DREAM

Data-dRiven PrEdictive FArMing in Telangana

Design Document - DD

POLITECNICO
MILANO 1863

Christian Grasso - Filippo Lazzati - Chiara Magri
Year: 2021/2022

Contents

List of Figures

List of Tables

1 Introduction

A DD[1], according to the definition provided by the **IEEE Std 1016™-2009** standard, is *a representation of a software design that is to be used for recording design information, addressing various design concerns, and communicating that information to the design's stakeholders.* It should be noticed that, in this document, instead of using the abbreviation SDD, which stands for "Software Design Description", is used DD. In other words, a DD is a document that is used by *acquirers, project managers, quality assurance staff, configuration managers, software designers, programmers, testers, and maintainers* to retrieve the specific design information needed by the specific stakeholder.

1.1 Purpose

`DREAM`, the system to-be, aims to help policy makers formulating policies in the field of agriculture. Moreover, `DREAM` aims to help farmers by putting them in contact with each other so that they can exchange advice and aids. Finally, `DREAM` also schedules the daily work of agronomists. These are the main goals of `DREAM` which shall be data-driven, namely it shall intensively exploit data to perform the tasks it is asked to do. The requirements that allow the system to satisfy the goals as well as a detailed list of all the goals are present respectively in *RASD - 3.2 Functional requirements* and in *RASD - 1.1 Purpose*. In this document the focus is on the design, that is on the description of the components and the interaction among them and with external systems through interfaces which compose the system and allow the satisfaction of the requirements and, at the end, of the goals.

1.2 Scope

The system to-be has three major stakeholders: the policy makers, the agronomists and the farmers. The needs that `DREAM` aims to satisfy differ among them, and the main goals of the system have been identified in section1.1. For a detailed description and explanation of the context in which `DREAM` operates the reader should refer to the *RASD - 1.2 Scope* section. In here, we briefly tell about the main design concerns of such stakeholders.
First of all, all of them have goals that, in order to be satisfied, require the storage of a large volume of data (after all, `DREAM` is a data-driven system), and therefore scalable technologies suitable to manage the required amount of information have to be used. Furthermore, the stakeholders are not assumed to be expert in computer science, thus a graphical interface is needed; more specifically, the idea is to implement a web application to facilitate the access to the system to all the users and from any kind of device (computer, smartphone, ...).

[1] Design Document. See section 1.3

1.3 Definitions, Acronyms, Abbreviations

This section provides a list of the main abbreviations, acronyms and definitions adopted in the document:

- Design Document (DD): "An SDD is a representation of a software design to be used for communicating design information to its stakeholders. The requirements for the design languages (notations and other representational schemes) to be used for conformant SDDs are specified.
- RASD = document that aims to present all the requirements of the system to be developed, explaining the domain in which it has to operate.
- UI = user interface
- farmer = citizen of Telangana who works on a piece of land
- agronomist = expert of agriculture in Telangana
- policy-maker = devises policies to regulate the agricultural production of Telangana
- best/worst -performing farmer = farmer chosen by a policy-maker (or an agronomist) taking into account its performances This standard is applicable to automated databases and design description languages but can be used for paper documents and other means of descriptions";
- initiative = either a visit or an help request or a collaboration between a farmer and an agronomist to improve the performances of the harvest.
- TSDPS = Telangana State Development Planning Society
- ITD = Implementation and Testing Document
- IT = Implementation and Testing
- OS = Operating System

1.4 Revision history

revision	*changes*
1.0	initial version
1.1	revision after IT delivery

1.5 Reference Documents

- *IEEE Std 1016™-2009*, the standard for information technology, systems design, Software Design Descriptions. It describes the "required information content and organization for software design descriptions (SDDs)";
- *ISO/IEC/IEEE 42010*, the standard for architecture description in systems and software engineering, that "addresses the creation, analysis and sustainment of architectures of systems through the use of architecture descriptions";
- course slides;
- assignment document;
- The Unified Modeling Language user guide, ISBN: 0-201-57168-4;
- January 1994, CMUCS- 94-166, see "An Introduction to Software Architecture" at `http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf`.

1.6 Document Structure

This document complies with the SDD[2] standard structure as it is defined in the *IEEE Std 1016™-2009*, sections 4 and 5. Nevertheless, the contents have been ordered in a way that best fits the specific topics of this document and to facilitate the readers in the reading of this DD[3]. The document is divided in 5 main parts:

1. the first part (to which this section belongs) provides an introduction to the system to-be, `DREAM`, helping in the reading of the following sections;
2. the second part provides the out-and-out design decisions under which `DREAM` is implemented. This part contains UML[4] diagrams as well as text descriptions of the components of the system along with their interfaces, and explanations of the selected architectural styles and design patterns and possible other design decisions;
3. the third part describes the user interface. This part is a continuation of the *RASD - 3.1.1 User Interfaces* section, which enters the details of the design choices;
4. the fourth part shows how the requirements identified in the *RASD - 3.2 Functional requirements* section have brought to the introduction of a specific component/interface in the system;

[2] Software Design Descriptions
[3] Design Document.
[4] Unified Modeling Language.

5. the fifth and last part is about a plan to follow for the implementation of the system, namely from which subcomponent to start, which subcomponents can be implemented in parallel, and about a plan for the integration of such subcomponents and also a plan for testing the integration.

It should be remarked that the structure of this document does not follow a logic or temporal order, but whoever is interested in the reading can jump from a section to another, because the purpose of it is to be a reference document.

2 Architectural design

2.1 Overview

In this section, an high-level overview of the design components is provided. The architecture chosen is a three-tier one, in which there is a clear-cut separation between the three tiers. The reasons behind this choice comprise mainly the scalability, the accessibility and the security of this solution; they are explained more in detail in section 2.6. An high-level view of the entire architecture of `DREAM` is illustrated in figure 1:

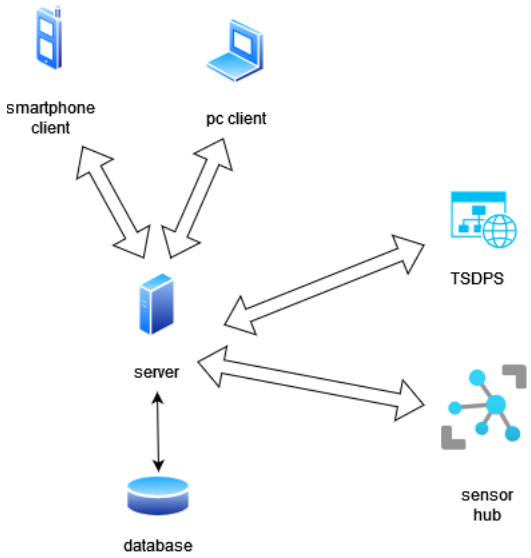


Figure 1: The main high-level components of the system.

As previously said, the system has a three-tier architecture.

Client Tier The user interface is visualized through a browser installed on the user's device (in the figure this device is exemplified through a smartphone and a pc). This is the tier closest to the user, and its role is basically related to managing the interaction with the user. It should be remarked that the logic of the system is not present here, but is contained in the application layer (web tier).

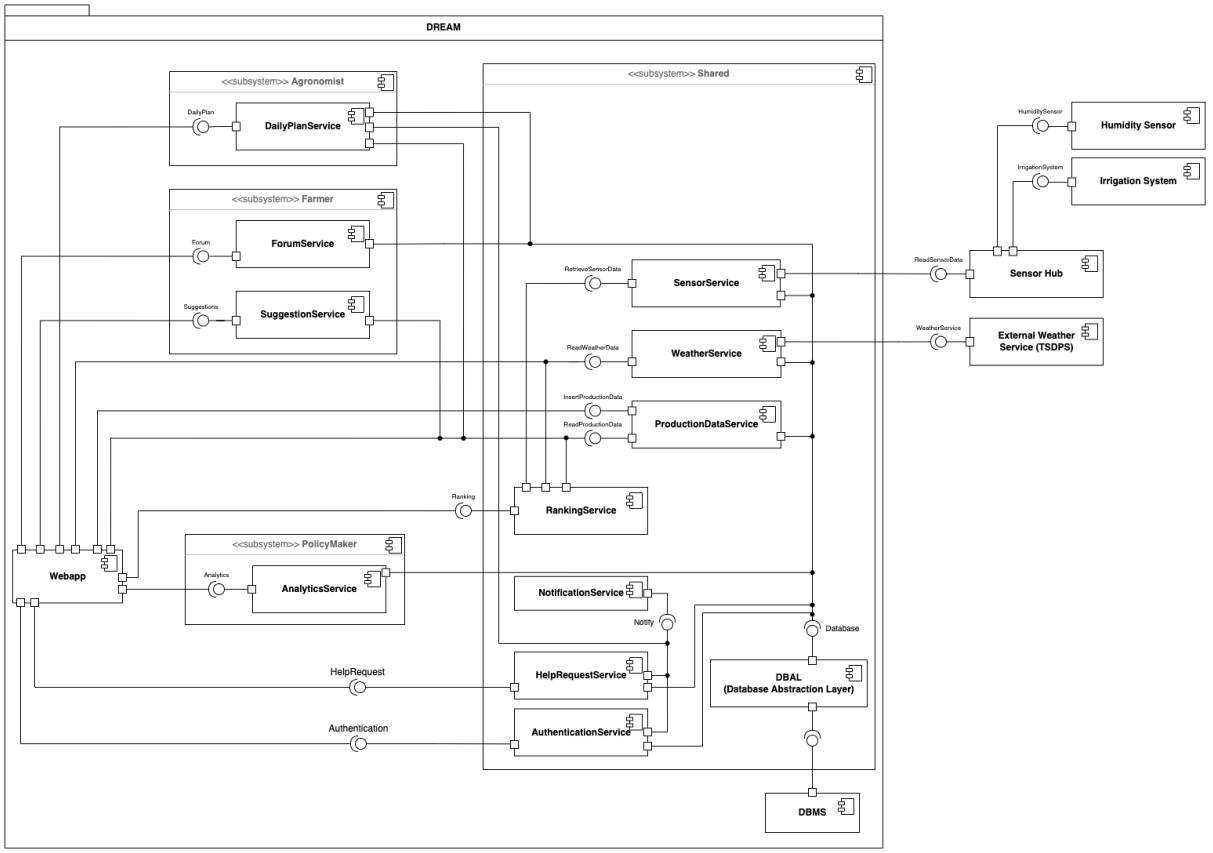
Web Tier The second tier of the application has the fundamental role of managing all the functions that are available to the user. Here lies the business logic of `DREAM`, which interacts with the other two tiers and with external services to perform all the activities of the system. This layer is exemplified by the high-level server component in figure 1. As it is explained

in the following section, such layer is made of various *services*, each of them associated to a specific function.

Data Tier The third tier is exemplified with a database and contains all the data `DREAM` has to store. As a matter of fact, `DREAM` is a data-driven application, and as such has to cope with a large amount of data to carry out its activities. All this data is stored in a database, which can be accessed by the application layer through a particular interface, as can be noticed in the following section.

External services TSDP (external weather service) and the sensor hub are two external high-level components `DREAM` has to interact with. The former provides weather reports and forecasts, while the latter allows `DREAM` to access sensors and water irrigation system data.

2.2 Component view



2.3 Deployment view

The deployment diagram of DREAM is the following:

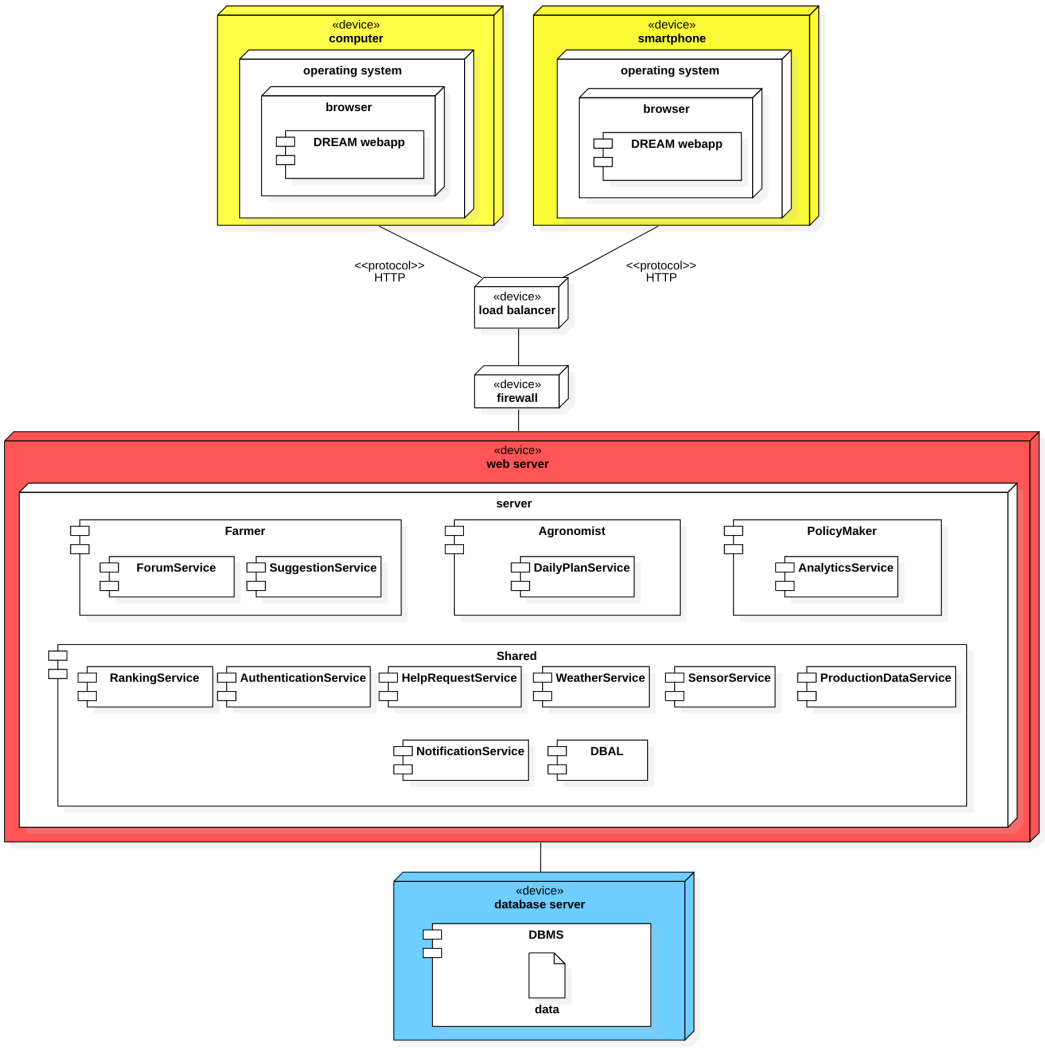


Figure 2: Deployment diagram of the system.

Some observations are required:

- as previously specified, we opt for a 3-tiers architecture, with a web server that contains the whole business logic of the system and a database server with the data;
- some colours have been used for an easier visualization of the tiers;
- for the sake of simplicity, we have decided to represent only 2 possible clients, while the system is designed to support a much larger number;
- the number of web servers is more than one in order to offer faster responses and to scale better with more users;
- according to the amount of data to manage[5], the data might be partitioned in more physical devices; for the sake of simplicity we have represented only one device since the access is hidden behind the DBAL.

2.4 Runtime view

In this section the interactions among the components (identified at section 2.2) are shown. In order to make the presentation clearer, the UML notation of sequence diagrams is used.
It should be noticed that not all the use cases and scenarios identified in *RASD - 3.2.3 Scenarios* and *RASD - 3.2.4 Use cases* are represented here. We have decided to explicitly describe only the most relevant use cases, since in many occasions the interaction among the components is more or less the same. As a matter of fact, it often occurs that the difference lies only in the function invoked on a specific component; since all the interfaces are well-explained in the rest of this document (see section **??**), we opted to avoid this repetition.

[5] see *RASD - 3.3 Performance requirements* for an estimate of the storage capacity required.

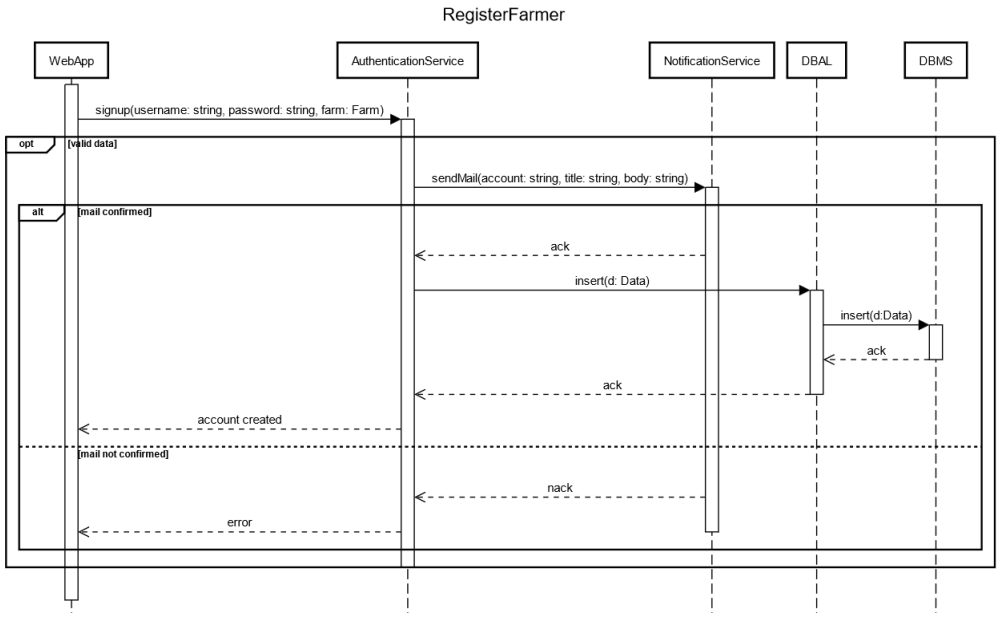


Figure 3: Sequence diagram that describes the RegisterFarmer use case.

This sequence diagram describes the process of registration of a farmer in the system. The farmer invokes the signUp method on the AuthenticationService, which checks the validity of the data (length of the password, ...) and, in case data is valid, it exploits the NotificationService to send an e-mail to the farmer with a link to verify the account. In case the account is verified, the AuthenticationService stores the new account in the database and positively notifies the farmer, otherwise an error is sent.

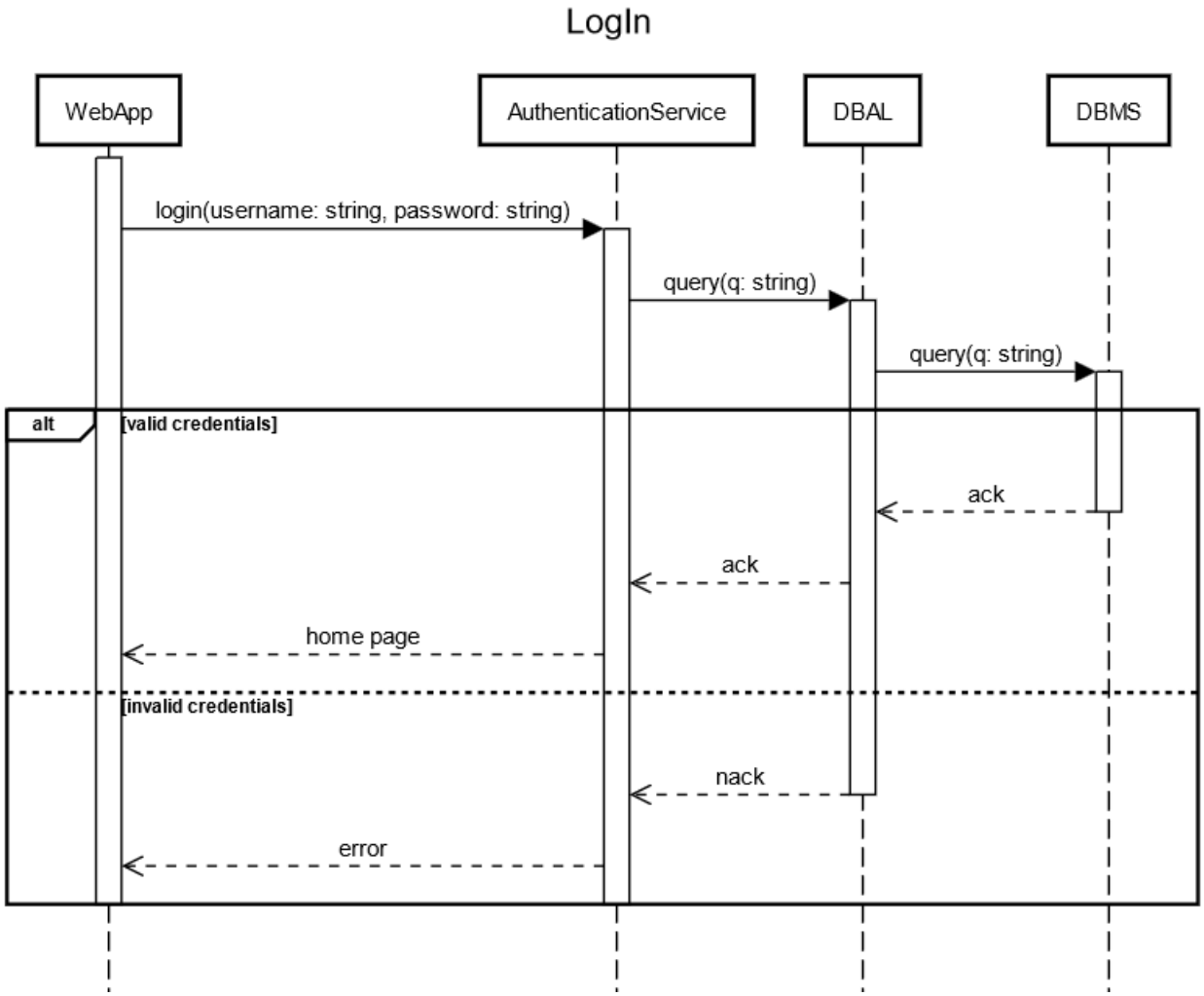


Figure 4: Sequence diagram that describes the LogIn use case.

This sequence diagram describes how a user (farmer, agronomist or policy maker) can access to the system. The procedure is rather simple: the WebApp invokes the login method on the AuthenticationService which checks that the inserted credentials do not contain code for an SQL injection attack (in such a case, an error is returned to the WebApp; this branch is not represented in the diagram) and then queries the database (through the DBAL) to check whether a profile associated to the specified credentials actually exists. In such a case, the user logs in, otherwise he receives an error.
OBS.: from now on, checks on the data inserted by the user in order to avoid

SQL injection attacks are not mentioned anymore, but they are present every time the user invokes methods of `DREAM` passing arguments (since it is a functionality offered by the software framework adopted, it is not meaningful neither to represent nor to mention it every time).

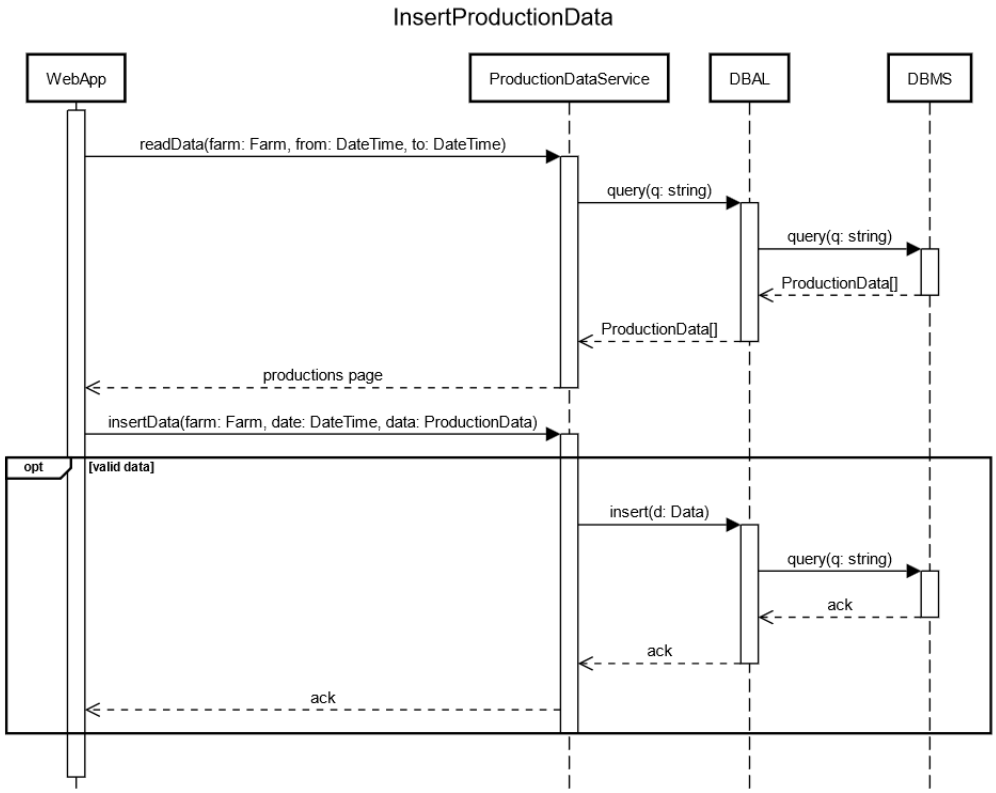


Figure 5: Sequence diagram that describes the InsertProductionData use case.

This sequence diagram describes the procedure through which a Farmer, starting from the home page, can insert production data in the system. According to the corresponding use case, the farmer automatically invokes the readData method when opening the "my productions" page. This call triggers an underlying retrieval of data from the database. Next, the Farmer calls insertData with all the details of the production to insert and, after a check about the validity, the data is inserted into the database and an ack is returned to the WebApp.

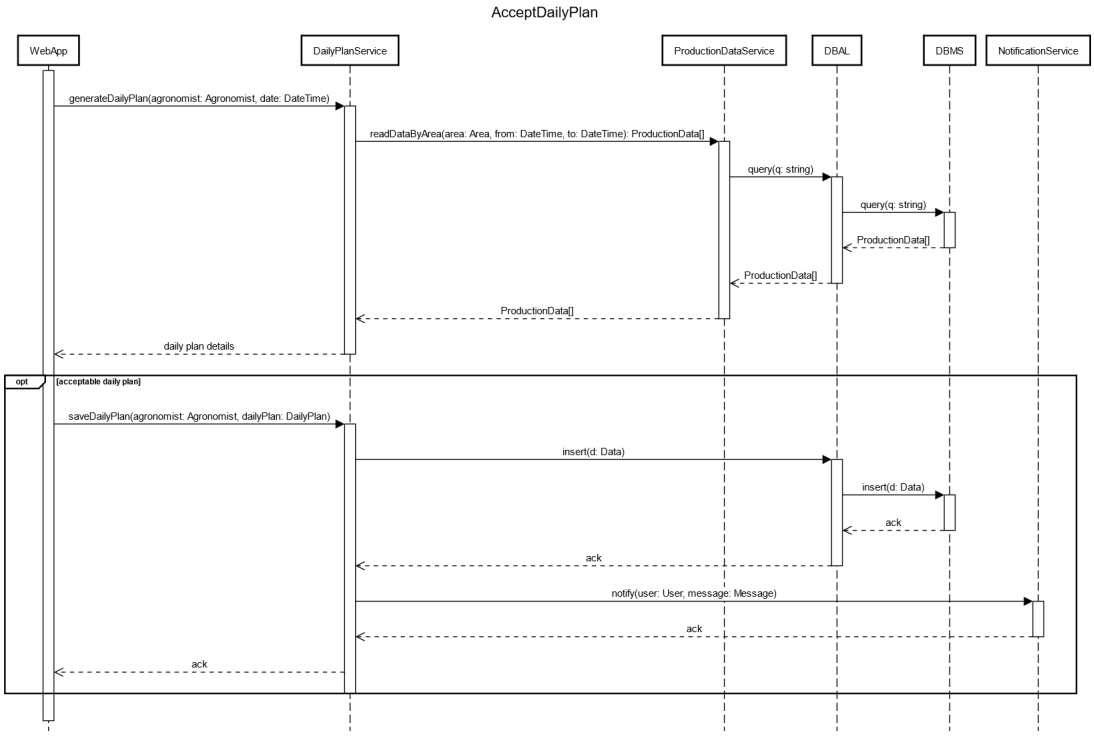


Figure 6: Sequence diagram that describes the AcceptDailyPlan use case.

This sequence diagram describes the interactions that occurs when an agronomist visualizes for the first time a new daily plan and accepts it.
First of all, the agronomist opens the daily plan dates page and then he When he selects the date of the daily plan he wants to visualize, the getDailyPlan interface of the DailyPlan component is invoked. As it is the first time that the agronomist visualizes the daily plan, getDailyPlan generates the daily plan (otherwise, it would just retrieve the already existing one from the database). In order to do so, it retrieves from the database the needed data, regarding:

- the farmers (and the corresponding farms) in the area of the agronomist
- how many visits each farmer in the area has received from agronomists in the last 365 days
- the date of the last visit from an agronomist for each farmer in the area
- the farmers in the area who requested a visit from an agronomist (when inserting a problem)

After retrieving these data, the DailyPlanService component generates the new daily plan and saves it into the database.

Notice that the data retrieved from the database in order to generate a new daily plan depend on the algorithm used by the DailyPlanService component; if this algorithm changes to implement a more sophisticated logic of daily plan generation, also the retrieved data will change. This does not change the fact that data regarding the present and past situation of farmers and visits are needed to generate a new daily plan.
When the agronomist visualizes the generated daily plan, he/she can modify it by adding, moving or removing visits. While he/she is doing this, invalid daily plan states can exist: for example, overlapping visits.
When the agronommist accepts the daily plan, the acceptDailyPlan interface of the DailyPlanService component is invoked. If the daily plan is valid, it saves the daily plan state as accepted and notifies the farmers whose farms are going to be visited by the agronomist through the NotificationService component.

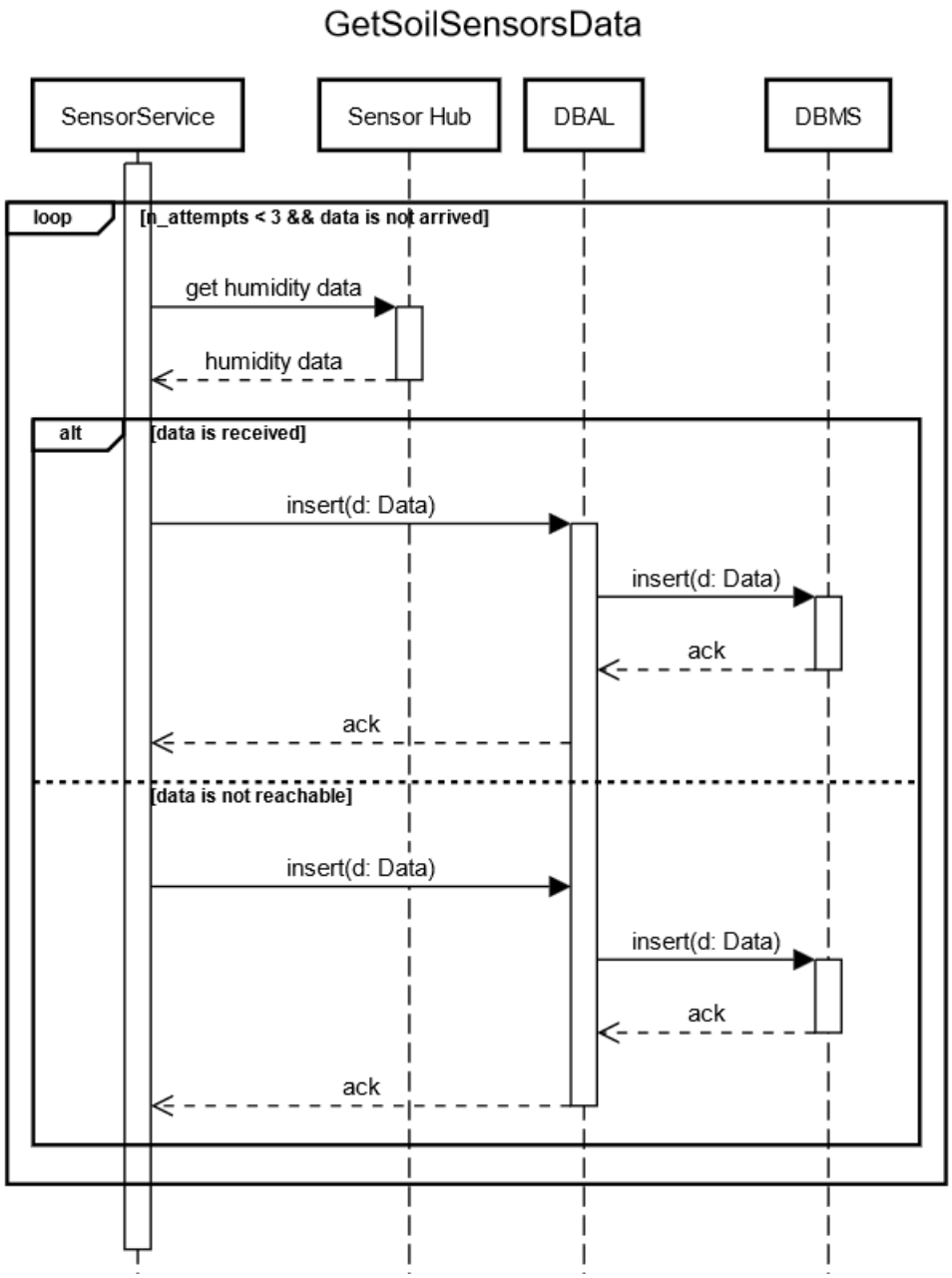


Figure 7: Sequence diagram that describes the GetSoilSensorsData use case.

This sequence diagram describes the process with which `DREAM` retrieves

data from the humidity sensors (the process for the water irrigation system is analogue). Every time a timeout expires, the SensorService gets in contact with the Sensor Hub through its interface. If the data arrives, then such data is stored in the database, otherwise the SensorService stores in the database that at that specific timestamp the sensor was not reachable.

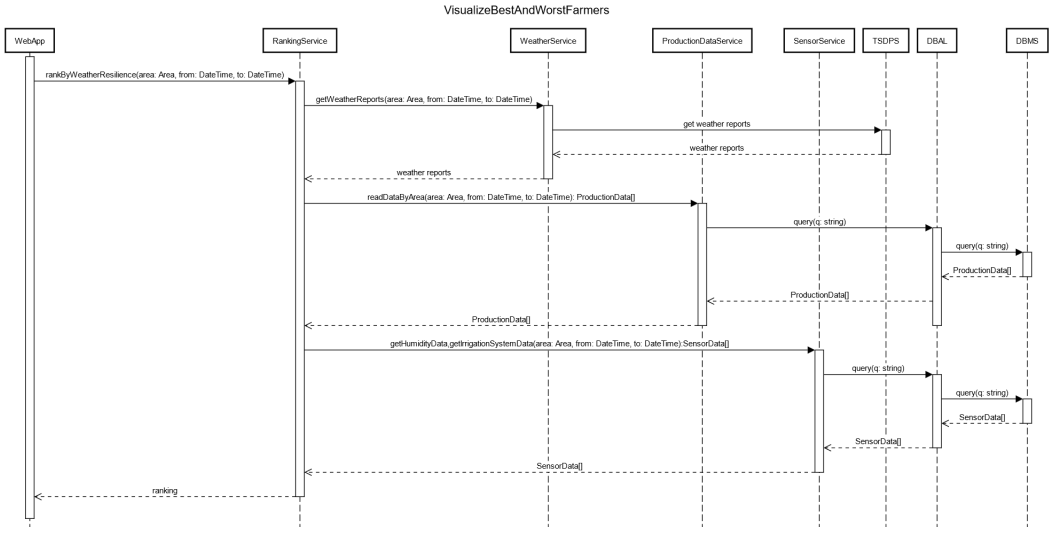


Figure 8: Sequence diagram that describes the VisualizeBestAndWorstFarmers use case.

This sequence diagram describes the process that leads to the ranking of farmers. The policy maker invokes the rankFarmers method on the RankingService, which exploits the WeatherService to retrieve the weather reports from the TSDPS; moreover, RankingService asks also the ProductionDataService and the SensorService to query the database to retrieve all the data it needs to rank the farmers (notice that getHumidityData and getIrrigationSystemData are invoked on the same arrow for graphical reasons), and finally it provides the policy maker the ranking.

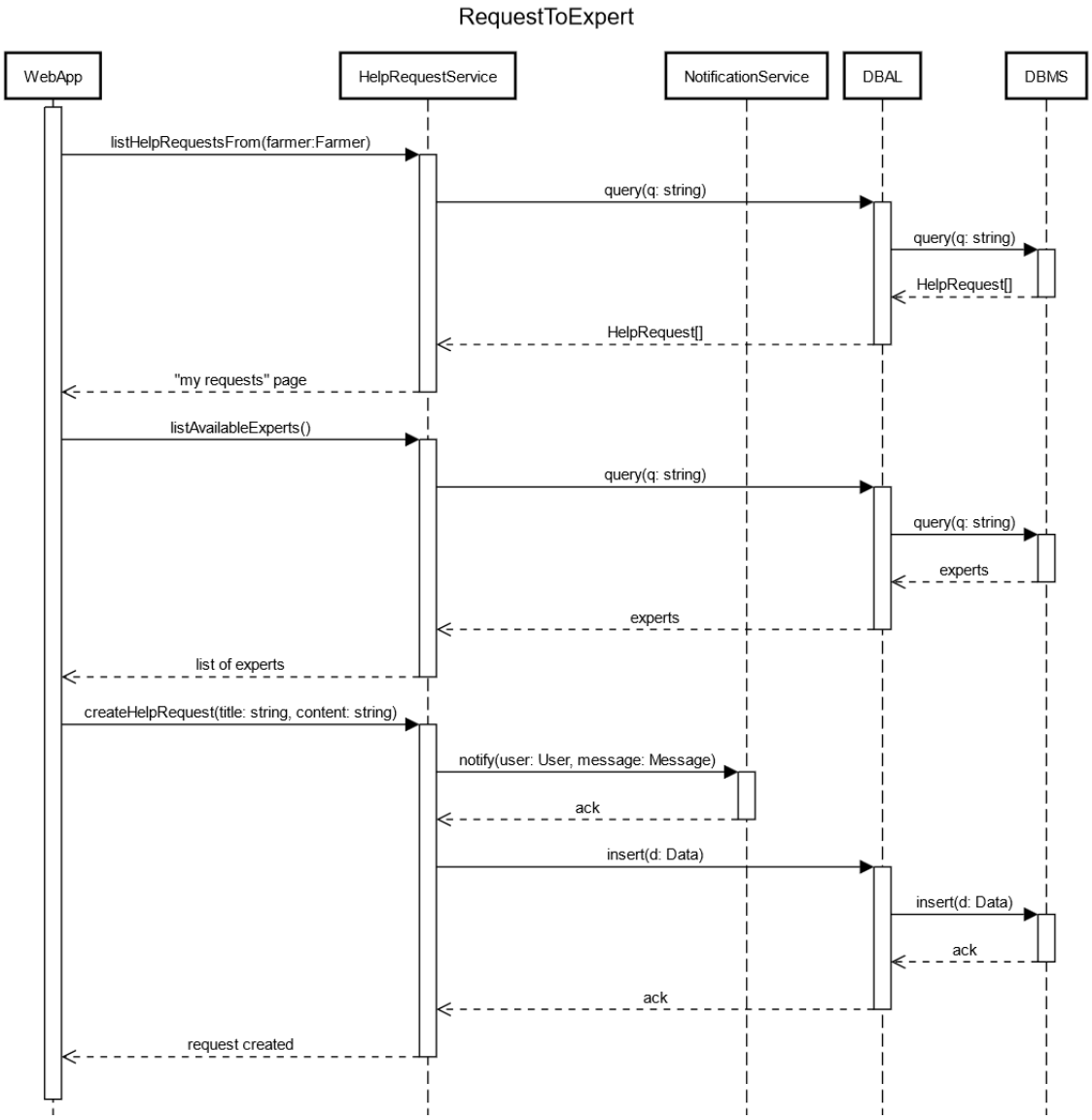


Figure 9: Sequence diagram that describes the RequestToExpert use case.

This sequence diagram represents the way a farmer can ask for help to an expert (agronomist or best-performing farmer). Starting from the home page, the farmer open the "my requests" page which implies a query to retrieve all the past requests of the farmer. Next, the farmer selects the expert from which he wants to receive some help and invokes the createHelpRequest method on the HelpRequestService. This service exploits the NotificationService to notify

the expert: the request is inserted in the database and the farmer is positively notified.

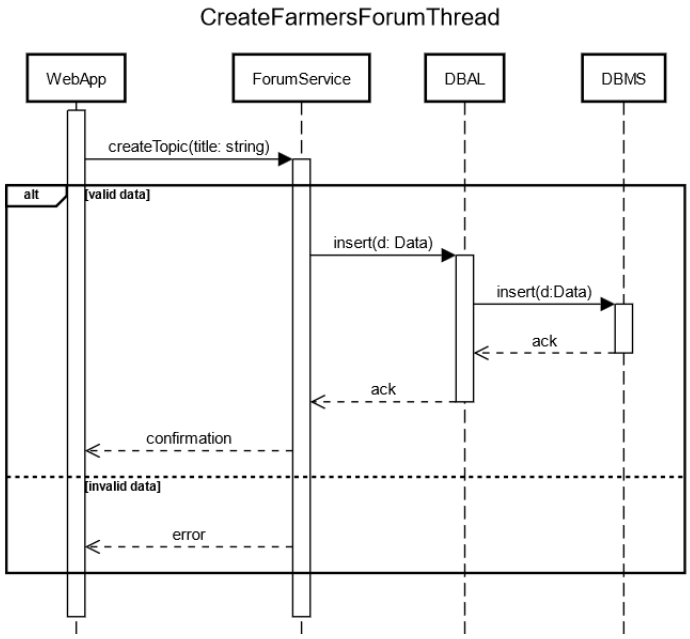


Figure 10: Sequence diagram that describes the CreateFarmersForumThread use case.

This diagram describes the process of creation of a new discussion thread by a farmer. It invokes the createTopic method on the HelpRequestService and, if the data inserted are valid, the service stores the information in the database and positively notifies the farmer, otherwise an error is shown.

2.5 Component interfaces

2.5.1 DBAL

Database Abstraction Layer - Allows to query the database in an object-oriented style abstracting over the specific database system used.

- `query(q: string)`

2.5.2 NotificationService

Service for notifications.

Notify

- `notify(user: User, message: Message): void`

  Sends a push notification to the specified user with the specified message.

- `sendMail(account: string, title: string, body: string): void`

  Sends an email to the specified account.

2.5.3 AuthenticationService

Handles user authentication and signup.

Authentication

- `login(username: string, password: string): void`

  Logs in the user with the specified username and password. If no such user exists or the password is incorrect, they will be prompted to check their credentials; otherwise, they will be redirected to the correct page based on the type of the user.

- `signup(username: string, password: string, farm: Farm): void`

  Creates a new user account for a farmer.

2.5.4 SensorService

Allows for interactions with ambient and soil sensors (humidity sensors, irrigation systems...) through a sensor hub.

RetrieveSensorData

- `getHumidityData(area: Area, from: DateTime, to: DateTime): SensorData[]`

  Returns soil humidity data for the specified area in the specified time interval. The items in the returned array contain the specific sensor identifier, the reading value, the date of the reading and the location.

- `getIrrigationSystemData(area: Area, from: DateTime, to: DateTime): SensorData[]`

  Returns irrigation system statistics for the specified area in the specified time interval. The items in the returned array contain the specific sensor identifier, the reading value, the date of the reading and the location.

2.5.5 WeatherService

Allows to read weather forecasts and reports.

ReadWeatherData

- `getWeatherReports(area: Area, from: DateTime, to: DateTime): WeatherData[]`

  Returns historic weather reports for the specified area and time interval. The items in the returned array contain the date and time of the report and the weather data.

- `getWeatherForecasts(area: Area, date: DateTime): WeatherData[]`

  Returns weather forecasts for the specified area and the specified date. The items in the returned array contain the time of the report and the weather data. The granularity of the reports varies based on how further in time the specified date is.

2.5.6 ProductionDataService

Handles farmer production data and issues.

InsertProductionData

- `insertData(farm: Farm, date: DateTime, data: ProductionData): void`

  Inserts production data for the specified farm and date.

- `insertIssue(farm: Farm, date: DateTime, issue: ProductionIssue, requestVisit: boolean):`

  Inserts a production issue for the specified farm and date. If requestVisit is true, invokes the addRequestForVisit interface of the DailyPlanService component.

ReadProductionData

- `readData(farm: Farm, from: DateTime, to: DateTime): ProductionData[]`

  Retrieves production data for the specified farm and time interval.

- `readIssues(farm: Farm, from: DateTime, to: DateTime): ProductionIssue[]`

  Reads production issues for the specified farm and time interval.

- `readDataByArea(area: Area, from: DateTime, to: DateTime): ProductionData[]`

  Reads the data by area.

2.5.7 RankingService

Allows ranking farmers based on various parameters.

Ranking

- `rankFarmers(area: Area, from: DateTime, to: DateTime, criteria:Map<Criterion, Float>):`

  Ranks farmers of a certain area, considering their production in a certain time interval, according to one or more criteria, each of them associated with a weight.

- `markFarmer(farmer:Farmer, best:boolean):void`

  Marks a farmer as best-performing if best is true, as worst-performing otherwise. If the farmer is being marked as best-performing, uses the NotificationService component to notify him/her.

- `unmarkFarmer(farmer:Farmer)`

  Unmarks a farmer from being best- or worst-performing.

2.5.8 HelpRequestService

Handling of help request / replies.

HelpRequest

- `createHelpRequest(title: string, content: string): HelpRequest`

  Creates a new help request from the current user.

- `listHelpRequestsFrom(farmer:Farmer): HelpRequest[]`

  Retrieves list of help requests sent from the farmer.

- `listHelpRequestsTo(expert:Expert): HelpRequest[]`

  Retrieves list of help requests sent to the expert.

- `listAvailableExperts(area:Area): User[]`

  Retrieves list of available experts (best farmers or agronomists in the given area).

- `answerHelpRequest(request: HelpRequest, reply: string): void`

  Adds a reply to an open help request.

- `addFeedbackForReply(helpReply: helpReply, feedback: string)`

  Adds a feedback to the given help reply.

2.5.9 ForumService

Provides an interface for farmers to interact with the forum.

Forum

- `createTopic(title: string): Topic`

  Creates a new empty topic in the forum.

- `addMessage(topic: Topic, comment: string): Message`

  Adds a new message to an open topic.

- `closeTopic(topic: Topic)`

  Closes an open topic to prevent any further comments.

2.5.10 DailyPlanService

Service for agronomist daily plans.

DailyPlan

- `getDailyPlan(agronomist: Agronomist, date: DateTime): DailyPlan`

  If the daily plan of the given agronomist in the given date already exists, returns it; otherwise, generates a new daily plan for the given agronomist and date and returns it.

- `addDailyPlanEntry(plan: DailyPlan, farm: Farm, time: Time): FarmVisit`

  Adds a new visit to the specified daily plan at the specified time.

- `removeDailyPlanEntry(entry: FarmVisit)`

  Removes the specified farm visit from the related daily plan.

- `moveDailyPlanEntry(entry: FarmVisit, time: Time)`

  Updates the time of the specified farm visit.

- `acceptDailyPlan(agronomist: Agronomist, dailyPlan: DailyPlan): void`

  Called when an agronomist accepts a daily plan for a future date, saves the state of the daily plan as accepted and notifies the farmers interested by the visits in the daily plan through the NotificationService component.

- `confirmDailyPlan(agronomist:Agronomist, dailyPlan: DailyPlan): void`

  Called when an agronomist confirms a daily plan after the end of the working day, saves the state of the daily plan as confirmed.

- `addRequestForVisit(farmer:Farmer): void`

  Called when an agronomist confirms a daily plan after the end of the working day, saves the state of the daily plan as confirmed.

2.5.11 SuggestionService

Service for automated farmer suggestions.

Suggestions

- `getSuggestions(farmer: Farmer): Suggestion[]`

  Retrieves a list of suggestions for the specified farmer.

2.5.12 AnalyticsService

Allows retrieving global analytics for initiatives.

Analytics

- `getInitiatives(): Initiative[]`

  Retrieves the list of currently open initiatives.

- `analyseInitiative(initiative:Initiative, timeInterval:TimeInterval):InitiativeResult`

  Returns a series of indicators comparing the activity of the farmers involved in the initiative, before and after the beginning of the collaboration. TimeInterval specifies the interval of the indicators.

2.6 Selected architectural styles and patterns

According to the definition provided by David Garlan and Mary Shaw[6], *an architectural style determines the vocabulary of components and connectors that can be used in instances of that style, together with a set of constraints on how they can be combined.*

Architectural style Given that the tasks that `DREAM` has to achieve can be perfectly reached by implementing the system as a web application, we have decided to opt for it. As a matter of fact, every time a user (farmer, agronomist or policy maker), namely a client, has to access the system, it sends a request to `DREAM`, or better, to the server of `DREAM`, which processes it and answers with a response. Moreover, our choice features 3 tiers with a thin client.
The main advantages of this choice related to the system we have to design are:

- **scalability**: the number of resources can be easily increased when needed; thanks also to the decision of adopting a 3-tiers architecture, the data is apart from the business logic, therefore the storage capacity can be increased without affecting the anything else

- **accessibility**: the access to the system can be done with every possible device with a browser installed and from every location as long as an Internet connection is available. Despite this characteristic might not seem as important as the others identified in the *RASD*, it is relevant because it provides much flexibility to the users: given their credentials, they can access from everywhere;

[6] January 1994, CMUCS- 94-166, see "An Introduction to Software Architecture" at `http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf`.

- **security**: users can access only through their credentials; this is a method for access controls and guarantees that only authorized users are granted access.

2.7 Other design decisions

Relational database We have decided to opt for a relational database because they perfectly match the use case of `DREAM`. As a matter of fact, according to *RASD - 3.3 Performance requirements*, there is not a huge number of relations between the entities of the data, thus a graph database would be counterproductive, because of its characteristic of being schema-less; the system has not strict time requirements, therefore adopting a key-value database would be uselessly expensive; the amount of data to manage is big, but not in the order required for matching the use case of a columnar database; finally, since data has to be pre-processed by `DREAM` before reaching the users (in other words we accept data exiting the database at a low level), then we avoid a document-oriented database (also because it would make us waste a lot of space).

Database abstraction layer A DBAL is useful way to abstract from the details of the specific technology adopted for the database. It allows to reduce the amount of work required for the implementation of the components to manage the data because it provides APIs that hide the technicalities of the specific database chosen.

Server-side framework For what concerns the application layer, a software framework will be used as a foundation to build upon. In addition to providing an established and streamlined development flow, the framework will provide basic interfaces to work with the HTTP protocol and handle common tasks such as webpage rendering. Additional pre-existing components will also be used to handle other generic tasks such as email sending.

Model view controller The MVC is a software design pattern that allows to design a software application using 3 different main elements:

- the model, which provides all the methods to access the data of the application;
- the view, which allows to visualize the data in the model and deals with the interaction between the system and the user;
- the controller, which receives the commands of the users and carries them out by modifying the state of the other two components.

The adoption of the MVC brings many advantages during the implementation phase of `DREAM`. It allows to easily organize large-size web applications, to implement graphical interfaces with less effort, to easily doing planning and maintenance and many others.

Multi-page application `DREAM` will be implemented as a multi-page application; that is, every time the user changes the current page - by following a link or performing an action in the application - a request will be made to the server to provide the full content of the new page.
The rationale for this choice is DREAM being an application which does not require an high degree of interactivity: a multi-page application is sufficient to satisfy all requirements while being easier to develop and maintain compared to a client-side single-page application (SPA).

Thin client For the same reasons outlined in the above paragraph, we have opted to keep most of the business logic on the server-side (thin client architecture). The server will also handle part of the presentation logic, by serving webpages that already include information relevant to the user without the need to perform further network requests, such as API calls. While this requires an higher load on the server, the use of caching techniques can keep the performance impact to a minimum without requiring extensive resources. A minimal amount of client-side logic will be required to handle some user interface related tasks and to perform early validation on user-generated content, in order to reduce unnecessary server load.

2.8 Algorithms

In this subsection are illustrated the main algorithms to be implemented:

Suggestions algorithm The *Suggestions* algorithm is in charge of providing suggestions to the farmer about what fertilizer to use for a certain kind of crop or what kind of crop to plant given a certain available area.
The idea behind it is to show to an expert a certain number of past weather reports about a certain location and to ask him what kind of fertilizer he would use for fertilizing a certain crop. After having gathered this (training) set of data, a feed-forward neural network is trained to learn to provide these *Suggestions*. With regards to the kind of crop to plan, instead of asking what kind of fertilizer to use for a certain crop we can ask to the expert what kind of crop for a certain area size.
The pseudo-code of the algorithm is:

Algorithm 1 Suggestions algorithm

$i \leftarrow N$ ▷ N is the size of the dataset.
$dataset \leftarrow empty$
while i is not 0 **do**
    $reports \leftarrow pastReports$
    $crop \leftarrow cropToFertilize$
    $fertilizer \leftarrow expertOpinion$
    $sample \leftarrow reports \cup crop \cup fertilizer$
    $dataset \leftarrow dataset \cup sample$
    $i \leftarrow i - 1$
end while
$NN \leftarrow randomInitialization$
$parameters \leftarrow training(NN, dataset)$

Since the problem is a multi-classification problem (there are multiple output labels), it might be useful to apply *Softmax* to the output layer.

3 User Interface design

Since the front-end of the DREAM system is a web application, the content of its user interface consists in HTML pages displayed through a browser. Some wireframes of the UI[7] can be found in *RASD - 3.1.1 User interface*. In this section, IFML[8] diagrams are used to show the pages composing the DREAM website, the components contained in these pages, the events that the user can fire through his/her interaction with the UI and how these events are handled by the DREAM system. As explained in the previous section, the DREAM website will be implemented as a multi-page application, whose pages will be generated by server-side templating. As a consequence of this design choice, most of the UI events cause an HTTP request to the web server, which replies with an HTTP response containing the new HTML that the browser displays. As the MVC pattern is adopted - see previous section -, HTTP requests (that can be interpreted as messages produced by the UI events), are handled by the Controller components (hosted in the DREAM server): in the following diagrams, actions of these components are displayed in grey hexagons with abstract names - these actions should be a guideline for the implementation of Controller components, but their mapping with software implementation aspects such as classes or methods is not discussed here. The following diagrams are referred to the PC version of the website; adjustments for the mobile version of the website - due to visualization issues - are specified in the textual descriptions of the diagrams.

[7] User Interface

[8] Interaction Flow Modeling Language, see https://www.ifml.org/

3.1 General Overview

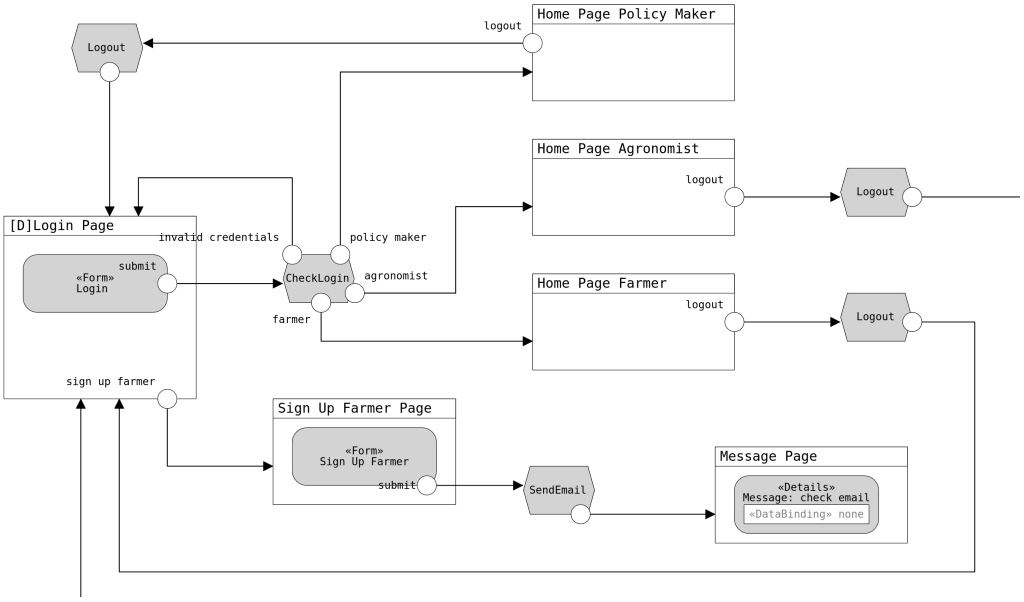


Figure 11: IFML diagram with login and home pages.

The DREAM website has a public landing page - Login page - which contains a form for logging in the system and a button for accessing a registration page for farmers accessing the system for the first time. By submitting the form for logging in, if the credentials inserted are correct - CheckLogin action -, the user is shown the home page corresponding to his role - either farmer, agronomist or policy maker.[9] If the user clicks on the button for signing up as a farmer, he/she is shown a new page containing a form for inserting his/her personal and working data and the credentials he/she wants to use to access the DREAM system. By submitting this form, the user is sent an email containing a link for confirming the email address inserted and is shown a new page with a message asking him/her to check the email.[10] Each page of the website, except for Login Page, Sign Up Farmer Page and Message Page, contains a button for logging out, that, if selected, brings to the Login Page.

[9] See *RASD - 3.2.4 Use cases*, Use Case 3.
[10] See *RASD - 3.2.4 Use cases*, Use Case 1.

3.2 Farmer Interface

The Home Page for farmers contains a menu, from which the user can access the five sections of the interface: "My Production", "Weather Forecasts", "Suggestions", "My Requests", "Forum". If the farmer is marked as best-performing, the menu shows also the menu entry for a sixth section, "My Replies", as best-performing farmers can be asked questions by other farmers, as agronomists do. If there are any notifications for the farmer - e.g., a request he/she made has been answered and he/she has not read the answer yet -, they are displayed in the Home Page.
All pages represented in the diagram contain the main menu (not displayed in the diagram for the sake of simplicity), so that the user can freely navigate from one section to another without passing through the Home Page.

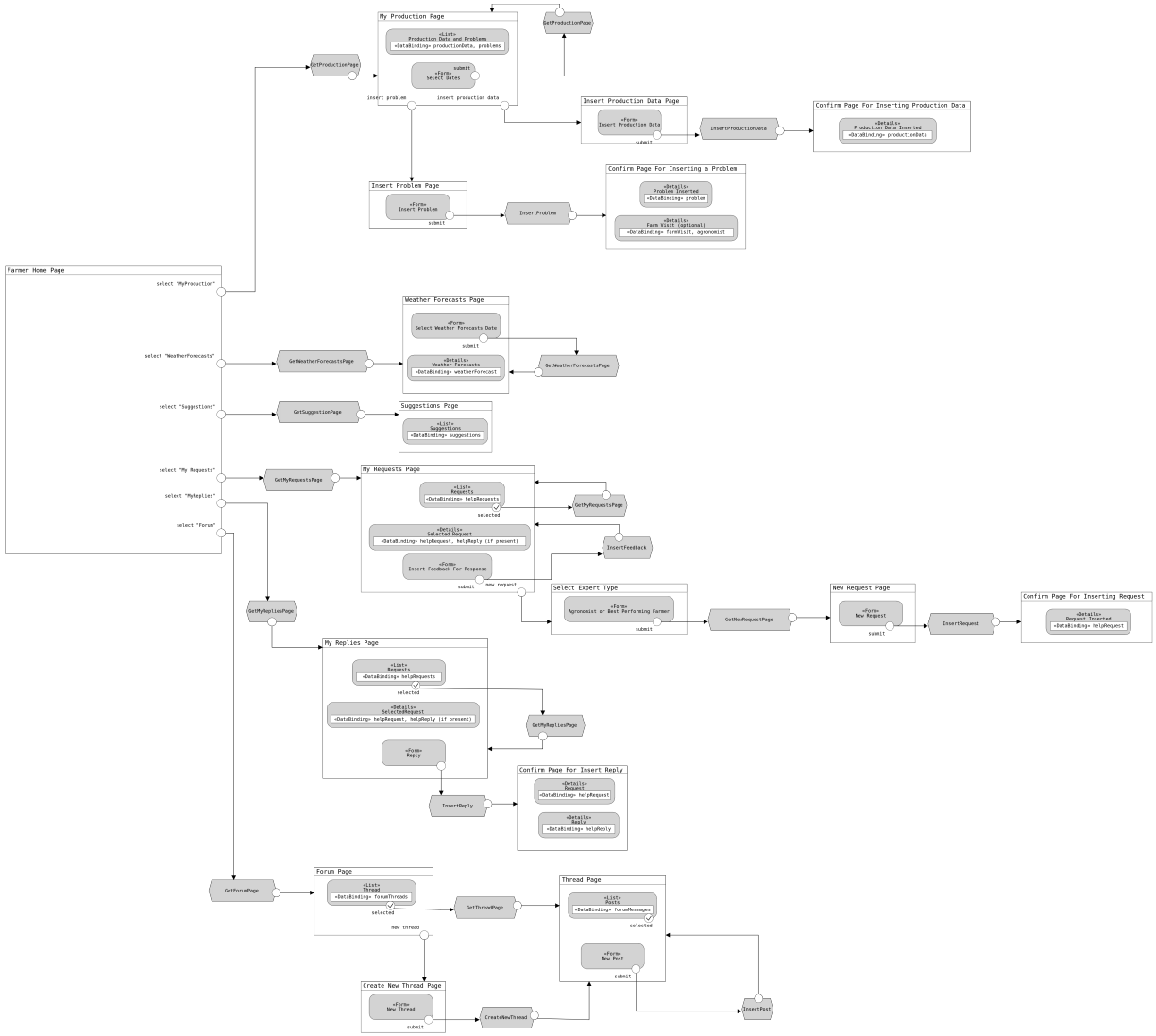


Figure 12: IFML diagram for the farmer portion of the interface.

3.2.1 "My Production" section

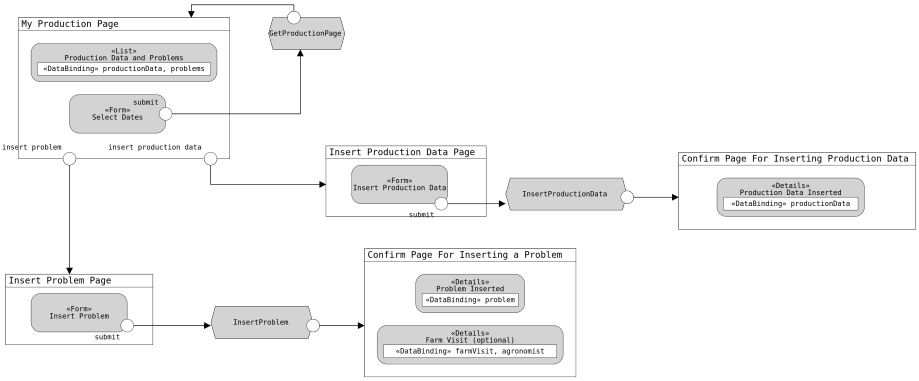


Figure 13: IFML diagram for the "My Production" section of the farmer interface.

By selecting the "My Production" entry of the main menu, the user is displayed the My Production Page, which contains a list of the production data and problems inserted by the user in descending temporal order. In the PC version of the website, only the last 20 production data and/or problems inserted are displayed, while in the mobile version of the website only the last 10 are displayed. The page also displays a "Previous" and a "Next" buttons, that allow the user to change the set of the 20 items displayed by going back or forward in time, and a form for selecting the starting and ending dates of the items to visualize (always considering the limit of 20 or 10 items that can be visualized at the same time).[11] The My Production page also contains a button "Insert Production Data", a button "Insert Problem" and a button "Back" that takes to the Home Page - back buttons are not displayed in the diagrams for the sake of simplicity-.
By selecting the "Insert Production Data" button in the My Production Page, the user is displayed the Insert Production Data page, which contains a form for inserting the production data for a work day. By submitting this form, the user is displayed a confirmation page, that shows the data inserted together with a confirmation message.[12]
By selecting the "Insert Problem" button in the My Production Page, the user is displayed the Insert Problem Page, which contains a form for inserting a de-

[11] See *RASD - 3.2.4 Use cases*, Use Case 4.
[12] See *RASD - 3.2.4 Use cases*, Use Case 5.

scription of a problem the user is facing and for optionally requesting the visit of an agronomist as soon as possible. By submitting this form, the user is displayed a confirmation page, that shows the problem inserted, a confirmation message, and, if he/she requested the agronomist's visit, the scheduled date of the visit and the agronomist who will perform the visit.[13]
All the pages Insert Production Data, Insert Problem, Confirm Page For Inserting Production Data, Confirm Page for Inserting a Problem contain a "Back" button that takes to the My Production Page.

3.2.2 "Weather Forecasts" section

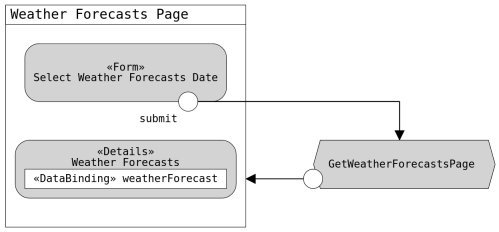


Figure 14: IFML diagram for the "Weather Forecasts" section of the farmer interface.

By selecting the "Weather Forecasts" entry of the main menu, the user is displayed the Weather Forecasts Page, which contains an input field for selecting the date of the forecasts to visualize. When the user chooses the date, the weather forecasts are shown in the same page if the website is visualized from a PC, while they are shown in a different page if the website is visualized from a mobile device.[14]
A "Back" button in the Weather Forecasts Page brings to the Farmer Home Page.

[13] See *RASD - 3.2.4 Use cases*, Use Case 6.
[14] See *RASD - 3.2.4 Use cases*, Use Case 19.

3.2.3 "Suggestions" section

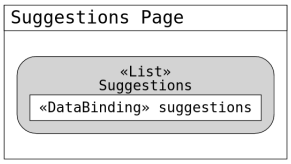


Figure 15: IFML diagram for the "Suggestions" section of the farmer interface.

By selecting the "Suggestions" entry of the main menu, the user is displayed the Suggestions Page, where some suggestions tailored for him/her are displayed.[15] A "Back" button in this page brings to the Farmer Home Page.

3.2.4 "My Requests" section

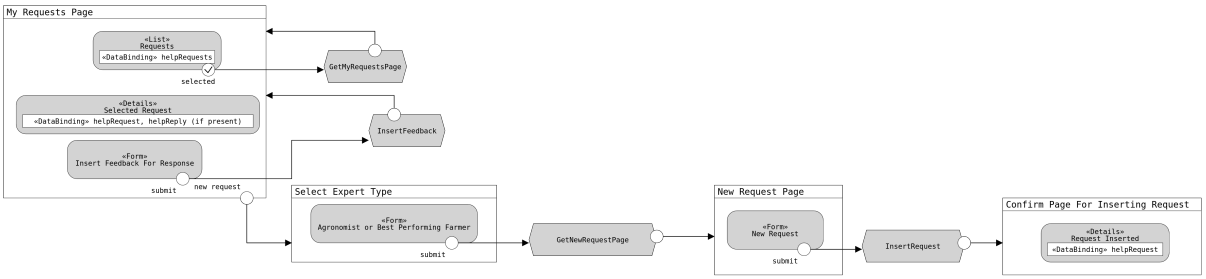


Figure 16: IFML diagram for the "My Requests" section of the farmer interface.

[15] See *RASD - 3.2.4 Use cases*, Use Case 20.

By selecting the "My Requests" entry of the main menu, the user is displayed the My Requests Page, which contains a list of the requests issued by him/her, ordered in descending temporal order. For each request, the title, the date and the receiver of the request are shown. As for the production data items and problems in the My Production Page, only the last 20 requests are shown in the PC version of the website, while only the last 10 ones are displayed in the mobile version of the website, and a "Previous" and "Next" button are present to visualize other requests. Requests which have been already replied are marked to distinguish them from unanswered ones, and requests which corresponding reply has not been read yet by the farmer are highlighted.
By selecting a request in the list, the farmer is shown in the My Requests Page also the text of the request, and if the request have been replied also the text of the response. If the farmer has already inserted a feedback for the response, it is shown, otherwise an input field for inserting such feedback is shown.[16] If the website is visited from a mobile device, the details about the request and the response are visualized in a different page from the My Production Page.
In the My Production Page, a "New Request" button is present, and if the user selects it, he/she is shown a page for choosing whether to submit the request to an agronomist or to a best-performing farmer. After this choice, the user is displayed a New Request Page, containing a form for creating a new request. After submitting this form, the user is displayed a confirmation page, that shows the details of the request inserted together with a confirmation message.[17]
In the My Requests page there is a "Back" button that brings to the Farmer Home Page; in the New Request Page and in the Confirm Page For Inserting Request there is a "Back" button that brings to the My Requests Page.

[16] See *RASD - 3.2.4 Use cases*, Use Case 24.
[17] See *RASD - 3.2.4 Use cases*, Use Case 22.

3.2.5 "My Replies" section

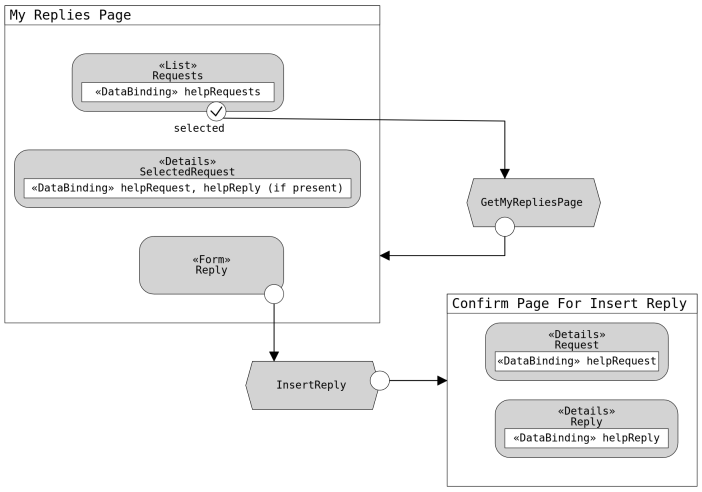


Figure 17: IFML diagram for the "My Replies" section of the farmer interface.

If the farmer is a best-performing farmer, in the main menu there is also a "My Replies" entry, that, if selected, brings to the My Replies Page. In this page, a list of the requests issued to the farmer is shown, ordered in descending temporal order. For each request, the asking farmer, the title and the date are shown. As in the My Requests section, only the last 20 requests are shown in the PC version of the website, while only the last 10 ones are displayed in the mobile version of the website, and a "Previous" and "Next" button are present to visualize other requests. Requests which have been already replied by the farmer are marked to distinguish them from unanswered ones, and requests that haven't been read yet by the farmer are highlighted.
By selecting a request in the list, the farmer is shown in the My Replies Page also the text of the request, and if he/she has already replied to the request also the text of the response, together with the related feedback, if present. If the farmer has not replied to the request, a form for inserting the reply is present. After submitting this form, the farmer is shown a confirmation page, with all details regarding the reply and the corresponding request, together with a confirmation message.[18] In the mobile version of the website, when selecting a request from the list in the My Replies Page, the request and the response (if present) are shown in another view, and also the form for inserting the request is shown in another dedicated view, accessible from the previous one.
In the My Replies Page there is a "Back" button for going back to the Farmer

[18] See *RASD - 3.2.4 Use cases*, Use Case 23.

Home Page, while in the Confirm Page For Insert Reply there is a "Back" button for going back to the My Replies Page.

3.2.6 "Forum" section

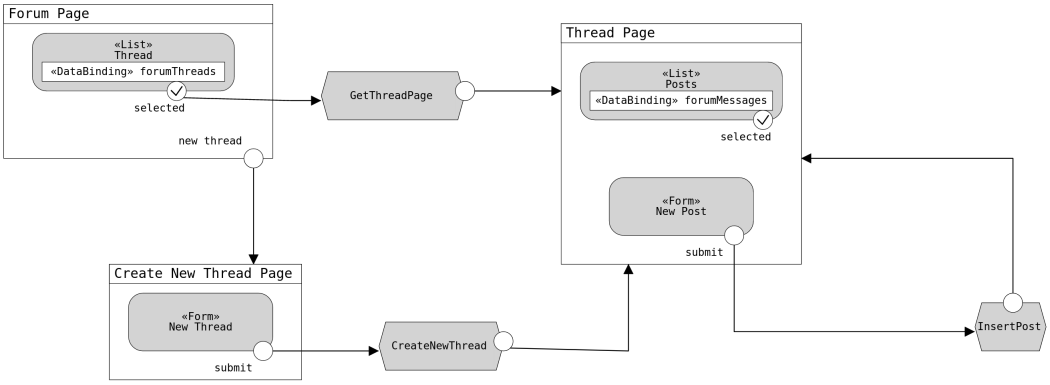


Figure 18: IFML diagram for the "Forum" section of the farmer interface.

By selecting the "Forum" entry of the main menu, the user is displayed the Forum Page, which contains a list of the threads present in the forum ,together with a "New Thread" button.
By selecting one of the threads in the Forum Page, the user accesses a Thread Page, which contains a list of the messages posted in the forum in descending temporal order, together with a form for inserting a new post. Only the last 20 messages posted are present - 10 in the mobile version of the website -, and there are a "Previous" and "Next" button for navigating between the groups of messages. When the user submits the form for posting a new message, the Thread Page is shown again, containing the new message in the list of the posted messages.[19] If the website is visited from a mobile device, the form for inserting a new message is displayed in a dedicated page.
If the user selects the "New Thread" button in the Forum Page, a page with a form for creating a new thread is shown. When the user submits this form,

[19] See *RASD - 3.2.4 Use cases*, Use Case 26.

he/she is shown the Thread Page for the new thread.[20]
In the Forum Page there is a "Back" button for going back to the Farmer Home Page, while in the Thread Page and in the Create New Thread Page there is a "Back" button for going back to the Forum Page.

3.3 Agronomist Interface

The Home Page for agronomists contains a menu, from which the user can access the four sections of the interface: "Daily Plan", "Farms", "Weather Forecasts", "My Replies".
If there are any notifications for the agronomist - e.g., a new request has been issued to him/her -, they are displayed in the Home Page.
If it's the first time that the agronomist accesses the `DREAM` system, a form for entering his personal and working data is present in the Home Page.[21]
All pages represented in the diagram contain the main menu (not displayed in the diagram for the sake of simplicity), so that the user can freely navigate from one section to another without passing through the Home Page.

[20] See *RASD - 3.2.4 Use cases*, Use Case 25.
[21] See *RASD - 3.2.4 Use cases*, Use Case 2.

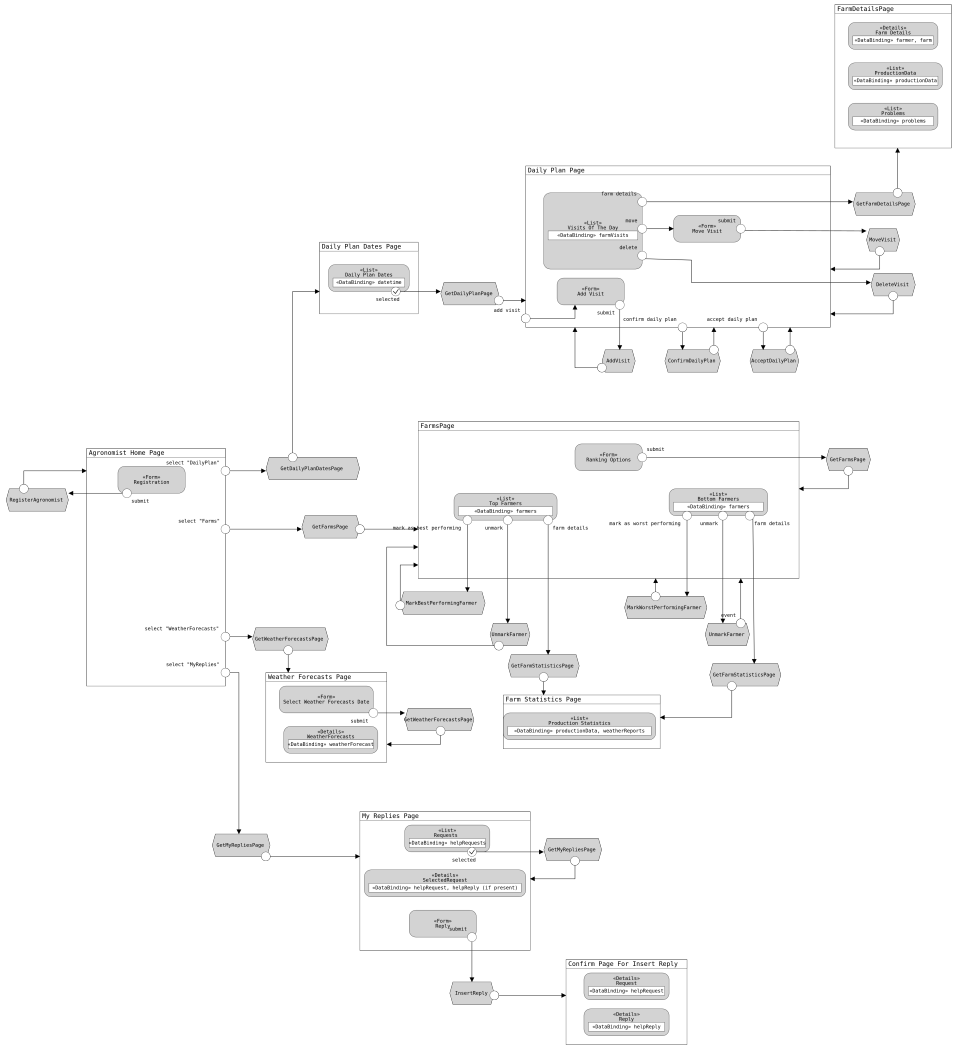


Figure 19: IFML diagram for the agronomist portion of the interface.

3.3.1 "Daily Plan" section

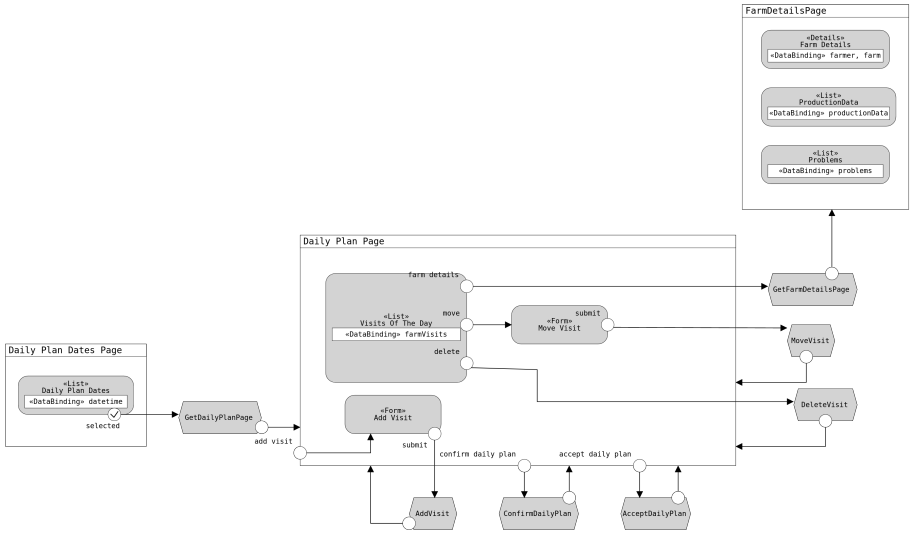


Figure 20: IFML diagram for the "Daily Plan" section of the agronomist interface.

By selecting the "Daily Plan" entry of the main menu, the user is shown the Daily Plan Dates Page, which displays a list of 7 working days, beginning from the current one.
When the user selects one of these days, he/she accesses the Daily Plan Page, which displays the list of visits scheduled for the chosen day.
If the daily plan has not been accepted yet, there are a "Farm Details", "Move" and "Delete" buttons for each visit, together with an "Add Visit" button and an "Accept Daily Plan" button. By clicking on a "Move" button related to a certain visit, the user is shown a form for selecting the new starting time for the visit, and after submitting this form, he/she is shown the daily plan with the visit moved to the desired starting time.[22] By clicking on the "Delete" button related to a certain visit, the user deletes the visit from the daily plan.[23] By clicking on the "Add Visit" button, the user is shown a form for adding a new visit to the daily plan, and when he/she submits this form, the visit is added to the daily plan.[24] If the user clicks on the "Accept Daily Plan" button, and if the `DREAM` system allows this operation, the Daily Plan Page is shown to the user without the "Move", "Delete", "Add Visit" and "Accept Daily Plan" button,

[22] See *RASD - 3.2.4 Use cases*, Use Case 9.
[23] See *RASD - 3.2.4 Use cases*, Use Case 10.
[24] See *RASD - 3.2.4 Use cases*, Use Case 11.

and with a confirmation message that the daily plan has been accepted. If the DREAM does not allow the agronomist to accept the daily plan, the Daily Plan Page is shown again with the same visits and buttons as before clicking the "Accept Daily Plan" button, together with an error message.[25]
If the daily plan has been accepted, and the last hour of the working day of a daily plan has passed, there are a "Farm Details", "Move" and "Delete" buttons for each visit, an input field for each visit for inserting a report, and an "Add Visit" button and a "Confirm Daily Plan" button. The "Move", "Delete" and "Add Visit" buttons are exactly the same as when the daily plan has not been accepted yet. When the user selects the "Confirm Daily Plan" button, the Daily Plan Page is shown to the user without the "Move", "Delete", "Add Visit" and "Accept Daily Plan" button, and with a confirmation message that the daily plan has been confirmed.[26]
If the user selects the "Farm Details" button related to a certain farm, the Farm Details Page of the selected farm is shown. It contains some details about the farm and the farmer who owns the farm - location of the farm, name and surname of the farmer - and the list of production data and problems inserted by the farmer since the last visit of the agronomist to the farm.[27]
A "Back" button in the Daily Plan Dates Page brings back to the Agronomist Home Page, a "Back" button in the Daily Plan page brings back to the Daily Plan Dates Page, and a "Back" button in the Farm Details Page brings back to the Daily Plan Page.

[25] See *RASD - 3.2.4 Use cases*, Use Case 7.
[26] See *RASD - 3.2.4 Use cases*, Use Case 12.
[27] See *RASD - 3.2.4 Use cases*, Use Case 8.

3.3.2 "Farms" section

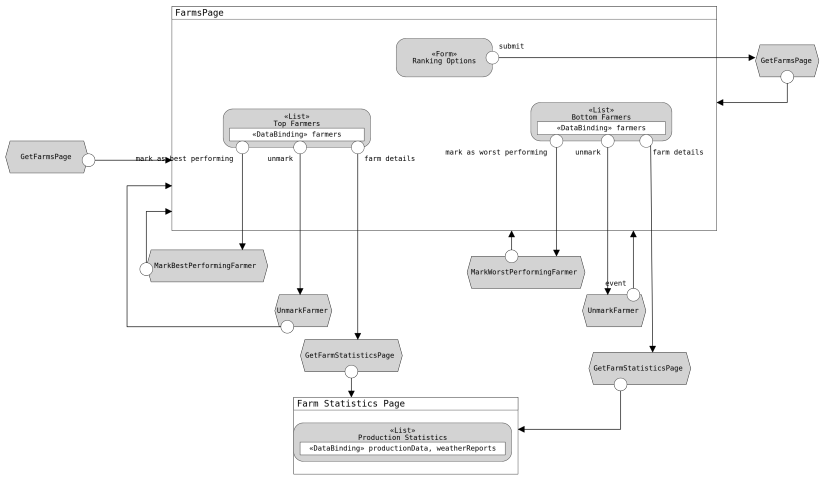


Figure 21: IFML diagram for the "Farms" section of the agronomist interface.

By selecting the "Farms" entry of the main menu, the user is shown the Farms Page, which contains a form that allows the agronomist to select some options for ranking the farmers in the area he/she is assigned to. When the user submits this form, the Farms Page displays a list of the best farmers according to the criteria chosen by the agronomist, and a list of the worst farmers according to the same criteria.[28] For each farmer there is a "Farm Details" button, which brings to a Farm Statistics Page, where monthly data about the farm production are shown. For each farmer in the first list, if not already marked as best- or worst-performing, there is a "Mark as best-performing" button[29]; for each farmer in the second list, if not already marked as best- or worst-performing, there is a "Mark as worst-performing" button. For each best- or worst-performing farmer there is an "Unmark" button[30].
A "Back" button in the Farms Page allows the user to go back to the Agronomist Home Page, while a "Back" button in the Farm Statistics Page allows the user to go back to the Farms Page.

3.3.3 "Weather Forecasts" section

This section of the website is analogous to the farmers' "Weather Forecasts" section, therefore the reader is redirected to section 3.2.2 of this document. The

[28] See *RASD - 3.2.4 Use cases*, Use Case 15.
[29] See *RASD - 3.2.4 Use cases*, Use Case 16.
[30] See *RASD - 3.2.4 Use cases*, Use Case 17.

only difference between the agronomists' and farmers' Weather Forecasts Page is that the form in the agronomists' page allows the user to select not only the date, but also the location (in the area where the agronomist works) to consider for the forecasts.

3.3.4 "My Replies" section

This section of the website is analogous to the farmers' "My Replies" section, therefore the reader is redirected to section 3.2.5 of this document.

3.4 Policy Maker Interface

The Home Page for policy makers contains a menu, from which the user can access the two sections of the interface: "Farms" and "Initiatives Analysis". All pages represented in the diagram contain the main menu (not displayed in the diagram for the sake of simplicity), so that the user can freely navigate from one section to another without passing through the Home Page.

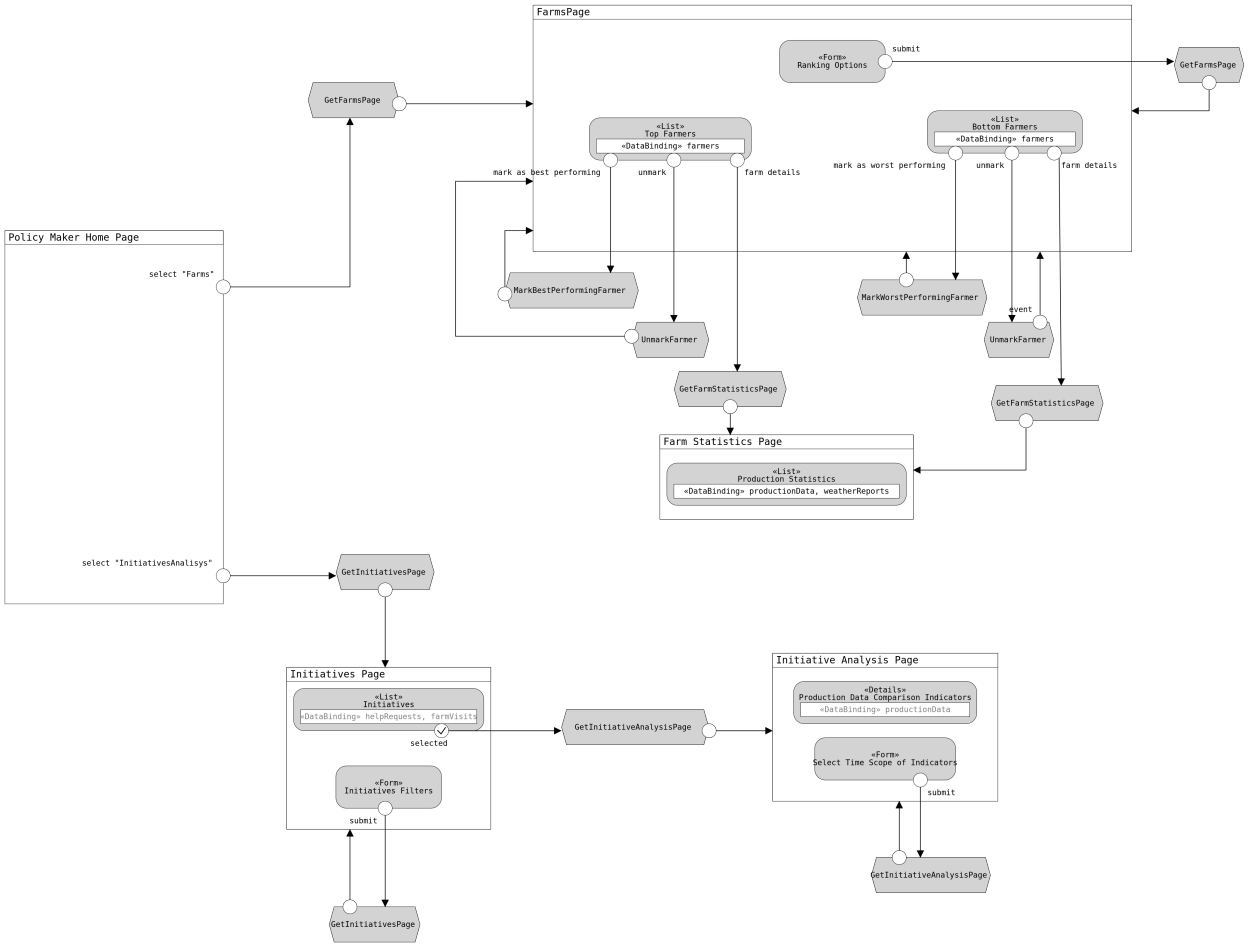


Figure 22: IFML diagram for the policy maker portion of the interface.

3.4.1 "Farms" section

This section of the website is analogous to the agronomist' "Farms" section, therefore the reader is redirected to section 3.3.2 of this document. The only difference is that policy makers can visualize farms located in all the Telangana state, while agronomists can visualize only the ones belonging to the area they are assigned to.

3.4.2 "Initiatives Analysis" section

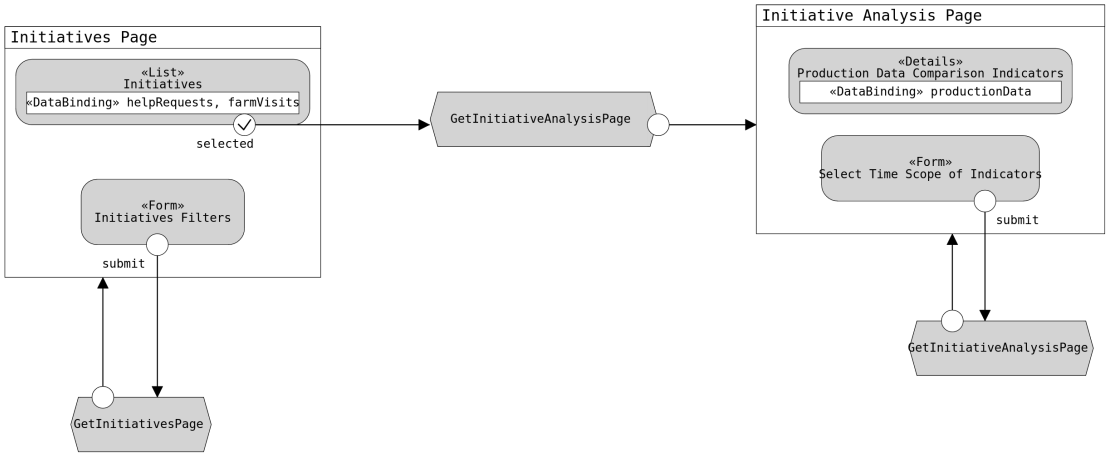


Figure 23: IFML diagram for the "Initiatives Analysis" section of the policy maker interface.

By selecting the "Initiatives Analysis" entry in the main menu, the user is shown the Initiatives Page, which contains the list of initiatives recorded into the DREAM system, ordered in descending temporal order. Only the last 50 initiatives are displayed, and two buttons "Next" and "Previous" allow the user to navigate through other sets of initiatives. In the page there is also a form that allows the user to filter the initiatives according to the type, date, area, etc..
By selecting an initiative in the list, the user is shown the corresponding Initiative Analysis Page, where some indicators comparing the farm production one month before and one month after the initiative are displayed, together with a

form that allows to user to change the time scope of these indicators.[31]
A "Back" button in the Initiatives Page brings to the Policy Maker Home Page, while a "Back" button in the Initiatives Analysis Page brings to the Initiatives Page.

[31] See *RASD - 3.2.4 Use cases*, Use Case 18.

4 Requirements Traceability

This section shows the mapping between the functional and non functional requirements presented in the RASD and the components presented in the section 2.2 of this document.

4.1 Functional Requirements Traceability

The following table shows how business logic components presented in the section 2.2 of this document implement the functional requirements of the `DREAM` system, listed in *RASD - 3.2.1 Requirements*.
Some of the components presented in section 2.2 are not present in the table:

- Webapp: It is involved in the implementation of all the requirements involving the interaction with the user (R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R16, R17, R18, R21, R22, R24, R25, R26, R28, R29, R38, R42, R43, R44, R45, R46, R47, R48, R49).
- NotificationService: It is not included in the table, as only business logic components are. It is involved in the implementation of requirements R33, R35, R45.
- DBAL: It is necessary for providing an implementation for all requirements involving the access to the database (all requirements except R26, R27, R28, R30).
- DBMS: It is necessary for providing an implementation for all requirements of the system involving the storage of or access to persistent data (all requirements except R26, R27, R28, R30).

Table 1: Traceability matrix of components to functional requirements.

Mapping between Components and Functional Requirements										
	FS	DPS	AS	SS	WS	PDS	RS	HRS	SUG	AUTHS
R1				X						
R2					X					

	FS	DPS	AS	SS	WS	PDS	RS	HRS	SUG	AUTHS
R3				X						
R4		X								
R5										X
R6						X				
R7						X				
R8						X				
R9						X				
R10						X				
R11						X				
R12		X				X				
R13		X								
R14		X								
R15				X	X	X	X			
R16										
R17										
R18							X			

	FS	DPS	AS	SS	WS	PDS	RS	HRS	SUG	AUTHS
R19			X			X				
R20			X	X	X	X				
R21			X							
R22			X							
R23				X	X	X			X	
R24				X	X	X			X	
R25				X	X	X			X	
R26					X					
R27					X					
R28					X					
R29										X
R30					X					
R31								X		
R32								X		
R33								X		
R34								X		

	FS	DPS	AS	SS	WS	PDS	RS	HRS	SUG	AUTHS
R35								X		
R36	X									
R37	X									
R38		X								
R39		X								
R40		X								
R41		X								
R42		X								
R43		X								
R44		X								
R45		X								
R46						X				
R47						X				
R48										X
R49						X				

Legend:

- FS: ForumService

- DPS:DailyPlanService
- AS: AnalyticsService
- SS: SensorService
- WS: WeatherService
- PDS:ProductionDataService
- RS: RankingService
- HRS: HelpRequestService
- SUGGS: SuggestionService
- AUTHS: AuthenticationService

Here a brief explanation of the mapping presented in the previous table is provided.

Table 2: Explanation of mapping between components and functional requirements.

R1 - DREAM collects soil moisture data from the soil moisture sensors: SensorService As showed in the section 2.4 (Runtime view) of this document, in the GetSoilSensorsData sequence diagram, the SensorService component is responsible for asking the data to soil moisture sensor system and storing them in the database of the `DREAM` system. It also provides and interface to other components for accessing these data - getHumidityData, see section 2.5.4 of this document.
R2 - DREAM accesses weather reports data provided by the TSDPS system: WeatherService The WeatherService component is responsible for asking weather reports data to the TSDPS[32] system and storing them in the database of the `DREAM` system. It also provides and interface to other components for accessing these data - getWeatherReports, see section 2.5.5 of this document.

[32]Telangana State Development Planning Society

R3 - DREAM collects watering data from the water irrigation system: SensorService

The SensorService component is responsible for asking water irrigation data to the water irrigation system and storing them in the database of the `DREAM` system. It also provides and interface to other components for accessing these data - getIrrigationSystemData, see section 2.5.4 of this document.

R4 - DREAM shall allow agronomists to insert feedback about the farmers they visited: DailyPlanService

Agronomists insert feedback about their visits to the farms when confirming their daily plan, and this operation is managed by the DailyPlanService with the confirmDailyPlan interface (see section 2.5.10 of this document).

R5 - DREAM shall allow farmers to insert the location and the area of their plot of land: AuthenticationService

The insertion of these data by the farmers is managed by the AuthenticationService component with the signup interface (see section 2.5.3 of this document).

R6 - DREAM shall allow farmers to insert the type of products they grow in a certain area of their farms: ProductionDataService

Farmers can insert the type of products they grow in a certain area of their farms when inserting production data. This interaction is managed by the ProductionDataService, that offers an insertData interface (see section 2.5.6 of this document).

R7 - DREAM shall allow farmers to insert data about their sowing activity: ProductionDataService

R8 - DREAM shall allow farmers to insert data about their planting activity: ProductionDataService

R9 - DREAM shall allow farmers to insert data about their harvesting activity: ProductionDataService

R10 - DREAM shall allow farmers to insert data about the irrigation of their plantations: ProductionDataService

R50 - DREAM shall allow farmers to insert data about their fertilizing activity: ProductionDataService

Farmers can insert their production data, regarding their sowing, planting, harvesting, fertilizing and watering activity - through the webapp. The insertion of these data in the database is performed by the Production-DataService component, that offers an insertData interface (see section 2.5.6 of this document and section 2.4 - sequence diagram InsertProductionData). It also offers a ReadProductionData interface to other components, that allows them with readData and readDataByArea to access these data.

R11 - DREAM shall allow farmers to insert descriptions of problems they face: ProductionDataService

Farmers can insert problems they face in the "MyProduction" section of the website. The insertion of such problems in the database of the `DREAM` system is performed by the ProductionDataService, which offers an insertIssue interface (see section 2.5.6 of this document). It also offers a readIssues interface to other components that allows them to access problems inserted by farmers.

R12 - DREAM shall allow farmers to request a visit of an agronomist when inserting a problem: ProductionDataService, DailyPlanService

R13 - When a farmer requests a visit of an agronomist, DREAM inserts the visit to the farmer into the daily plan of an agronomist assigned to the area of the farmer: DailyPlanService

If a farmer requests a visit of an agronomist when inserting a problem, the ProductionDataService, which processes the insertion of problems by farmers (interface insertIssue, see section 2.5.6 of this document), uses the DailyPlanService interface addRequestForVisit (see section 2.5.10 of this document) to insert the requested visit into the daily plan of an agronomist assigned to the area of the farmer.

R15 - DREAM can use different criteria (namely, productivity in terms of harvested quantity over sowed quantity, productivity in presence of adverse meteorological events, productivity in presence of drought, and so on) and/or combinations of them to rank farmers: SensorService, WeatherService, ProductionDataService, RankingService

R16 - DREAM shall allow policy makers and agronomists to choose the criteria to use for ranking farmers and the associated weights (for combinations of them): SensorService, WeatherService, ProductionDataService, RankingService

R17 - DREAM shall allow policy makers and agronomists to choose the time period and the area to consider for ranking the farmers: RankingService

The RankingService component offers a rankFarmer interface (see section 2.5.7 of this document), that allows to rank the farmers according to different criteria and with respect to different time periods and areas. In order to rank the farmers, it uses interfaces provided by SensorService (getHumidityData, getIrrigationSystemData, see section 2.5.4 of this document), WeatherService (getWeatherReports, see section 2.5.4 of this document), ProductionDataService (readDataByArea, see section 2.5.6 of this document). See also section 2.4 of this document, sequence diagram VisualizeBestAndWorstFarmers.

R18 - DREAM allows policy makers and agronomists to mark/unmark a farmer as best-performing or worst-performing: RankingService

The RankingService component offers a markFarmer and unmarkFarmer interfaces for implementing this requirement. (See also section 2.5.7 of this document)

R19 - DREAM can compare different time periods of a farmer work according to various production criteria (e.g. production volume, fertilizers adopted, fraction of harvested plants over sowed ones, etc): AnalyticsService, ProductionDataService

R20 - DREAM can compare different time periods of a farmer work with respect to environmental factors (e.g. weather reports data, soil moisture, ...): AnalyticsService, SensorService, WeatherService, ProductionDataService

R21 - DREAM allows policy makers to choose an initiative taken by an agronomist or a farmer to help a farmer - i.e., visit to the farm or reply to a question - and two time periods of the farmer to compare the two time periods: AnalyticsService

R22 - DREAM can show the impact of a certain initiative taken by an agronomist or a farmer to help a farmer - i.e., visit to the farm or reply to a question - during a certain time period: AnalyticsService

In order to allow policy makers to analyze the impact of initiatives involving farmers and agronomists on the farmers' production, the AnalyticsService component offers two interfaces: getInitiatives ad analyseInitiative (see section 2.5.12 of this document). To analyze the impact of an initiative, that is comparing the farmer's production before and after the initiative, the AnalyticsServiceComponent uses interfaces provided by the ProductionDataService (readData, see section 2.5.6 of this document), SensorService (getHumidityData, getIrrigationSystemData, see section 2.5.4 of this document) and WeatherService (getWeatherReports, see section 2.5.5 of this document) components.

R23 - DREAM is able to find correlations among environmental factors, fertilisers adopted and crops planted with the volume of production of the farmers: SuggestionService, SensorService, WeatherService, ProductionDataService

R24 - DREAM is able to send suggestions to farmers about fertilizers to useSuggestionService, SensorService, WeatherService, ProductionDataService

R25 - DREAM is able to send suggestions to farmers about crops to plant: SuggestionService, SensorService, WeatherService, ProductionDataService

The SuggestionService component offers a getSuggestions interface (see section 2.5.11), that computes some suggestions for a farmer about fertilizers to use or crops to plant. In order to achieve this result, the SuggestionService component uses interfaces provided by: SensorService (getHumidityData, see section 2.5.4 of this document), WeatherService (getWeatherReports, getWeatherForecasts, see section 2.5.5 of this document), ProductionDataService (readData, see section 2.5.6 of this document).

R26 - DREAM allows farmers to choose the date of the forecasts to visualize: WeatherService

R27 - DREAM is able to connect to the Telangana government website to fetch forecasts for the chosen date: WeatherService

R28 - DREAM can show weather forecast data for a certain location and date: WeatherService

R30 - DREAM shall allow agronomists to choose the date of the weather forecasts to visualize

Requirements pertaining the weather forecasts are implemented by the WeatherService component, which offers a getWeatherForecasts interface (see section 2.5.5 of this document), that retrieve weather forecasts for a chosen date from the TSDPS website.

R29 - DREAM shall allow agronomists to insert the area they are responsible of: AuthenticationService

The insertion and modification of these data by the farmers is managed by the AuthenticationService component.

R31 - DREAM allows farmers to choose the (best-performing) farmer or agronomist (assigned to his area) who to issue a request: HelpRequestService

R32 - DREAM allows the farmers to send a request to a best-performing farmer or agronomist: HelpRequestService

R33 - DREAM notifies (best-performing) farmers and agronomists of the requests of help from other farmers: HelpRequestService

R34 - DREAM allows best-performing farmers and agronomists to insert a message of response to the farmers who have made a request for help to them: HelpRequestService

R35 - DREAM sends the response to the farmer who issued the corresponding request: HelpRequestService

The HelpRequestService component offers all the interfaces necessary to implement the requirements related to help requests from farmers to agronomists or best-performing farmers (see section 2.5.8 of this document and section 2.4 of this document - RequestToExpert sequence diagram). The listAvailableExperts interface allows farmers to see who are the agronomists and best-performing farmers he/she can issue a request to (R31). The createHelpRequest interface allows farmers to send an help request to the chosen agronomist or best-performing farmer (R32, R33) - that is, saving it in the database of the `DREAM` system, from where it will be retrieved via the interface listHelpRequestsTo when the agronomist or best-performing farmer will access the "MyReplies" section of the website. The answerHelpRequest interface allows recipients of requests to insert a reply (R34, R35) - that is, saving it in the database of the `DREAM` system, from where it will be retrieved via the interface listHelpRequestsFrom when the farmer will access the "MyRequests" section of the website. The HelpRequestService component uses the NotificationService component to notify users of new help requests or replies; this is not shown in the table as explained above.

R36 - DREAM allows farmers to initiate a new thread in the forum: ForumService

The requirement is implemented by the ForumService component, which offers a createTopic interface (see section 2.5.9 of this document).

R37 - DREAM allows the farmers to add a post to a thread in the forum: ForumService

The requirement is implemented by the ForumService component, which offers an addMessage interface (see section 2.5.9 of this document).

R38 - DREAM is able to generate a daily plan of visits for each agronomist evenly distributing (with exceptions to bad-performing farmers) over the year the number of visits to each farmer: DailyPlanService

R39 - DREAM enforces that every farmer is assigned a visit in the daily plan of an agronomist at least twice a year: DailyPlanService

R40 - DREAM enforces that no agronomist in a certain area is assigned more than twice of the visits than an agronomist of the same area: DailyPlanService

R41 - DREAM enforces that the daily plan of agronomists includes more visits to worst-performing farmers than to other ones: DailyPlanService

Requirements pertaining the generation of the daily plan for agronomists are implemented by the DailyPlanService component, which offers a generateDailyPlan interface (see section 2.5.10 of this document, section 2.4 of this document - AcceptDailyPlan sequence diagram).

R42 - DREAM allows agronomists to accept or modify -i.e.: remove, add or move a visit - the automatically generated daily plan: DailyPlanService

The DailyPlanService offers interfaces for modifying the daily plan of an agronomist - addDailyPlanEntry, removeDailyPlanEntry, moveDailyPlanEntry - and for accepting a daily plan (saveDailyPlan). (See section 2.5.10 of this document and section 2.4 of this document - sequence diagram AcceptDailyPlan).

R43 - DREAM allows agronomists to confirm a daily plan at the end of the working day: DailyPlanService

The DailyPlanService offers a confirmDailyPlan interface for confirming the daily plan of an agronomist (see section 2.5.10 of this document).

R44 - DREAM allows agronomists to specify deviations -i.e.: remove, add or move a visit- from a daily plan at the end of the working day: DailyPlanService

The DailyPlanService offers interfaces for modifying the daily plan of an agronomist - addDailyPlanEntry, removeDailyPlanEntry, moveDailyPlanEntry (see section 2.5.10 if this document).

R45 - DREAM is able to notify farmers involved by a visit of an agronomist: DailyPlanService

When an agronomist accepts a daily plan, the saveDailyPlan interface of the DailyPlanService component is used. When this happens, the DailyPlanService component uses the NotificationService component to notify farmers involved by visits of agronomists (the mapping is not showed in the table, as explained above).

R46 - DREAM allows agronomists to visualize data about farmers to visit - such as name, surname and location of the farm - and their production since their last visit to the farm: ProductionDataService

R47 - DREAM allows agronomists to visualize the problems inserted by a farmer to be visited since their last visit to the farm: ProductionDataService

These requirements are implemented by the ProductionDataService component, that offers the readData and readIssues interfaces (see section 2.5.6 of this document).

R48 - DREAM shall allow users to log in the system by using an email and a password: AuthenticationService

This requirement is implemented by the AuthenticationService component, that offers the login interface (see section 2.5.3 of this document).

R49 - DREAM shall allow farmers to visualize data inserted by them about their production: ProductionDataService

This requirements is implemented by the ProductionDataService component, that offers the readData and readIssues interfaces (see section 2.5.6 of this document).

4.2 Non Functional Requirements Traceability

Non functional requirements for the `DREAM` system are presented in *RASD - 3.3 Performance Requirements* and in *RASD - 3.5 Software System Attributes*. In this section, an explanation of how this requirements are met by elements presented in this document is provided.

4.2.1 Performance Requirements

Performance requirements regarding the number of simultaneous users and transactions supported by the `DREAM` system are achieved by the possibility of replicating the web server and the database showed in the deployment diagram in section 2.3 - Deployment view - on more than one physical machine.
Performance requirements regarding the quantity of data stored by the `DREAM` system are achieved by the possibility of distributing the database on more than one physical machine.

4.2.2 Software System Attributes

As discussed in RASD, reliability and availability are not main concerns of the DREAM system; nevertheless, they are increased by the replication of web server and database.
Elements supporting the security requirement are:

- the usage of HTTPS[33] connections between clients and servers, to prevent interception of messages by unintended parties
- the usage of a firewall between clients and servers, to filter requests to the DREAM system, eliminating malicious ones
- the usage of a server-side framework that provides security functionalities off-the-shelf, such as authentication and protection against SQL injection attacks

Maintainability is mainly achieved through the rigorous alignment of implementation and documentation, that includes the RASD, the DD (this document itself) and the ITD[34].
Client-side portability is achieved by the choice of implementing the DREAM system as a web application: clients of web applications are browsers, and browsers are available for essentially all possible devices. Just one remark: portability of the web application through different browsers should be taken in consideration during the IT[35] phase. Server-side portability is achieved by the adoption of a server-side framework based on an interpreted programming language, executed on a platform available for all most popular OSs[36] (namely, Linux, MacOS and Windows).

[33] Hypertext Transfer Protocol Secure, provides secure communication over a network by the mean of cryptography
[34] Implementation and Testing document
[35] Implementation and Testing
[36] Operating Systems

5 Implementation, integration and test plan

5.1 Implementation and integration

As outlined in the component diagram, there is a fairly small number of dependencies between components; this is due to the fact DREAM is made up of many independent features.

The first part of the system to bring up is the **DBMS** along with the **Database Abstraction Layer**, since basically every other component relies on database access to store and retrieve data. The DBAL component will not be implemented in-house; instead, a robust pre-existing solution provided by the adopted framework will be used.

The first component that must be set up is the **AuthenticationService**, since authentication is required for all other components to function. Again, this being a generic feature, it will be provided by the framework and will only need to be configured for our use case.

Subsequently, work on the other backend components can begin and be carried on in parallel, with the exception of the **RankingService** which requires the development of other services to be complete (namely, the `Weather` and `ProductionData` components). Simultaneously with the development of each component, the required integrations will be performed. As already mentioned, components are loosely coupled, thus a more detailed integration plan would be redundant.

As for frontend development, thanks to the specific architecture chosen, it is possible to proceed with development of the web interface immediately after the completion of each component. This is preferable to developing the frontend strictly after the entire backend is ready, as it allows for immediate feedback on the specific feature being developed.

5.2 Testing

Due to the number of independent features in the system, it is crucial that **each component undergoes unit testing**. Testing will begin as soon as the development of each component is complete, and will focus mostly on the business logic, and to a lesser extent to presentation logic.

For our specific use case, integration testing is of minor significance, but still required for some specific cases (e.g. the `RankingService` as explained above).

After the implementation and integration phase is complete and the system is deemed to be in a production-ready state, **stress testing** is also required due to the large number of users that will access the system simultaneously. This will be carried forward using specialized external services that allow sending a large number of requests while simulating real user behaviour.

6 Effort spent

Table 3: The time Christian Grasso has spent on this document.

Effort spent	
Initial organization	2h
Interfaces and organization	4.30h
Component diagram	2.30h
Other design decisions and component interfaces	3h
Final recap	2.30h

Table 4: The time Filippo Lazzati has spent on this document.

Effort spent	
Initial organization	2h
Introduction	2h
Interfaces and organization	2h
Deployment diagram and other design choices	3.30h
Runtime view	3h

Overview	45mins
Final recap	2.30h

Table 5: The time Chiara Magri has spent on this document.

Effort spent	
Initial organization	2h
Interfaces and organization	2h
Component interfaces	1:30h
User interface	11h
Requirements traceability	7h
Final recap	2.30h

7 References

7.1 Used tools

- IFMLedit: to produce IFML diagrams
- StarUML: to draw UML diagrams;
- draw.io: another tool to draw UML diagrams;
- Alloy: to specify formally some requirements;
- Overleaf: Latex editor;
- Github: to share the files;