DD

January 9, 2022

Contents

1 Introduction

1.1 Purpose

Recently, the importance and necessity of resilient food systems have become evident with the large disturbances caused by the Covid-19 pandemic. This however has always been the case as the continuing threat of climate change to the agriculture sector demands a change in how we produce food.

With this in mind, the state of Telengana aims at developing a new system in order to aid an anticipatory, data driven governance model. It will combine information from several sources, such as water irrigation systems and the farmers themselves, in order to track the performance of individual farmers and analyze the impact of steering initiatives. The new system, DREAM, will moreover act as a platform where farmers and agronomists can communicate and share their knowledge.

This document contains a description of the architectural design for the system, including the components involved and how they interact. Additionally mockups of the user interface are presented and a plan for the implementation, testing and integration of the system. All in all, this document should guide the development of the system.

1.2 Scope

While there are several stakeholders to consider, this document only concerns itself with that of policy makers and farmers. Occasionally agronomists are mentioned as well but they should not be considered part of the project scope and are only mentioned to clarify or to help in understanding.

Furthermore, the system only concerns itself with farmers of Telengana and policy makers of the same state. This might mean that certain decisions, or requirements, might be inapplicable for other geographical areas.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Definition	Description
Performance	Performance w.r.t to farmers is used as a general term that for example could mean productivity. The performance metric can be decided on later is the idea.
Farm Identifier	To identify a specific farm, this could be e.g. a plot number or an organization number.

1.3.2 Abbreviations

Abbreviation	Description
RASD	Requirements Analysis and Specification Document
API	Application Programming Interface
RX	Requirement number X
DREAM	Data-dRiven PrEdictive FArMing in Telengana
CLI	Command Line Interface
DBMS	Database Management System
DB	Database
SI	Steering Initiative
MVC	Model View Controller

1.4 Revision History

1.5 Reference Documents

- The specification document "01. Assignment RDD AY 2021-2022.pdf"
- RASD

1.6 Document Structure

This document contains seven sections, detailed below.

- ~~In the first section background to the problem is given, along with the purpose of this specific report. Additionally, some necessary information in order to read the report is given, such as definitions and abbreviations.~~
- The second section focuses on describing and motivating the architectural design of the system to be. It starts with a high-level overview of the architecture and then breaks each part down into components. The components are described and their interdependence are shown in the component diagram. Moreover, the section contains a component interface diagram, a deployment view and sequence diagrams describing the interactions between components.
- In the third section design mockups of the user interface is presented.
- The fourth section contain the requirement traceability matrix, where each of the components described in section two is mapped to the requirements specified in the RASD. The mapping is based on whether the component contributes to the fulfilment of the requirement.
- Section five describes the suggested implementation and test plan for the system to be.
- Section six contains the effort spent on this report by the authors.
- Section seven contains the references used.

2 Architectural design

2.1 Overview: High-level components and their interaction

The architecture of DREAM follows the three-tier pattern with a presentation layer, business logic layer and a data layer.

A typical client-server architecture is also reasonable, considering that the system is a distributed web application. However, the decision to use a three-tier architecture was made in order to separate the business logic of the system from the data. The data in itself could reasonably be used for other applications in the future, and so decoupling them facilitates that. Moreover it provides an additional layer of safety as all data access will go through the middle layer.

Below, there is first a short description of every tier, and then a graphical representation of the architecture.

- **Presentation**

The presentation tier handles the interaction with the user, by communicating with the business tier and presenting the information retrieved. The idea is that this layer will be a simple website and it will only communicate with the server containing the business logic.

- **Business logic**

This tier contains all of the application's business logic, which in this case is e.g. the farmer performance calculations. Being the middle layer, it also handles the moving, and translation, of data back and forth between the data tier and presentation tier.

Moreover, the business logic tier is responsible for the communication with external services. In our case, this is for example the retrieval of water irrigation data for a farm.

- **Data**

The data tier contains and handles the persistent data of the system.

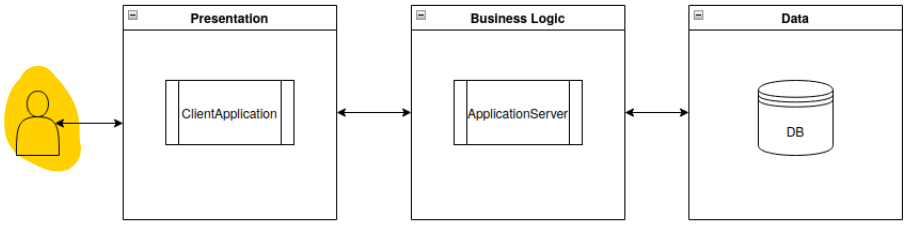

Figure 1: Overview of the system's three-tier architecture

2.2 Component view

In this section the component view is presented with a component diagram and corresponding descriptions. The diagram is constructed in the same fashion as the overview diagram with three tiers. Viewed left to right, starting with the presentation tier (in ClientApplication), then the business logic is contained in the ApplicationServer, and lastly the data tier is represented with the DBMS component.

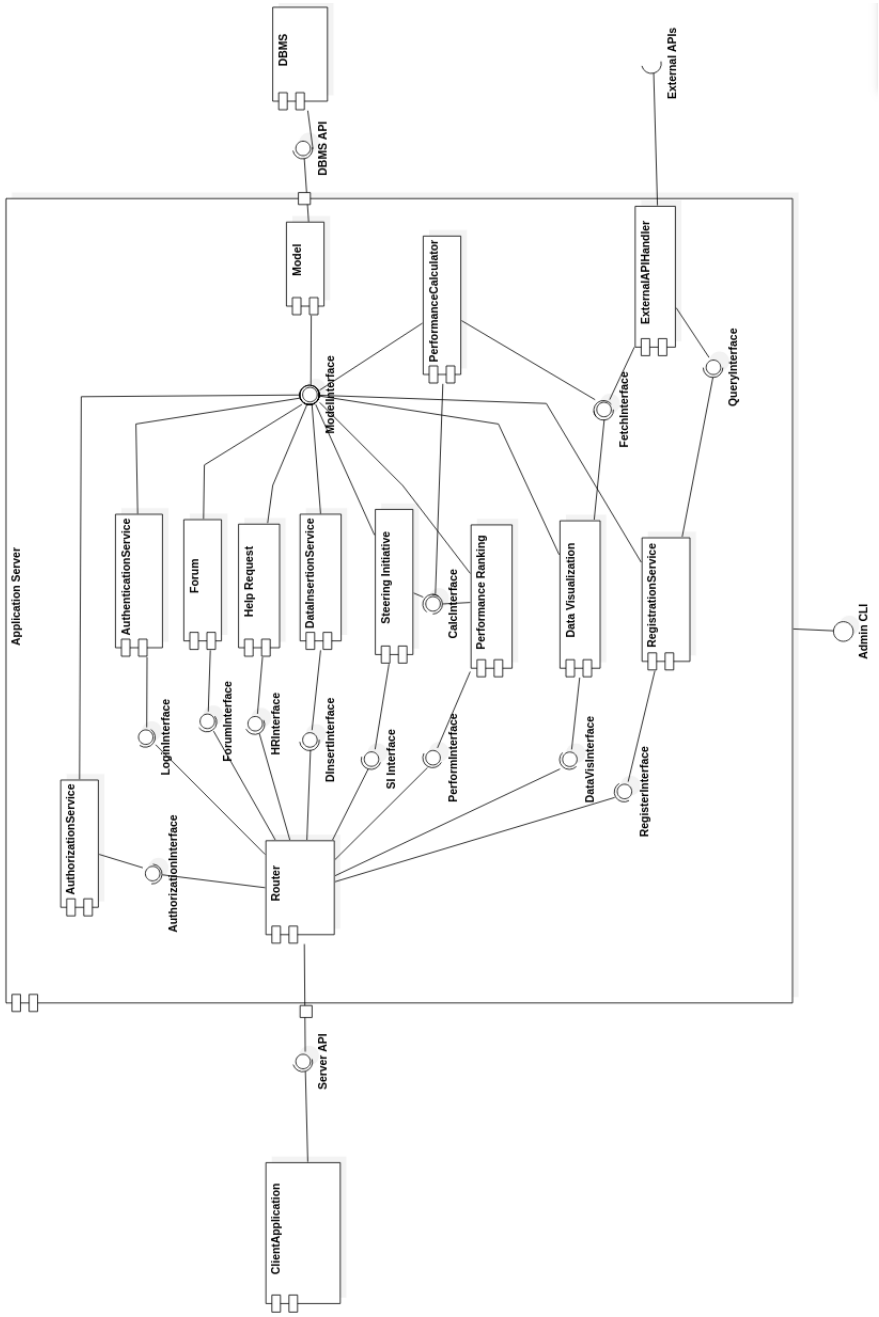

Figure 2: Component diagram of the DREAM

In the list below, all components are briefly described.

- **ClientApplication** - Client application component. E.g web browser session on a mobile phone.
- **Router** - Entry point of client requests to the server. Handles request by rerouting them to the correct component if authorized.
- **AuthorizationService** - Component that handles and verifies allowances depending on user type. E.g a farmer is not allowed to access pages meant for policy makers.
- **AuthenticationService** - For verifying user credentials and allowing farmers/policy makers to log in.
- **Forum** - Component that handles interaction with the forum service. Allows farmers to create posts or comment on existing ones.
- **Help Request** - Handles creating and responding to help requests by farmers.
- **DataInsertionService** - Component used when farmers input farm data, e.g crop amount.
- **SteeringInitiative** - Component handling viewing of steering initiatives on farmers. This component is accessible by policy makers.
- **Performance Ranking** - Responsible for ranking the performance of farmers and viewing performance graphs. This component is accessible by policy makers.
- **Data visualization** - The component responsible for creating the personalized view with data relevant to a specific farmer. Involves fetching of data from the DB and APIs.
- **RegistrationService** - The component containing the service used to handle the registration of farmers to DREAM.
- **PerformanceCalucator** - Responsible for calculating the performance of farmers. Placed in a separate component in order to easily add new metrics when determining farmer performance.
- **ExternalAPIHandler** - This component is responsible for handling communication with external services. In our case this is currently of two types, either fetching of e.g. water irrigation data, or querying for confirming farmer connection to farm (when registering).
- **Model** - This component contains an object-relation mapping (ORM) that other components use when retrieving data from the database. Thus, it is the component solely responsible for communicating with the data tier.
- **DBMS** - Database Management System. Contains the database for storing persistent data.

2.3 Deployment view

This section presents a deployment diagram of the system. It specifies the environments and tools to be used to build the system, coupled with the protocols needed to communicate between the various parts.

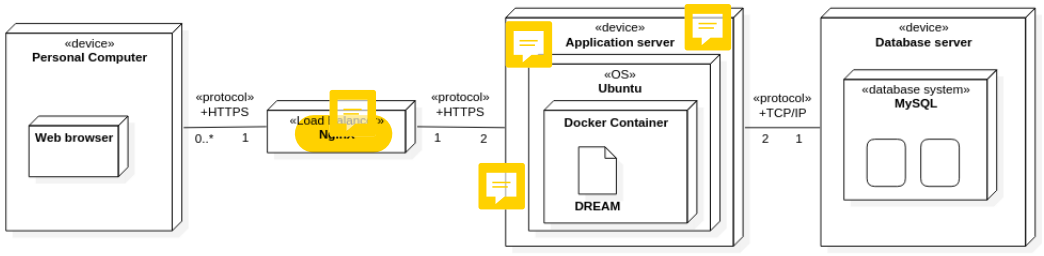

Figure 3: Deployment diagram

- **Personal Computer** - Personal computer running a web browser with internet access. Also includes mobile devices.
- **Load Balancer** - A web server that caches static files, handles incoming requests and balances the workload among the two application servers. NginX is used as the web server.
- **Application server** - Hosts the application logic of the system. Interacts with the database server. Two application servers will be used to increase reliability and to handle many concurrent requests.
- **Database server** - Hosts the database of the system. The idea is to have two databases running, a main one that will be used by DREAM and one replica that can be instated if something happens with the main one.

2.4 Runtime view

In this section sequence diagrams describing the way in which components of the system interact for the main functionalities are presented. We begin by describing the flow of a general request to the system, with authorization, in order to omit that part of the sequence for the following diagrams.

Since some functionalities are similar, or contained in one another, only one is presented and the rest briefly described.

For all the sequence diagrams, except for Login, it is assumed that the policy maker or farmer is already logged in.

2.4.1 General Request with Authorization

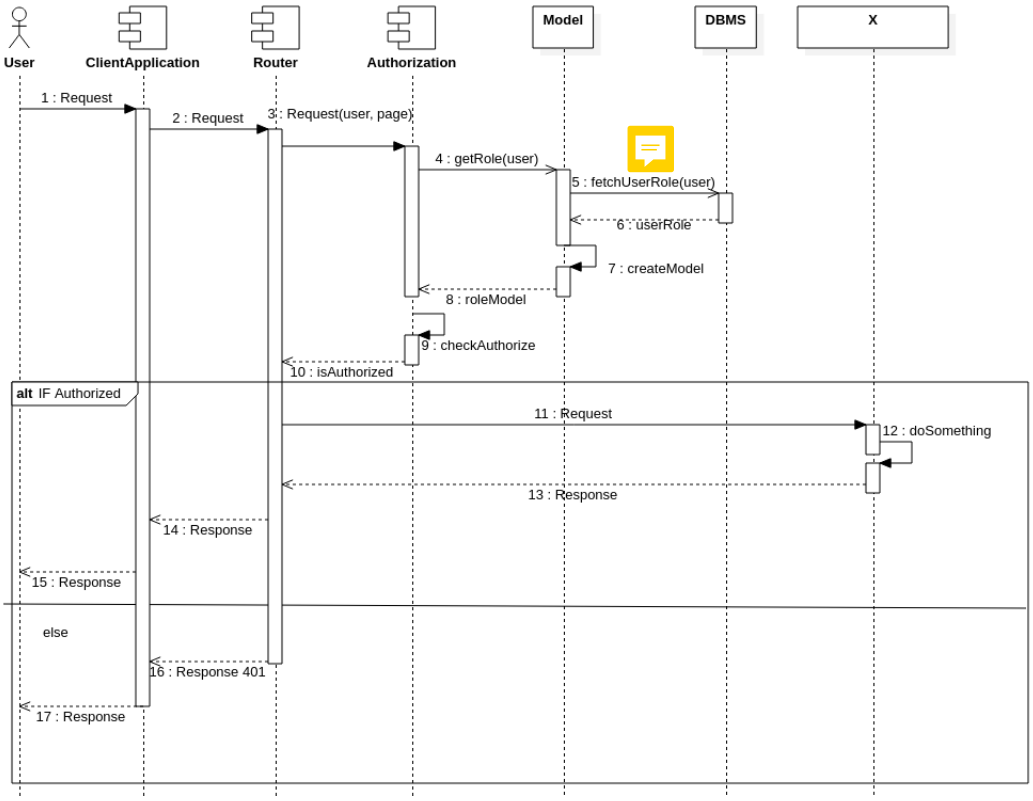

Figure 4: General request and authorization sequence

The sequence diagram above describes the process of authorizing a user before granting access to the requested page. This sequence happens on every request, but of course some caching should be used in order to avoid excessive database calls. In this diagram it is assumed that the user is already authenticated, and that their request contain information such that the system can identify them, e.g. cookie or some token.

The component "X" in the diagram just refers to any component, and message 12 is used to represent the whole flow starting and ending in X.

In the following diagrams, this sequence will be omitted. Meaning, the client application will make the request to the component X directly.

2.4.2 Farmer registering

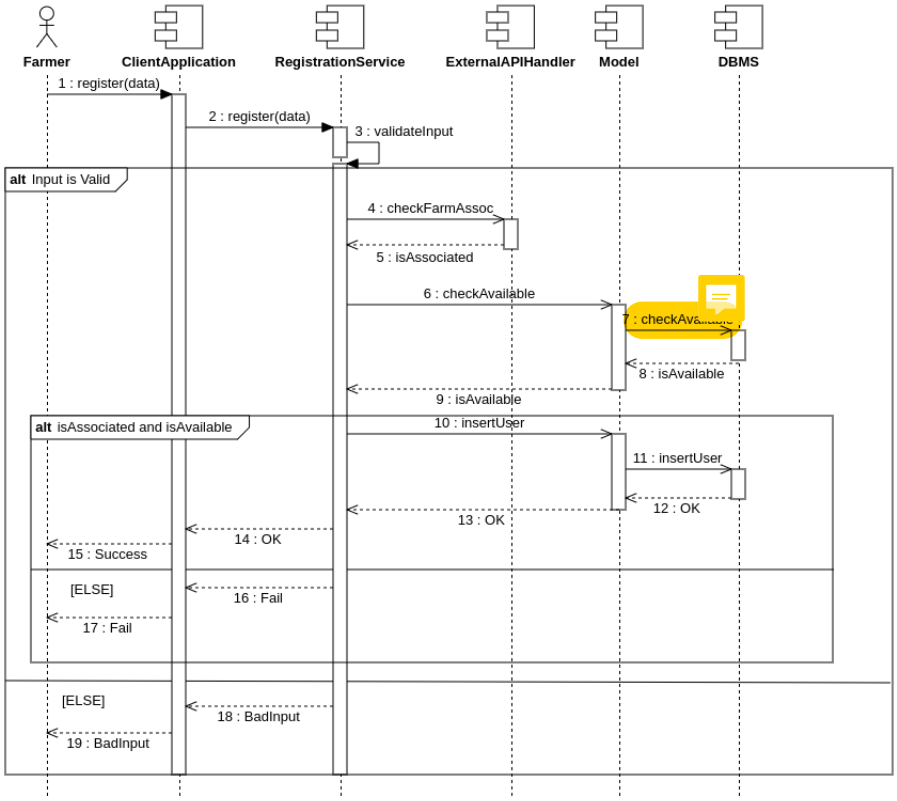

Figure 5: Sequence diagram for farmer registration

As mentioned previously, e.g. in the RASD document, it is assumed that their exists an external API where a farmer's association to a farm can be verified. This is done in the diagram above in the asynchronous message 4.

The diagram assumes that the farmer is already on the register view. Other than that, the diagram should be clear.

2.4.3 Login

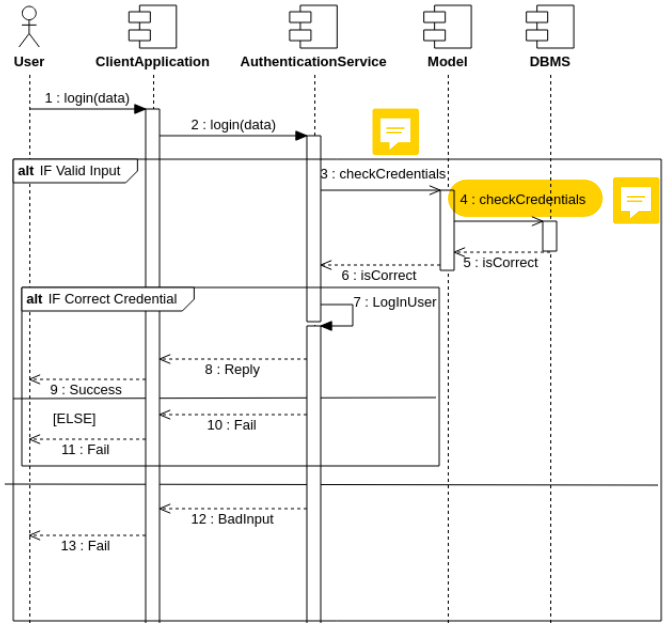

Figure 6: Sequence diagram for login

User is either a policy maker or a farmer, and is already on the login view.

In message 8 we could imagine that the authentication service will reply with a token that the user then can pass to the system in order to remain authenticated. Such a token is needed in order to authorize the user's request to the system. Exactly how, e.g. whether to use JSON Web Tokens (JWT) or regular cookie authentication, is not specified here.

2.4.4 Commenting on forum thread

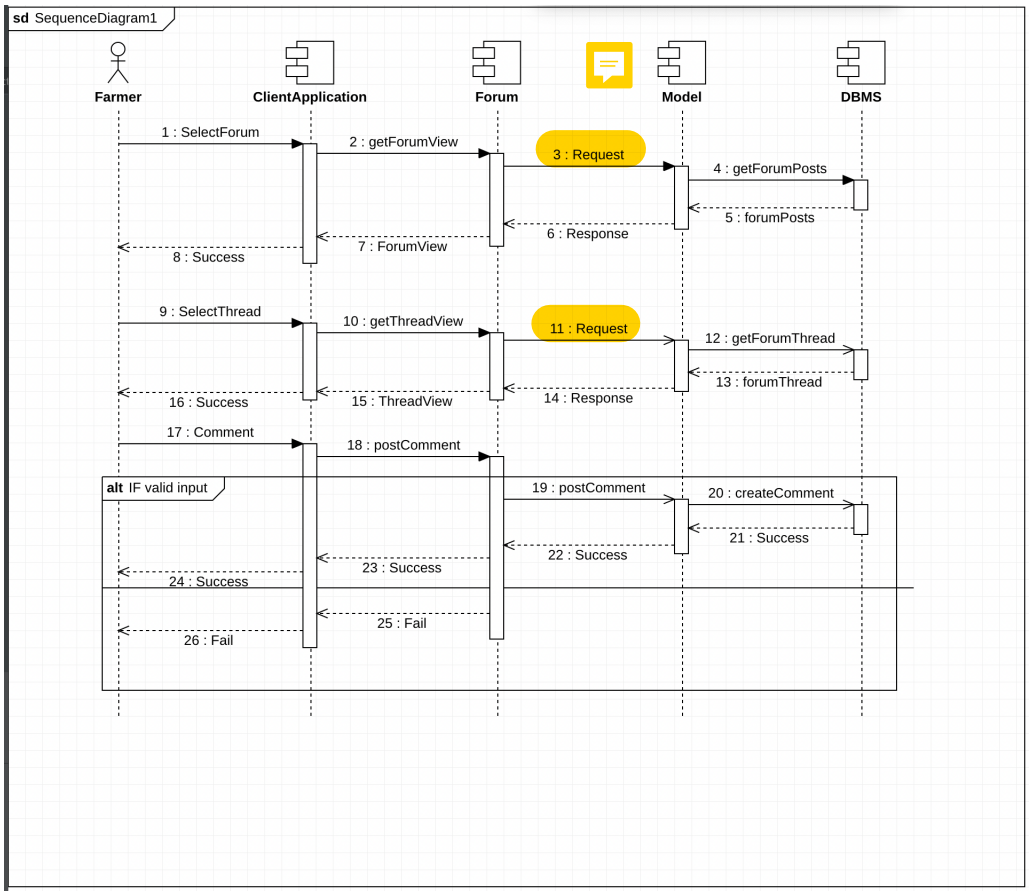

Figure 7: Sequence diagram for forum

This sequence diagram begins by the farmer wanting to view the forum (i.e a list of forum threads), followed by selecting a specific thread and then commenting on a thread.

2.4.5 Create forum thread

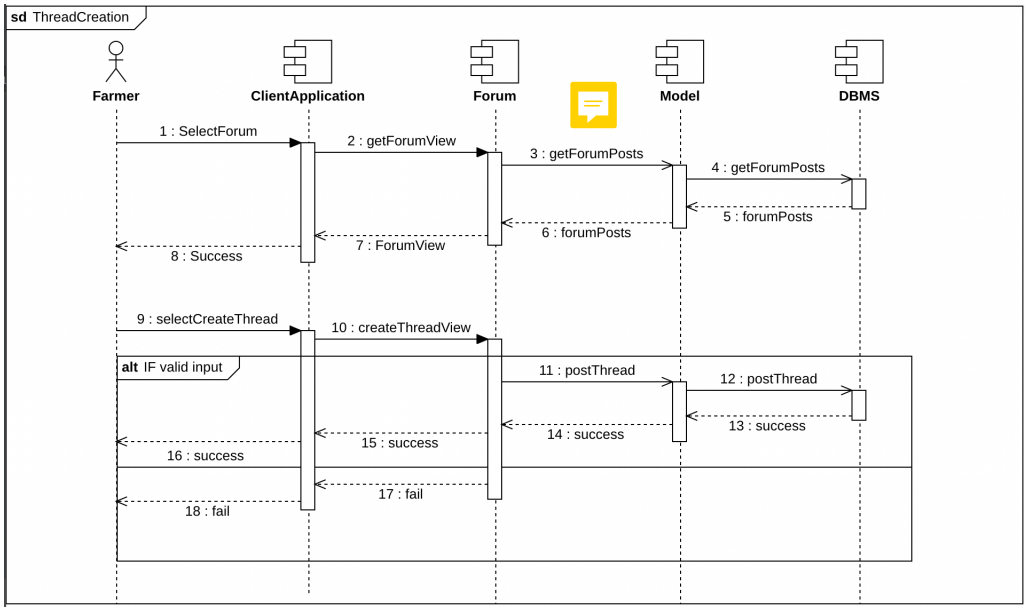

Figure 8: Sequence diagram for forum thread creation

2.4.6 Help Request

The sequence diagram(s) of the help request functionality is not included in the report as they are essentially equal to the one's in section 2.4.4 and 2.4.5.

2.4.7 Steering initiative

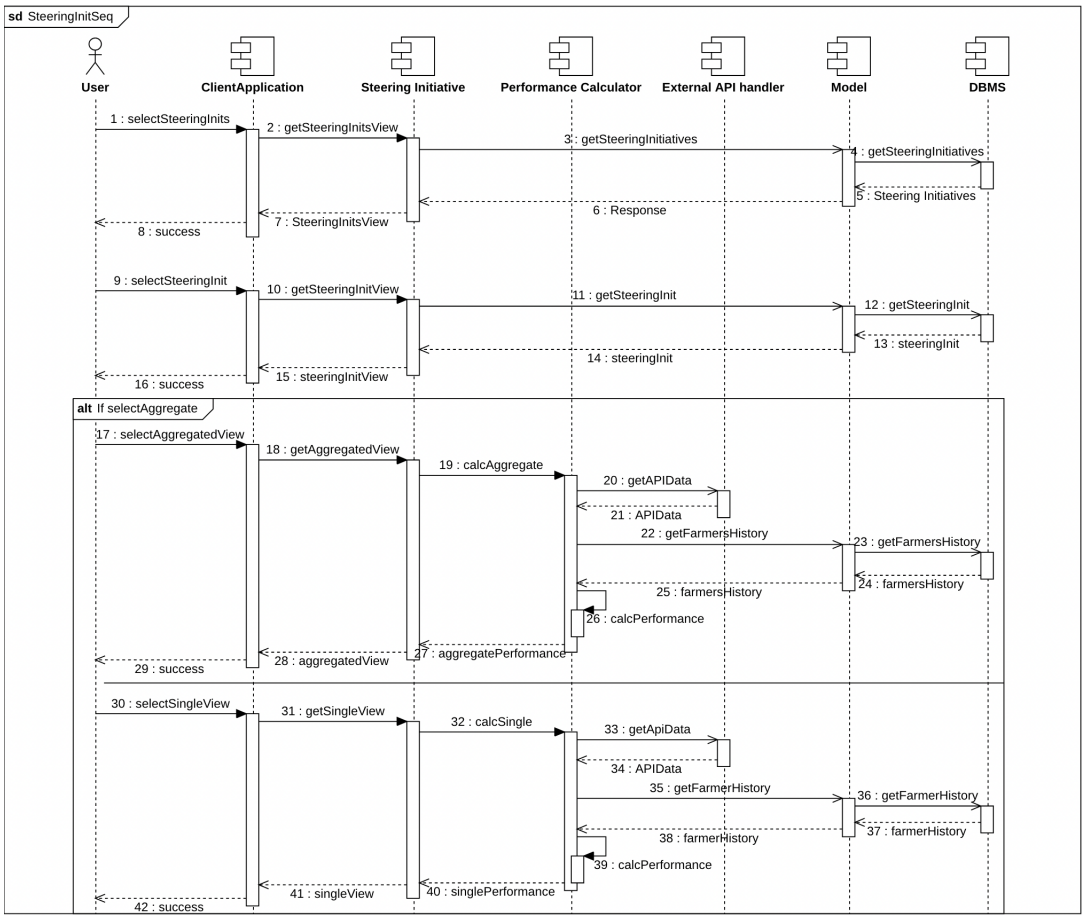

Figure 9: Sequence diagram for policy maker evaluating steering initiatives

From 1 - 16, this sequence diagram is quite similar to forum, essentially a policy maker selects to view a steering initiative. The difference is how he/she wants to view it, either as an aggregated view, or for a single farmer. The policy maker wants to analyze if the steering initiative has been effective or not, and for that the system needs to calculate the performance. This is done in the Performance Calculator, which fetches various data from the APIs, and past data inputted by farmers to calculate performance. The end goal of this process is essentially the same for both aggregated and single farmer, as the system will show a performance graph with the start of the steering initiative marked out.

2.4.8 Performance list

The sequence diagram for viewing a farmers performance is not included in this report since it is very similar to the performance calculation in the steering initiative diagram from section **2.4.7** if messages 9-16 are disregarded.

2.4.9 Inserting data

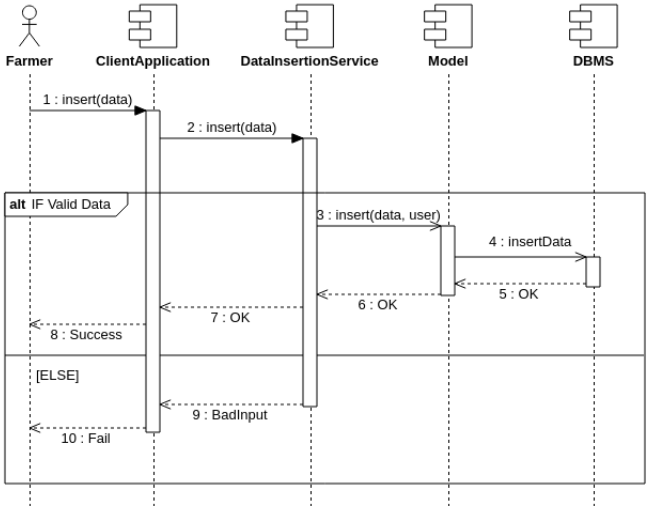

Figure 10: Sequence diagram for farmer inserting data about their production

The farmer is already on the data insertion view at the start of this diagram.

2.4.10 Visualize personal data

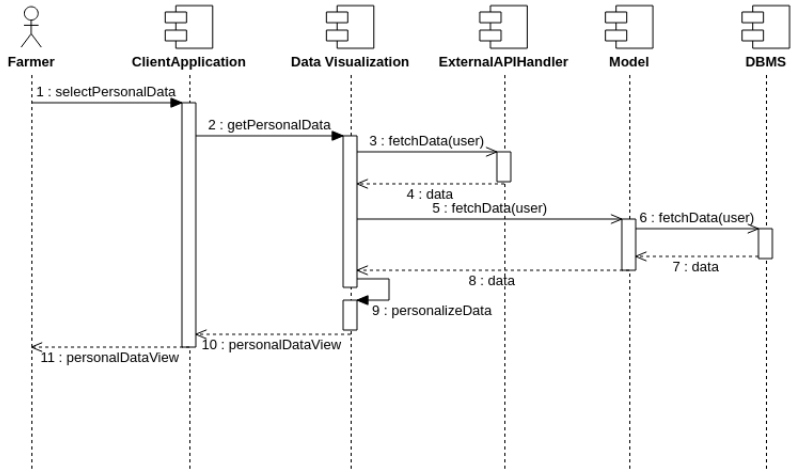

Figure 11: Sequence diagram for farmer viewing personalized data

2.5 Component interfaces

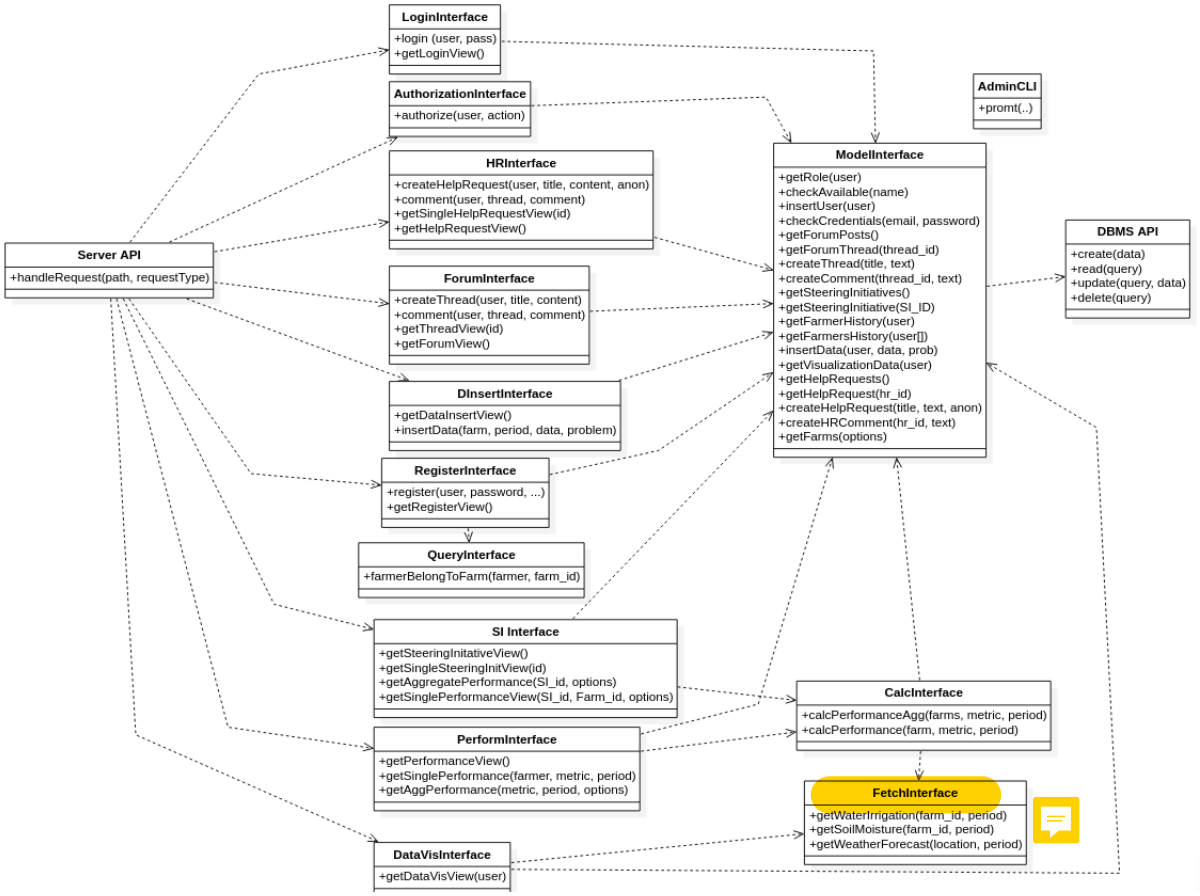

Figure 12: Diagram describing the component interfaces, and their dependencies

2.6 Selected architectural styles and patterns

- **Model View Controller (MVC)**
  For manageability and separation of concern, DREAM is built with the MVC pattern. The MVC pattern separates the components into:
  - **Model** - Backend data logic
  - **View** - Presentation layer (user interface)
  - **Controller** - Middle hand, gets input from view and converts into commands

Separating the presentation logic and data logic in this fashion results in a system that is much easier to manage, modify and test. This gets even more prominent as systems grow larger in size. The view layer in DREAM is represented by the ClientApplication component, model by the Model component and DBMS. The controller is represented by all the components that offers a interface that the router uses, such as the Forum component (which can be thought of as a ForumController). See the component diagram in Figure 2 for all such components.

- **Three tier architecture**

  The system is structured in three tiers, a presentation tier, business logic tier and a data tier. As previously mentioned, the reason for choosing this style instead of a traditional client-service architecture was mainly to increase the reusability of the data tier in particular. It is likely that the data acquired over time will be useful for other services, and thus it makes sense to decouple the data and business tier at this stage. Moreover, it facilitates scalability in an easy way by essentially allowing one tier to be replaced or given more resources without impacting the other two.

2.7 Other design decisions

The system is designed with the premise that data will be acquired from external APIs whenever needed. Another way would have been to continuously, on a schedule, fetch e.g. all water irrigation data. There are mainly two reasons behind this choice. Firstly it can be more flexible, introducing a new data source only imply updating the business logic layer. Secondly, duplication of the data does not seem necessary. Instead, having clear tasks for every service in the ecosystem makes more sense as they may be easier combined in different ways when decoupled.

Though, it should be noted that this approach makes the system more vulnerable in the sense that external services are required for full functionality. With that in mind, storing computations, i.e. when external data is combined in order to for example calculate farmer performance, would be a good idea. Especially so for the larger computations such as when computing an aggregated performance of many farmers during a long period of time.

3 User interface design

In this section mockups of the user interface are presented. The mockups are here only shown in the *PC* view, i.e. not the mobile design.

3.1 Farmer

3.1.1 Login

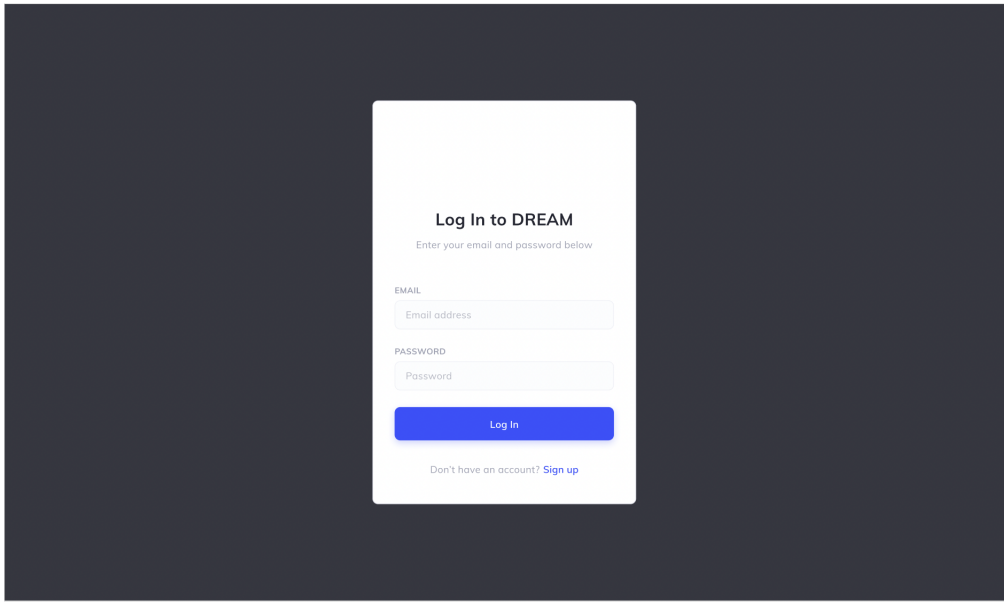

Figure 13: UI for signing in

3.1.2 Register

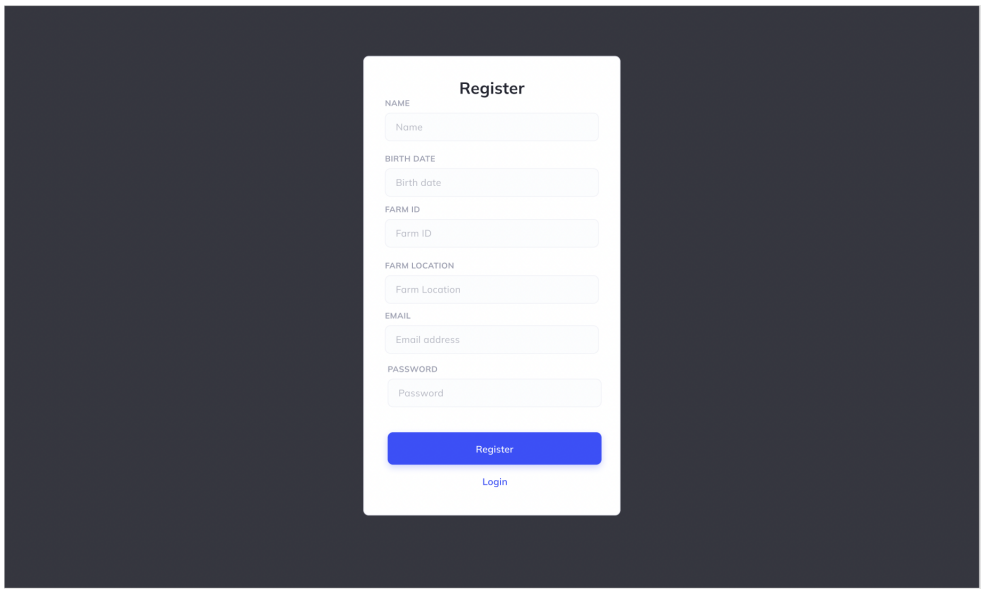

Figure 14: UI for registering an account as a farmer

3.1.3 Input data

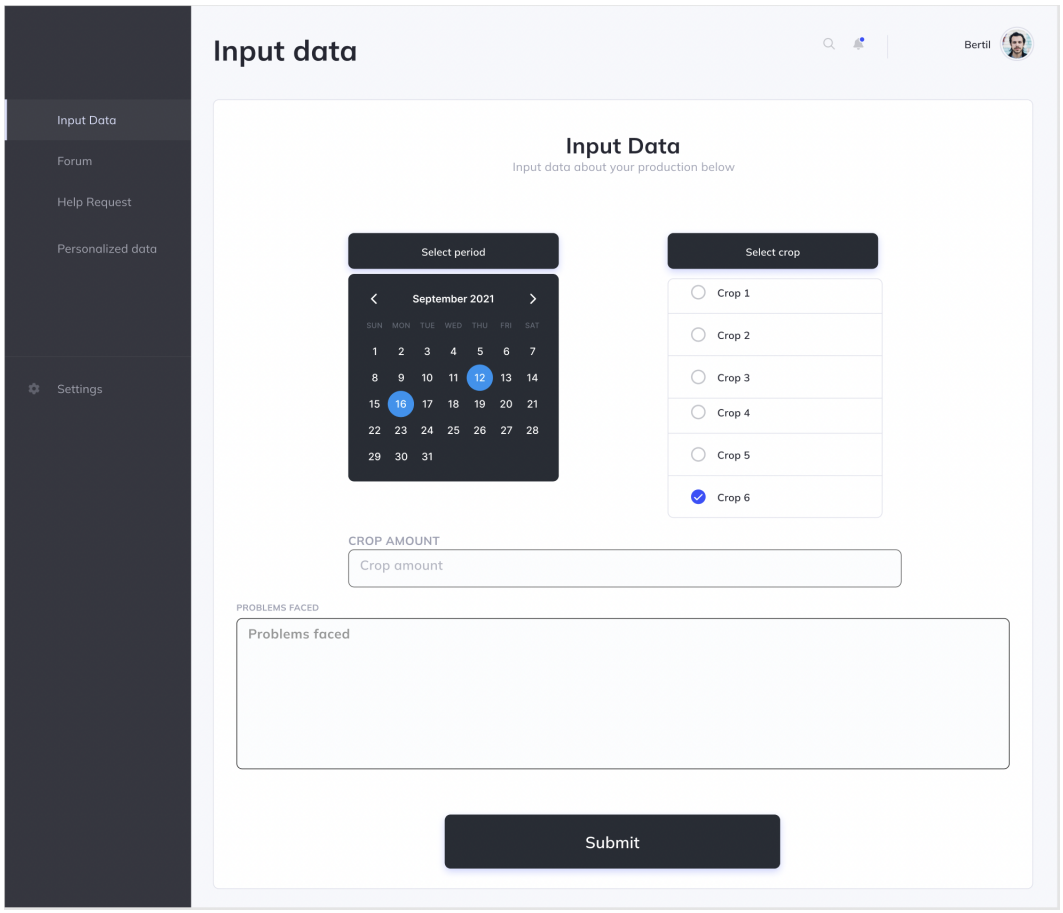

Figure 15: UI for inputting data

3.1.4 Forum

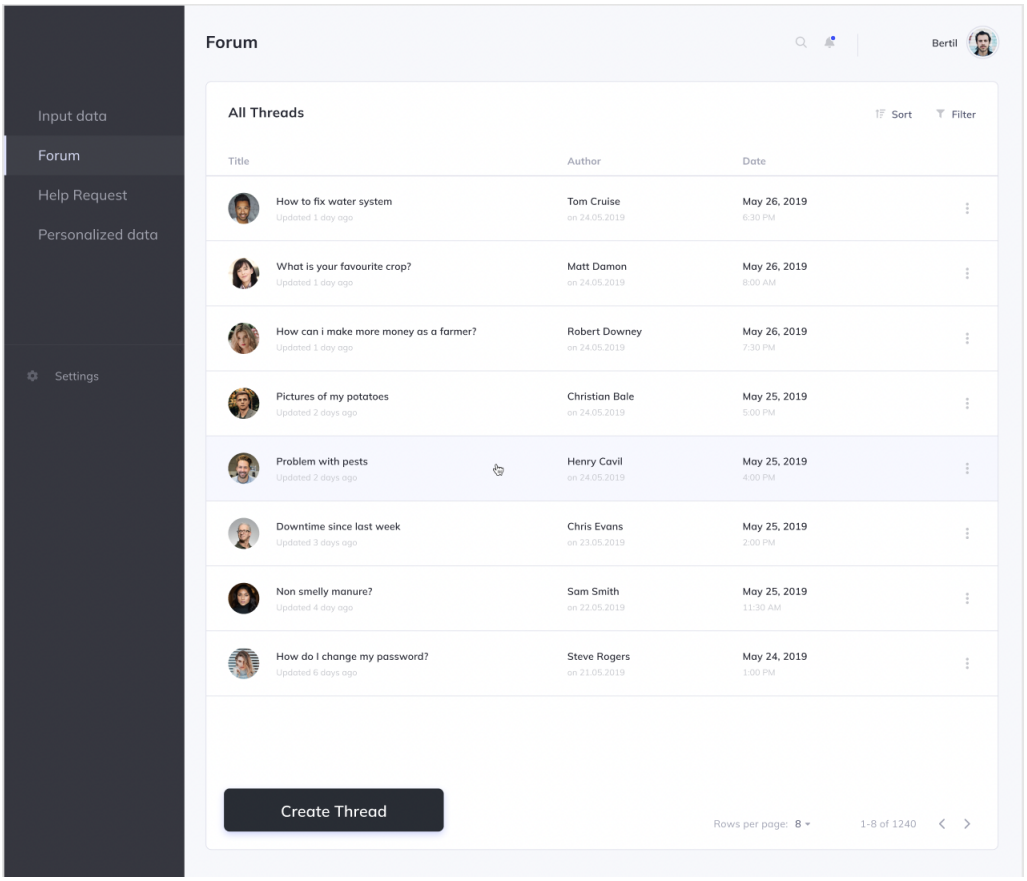

Figure 16: Forum UI for farmers

3.1.5 Create forum thread

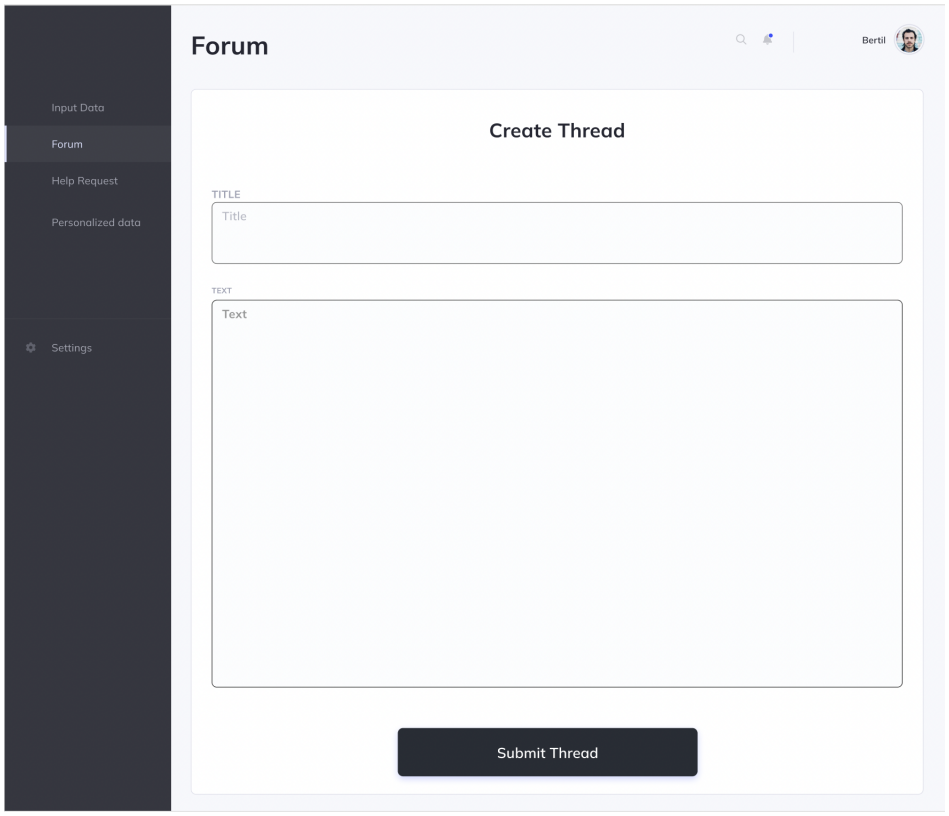

Figure 17: Form UI to create forum thread

3.1.6 Forum thread

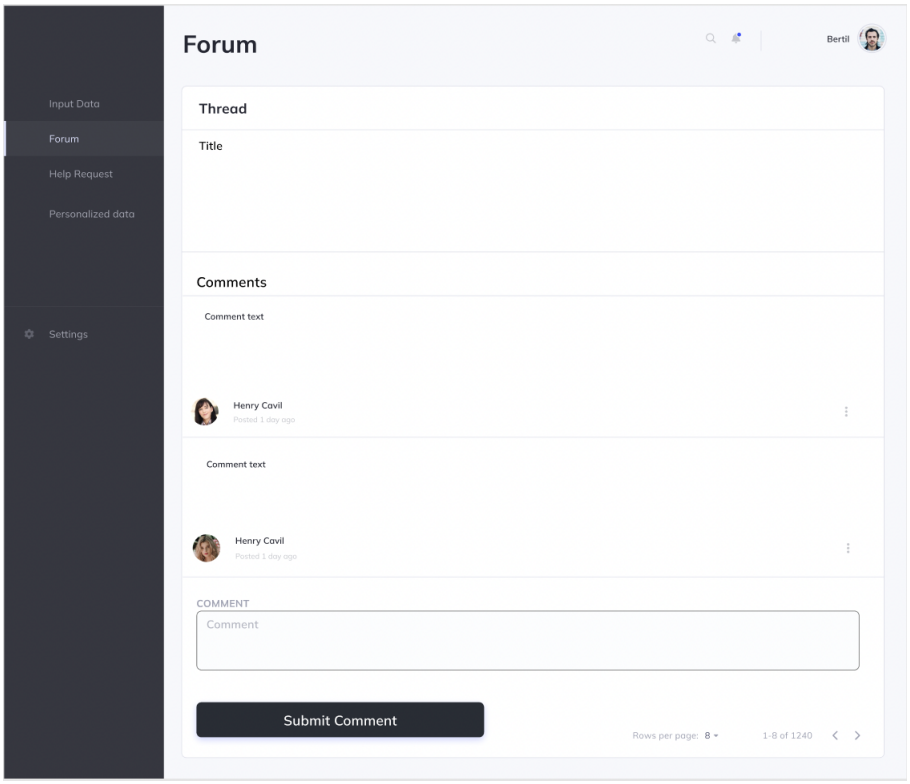

Figure 18: UI of thread with comments

3.1.7 Help Request

Like in previous sections, the user interfaces for the help request functionality is omitted. This is because of the heavy similarities to Forum and that those user interfaces should be sufficient.

3.1.8 View personalized data

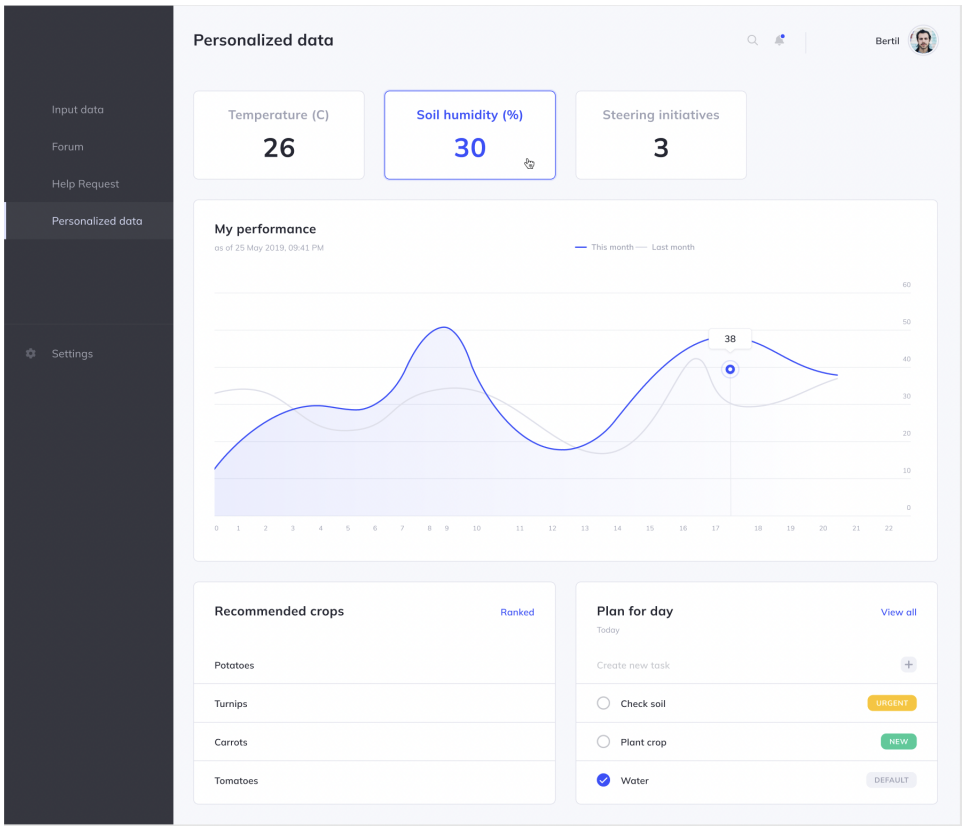

Figure 19: UI for page showing personalized data

3.2 Policy Maker

The two functions available for Policy makers have quite a lot of overlap. To simplify, one interface is included per functionality and the rest are described.

3.2.1 View Performance

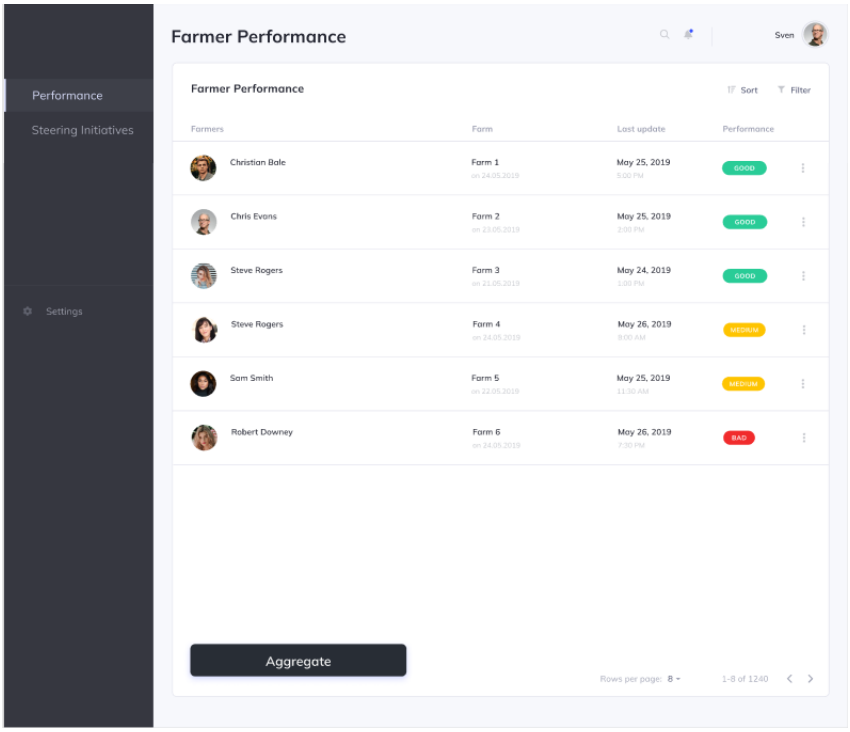

Figure 20: UI for farmer performance

Here the list of farmers are shown, ordered by performance. It is also possible to select a farmer, and see their individual performance graph. The mockup for that can be seen in the next section (3.2.2), though the vertical line should not be included here. Also, pressing "Aggregate" will aggregate the performance of the farmers displayed and show a graph similar to the one for an individual farmer.

3.2.2 Steering initiative

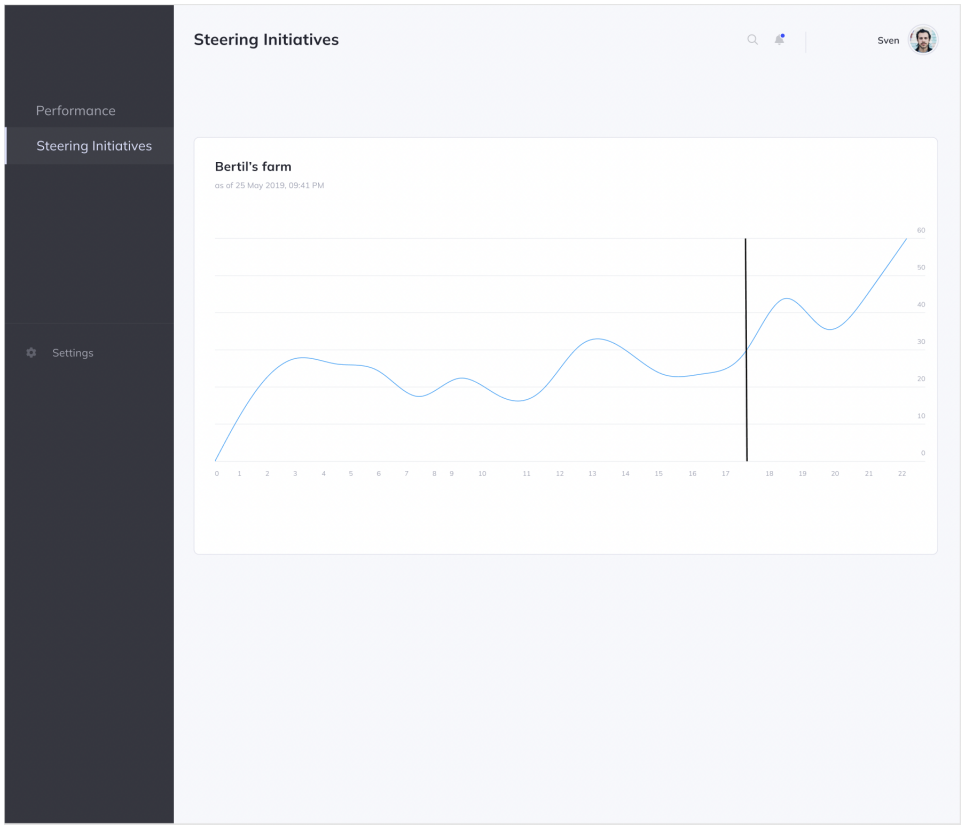

Figure 21: UI showing effect of steering initiative

Here only the graph the performance graph is shown. The vertical line identifies the time at which a steering initiative was started. For steering initiatives there are two other user interfaces that needs to be described. The initial view, when selecting *Steering Initiatives* from the left sidebar, shows a list showing all steering initiatives. That list has the same design as e.g. forum and the farmer performance list. After selecting a steering initiative from the list, you are redirected to a view that shows the farms who have adopted the steering initiative. That view is identical to the one described in the previous section (3.2.1).

4 Requirments traceability

This section contains a table explaining what components are required in order to fulfil each of the requirements specified in the RASD. To save some space,

the components have been given abbreviations, see list below.

- **CA** - ClientApplication
- **RO** - Router
- **AZS** - AuthorizationService
- **ACS** - AuthenticationService
- **FM** - Forum
- **HR** - Help Request
- **DIS** - DataInsertionService
- **SI** - SteeringInitiative
- **PR** - Performance Ranking
- **DV** - Data visualization
- **RS** - RegistrationService
- **PC** - PerformanceCalucator
- **EPH** - ExternalAPIHandler
- **M** - Model
- **DB** - DBMS

R	CA	RO	AZS	ACS	FM	HR	DIS	SI	PR	DV	RS	PC	EPH	M	DB
R1	✓	✓									✓		✓	✓	✓
R2	✓	✓	✓	✓			✓							✓	✓
R3	✓	✓	✓	✓			✓							✓	✓
R4	✓	✓	✓	✓		✓								✓	✓
R5	✓	✓	✓	✓		✓								✓	✓
R6	✓	✓	✓	✓	✓									✓	✓
R7	✓	✓	✓	✓	✓									✓	✓
R8	✓	✓	✓	✓						✓				✓	✓
R9	✓	✓	✓	✓					✓			✓	✓	✓	✓
R10	✓	✓	✓	✓				✓						✓	✓
R11	✓	✓	✓	✓				✓				✓	✓	✓	✓
R12	✓	✓	✓	✓				✓				✓	✓	✓	✓
R13	✓	✓									✓		✓	✓	✓
R14	✓	✓	✓	✓		✓								✓	✓
R15	✓	✓	✓	✓		✓								✓	✓
R16	✓	✓	✓	✓	✓									✓	✓
R17	✓	✓	✓	✓	✓									✓	✓
R18	✓	✓	✓	✓					✓			✓		✓	✓
R19	✓	✓	✓	✓					✓			✓		✓	✓
R20	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓			✓	
R21	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓			✓	
R22														✓	✓
R23	✓	✓	✓											✓	✓
R24	✓	✓	✓											✓	✓

5 Implementation, integration and test plan

In this section the plans to follow for the implementation and testing of the system are described.

5.1 Implementation

The implementation of this system would follow a planned structure, dividing the components into various groups in order to maintain a clear development path. This division is strongly based on the component diagram, see Figure 2. The system is to be developed according to the five subparts listed below, with the aim of following a right to left path of development (considering the component diagram). The intention with this is to have the data requirements (i.e DBMS and Model component) dictate the rest of the system structure.

1. **DBMS & Model**
   This part encapsulates creating the data model. The database tables with the specified rows will be created, in tandem with the Model component.

2. **Calculation & fetch interfaces**
   These supply the controllers with the rest of the data they need to function correctly.

3. **Controllers**
   The controllers will be developed with respect to the component diagram. All controllers (e.g. Forum, Authentication, Help Request, etc), are independent and can be developed in parallel. Their individual functionality is specified in the component interface diagram and they often resemble each other heavily.

4. **Router**
   After the controllers have been developed, the router should be implemented. There are various libraries and standard solutions for this. It should not be forgotten that the router must authorize the request before redirecting it to the correct controller.

5. **ClientApplication**
   This is the view part of the system, where all the user interactivity will be developed. The modular way in which this system is structured means that this component only has to be responsible for the graphical user interface and nothing else.

5.2 Integration & Test plan

The sequence in which we integrate components will follow the implementation order described in the previous section. The right-to-left order, as seen from the component diagram, corresponds to a bottom-up testing strategy which should work well considering the quite simple hierarchical structure of the system. With this approach drivers needs to be implemented for every component that is integration tested.

The choice of a bottom-up strategy was natural and with it incrementally testing throughout the development of the system should be the same. Before testing to integrate a component, two conditions should be fulfilled. Firstly, the component's interface should offer all functionality specified in the component interface diagram, see Figure 12. Secondly, the component should pass all unit tests.

Of course, when replacing a driver with the implementation, the integration tests needs to be checked again.

To further clarify, below the ordering in which to run the integration tests is presented with explanatory graphics. The graphics only show one of the controllers integrated, though the rest of them follow in the same way as they are on the same level in the hierarchy.

1. **DBMS**

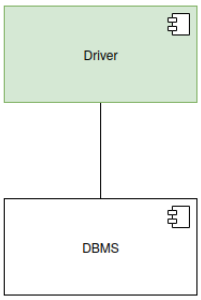

Figure 22: Integration test of DBMS

2. **Model**

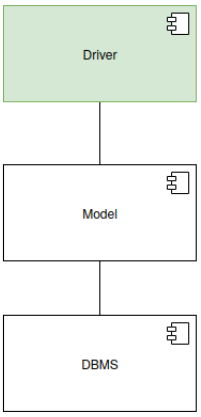

Figure 23: Secondly we integrate the model component

3. **ExternalAPI**

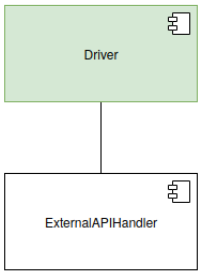

Figure 24: The integration of the external API component

4. **PerformanceCalculator**

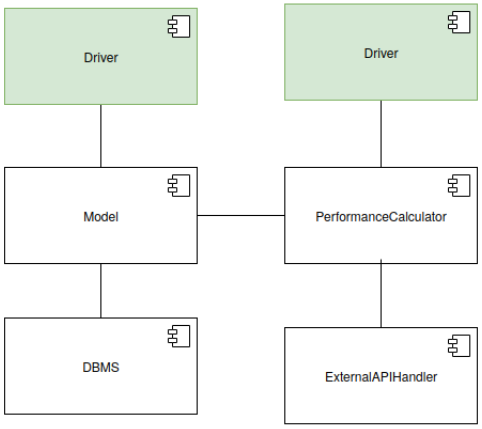

Figure 25: Integrating the performance calculator

5. **The controllers**

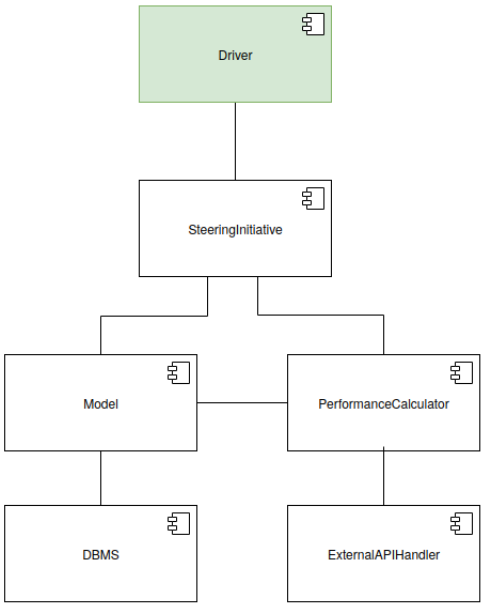

Figure 26: Integration of controllers, here only showing Steering Initiative. Other controllers would lie parallel to Steering Initative and all depend on the Model component. Some would also, like Steering Initiative, depend on the performance calculator. Finally the RegistrationService component depends, in addition to Model, on the ExternalAPIHandler.

6. **Router**

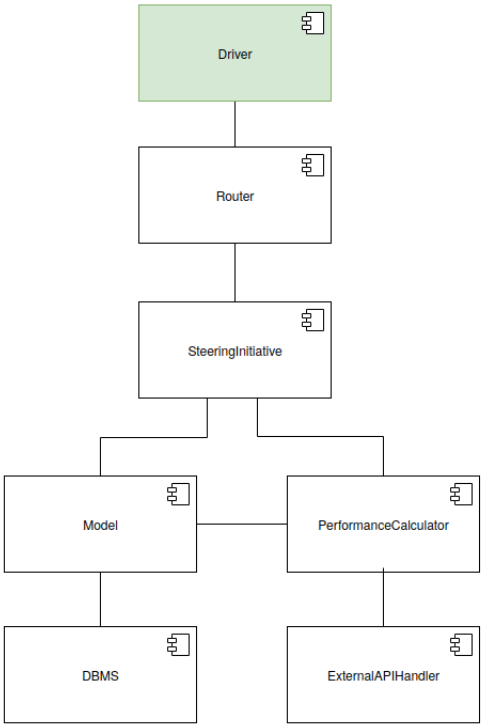

Figure 27: Integration of the Router component. Also here only the Steering Initiative controller is shown, though the others follow in the same way.

7. **ClientApp**

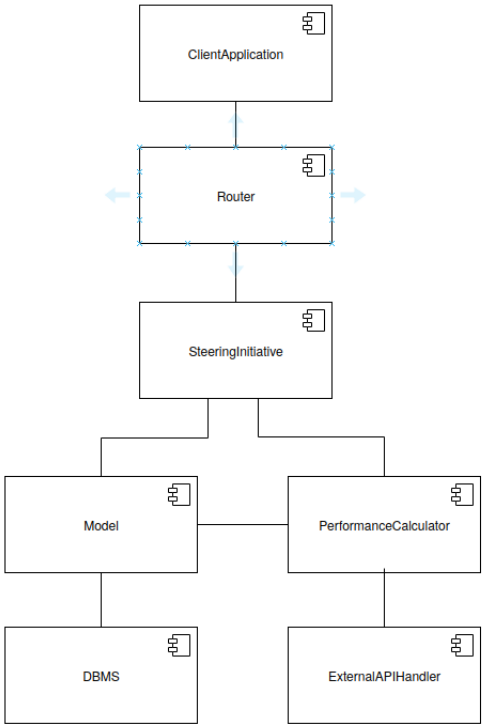

Figure 28: The final integration to complete the system, ClientApplication.

As soon as the whole system has been integrated, system testing can begin in order to assess the fulfilment of functional and non-functional requirements. If possible, some form of usability testing should be conducted after the system tests pass. This, among other things, to evaluate the design on its user friendliness.

6 Effort spent

6.1

Task	**Time spent**
Introduction	2h
Architectural design	25h
User Interface Design	1h
Requirements Traceability	1.5h
Implementation, Integration and Test Plan	4h
Reasoning	5h
Total	38.5h

6.2

Task	Time spent
Introduction	0h
Architectural design	24h
User Interface Design	3h
Requirements Traceability	2.5h
Implementation, Integration and Test Plan	4h
Reasoning	5h
Total	38.5h

7 References