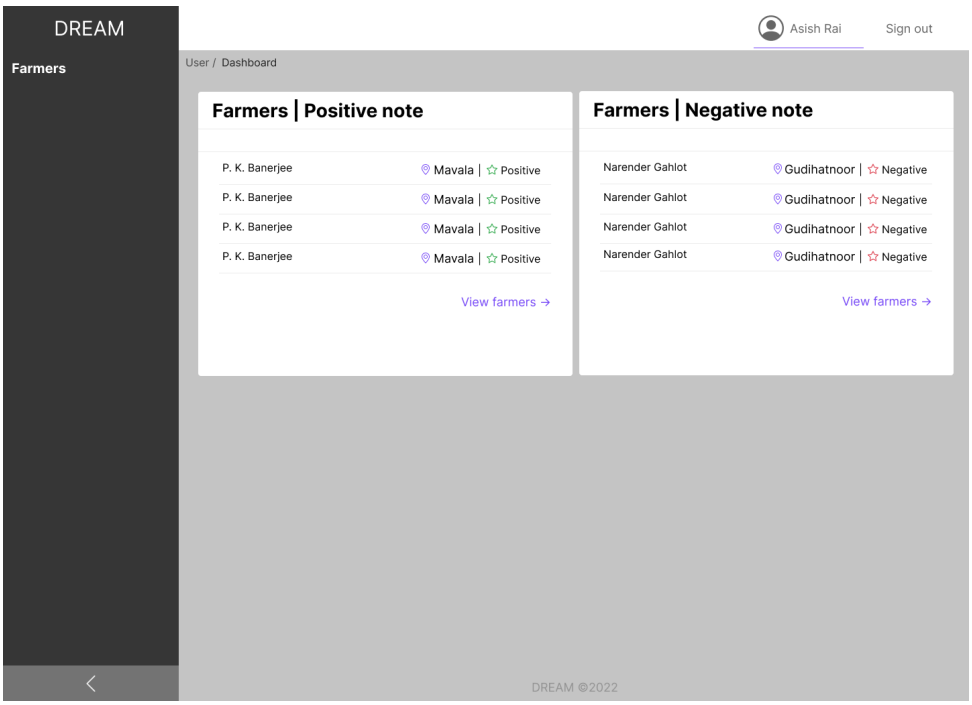

Figure 3.22: Policy maker's dashboard

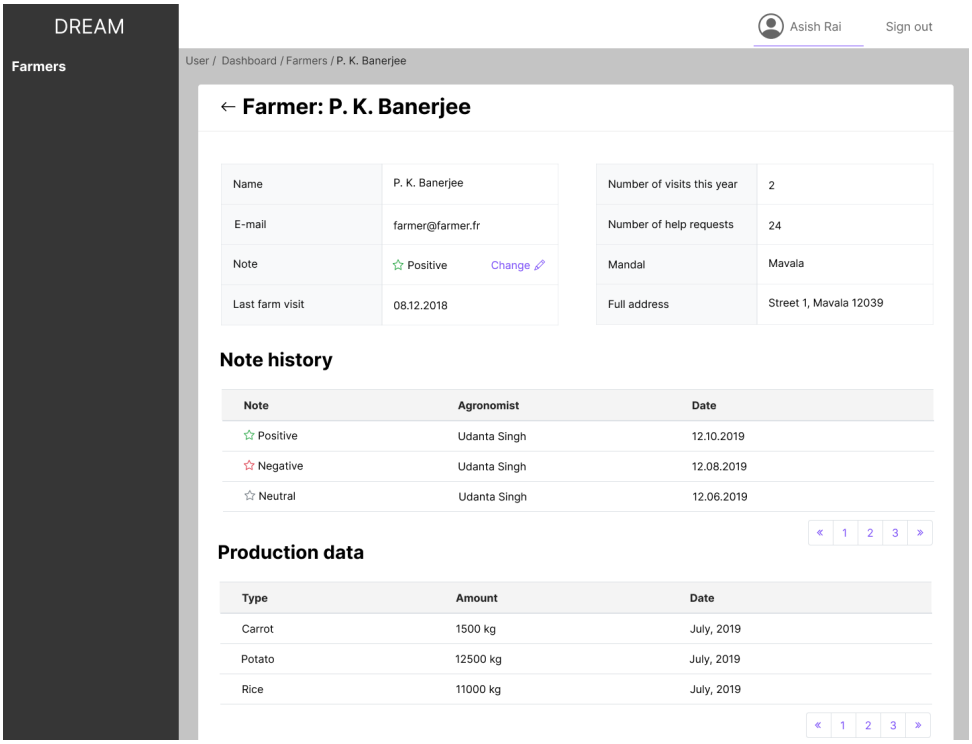

Figure 3.23: Partial view of farmer's summary, with option to change note

3.1.2 Hardware Interfaces

DREAM is a web application, as such each user should own a device capable of installing a modern browser. Form factor of the device does not matter, as DREAM application is fully responsive.

3.1.3 Software Interfaces

A modern browser is necessary to use the system. DREAM supports most major browsers. However, in order to ensure full functionality, below is a list of browsers that are fully supported.

- Google Chrome
- Firefox
- Safari
- Microsoft Edge
- Opera

User should always update their browser to keep up with security, compatibility and functionality.

3.1.4 Communication Interfaces

To perform any kind of operation using the system, a stable internet connection is required (Wi-Fi or cellular).

3.2 Functional Requirements

ID	Requirement
R1.	The system must uniquely identify each user by his e-mail.
R2.	The system must allow an unregistered user to create an account with a chosen role.
R3.	The system must ensure that an agronomist chooses the area of responsibility during the registration process.
R4.	The system must ensure that a farmer inserts his farm data during the registration process.
R5.	The system must ensure that two casual farm visits are scheduled for each year of the farm's existence.
R6.	The system must allow a registered user to log in to the application.
R7.	The system must allow a registered user to reset his password.
R8.	The system must allow a logged-in user to sign out of the application.
R9.	The system must allow a registered user to delete his account.
R10.	The system must allow a policy maker to assign a note to a farmer.
R11.	The system must ensure that every farmer initially has a neutral note.
R12.	The system must have a list of predefined farmer's problem types.
R13.	The system must allow a policy maker to specify a problem type when assigning a negative note.
R14.	The system must ensure that a farm is visited more often in the event of any problems.
R15.	The system must have a specified number of additional visits caused by each predefined problem type.
R16.	The system must ensure that only the most recently specified problem type is taken into account when determining the number of visits.
R17.	The system must ensure that an automatic help request followed by additional farm visits are created, if a farmer receives a negative note.
R18.	The system must ensure that all the additional farm visits created due to obtaining a negative note are deleted after a farmer's negative note is updated with a positive or a neutral one.
R19.	The system must allow a policy maker to see a list of all farmers in Telangana.

R20. The system must allow a policy maker to see a list of all farmers with a specific note.

R21. The system must allow a policy maker to see a list of all farmers in a given mandal.

R22. The system must allow a policy maker to find a farmer's summary by his name and surname.

R23. The system must allow a policy maker to see a farmer's summary.

R24. The system must allow a farmer to see his own farmer's summary.

R25. The system must allow a farmer to see personalized suggestions.

R26. The system must allow a farmer to manage his monthly production data.

R27. The system must allow a farmer to see a list of help requests created by him.

R28. The system must allow a farmer to find a help request created by him by the topic.

R29. The system must allow a farmer to see a specific help request created by him.

R30. The system must allow a farmer to delete a specific help request created by him.

R31. The system must allow a farmer to create a new help request.

R32. The system must allow a farmer with a positive note to see a list of all help requests in his mandal.

R33. The system must allow a farmer with a positive note to find a help request in his mandal by the topic.

R34. The system must allow a farmer with a positive note to see a specific help request in his mandal.

R35. The system must allow a farmer with a positive note to respond to a specific help request in his mandal.

R36. The system must allow a farmer with a positive note to delete a help response, only if it was created by him.

R37. The farmer is able to see farmer's summary of a farmer with a negative note whose help request he received.

R38. The system must allow a farmer to see a list of all forum threads in Telangana.

R39. The system must allow a farmer to find a forum thread by the topic.

R40. The system must allow a farmer to see a specific forum thread with its comments.

R41. The system must allow a farmer to create a comment in a forum thread.

R42. The system must allow a farmer to delete a comment in a forum thread, only if it was created by him.

R43. The system must allow a farmer to create a forum thread.

R44. The system must allow an agronomist to manage his area of responsibility.

R45. The system must allow an agronomist to see the list of suggestions for mandals in his area of responsibility.

R46. The system must allow an agronomist to manage the list of suggestions for mandals in his area of responsibility.

R47. The system must allow an agronomist to see a list of all farmers in his area of responsibility.

R48. The system must allow an agronomist to see a list of all farmers with a specific note in his area of responsibility.

R49. The system must allow an agronomist to see a list of all farmers in a given mandal in his area of responsibility.

R50. The system must allow an agronomist to find a farmer's summary in his area of responsibility by his name and surname.

R51. The system must allow an agronomist to see a farmer's summary in his area of responsibility.

R52. The system must allow an agronomist to see a list of all help requests in his area of responsibility.

R53. The system must allow an agronomist to find a help request in his area of responsibility by the topic.

R54. The system must allow an agronomist to see a specific help request in his area of responsibility.

R55. The system must allow an agronomist to respond to a specific help request in his area of responsibility.

R56. The system must allow an agronomist to delete a help response, only if it was created by him.

R57. The system must allow an agronomist to see his daily plans.

R58. The system must allow an agronomist to submit a daily plan's execution state, by rejecting or confirming each visit, on the same date or after the daily plan's date has passed.

R59. The system must allow an agronomist to provide a comment for a visit he is confirming.

R60. The system must ensure that after an agronomist submits his daily plan, for all the causal visits marked as confirmed, new ones are created in approximately half of a year.

R61. The system must allow an agronomist to delete a visit before its date.

R62. The system must ensure that a deleted visit is marked as rejected.

R63. The system must ensure that in case of rejecting a casual visit, a new one is created in maximally 5 days.

R64. The system must allow replanning any different visit than a casual one without any constraints.

R65. The system must read and update weather forecasts every day.

R66. The system must read and store data from humidity sensors every day.

R67. The system must read and store data from water irrigation systems every day.

R68. The system must ensure that a farmer who creates a help request cannot be its recipient.

3.2.1 Use cases

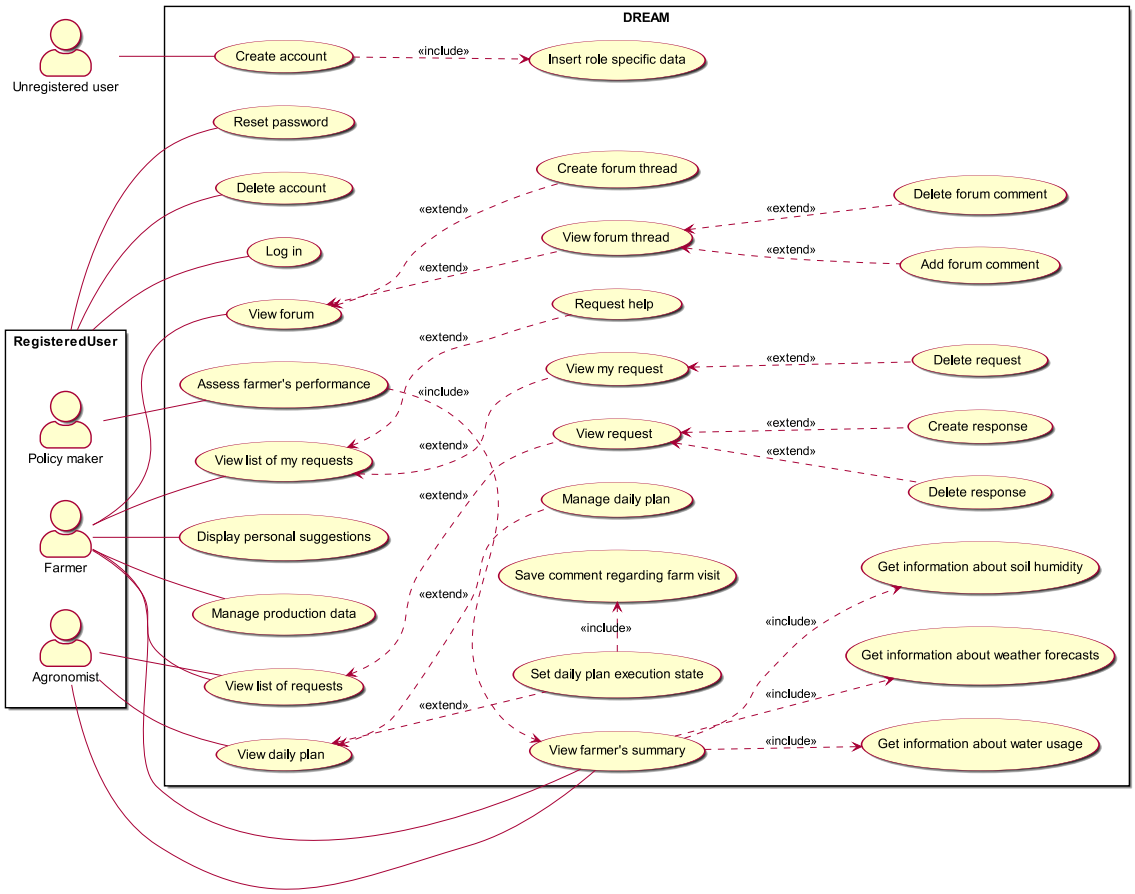

Figure 3.24: Use case diagram.

Given the great number of different use cases, the descriptions of only the most important ones are provided below.

Name	Create account
Actors	Unregistered user
Goals	G1, G2, G3, G4, G5, G6
Entry conditions	• The unregistered user, who wants to create an account, is on the welcome screen of the application.

Flow of events	1. The unregistered user clicks on *Create account* button on top. 2. The application shows a form with role-irrelevant data (email, name, surname and password) to fill and a dropdown to choose the role. 3. The unregistered user fills the form and selects a role. 4. If a role of a farmer was chosen, the application shows another part of the form with text fields related to farm data to be filled. 5. If a role of an agronomist was chosen, the application shows another part of the form that asks to manage mandals in the agronomist's area of responsibility. 6. The unregistered user fills role-related data if there is any. 7. If a role of a farmer was chosen, the system creates a farm for a farmer and two casual visits, which will take place in the following year, starting from the current date. 8. If a role of an agronomist was chosen, the system adds selected mandals to the agronomist's area of responsibility. 9. The unregistered user clicks on *Create account* button. 10. The application creates a new user. 11. The application shows a message indicating successful registration. 12. The application shows the log in screen.
Exit conditions	A new user is created in the system. From now on, the credentials entered during the registration process can be used to log in to the application.
Exceptions	• The unregistered user cancels the operation, clicking on a Cancel button or a cross in the top right corner of the pop-up. The application shows the welcome screen. • The user with the given email address already exists. The application shows a message explaining that the user with given email already exists. • Some part of the form was not filled or was filled incorrectly. The application shows a message explaining the type of error made by the user. • The application is not able to create a new user in step 10. The application shows a message indicating a possible cause of the error.

Table 3.2: Use case description: Create account.

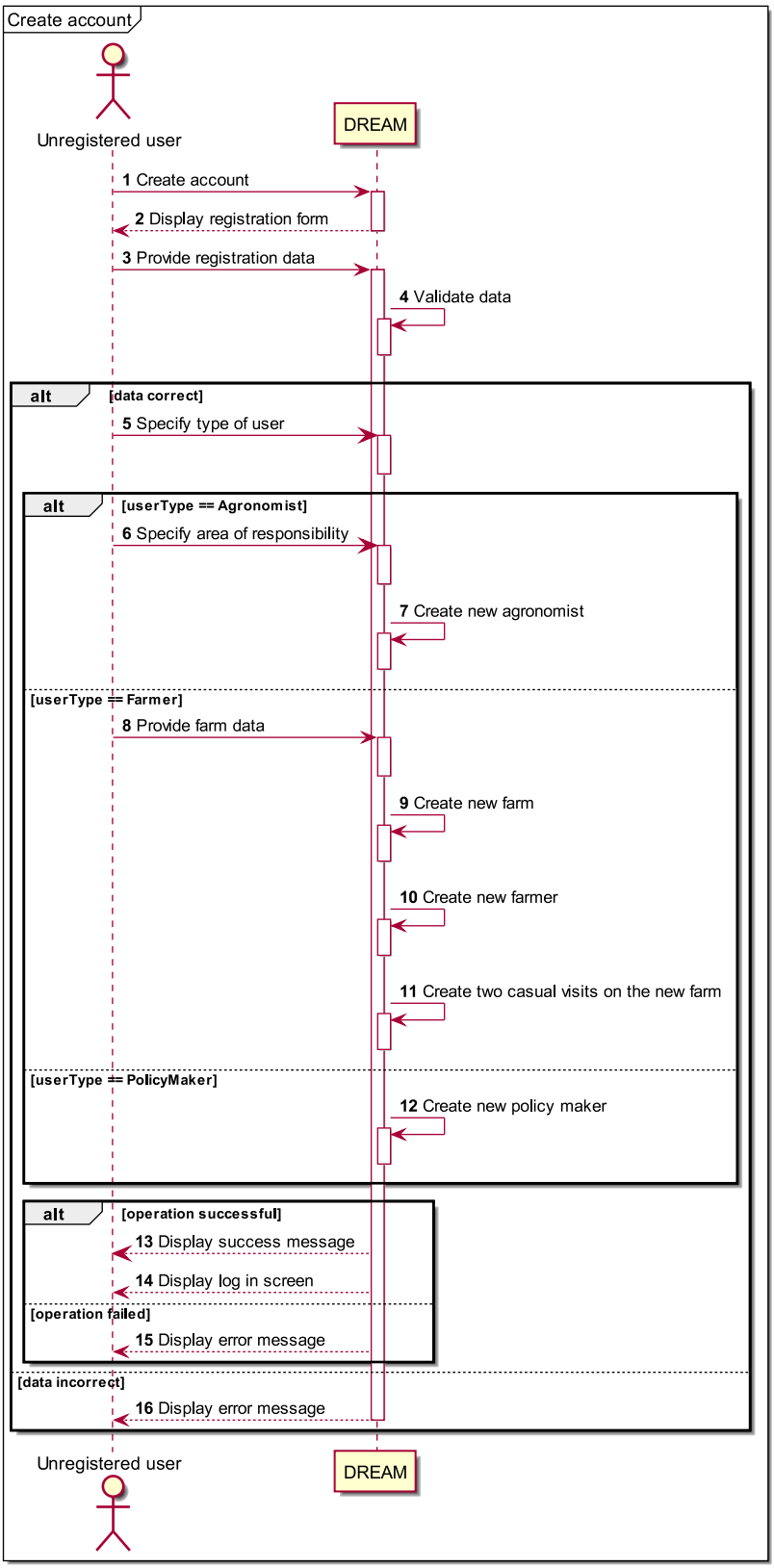

Figure 3.25: Sequence diagram presenting the process of creating a new user account.

Name	Log in
Actors	Policy maker, Agronomist, Farmer
Goals	G1, G2, G3, G4, G5, G6
Entry conditions	• The user, who wants to log in, is on the welcome screen of the application.
Flow of events	1. The user clicks on *Log in* button in the top bar. 2. The application shows a pop-up with text fields for e-mail and password. 3. The user enters an e-mail and password. 4. The application verifies the credentials. 5. The application shows the dashboard view.
Exit conditions	The user is successfully logged in. From now on, he can use role-specific functions of the DREAM application.
Exceptions	• The credentials entered in step 4 are not valid. The application shows a message explaining the issue. • The user cancels the operation, clicking on a *Cancel button* or a cross in the top right corner of the pop-up. The application comes back to the welcome screen. • The application is not able to verify the credentials in step 4. The application shows a message indicating a possible cause of the error.

Table 3.3: Use case description: Log in.

Name	Delete account
Actors	Policy maker, Agronomist, Farmer
Goals	G1, G2, G3, G4, G5, G6
Entry conditions	• The user is already logged in to the application. • The user is on a first view shown after a log-in (dashboard screen).

Flow of events	1. The user clicks on his name in the top bar. 2. The user scrolls to the bottom of the page. 3. The user clicks on the *Delete account* button. 4. The application shows a pop-up asking the user to confirm account deletion. 5. The user clicks on the *Delete account* button on the pop-up. 6. The application deletes the account. 7. The application shows a message indicating successful account deletion. 8. The application shows the welcome screen.
Exit conditions	The account together with all related data is deleted from the system.
Exceptions	• The user cancels the operation by clicking on the *Cancel* button or a cross in the top right corner of the pop-up in step 4. The pop-up is closed. • The application is not able to delete the account in step 6. The application shows a message indicating a possible cause of the error.

Table 3.4: Use case description: Delete account.

Name	Reset password
Actors	Policy maker, Agronomist, Farmer
Goals	G1, G2, G3, G4, G5, G6
Entry conditions	• The user, who wants to log in, is on the welcome screen of the application.

Flow of events	1. The user clicks on *Log in* button in the top bar. 2. The user clicks on *Forgot password?* button. 3. The application shows a pop-up with a text field to fill an e-mail. 4. The user enters his e-mail. 5. The application sends an e-mail with a password reset link. 6. The user open his mailbox, goes to the email from DREAM app and clicks on a password reset link. 7. The application opened in a given URL shows two text fields to enter the new password twice. 8. The user fills both text fields with a new password. 9. The user clicks on a *Save* button. 10. The application verifies if both text fields contain the same passwords. 11. The application saves the new password for a given user.
Exit conditions	The new password is saved and can be used to log in to the application.
Exceptions	• The two text fields contain different passwords in step 10. The application shows a message explaining the issue. • The user cancels the operation, clicking on a *Cancel button* or a cross in the top right corner of the pop-up. The application comes back to the welcome screen. • The user resigns from resetting a password and decides not to use the reset link in step 6. The user can still use the old password to log in to the application. • The application is not able to save a new password in step 11. The application shows a message indicating a possible cause of the error.

Table 3.5: Use case description: Reset password.

Name	Assess farmer's performance
Actors	Policy maker
Goals	G3
Entry conditions	• The policy maker is already logged in to the application.

Flow of events	1. The policy maker opens *Farmers* view. 2. The policy maker clicks on one of the farmers. 3. The application shows the selected farmer's summary. 4. The policy maker clicks on *Change* button next to the current note of the farmer. 5. The policy maker chooses a note from a dropdown list. 6. In case of the note being a negative note, the system displays a dropdown list of available problem types. 7. The policy maker chooses a matching problem type. 8. The policy maker clicks on *Save* button. 9. The system issues a help request followed by a creation of additional farm visits. 10. In case of the note being neutral or positive, while the previous note was negative, the system deletes all the additional farm visits created due to obtaining a negative note. 11. The application saves a new farmer's note. 12. The application shows a message indicating successful assignment of the note.
Exit conditions	The new note is saved in the system, visits are adjusted and in case of a negative note, a new help request is created.
Exceptions	• The policy maker resigns from changing the current note held by a farmer by clicking the cross next to the *Save*. The system shows the farmer's summary view with the previous note in the note field. • The application is not able to save the note assigned in step 5. The application shows a message indicating a possible cause of the error.

Table 3.6: Use case description: assess farmer's performance.

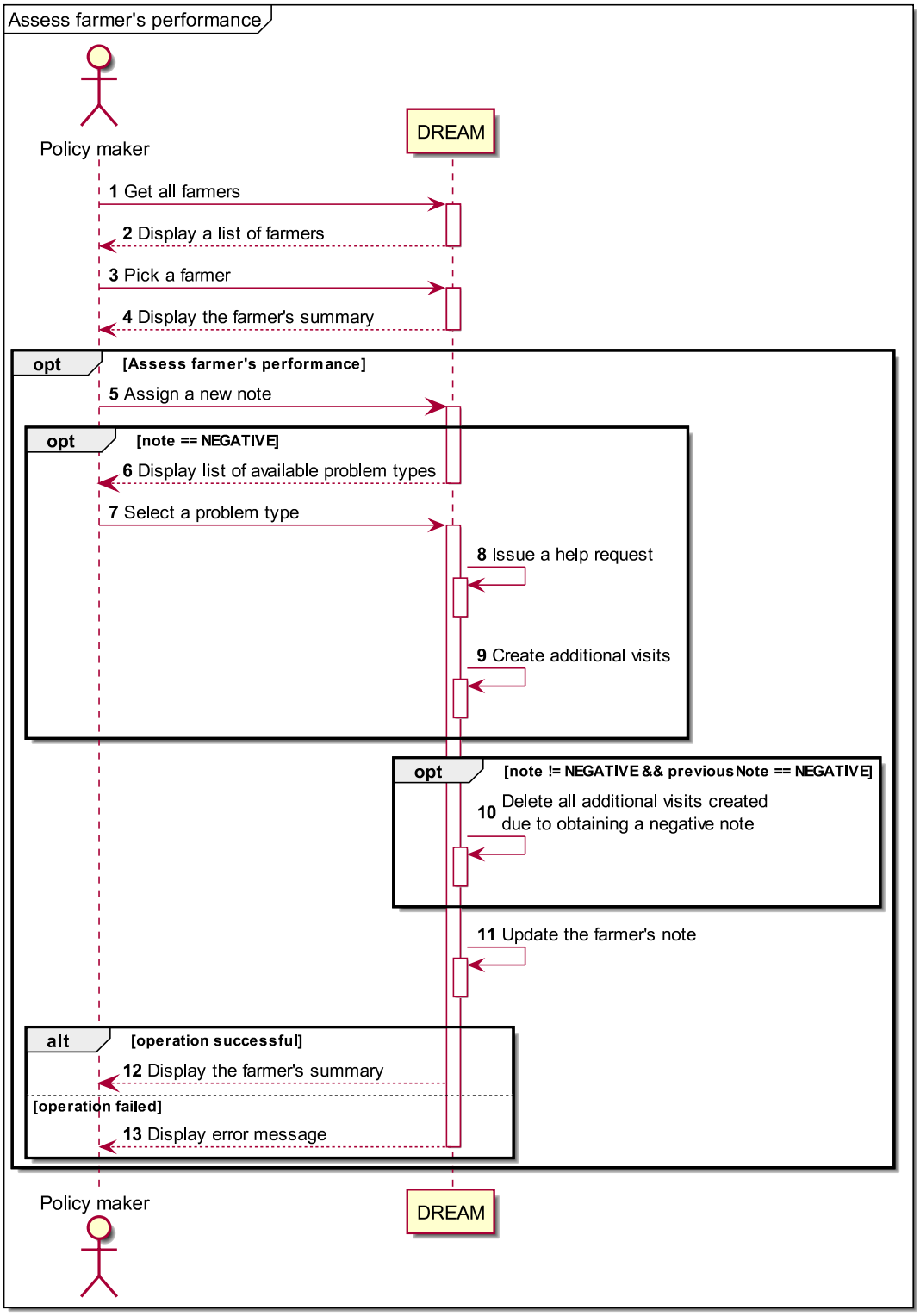

Figure 3.26: Sequence diagram presenting farmer's performance assessment.

Name	Create forum thread
Actors	Farmer
Goals	G6
Entry conditions	• The farmer is already logged in to the application. • The farmer is in the *Forum* view, accessible from the sidebar of the application.
Flow of events	1. The farmer clicks on *Create forum thread* at the bottom of the page. 2. The application shows a pop-up with textboxes to fill. 3. The farmer enters the topic and description of the thread. 4. The farmer clicks on *Create* button. 5. The application creates a new forum thread. 6. The application displays the new thread.
Exit conditions	The application successfully created a new forum thread. From now on, the new forum thread is visible to all other farmers. User sees the new thread.
Exceptions	• The farmer closes the pop-up showed in step 2 by clicking on a cross in the top right corner or the *Cancel button* in the bottom. The system comes back to the *Forum* view. • The application is not able to create a forum thread in step 5. The application shows a message indicating a possible cause of the error.

Table 3.7: Use case description: Create forum thread.

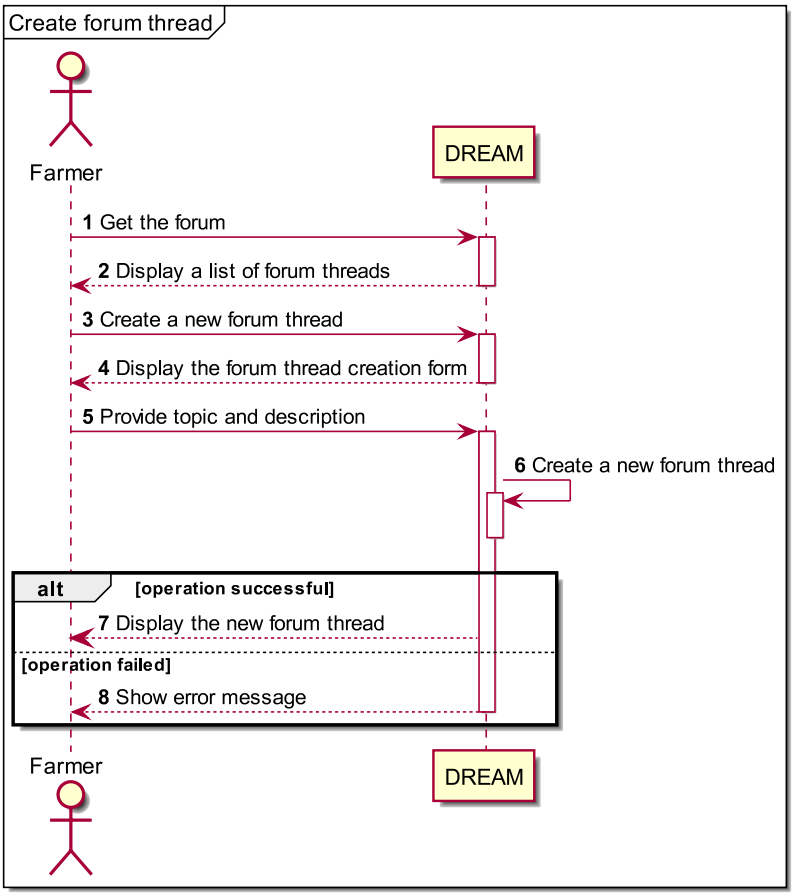

Figure 3.27: Sequence diagram presenting the process of creating a new forum thread.

Name	Add forum comment
Actors	Farmer
Goals	G6
Entry conditions	• The farmer is already logged in to the application. • The farmer is in the *Forum* view, accessible from the sidebar of the application.
Flow of events	1. The farmer selects a forum thread, by clicking on the thread's topic in the list. 2. The application shows the contents of the selected forum thread. 3. The farmer fills the textbox with the content of his comment. 4. The farmer clicks on *Send* button. 5. The application adds a new forum comment. 6. The application updates thread view, showing the new comment.

Exit conditions	The application saved the new comment in the thread. From now on, the new comment is visible to all farmers viewing the thread. Farmer can see new comment in thread view.
Exceptions	• The application is not able to add a forum comment in step 5. The application shows a message indicating a possible cause of the error.

Table 3.8: Use case description: Add forum comment.

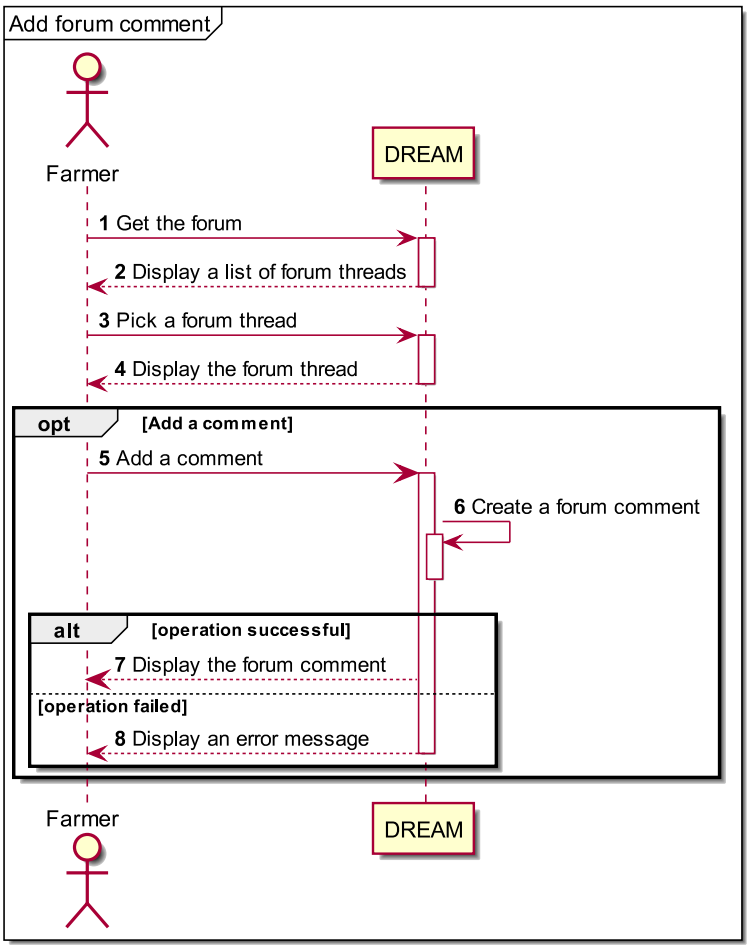

Figure 3.28: Sequence diagram presenting the addition of a comment in a forum thread.

Name	Display personal suggestions
Actors	Farmer
Goals	G1

Entry conditions	• The farmer is already logged in to the application. • The farmer is on the dashboard view.
Flow of events	1. The application shows personal suggestions on the top right part called *Tips and suggestions* of the dashboard. 2. The farmer reads personal suggestions from the list.
Exit conditions	The farmer can read personalized suggestions showed in the dashboard view.
Exceptions	• The application is not able to load personalized suggestions. The application shows a message indicating a possible cause of the error.

Table 3.9: Use case description: Display personal suggestions.

Name	View forum
Actors	Farmer
Goals	G6
Entry conditions	• The farmer is already logged in to the application. • The farmer is on the dashboard view.
Flow of events	1. The farmer clicks on *Forum* in the sidebar. 2. The application shows the forum view with a list of already created forum threads.
Exit conditions	The farmer is able to see a list of already created forum threads.
Exceptions	• The application is not able to load the list of already created forum threads. The application shows a message indicating a possible cause of the error.

Table 3.10: Use case description: View forum.

Name	View forum thread
Actors	Farmer
Goals	G6

Entry conditions	• The farmer is already logged in to the application. • The farmer is on the forum view.
Flow of events	1. The farmer finds a forum thread he is interested to see in the list. 2. The farmer clicks on the forum thread he found in the previous step. 3. The application loads the contents of the selected forum thread.
Exit conditions	The farmer is able to read the contents of the forum thread.
Exceptions	• The application is not able to load the contents of the selected forum thread. The application shows a message indicating a possible cause of the error.

Table 3.11: Use case description: View forum.

Name	Request help
Actors	Farmer
Goals	G5, G6
Entry conditions	• The farmer is already logged in to the application.
Flow of events	1. The farmer clicks on *My help requests* view in the sidebar. 2. The farmer clicks on *Create help request* button. 3. The application shows a pop-up with textboxes to fill. 4. The farmer enters a topic and message. 5. The farmer clicks on *Send request* button. 6. The application sends requests to the agronomists and several farmers with a positive note working in the given area. 7. The application shows the created request.
Exit conditions	The request can be seen in the *Provide help* view by the recipients selected in step 6. Additionally, the farmer who created the request can see it in the *My help requests* view.
Exceptions	• The farmer closes the pop-up showed in step 3 by clicking on *Cancel* button or on a cross in a top right corner. The system comes back to the *My help requests* view. • The application is not able to create a request for help in step 6. The application shows a message indicating a possible cause of the error.

Table 3.12: Use case description: Request help.

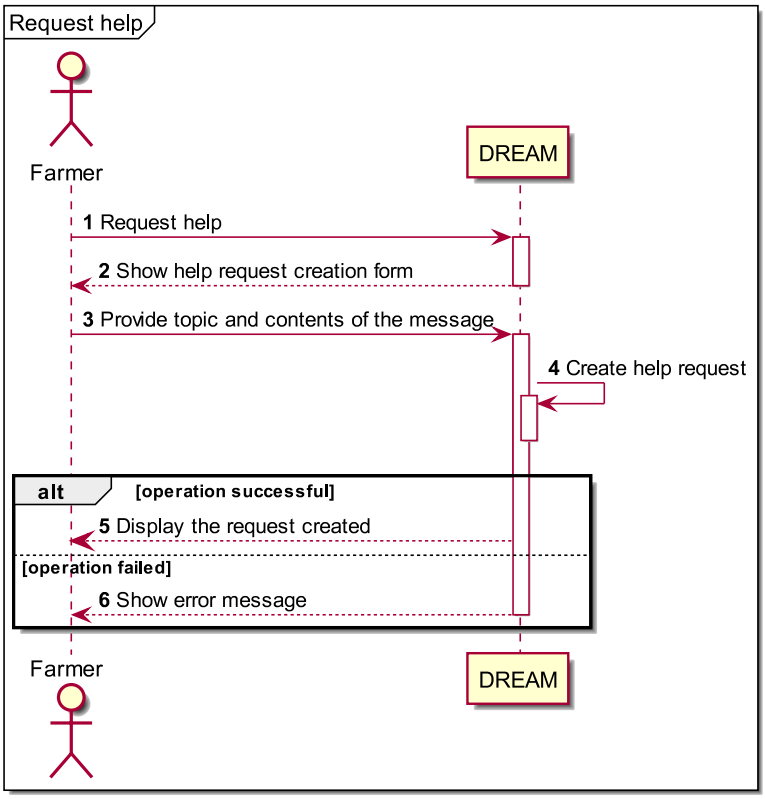

Figure 3.29: Sequence diagram presenting the creation of a help request.

Name	Create response
Actors	Agronomist, Farmer (positive note)
Goals	G5, G6
Entry conditions	• The farmer or the agronomist is already logged in to the application. • The farmer or the agronomist is in the *Provide help* view.
Flow of events	1. The farmer or the agronomist clicks on the requests he wants to reply to. 2. The application shows a pop-up with textboxes to fill. 3. The farmer or the agronomist enters the contents of the reply message. 4. The farmer or the agronomist clicks the *Send* button. 5. The application saves the answer to the request. 6. The application shows a message indicating successful sending of the message.

Exit conditions	The farmer can see the response by clicking on the request in the *My help request tab*.
Exceptions	• The application is not able to create an answer to a help request in step 5. The application shows a message indicating a possible cause of the error. • The user resigns from creating a response to the request by clicking on the cross in the top right corner or *Cancel button*. The application comes back to *Provide help* view.

Table 3.13: Use case description: Create response.

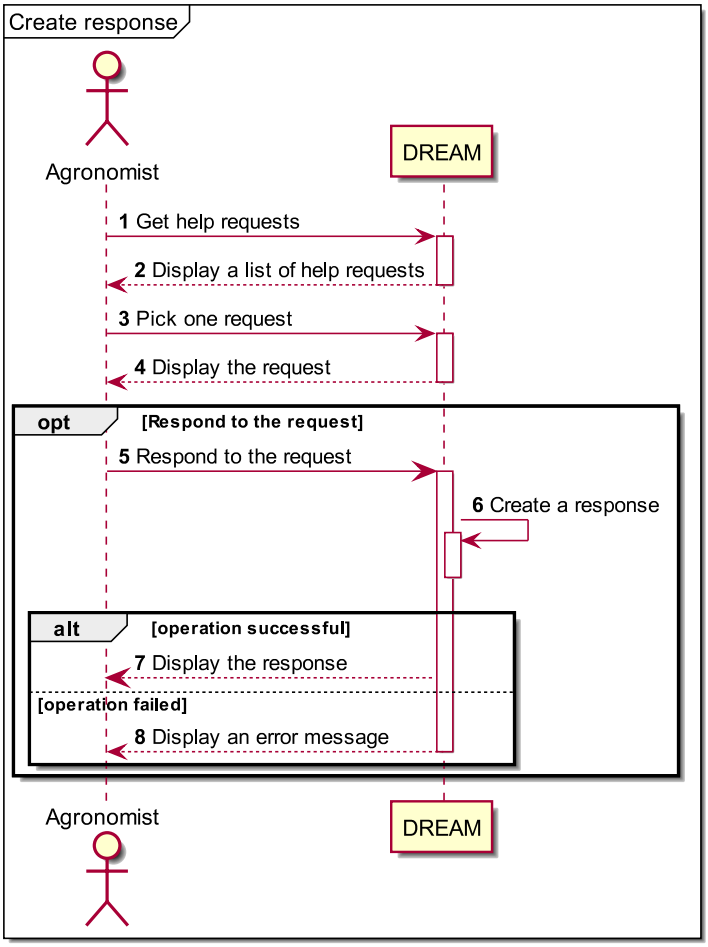

Figure 3.30: Sequence diagram presenting the event of answering to a help request.

Name	View farmer's summary
Actors	Agronomist, Farmer, Policy maker
Goals	G2, G3, G5, G6
Entry conditions	• The user is already logged in to the application. • The user is on the dashboard view.
Flow of events	1. If the user is Agronomist or Policy Maker a) The user clicks on *Farmers* in the sidebar. b) The application shows a list of farmers. c) The user finds in a list a farmer whose summary he wants to see. d) The user clicks on an eye icon. 2. If the user is Farmer a) The user clicks on *Summary* in the sidebar. 3. The application shows a farmer's summary page.
Exit conditions	The user is able to see the contents of farmer's summary.
Exceptions	• The application is not able to load the farmer's summary of the selected farmer. The application shows a message indicating a possible cause of the error.

Table 3.14: Use case description: View farmer's summary.

Name	View daily plan
Actors	Agronomist
Goals	G4
Entry conditions	• The agronomist is already logged in to the application. • The agronomist is on the dashboard view.
Flow of events	1. The agronomist clicks on *Daily plan* in the sidebar. 2. The application shows daily plan view for the current day.
Exit conditions	The agronomist can see daily plan for the current day. The agronomist can choose another day from the calendar on the left side of the view.

Exceptions	• The application is not able to load the daily plan of the agronomist. The application shows a message indicating a possible cause of the error.

Table 3.15: Use case description: View daily plan.

Name	Set daily plan execution state
Actors	Agronomist
Goals	G4
Entry conditions	• The agronomist is already logged in to the application. • The agronomist is in the *Daily plan* view.
Flow of events	1. The agronomist clicks on a day in the calendar for which execution state is to be set. 2. The agronomist clicks on *Submit execution state* button. 3. The application shows a pop-up with a list of visits in a selected day. 4. The agronomist marks each checkbox next to the visits he has done, leaving unmarked checkboxes next to the visits he skipped. 5. The application shows text fields below each visit with marked checkbox. 6. The agronomist fills text fields shown in the previous step with comments he collected during the visits. 7. The agronomist clicks on *Submit* button to save the execution state. 8. The application saves the execution state. 9. The application plans new visits to farms that were visited in terms of a casual visit in approximately half of a year. 10. The application plans new visits to the farms that were not visited, but should be visited in terms of a casual visit, in approximately the next 5 days.
Exit conditions	The state of visits for the given date is saved in the system. New visits are planned in dates depending on the state of the last visit (if it was *confirmed* or *rejected*).

Exceptions	• The agronomist cancels the operation in the steps 3-6 by clicking on the *Cancel* button or a cross in the top right corner of the pop-up. The application comes back to the daily plan view. • The application is not able to save the execution state in the step 8. The application shows a message indicating a possible cause of the error.

Table 3.16: Use case description: Set daily plan execution state.

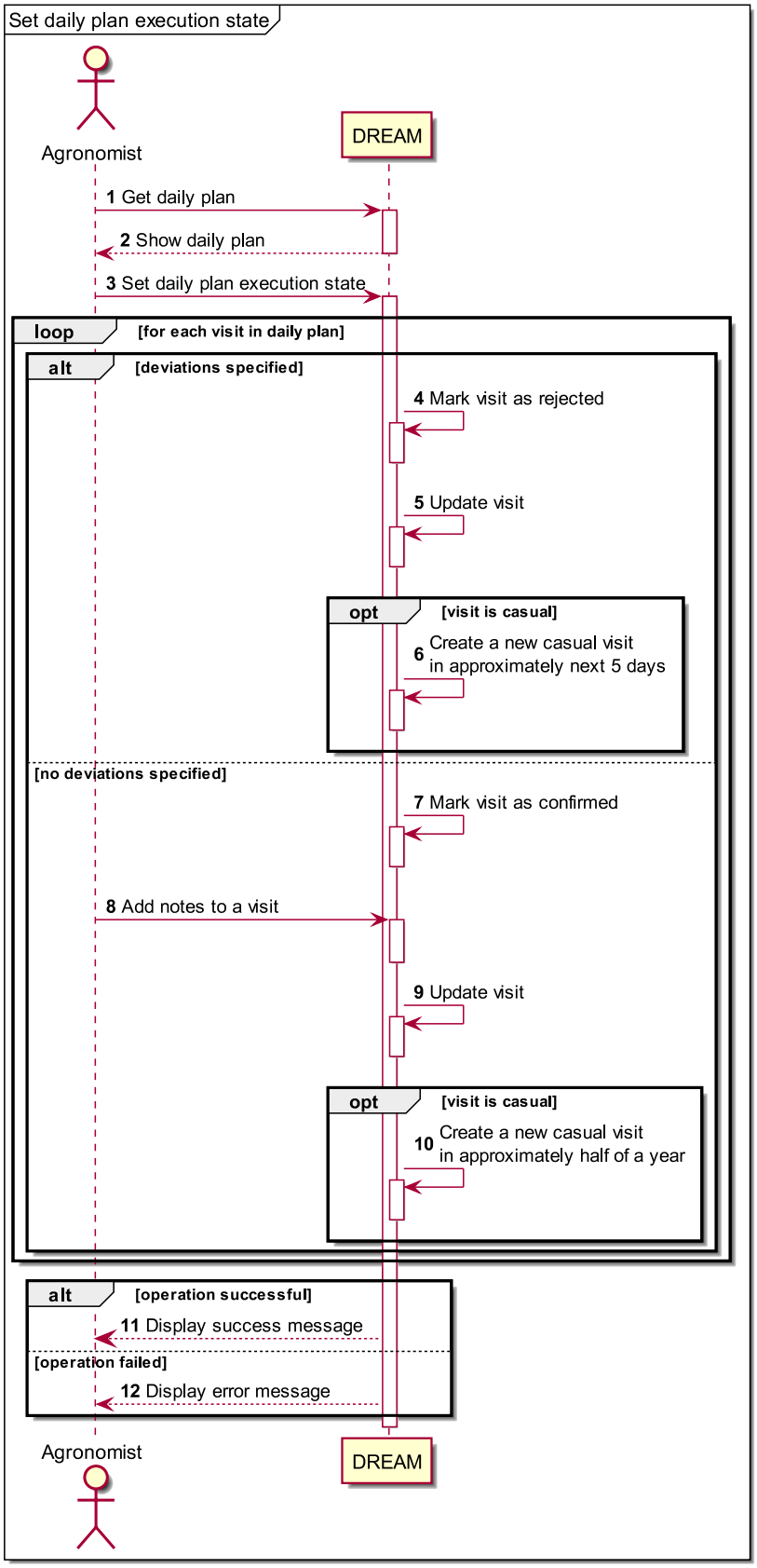

Figure 3.31: Sequence diagram presenting the action of displaying a daily plan.

Name	Manage daily plan
Actors	Agronomist
Goals	G4
Entry conditions	• The agronomist is already logged in to the application. • The agronomist is in the daily plan view.
Flow of events	1. The agronomist clicks on the day in the calendar in which he wants to rearrange the visits. 2. The application shows a list of visits in the selected day. 3. The agronomist clicks on an edit icon to change the date of the selected visit. 4. The application shows a pop-up with a field to choose a new date for the visit. 5. The agronomist selects a new date for the visit selected in the previous step. 6. The agronomist confirms his choice. 7. The application saves changes made to the daily plan. 8. The application shows a message indicating successful save of the visit's date.
Exit conditions	The calendar in the *Visit plan* view was modified accordingly.
Exceptions	• An agronomist closes the pop-up shown in the step 4. The system shows the daily plan tab view. • An agronomist wants to delay a casual visit for more than 5 days. The application shows an error, not allowing to delay the visit too much. • The application is not able to save updates to the daily plan in step 6. The application shows a message indicating a possible cause of the error.

Table 3.17: Use case description: Update daily plan.

3.2.2 Mapping on requirements

G1	Improve farmers performance by providing them with personalized suggestions.
A5.	Each farm is inside exactly one mandal.
A6.	Each farmer reliably updates his production data each month.
A8.	Agronomist's area of responsibility contains at least one mandal.

A10.	The information provided by a user during registration process is valid.
A11.	Well-performing farmers and agronomists are eager to help farmers with negative notes.
A12.	Suggestions created by agronomists are relevant and helpful for the farmers.
A13.	Each mandal has at least one agronomist who is responsible for the farms inside that area.
A20.	Production types are predefined.
R1.	The system must uniquely identify each user by his e-mail.
R2.	The system must allow an unregistered user to create an account with a chosen role.
R3.	The system must ensure that an agronomist chooses the area of responsibility during the registration process.
R4.	The system must ensure that a farmer inserts his farm data during the registration process.
R6.	The system must allow a registered user to log in to the application.
R7.	The system must allow a registered user to reset his password.
R8.	The system must allow a logged-in user to sign out of the application.
R9.	The system must allow a registered user to delete his account.
R25.	The system must allow a farmer to see personalized suggestions.
R26.	The system must allow a farmer to manage his monthly production data.
R44.	The system must allow an agronomist to manage his area of responsibility.
R45.	The system must allow an agronomist to see the list of suggestions for mandals in his area of responsibility.
R46.	The system must allow an agronomist to manage the list of suggestions for mandals in his area of responsibility.

Table 3.18: G1 mapping on assumptions and requirements.

G2	Acquire, combine, and visualize data from external systems.
A3.	Each farm has at least one humidity sensor.
A4.	Each farm has a water irrigation system.
A5.	Each farm is inside exactly one mandal.
A7.	Weather is consistent in a given mandal.
A10.	The information provided by a user during registration process is valid.
A17.	External systems are reliable and highly available.
A18.	Every sensor system has a unique ID assigned by the sensor system provider, which allow accessing the water usage data via an API.

A19.	Every water irrigation system has a unique ID assigned by the water irrigation system provider, which allow accessing the soil humidity data via an API.
R1.	The system must uniquely identify each user by his e-mail.
R2.	The system must allow an unregistered user to create an account with a chosen role.
R3.	The system must ensure that an agronomist chooses the area of responsibility during the registration process.
R4.	The system must ensure that a farmer inserts his farm data during the registration process.
R5.	The system must ensure that two casual farm visits are scheduled for each year of the farm's existence.
R6.	The system must allow a registered user to log in to the application.
R7.	The system must allow a registered user to reset his password.
R8.	The system must allow a logged-in user to sign out of the application.
R9.	The system must allow a registered user to delete his account.
R23.	The system must allow a policy maker to see a farmer's summary.
R24.	The system must allow a farmer to see his own farmer's summary.
R37.	The farmer is able to see farmer's summary of a farmer with a negative note whose help request he received.
R51.	The system must allow an agronomist to see a farmer's summary in his area of responsibility.
R65.	The system must read and update weather forecasts every day.
R66.	The system must read and store data from humidity sensors every day.
R67.	The system must read and store data from water irrigation systems every day.

Table 3.19: G2 mapping on assumptions and requirements.

G3	Facilitate performance assessment of the farmers.
A1.	Each farmer possess exactly one farm.
A2.	Each farm belongs to exactly one farmer.
A3.	Each farm has at least one humidity sensor.
A4.	Each farm has a water irrigation system.
A6.	Each farmer reliably updates his production data each month.
A7.	Weather is consistent in a given mandal.
A9.	Each registered user can be only one of the actors: an agronomist, a farmer, or a policy maker.
A10.	The information provided by a user during registration process is valid.

A14.	Based on the farmer's summary, a policy maker is able to understand whether the help given by farmers and agronomists produces significant results.
A17.	External systems are reliable and highly available.
A20.	Production types are predefined.
A21.	Problem types are predefined.
R1.	The system must uniquely identify each user by his e-mail.
R2.	The system must allow an unregistered user to create an account with a chosen role.
R3.	The system must ensure that an agronomist chooses the area of responsibility during the registration process.
R4.	The system must ensure that a farmer inserts his farm data during the registration process.
R5.	The system must ensure that two casual farm visits are scheduled for each year of the farm's existence.
R6.	The system must allow a registered user to log in to the application.
R7.	The system must allow a registered user to reset his password.
R8.	The system must allow a logged-in user to sign out of the application.
R9.	The system must allow a registered user to delete his account.
R10.	The system must allow a policy maker to assign a note to a farmer.
R11.	The system must ensure that every farmer initially has a neutral note.
R12.	The system must have a list of predefined farmer's problem types.
R13.	The system must allow a policy maker to specify a problem type when assigning a negative note.
R19.	The system must allow a policy maker to see a list of all farmers in Telangana.
R20.	The system must allow a policy maker to see a list of all farmers with a specific note.
R21.	The system must allow a policy maker to see a list of all farmers in a given mandal.
R22.	The system must allow a policy maker to find a farmer's summary by his name and surname.
R23.	The system must allow a policy maker to see a farmer's summary.

Table 3.20: G3 mapping on assumptions and requirements.

G4	Promote regular farms' visits by agronomists.
A1.	Each farmer possess exactly one farm.
A2.	Each farm belongs to exactly one farmer.
A5.	Each farm is inside exactly one mandal.
A8.	Agronomist's area of responsibility contains at least one mandal.

A10. The information provided by a user during registration process is valid.

A11. Well-performing farmers and agronomists are eager to help farmers with negative notes.

A13. Each mandal has at least one agronomist who is responsible for the farms inside that area.

A21. Problem types are predefined.

R1. The system must uniquely identify each user by his e-mail.

R2. The system must allow an unregistered user to create an account with a chosen role.

R3. The system must ensure that an agronomist chooses the area of responsibility during the registration process.

R4. The system must ensure that a farmer inserts his farm data during the registration process.

R5. The system must ensure that two casual farm visits are scheduled for each year of the farm's existence.

R6. The system must allow a registered user to log in to the application.

R7. The system must allow a registered user to reset his password.

R8. The system must allow a logged-in user to sign out of the application.

R9. The system must allow a registered user to delete his account.

R10. The system must allow a policy maker to assign a note to a farmer.

R11. The system must ensure that every farmer initially has a neutral note.

R12. The system must have a list of predefined farmer's problem types.

R13. The system must allow a policy maker to specify a problem type when assigning a negative note.

R14. The system must ensure that a farm is visited more often in the event of any problems.

R15. The system must have a specified number of additional visits caused by each predefined problem type.

R16. The system must ensure that only the most recently specified problem type is taken into account when determining the number of visits.

R17. The system must ensure that an automatic help request followed by additional farm visits are created, if a farmer receives a negative note.

R18. The system must ensure that all the additional farm visits created due to obtaining a negative note are deleted after a farmer's negative note is updated with a positive or a neutral one.

R44. The system must allow an agronomist to manage his area of responsibility.

R57. The system must allow an agronomist to see his daily plans.

R58.	The system must allow an agronomist to submit a daily plan's execution state, by rejecting or confirming each visit, on the same date or after the daily plan's date has passed.
R59.	The system must allow an agronomist to provide a comment for a visit he is confirming.
R60.	The system must ensure that after an agronomist submits his daily plan, for all the causal visits marked as confirmed, new ones are created in approximately half of a year.
R61.	The system must allow an agronomist to delete a visit before its date.
R62.	The system must ensure that a deleted visit is marked as rejected.
R63.	The system must ensure that in case of rejecting a casual visit, a new one is created in maximally 5 days.
R64.	The system must allow replanning any different visit than a casual one without any constraints.

Table 3.21: G4 mapping on assumptions and requirements.

G5	Enable agronomists to exchange information with farmers.
A5.	Each farm is inside exactly one mandal.
A8.	Agronomist's area of responsibility contains at least one mandal.
A10.	The information provided by a user during registration process is valid.
A11.	Well-performing farmers and agronomists are eager to help farmers with negative notes.
A13.	Each mandal has at least one agronomist who is responsible for the farms inside that area.
A16.	Farmers and agronomists create meaningful and not offensive help requests and responses.
A21.	Problem types are predefined.
R1.	The system must uniquely identify each user by his e-mail.
R2.	The system must allow an unregistered user to create an account with a chosen role.
R3.	The system must ensure that an agronomist chooses the area of responsibility during the registration process.
R4.	The system must ensure that a farmer inserts his farm data during the registration process.
R5.	The system must ensure that two casual farm visits are scheduled for each year of the farm's existence.
R6.	The system must allow a registered user to log in to the application.
R7.	The system must allow a registered user to reset his password.

R8.	The system must allow a logged-in user to sign out of the application.
R9.	The system must allow a registered user to delete his account.
R17.	The system must ensure that an automatic help request followed by additional farm visits are created, if a farmer receives a negative note.
R27.	The system must allow a farmer to see a list of help requests created by him.
R28.	The system must allow a farmer to find a help request created by him by the topic.
R29.	The system must allow a farmer to see a specific help request created by him.
R30.	The system must allow a farmer to delete a specific help request created by him.
R31.	The system must allow a farmer to create a new help request.
R44.	The system must allow an agronomist to manage his area of responsibility.
R47.	The system must allow an agronomist to see a list of all farmers in his area of responsibility.
R48.	The system must allow an agronomist to see a list of all farmers with a specific note in his area of responsibility.
R49.	The system must allow an agronomist to see a list of all farmers in a given mandal in his area of responsibility.
R50.	The system must allow an agronomist to find a farmer's summary in his area of responsibility by his name and surname.
R51.	The system must allow an agronomist to see a farmer's summary in his area of responsibility.
R52.	The system must allow an agronomist to see a list of all help requests in his area of responsibility.
R53.	The system must allow an agronomist to find a help request in his area of responsibility by the topic.
R54.	The system must allow an agronomist to see a specific help request in his area of responsibility.
R55.	The system must allow an agronomist to respond to a specific help request in his area of responsibility.
R56.	The system must allow an agronomist to delete a help response, only if it was created by him.

Table 3.22: G5 mapping on assumptions and requirements.

G6	Enable farmers to exchange their knowledge.
A10.	The information provided by a user during registration process is valid.
A11.	Well-performing farmers and agronomists are eager to help farmers with negative notes.
A15.	Farmers create meaningful and not offensive threads and comments on the forum.

A16. Farmers and agronomists create meaningful and not offensive help requests and responses.

A20. Production types are predefined.

A21. Problem types are predefined.

R1. The system must uniquely identify each user by his e-mail.

R2. The system must allow an unregistered user to create an account with a chosen role.

R3. The system must ensure that an agronomist chooses the area of responsibility during the registration process.

R4. The system must ensure that a farmer inserts his farm data during the registration process.

R5. The system must ensure that two casual farm visits are scheduled for each year of the farm's existence.

R6. The system must allow a registered user to log in to the application.

R7. The system must allow a registered user to reset his password.

R8. The system must allow a logged-in user to sign out of the application.

R9. The system must allow a registered user to delete his account.

R17. The system must ensure that an automatic help request followed by additional farm visits are created, if a farmer receives a negative note.

R27. The system must allow a farmer to see a list of help requests created by him.

R28. The system must allow a farmer to find a help request created by him by the topic.

R29. The system must allow a farmer to see a specific help request created by him.

R30. The system must allow a farmer to delete a specific help request created by him.

R31. The system must allow a farmer to create a new help request.

R32. The system must allow a farmer with a positive note to see a list of all help requests in his mandal.

R33. The system must allow a farmer with a positive note to find a help request in his mandal by the topic.

R34. The system must allow a farmer with a positive note to see a specific help request in his mandal.

R35. The system must allow a farmer with a positive note to respond to a specific help request in his mandal.

R36. The system must allow a farmer with a positive note to delete a help response, only if it was created by him.

R37. The farmer is able to see farmer's summary of a farmer with a negative note whose help request he received.

R38. The system must allow a farmer to see a list of all forum threads in Telangana.

R39. The system must allow a farmer to find a forum thread by the topic.

R40. The system must allow a farmer to see a specific forum thread with its comments.

R41.	The system must allow a farmer to create a comment in a forum thread.
R42.	The system must allow a farmer to delete a comment in a forum thread, only if it was created by him.
R43.	The system must allow a farmer to create a forum thread.
R68.	The system must ensure that a farmer who creates a help request cannot be its recipient.

Table 3.23: G6 mapping on assumptions and requirements.

3.3 Performance Requirements

Given that the system is intended for Telangana farmers, agronomists, and policymakers, it is fair to assume that it will be utilized by about 30 000 people. Therefore, the following performance requirements, which were formulated taking into consideration the best practices [5], should be met.

3.3.1 Response Time

- During peak hours, the average response time should be 2 seconds.
- During peak hours, 99% of all response times must be shorter than 3 seconds.
- In each 10-minute period commencing on the hour, the average response time must be 2 seconds or less.
- 95% percent of all response times must be shorter than 2 seconds.

3.3.2 Workload

- The system must be capable of processing 20 transactions per second.
- The system must have no more than one hour of downtime every three months.

3.3.3 Scalability

- The system must be able to sustain a 5% yearly growth in new users.
- The system must be able to sustain a 10% yearly increase in the number of transactions per second.
- The system shall be able to extend its connections to new external systems providing relevant data as their number may arise.

3.3.4 Platform

DREAM has to be housed on a platform that meets the following criteria:

- Operating system: Windows 10 or Windows 11.
- *.NET 6.0* development platform installed.
- *PostgreSQL* object-relational database system.
- CPU: 1.4 GHz 64-bit processor compatible with x64 instruction set.
- Memory: Minimum 8 GB of RAM.
- Hard drive space: minimum of 20 GB of space.
- Network adapter: an Ethernet adapter capable of at least 1 gigabit per second throughput.

3.4 Design Constraints

This section describes the constraints of a design. These include uncontrolled limitations that are self-imposed in order to enhance the design.

3.4.1 Standards compliance

The system will collect data from external systems and sensors, as well as data entered by users during registration and application use, such as production data, agronomist's daily plan, farmer's summary, and notes. The data will be used exclusively for system purposes and will be treated discreetly in accordance with the General Data Protection Regulation [8] and IS 17428.1 [6] guidelines. In particular, no information relating to a specific person will be made public as a result of the statistical analyses undertaken.

3.4.2 Hardware limitations

The following are the hardware constraints that the end user must meet in order to use the system.

- Processor: 1.9 GHz x86- or x64-bit dual-core processor.
- Memory: 2 GB of RAM.
- Network: wired or wireless connection with bandwidth greater than 40 KBps (320 kbps) and latency under 200 ms.
- Web browser: Google Chrome, Firefox, Safari, Microsoft Edge, or Opera running on Windows, macOS, Linux, or mobile operating systems including Android and iOS, with *JavaScript* enabled.

3.4.3 Any other constraints

DREAM will employ stateless protocols to enable components to be handled and changed without impacting the overall system. Additionally, the system will catalog, retrieve and run queries on all the relevant data using a relational database.

3.5 Software System Attributes

The system's characteristics that facilitate the measurement of its performance are presented in this section.

3.5.1 Reliability

The mean time between failures shall be equal to 120 hours, which means that in the worst case scenario it may break once every 5 days. Furthermore, the system shall be fault-tolerant, with its architecture prepared for any potential damage to system components by having replicas ready to be substituted. In order to recover from eventual data losses, redundancy should be considered for the system's database implementation.

3.5.2 Availability

As mentioned in the section 3.3, the system shall have no more than one hour of downtime every three months, thus, shall be highly available. In case of any planned maintenance works, all the users should be notified at least 48 hours in advance.

3.5.3 Security

The system shall implement role-based access control, which is a method of restricting system access to authorized users and granting permissions based on the user's role. It is accomplished through the use of both authentication and authorization implemented in the backend. The former is based on verifying the identity of a user, which can be an agronomist, a farmer, or a policy maker, during the log in phase. The latter, on the other hand, validates the logged-in user's permissions to perform an action (for example, visualizing an agronomist's daily plan) before actually carrying it out. The system uses a pseudo-random number generator based on HMACSHA1 to implement password-based key derivation capabilities, PBKDF2. [7].

3.5.4 Maintainability

The system shall be written in a widely known programming language, which would ensure a high level of maintainability and a relatively simple induction for potential new members of the development team. Following that, it should be divided into modularized components to ease replacements and fixes in the event of a failure. The system's backend must be capable of supporting maintenance works on a copy, which will subsequently be deployed after being tested on pre-defined test suites, in order to ensure no downtime for the core functionalities.

Both the testing scenarios and the tests themselves should be defined. This contains both end-to-end testing and basic unit tests. The code coverage should then be monitored and controlled. In the event of a system failure, thorough crash reports must be sent to the developers.

Maintenance pauses will be arranged on occasion throughout the night, when user traffic is at its lowest.

3.5.5 Portability

DREAM will be offered as a web application, making it available on the vast majority of current web browsers, including desktop and mobile machines. Because of this approach, the system will be usable on a wide range of devices. User interface adjustments shall be made to make the application user-friendly on all the environments.

Chapter 4

Formal Analysis using Alloy

This chapter presents formal analysis of the model using Alloy. Alloy is both a language and a tool for defining and exploring structures [9]. The goal was to prepare an alloy model that would allow to us to verify the correctness of the most important requirements of our system. The first part of this chapter includes the alloy code that models our system. The second part presents requirements and assumptions proved by the instances of worlds generated during analysis.

4.1 Alloy model

```
sig Date in Int {} {this > 0}
let currentDate = 5

// Enums
enum Note {
    Positive,
    Neutral,
    Negative
}
enum VisitReason {
    NegativeNote,
    Casual,
    AgronomistDecision
}
enum WeatherType {
    Sunny,
    MostlyCloudy,
    Cloudy,
    Rainy,
    Lightning
}
enum VisitState {
    Planned,
    Confirmed,
    Rejected
}

// Signatures
```

```
sig ProblemType {}

abstract sig User {}
sig PolicyMaker extends User {}
sig Agronomist extends User {
    areaOfResponsibility: some Mandal,
    visits: disj set Visit
}
sig Farmer extends User {
    farm: disj one Farm,
    farmerNotes: disj set FarmerNote
}

sig FarmerNote {
    note: one Note,
    problemType: lone ProblemType,
    policyMaker: one PolicyMaker,
    owner: one Farmer,
    date: one Date
} {
    note != Negative => no problemType
    note = Negative => one problemType
}

sig Farm {
    mandal: one Mandal,
    sensorSystemResponses: disj set SensorSystemResponse,
    waterIrrigationSystemResponses: disj set WaterIrrigationSystemResponse,
    productions: disj set Production
}

sig Mandal {
    weatherSystemResponses: disj set WeatherSystemResponse
}

sig Production {
    productionType: one ProductionType
}

sig WaterIrrigationSystemResponse {}

sig SensorSystemResponse {}

sig WeatherSystemResponse {
    type: one WeatherType
}

sig ProductionType {}

sig HelpRequest {
    recipients: some (Agronomist + Farmer),
    author: one Farmer
} { author not in recipients }

sig HelpResponse {
    author: one (Agronomist + Farmer),
    helpRequest: one HelpRequest
} { author in helpRequest.recipients }
```

```
87
88  sig Visit {
89      reason: one VisitReason,
90      state: one VisitState,
91      farm: one Farm,
92      date: one Date
93  }
94
95  sig ForumThread {
96      author: one Farmer,
97      comments: disj set ForumComment
98  }
99
100 sig ForumComment {
101     author: one Farmer
102 }
103
104 sig Suggestion {
105     productionTypes: some ProductionType,
106     mandals: some Mandal
107 }
108
109 // FACTS, predicates, assertions
110 // FarmerNotes
111 fact {
112     ~owner = farmerNotes
113 }
114
115 pred isLatestFarmerNote [farmerNote: one FarmerNote, farmerNotes: set
     ↪ FarmerNote] {
116     no f: (farmerNotes - farmerNote) | f.date > farmerNote.date
117 }
118
119 pred latestFarmerNoteIsEq [f: Farmer, n: Note] {
120     one farmerNote: farmerNotes[f] |
121         (isLatestFarmerNote[farmerNote, farmerNotes[f]] && farmerNote.note
     ↪ = n)
122         ||
123         (n = Neutral && no farmerNotes[f])
124 }
125
126 fact OnlyAgronomistOrAFarmerWithPositiveNoteCanBeARecipientOfAHelpRequest{
127     all h: HelpRequest | all r: h.recipients | (r in Agronomist) || (r in
     ↪ Farmer && latestFarmerNoteIsEq[r, Positive])
128 }
129
130 assert VerifyOnlyNegativeNoteHasAProblemType {
131     all farmerNote: FarmerNote | farmerNote.note = Positive implies no
     ↪ farmerNote.problemType
132     all farmerNote: FarmerNote | farmerNote.note = Neutral implies no
     ↪ farmerNote.problemType
133     all farmerNote: FarmerNote | farmerNote.note = Negative implies one
     ↪ farmerNote.problemType
134 }
135 check VerifyOnlyNegativeNoteHasAProblemType
136
137 fact NoTwoNotesForTheSameDayAndFarmer {
138     no disj n1, n2: FarmerNote | n1.date = n2.date && n1.owner = n2.owner
```

```
}

// Mandals
fact NoMandalWithoutAnAgronomist {
    all m: Mandal | one a: Agronomist | m in a.areaOfResponsibility
}
assert VerifyEachMandalIsInsideAgronomistAreaOfResponsibility {
    no m: Mandal | all a: Agronomist | m not in a.areaOfResponsibility
}
check VerifyEachMandalIsInsideAgronomistAreaOfResponsibility

// Farms
fact NoFarmWithoutFarmer {
    all f: Farm | one frm: Farmer | frm.farm = f
}
assert VerifyNoFarmWithoutFarmer {
    no f: Farm | all frm: Farmer | frm.farm != f
}
check VerifyNoFarmWithoutFarmer

fact NoWaterIrrigationSystemResponseWithoutAFarm {
    all p: Production | one f: Farm | p in f.productions
}
assert VerifyNoProductionWithoutAFarm {
    no p: Production | all f: Farm | p not in f.productions
}
check VerifyNoProductionWithoutAFarm

fact NoWaterIrrigationSystemResponseWithoutAFarm {
    all res: WaterIrrigationSystemResponse | one f: Farm | res in f.
 ↪ waterIrrigationSystemResponses
}
assert VerifyNoWaterIrrigationSystemResponseWithoutAFarm {
    no res: WaterIrrigationSystemResponse | all f: Farm | res not in f.
 ↪ waterIrrigationSystemResponses
}
check VerifyNoWaterIrrigationSystemResponseWithoutAFarm

// HelpRequests
fact NoFarmerWithoutPositiveNoteIsARecipientOfAHelpRequest {
    no h: HelpRequest | one r: h.recipients | r in Farmer && (
 ↪ latestFarmerNoteIsEq[r, Neutral] || latestFarmerNoteIsEq[r, Negative])
}

assert VerifyAuthorOfHelpResponseMustBeInsideRequestRecipients {
    all hr: HelpResponse | one rec: hr.helpRequest.recipients | hr.author
 ↪ in rec
}
check VerifyAuthorOfHelpResponseMustBeInsideRequestRecipients

fact AgronomistIsARecipientOfAHelpRequestBasedOnAreaOfResponsibility {
    all h: HelpRequest | some r: h.recipients |
        r in Agronomist && h.author.farm.mandal in r.areaOfResponsibility
}

// SensorSystem
fact NoSensorSystemResponseWithoutAFarm {
```

```
192     all res: SensorSystemResponse | one f: Farm | res in f.
 ↪ sensorSystemResponses
193 }
194
195 assert VerifyNoSensorSystemResponseWithoutAFarm {
196     no res: SensorSystemResponse | all f: Farm | res not in f.
 ↪ sensorSystemResponses
197 }
198 check VerifyNoSensorSystemResponseWithoutAFarm
199
200 // WeatherSystem
201 fact NoWeatherSystemResponseWithoutAMandal {
202     all res: WeatherSystemResponse | one m: Mandal | res in m.
 ↪ weatherSystemResponses
203 }
204
205 assert VerifyNoWeatherSystemResponseWithoutAMandal {
206     no res: WeatherSystemResponse | all m: Mandal | res not in m.
 ↪ weatherSystemResponses
207 }
208 check VerifyNoWeatherSystemResponseWithoutAMandal
209
210 // Visits
211 fact NoVisitsWithoutAgronomist {
212     all v: Visit | one a: Agronomist | v in a.visits
213 }
214
215 fact CasualVisitMustBePlannedIfItWasRejectedOrConfirmed {
216     all disj v1, v2: Visit | (v1.state = Rejected || v1.state = Confirmed)
 ↪ && v1.reason = Casual
217         implies v2.reason = Casual && v1.date < v2.date && v2.state =
 ↪ Planned
218 }
219
220 fact NoVisitIsConfirmedBeforeItsDate {
221     no v: Visit | v.state = Confirmed && v.date > currentDate
222 }
223
224 fact NoVisitIsPlannedAfterItsDate {
225     no v: Visit | v.state = Planned && v.date <= currentDate
226 }
227
228 fact NoPlannedVisitCausedByNegativeNoteIfFarmerNoteIsNotNegative {
229     all v: Visit | v.reason = NegativeNote && v.state = Planned
230         implies latestFarmerNoteIsEq[~farm[v.farm], Negative]
231 }
232 assert VerifyNoPlannedVisitCausedByNegativeNoteIfFarmerNoteIsNotNegative {
233     no v: Visit | v.reason = NegativeNote && v.state = Planned
234         && not latestFarmerNoteIsEq[~farm[v.farm], Negative]
235 }
236 check VerifyNoPlannedVisitCausedByNegativeNoteIfFarmerNoteIsNotNegative
237
238 fact NoVisitDueToNegativeNoteIsPlannedBeforeTheDateOfTheLastNegativeNote {
239     all v: Visit | v.reason = NegativeNote && v.state = Planned
240         implies (one fn: farmerNotes[~farm[v.farm]] |
241             isLatestFarmerNote[fn, farmerNotes[~farm[v.farm]]]
242                 && fn.note = Negative && fn.date <= v.date )
243 }
```

```
244
245 fact NoMultipleVisitsDueToNegativeNoteOnTheSameDayToTheSameFarm {
246     no disj v1, v2: Visit |
247         v1.reason = NegativeNote
248         && v1.state = Planned
249         && v2.reason = NegativeNote
250         && v2.state = Planned
251         && v1.date = v2.date
252         && v1.farm = v2.farm
253 }
254
255 fact NoMultipleCasualVisitsOnTheSameDayToTheSameFarm {
256     no disj v1, v2: Visit |
257         v1.reason = Casual
258         && v1.state = Planned
259         && v2.reason = Casual
260         && v2.state = Planned
261         && v1.date = v2.date
262         && v1.farm = v2.farm
263 }
264
265 // ForumComment
266 fact NoForumCommentWithoutAForumThread {
267     all fc: ForumComment | one ft: ForumThread | fc in ft.comments
268 }
269 assert VerifyNoForumCommentWithoutAForumThread {
270     no fc: ForumComment | all ft: ForumThread | fc not in ft.comments
271 }
272 check VerifyNoForumCommentWithoutAForumThread
273
274 // Worlds
275
276 // World for testing correctness of help requests.
277 // There should be a farmer recipient, with multiple notes, being a
    ↪ recipient.
278 // His latest note should be positive.
279 pred showHelpRequestsForFarmersWithPositiveNote {
280     #WaterIrrigationSystemResponse = 0
281     #SensorSystemResponse = 0
282     #WeatherSystemResponse = 0
283     #ForumThread = 0
284     #ForumComment = 0
285     #ProductionType = 0
286     #Visit = 0
287     #HelpResponse > 2
288     #Suggestion = 0
289     #Production = 0
290     #Mandal = 1
291
292     #Agronomist = 1
293     #Farmer = 2
294     #HelpRequest > 1
295     #PolicyMaker <= 3
296     #recipients >= 1
297
298     some f: Farmer | some ~recipients[f] && #farmerNotes[f] = 3
299 }
300 run showHelpRequestsForFarmersWithPositiveNote for 8
```

```
301
302 pred showWorldWithPlannedVisitDueToNegativeNote {
303     #ForumThread = 0
304     #ForumComment = 0
305     #HelpResponse = 0
306     #WaterIrrigationSystemResponse = 0
307     #SensorSystemResponse = 0
308     #WeatherSystemResponse = 0
309     #Production = 0
310     #Suggestion = 0
311     #HelpRequest = 0
312
313     #Mandal <= 3
314     #Visit >= 3
315     #Farmer >= 1
316     #PolicyMaker <= 1
317     #Agronomist = 1
318     #FarmerNote >= 2
319
320     some v: Visit | v.state = Confirmed
321     some v: Visit | v.state = Planned && v.reason = NegativeNote
322 }
323 run showWorldWithPlannedVisitDueToNegativeNote for 8
324
325 pred showWorldWithRejectedCasualVisit {
326     #ForumThread = 0
327     #ForumComment = 0
328     #HelpResponse = 0
329     #WaterIrrigationSystemResponse = 0
330     #SensorSystemResponse = 0
331     #WeatherSystemResponse = 0
332     #Production = 0
333     #Suggestion = 0
334     #HelpRequest = 0
335
336     #Mandal <= 3
337     #Visit >= 3
338     #Farmer >= 1
339     #PolicyMaker <= 1
340     #Agronomist = 1
341     #FarmerNote >= 2
342
343     some v: Visit | v.state = Rejected && v.reason = Casual
344 }
345 run showWorldWithRejectedCasualVisit for 8
346
347 pred showWorldFocusedOnForum {
348     #ForumThread >= 2
349     #ForumComment >= 5
350     #HelpResponse = 0
351     #WaterIrrigationSystemResponse = 0
352     #SensorSystemResponse = 0
353     #WeatherSystemResponse = 0
354     #Production = 0
355     #Suggestion = 0
356     #HelpRequest = 0
357
358     #Mandal <= 3
```

```
359     #Visit <= 2
360     #Farmer >= 5
361     #PolicyMaker <= 1
362     #Agronomist = 1
363     #FarmerNote <= 2
364 }
365 run showWorldFocusedOnForum for 8
366
367 pred show {
368     #WaterIrrigationSystemResponse = 1
369     #SensorSystemResponse = 1
370     #WeatherSystemResponse = 1
371     #ForumThread = 2
372     #ForumComment = 5
373     #HelpRequest >= 0
374     #HelpResponse >= 0
375     #Visit >= 0
376     #Farmer >= 2
377     #PolicyMaker <= 3
378     #Agronomist <= 3
379     #FarmerNote <= 2
380 }
381
382 run show for 8
```

4.2 Generated worlds

In order to manage the great number of different signatures, different constraints for world instances generation were proposed. Each of the instances presented below depicts a world focused on specific facts and signatures. To preserve readability, the atoms related to enumeration types were removed from the figures.

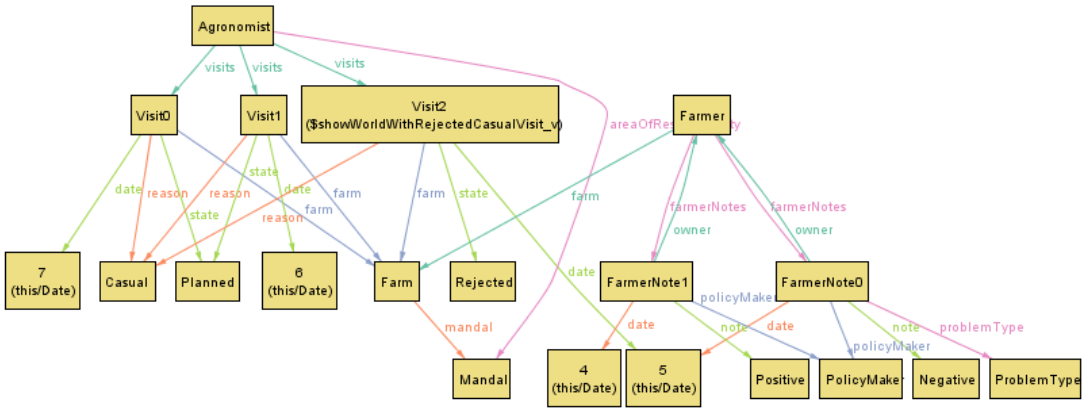

Figure 4.1: World instance focused on visits - rejected, causal visit

The first instance of worlds presented in Figure 4.1 shows that as stated in **R63**, if a casual visit is rejected, then a new casual visit is planned.

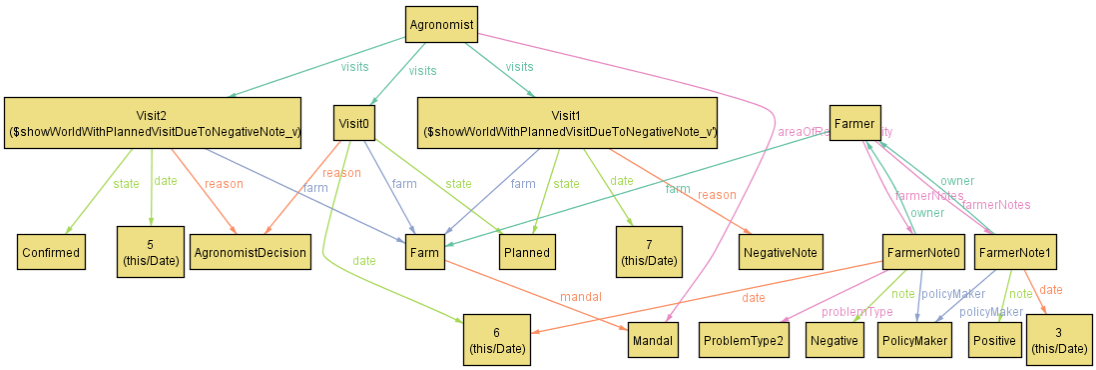

Figure 4.2: World instance focused on visits - planned, due to negative note visit

The world instance presented in figure 4.2 shows usage of the fact *NoVisitDueToNegativeNoteIsPlannedBeforeTheDateOfTheLastNegativeNote* as the visit due to negative note is planned on *date = 7* whilst the negative note was given on *date = 6*. Another requirement (**R58**) fulfillment that can be noticed is that only a visit on or after the current day can be confirmed (it follows the state chart presented in figure 2.2). The aforementioned requirement is exemplified with the date of the *Visit2* (*date = 5*, according to the alloy script in the previous section the *currentDay = 5*).

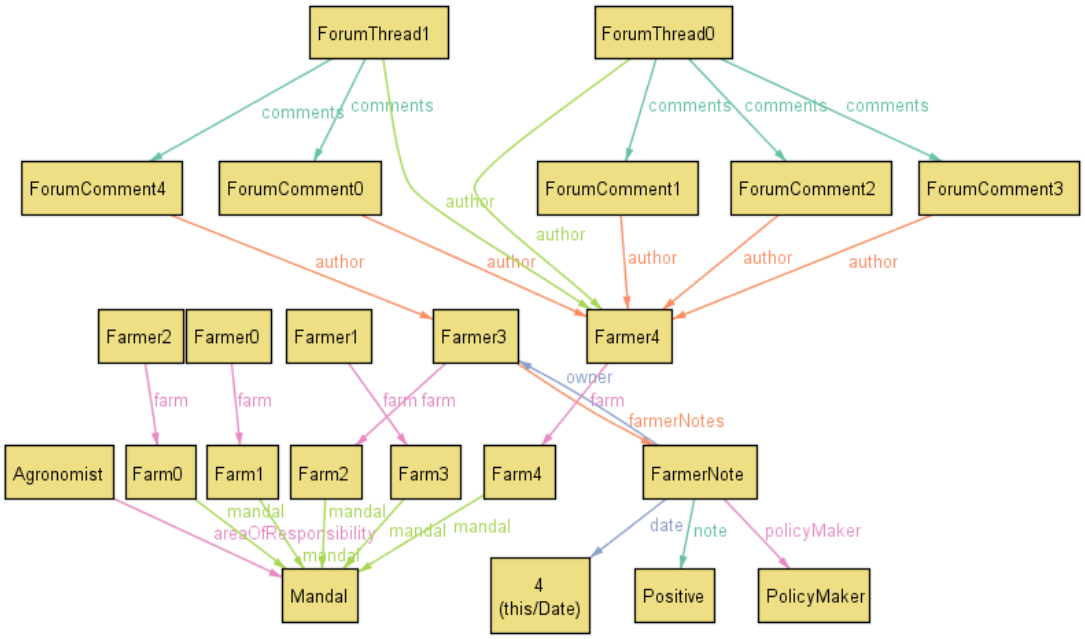

Figure 4.3: World instance focused on farmer's forum

The third world instance (figure 4.3) shows many atoms related to farmer's forum. Two forum threads are created by two different farmers and some comments are added. In addition, as assumptions **A1** and **A5** state, each farmer possesses exactly one farm that belongs to exactly one mandal.

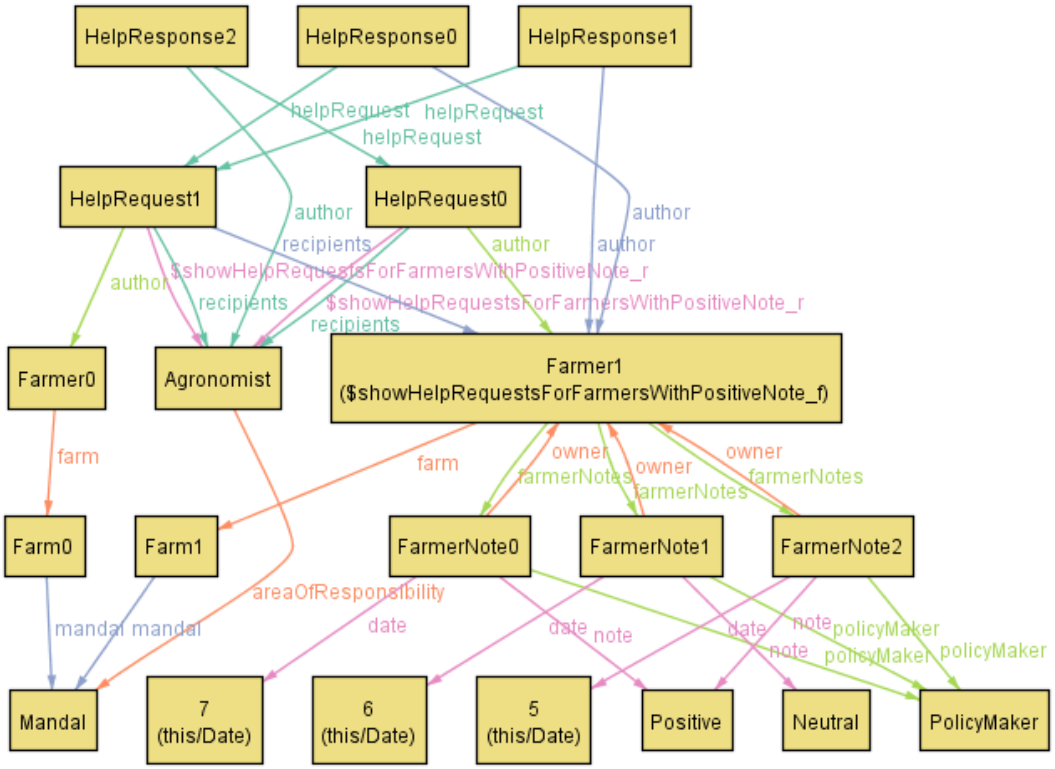

Figure 4.4: World instance focused on help requests

The fourth world instance (figure 4.4) focuses on help requests and responses. First, both help requests are obtained by an agronomist that takes care of the mandal (the mandal is inside the agronomist's area of responsibility) in which the author of the help request has his farm. Secondly, the *Farmer1* who created help responses is a farmer with a positive note as, which matches the requirement **R35**.

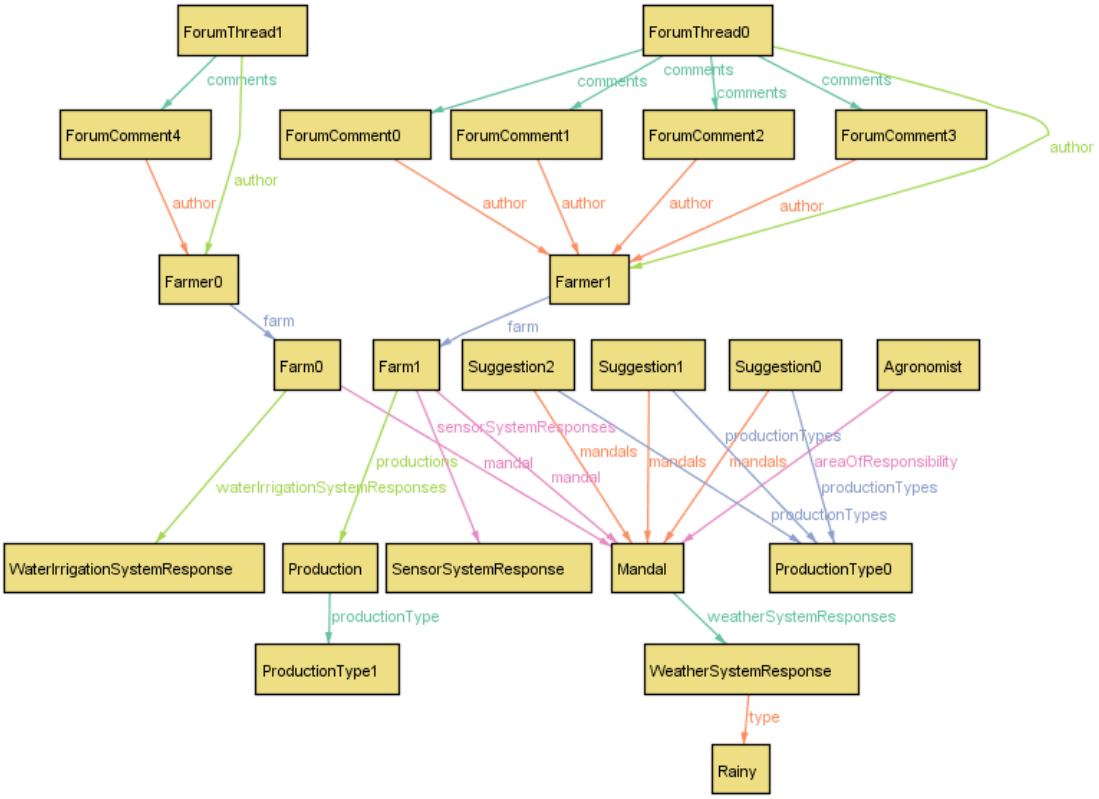

Figure 4.5: World instance without visits (focus shifted to other atoms)

The last figure, figure 4.5, presents an instance created with the least number of constraints. It contains some types that were not presented on previous figures for clarity. For instance, atoms presenting the production of *Farm1* are depicted.

Chapter 5

Effort Spent

Mariusz Wiśniewski

Topic	Hours	Date
Document Structure	1	14.11.2021
Getting familiar with the topic, discussion about the goals	2.5	17.11.2021
Purpose and scope	2.5	21.11.2021
Document structure adjustments	1	21.11.2021
Document scope definition	2	22.11.2021
Phenomena discussion	2	23.11.2021
Phenomena discussion, class diagram	3	24.11.2021
User characteristics	1.5	25.11.2021
Class diagram continuation	2.5	26.11.2021
Product functions	2	29.11.2021
Scenarios, product functions, and class diagram discussion	3	30.11.2021
Use case diagram adjustments	1	1.12.2021
Class and use case diagrams discussion, use case descriptions	2	2.12.2021
Performance requirements	3	2.12.2021
Design constraints and software system attributes	5	3.12.2021
Requirements discussion	2	8.12.2021
Class diagram description, state and sequence diagrams	4	10.12.2021
Use cases and UI mocks discussion	2	11.12.2021
Visit planning algorithm	3	12.12.2021
Verification of changes in mocks, requirements and assumptions	4	14.12.2021

Control of the coherence between diagrams and requirements	4	16.12.2021
Adjustments in requirements mapping and correction of typos	2	16.12.2021
Requirements discussion and mapping	4.5	17.12.2021
Document refinement and Alloy modelling	7	18.12.2021
Alloy modelling	3	19.12.2021
Document style and typos correction	0.5	22.12.2021
Formal Analysis using Alloy chapter verification	0.5	22.12.2021
Total	70.5	

Józef Piechaczek

Topic	Hours	Date
Document Structure	1	14.11.2021
Getting familiar with the topic, discussion about the goals	2.5	17.11.2021
Purpose and scope	3.5	21.11.2021
Phenomena	1.5	22.11.2021
Phenomena discussion	2	23.11.2021
Phenomena discussion, class diagram	3	24.11.2021
Scenarios	1.5	25.11.2021
Class diagram continuation	2.5	26.11.2021
Further work on scenarios	2	29.11.2021
Scenarios, product functions, and class diagram discussion	3	30.11.2021
Class diagram continuation, external interfaces requirements	1	1.12.2021
Class and use case diagrams discussion, use case descriptions	2	2.12.2021
UI Design	10	6-16.12.2021
Requirements discussion	2	8.12.2021
Use cases and UI mocks discussion	2	11.12.2021
Verification of changes in mocks, requirements and assumptions	4	14.12.2021
Class/Use case diagram adjustments	3	15.12.2021
Control of the coherence between diagrams and requirements	4	16.12.2021
Requirements discussion and mapping	4.5	17.12.2021
Document revision	3	17.12.2021
Document refinement and Alloy modelling	6	18.12.2021
Alloy modelling	3	19.12.2021

Total	67

Kinga Marek

Topic	Hours	Date
Document Structure	1	14.11.2021
Getting familiar with the topic, discussion about the goals	2.5	17.11.2021
Purpose and scope	3.5	21.11.2021
Phenomena	1.5	22.11.2021
Phenomena discussion	2	23.11.2021
Phenomena discussion, class diagram	3	24.11.2021
Scenarios	1.5	25.11.2021
Class diagram continuation	2.5	26.11.2021
Class diagram verification	2	29.11.2021
Scenarios, product functions, and class diagram discussion	3	30.11.2021
Use case descriptions	2	1.12.2021
Class and use case diagrams discussion, use case descriptions	2	2.12.2021
Functional requirements, domain assumptions improvements	3	3.12.2021
Requirements discussion	2	8.12.2021
Requirements mapping and use cases description	5	9.12.2021
Assumptions and scenarios consistency fixes	1	10.12.2021
Use cases and UI mocks discussion	2	11.12.2021
Fixes in UI mocks for consistency	1	13.12.2021
Use cases description and class diagram fixes for consistency	4	13.12.2021
Verification of changes in mocks, requirements and assumptions	4	14.12.2021
Requirement comments	1	15.13.2021
Requirements discussion and mapping	4.5	17.12.2021
Document refinement and Alloy modelling	6	18.12.2021
Alloy modelling	3	19.12.2021
Alloy modelling (code fixes, world instances preparation)	4	19.12.2021
Alloy description	1	21.12.2021
Total	68	

References

[5] André B. Bondi. “Best Practices for Writing and Managing Performance Requirements: A Tutorial”. In: *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. ICPE ’12. Boston, Massachusetts, USA: Association for Computing Machinery, 2012, pp. 1–8. ISBN: 9781450312028. DOI: 10.1145/2188286.2188288. URL: https://doi.org/10.1145/2188286.2188288.

[6] The Bureau of Indian Standards. *IS 17428.1 Data Privacy Assurance Part 1 Engineering and Management Requirements*. https://standardsbis.bsbedge.com/BIS_SearchStandard.aspx?Standard_Number=IS%2017428%20:%20Part%201&id=33561. 2020. (Visited on 12/03/2021).

[7] Microsoft. *Rfc2898DeriveBytes Class*. https://docs.microsoft.com/en-gb/dotnet/api/system.security.cryptography.rfc2898derivebytes?view=net-6.0. (Visited on 12/18/2021).

[8] The European Parliament and the Council of the European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council*. https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679. Apr. 2016. (Visited on 12/03/2021).

[9] MIT Software Design Group. *Alloy modeling language and analyzer*. https://alloytools.org. (Visited on 12/22/2021).