POLITECNICO
MILANO 1863

Software Engineering 2 project

Academic Year 2021-2022

DREAM - Data-dRiven PrEdictive FArMing in Telengana

Design Document

Version 0.2 – 08/01/2022

1.INTRODUCTION

1.1 Purpose

Agriculture plays a vital role in India's economy. However, with food demand increasing, climate change and COVID-19 pandemic, agriculture in Telengana India is facing severe challenges, like unstable food supply chains, vulnerabilities of marginalized communities and smallholders.

Telengana's government wants to build anticipatory governance models for food systems using digital public goods and community-centric approaches to strengthen data-driven policy making in Telengana.

This document contains an explanation of the design decisions that we have made for the whole system, going from the general architecture design to the specific components, their interfaces, their interactions and their physical deployment, along with a presentation of some graphical user interface mockups. Moreover, this document also contains a discussion on the implementation and testing plan in order to give the developers a general roadmap.

1.2 Scope

The Dream System is an easy-to-understand interface which aims to helping the user to complete their own tasks on the process of Farming.

The Dream System allows the Policy Makers to identify those farmers who are performing well or badly and understand whether the steering initiatives carried out by agronomists.

The Dream System allows farmers to visualize data relevant to them, provide information about their production, ask for help and suggestion by agronomists and other farmers and create discussion forums with the other farmers.

The Dream System also allows agronomists to receive requests for help and answer to these requests, visualize data concerning weather forecasts in the responsible area and the best performing farmers in the responsible area, create and modify a daily plan to visit farms in the responsible area and confirm the execution of the daily plan at the end of each day or specify the deviations from the plan.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- DREAM System (or “The System”): refers to the whole system to be developed.

1.3.2 Acronyms

- API: Application Programming Interface
- RASD: Requirement Analysis and Specification Document.
- UML: Unified Modelling Language.
- GPS: Global Positioning System.

1.3.3 Abbreviations

- C.i: i-th component

1.4 Revision history

Version	Date	Authors	Summary
0.1	06/01/2022	Rui Zhang	Update section1 and section 3
0.2	08/01/2022	Zhijun Hu	Update section2 Fix section3

1.5 Reference Documents

- Specification document: Project Assignment A.Y. 2021-2022.pdf
- RASD of DREAM
- Software Engineering 2 course slides
- IEEE Standard on Requirement Engineering (ISO/IEC/IEEE 29148)

1.6 Document Structure

This document is structured as follows:

1. ***Introduction*** – A general introduction of the system-to-be, which aims at giving general information about what this document is going to explain.
2. ***Architectural Design*** – An overview of the high-level components and their interactions, with a focus on both static and dynamic view, with the help of diagrams.
3. ***User Interface Design*** – A representation of how the User Interface will look like.
4. ***Requirements Traceability*** – An explanation about how the requirements defined in the RASD map to the design elements defined in this document.
5. ***Implementation, Integration and Test Plan*** – Identification of the order in which the sub-components of the system should be implemented, integrated and tested.
6. ***Effort spent –*** Effort spent by all team members shown as the list of all the activities done during the realization of this document
7. ***References*** – References to documents that this project was developed upon.

2. ARCHITECTURAL DESIGN

2.1 Overview

The figure shown below represents a high-level description of the main components which make up the System. 4-tier architecture is used to facilitate maintainability and scalability. Further details will be provided in sections 2.6 and 2.7.

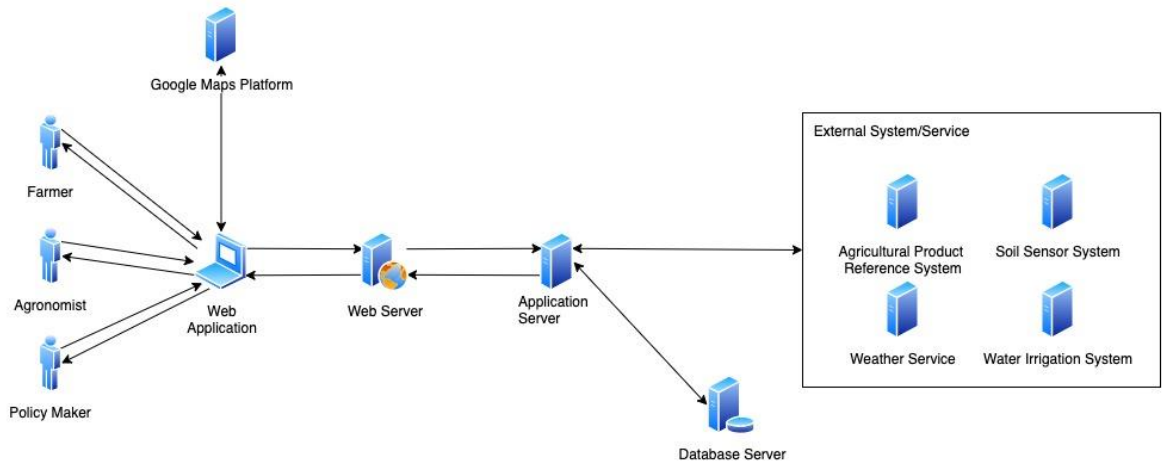


Figure 1 Overall architecture of the System

The main components of the Systems are the following:

- **Web Application**

  A web application accessible through Farmer/Agronomist/Policy Maker's browser that allows him/her to access DREAM Services. The web app will work with the most modern internet browsers, which communicate with the System by sending requests to the Web Server

- **Web Server**

  It is the backend component that communicates with Farmer/Agronomist/Policy Maker's browser on one hand, and with Application Server on the other.

- **Application Server**

  It is the backend component of the System which provides business functionality, i.e., the business logic tier. It communicates with the various External Systems / Services and interacts with the data layer and the web layer.

- **Database Server**

  It is the component which is responsible for data storage and can only be accessed by Application Server.

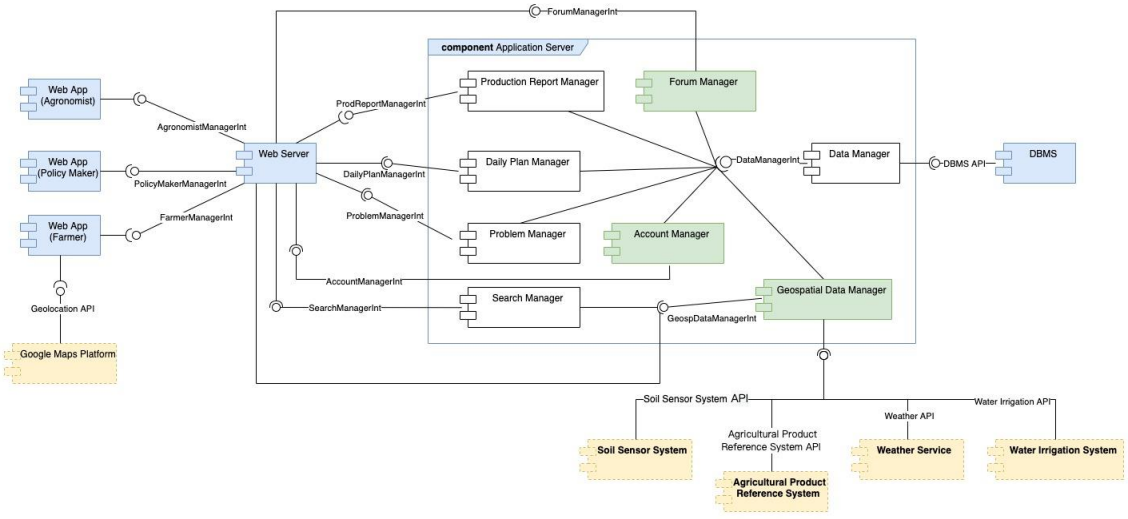
- **External Systems / Services**

  These are systems and services which provide functionalities that are not internally developed.

  - **Google Maps Platform**: this platform is responsible for providing the Map Services including *GPS Service* necessary for retrieving the location of the farmers.
  - **Weather Service**: this service is responsible for providing meteorological short-term and long-term forecasts.
  - **Soil Sensor System**: this system is responsible for providing data about the humidity of soil.
  - **Water Irrigation System**: this system is responsible for providing water usage per farmer.
  - **Agricultural Product Reference System**: this system is responsible for providing suggestions concerning specific crops.

2.2 Component view

In this section, every high-level component is analyzed in terms of its subcomponents. External Systems, such as *Google Maps Platform* and *Weather Service*, are presented as black boxes that expose only the interfaces used by DREAM. Further details about the component interfaces are shown in section 2.5.



Component Diagram 1: Main components

2.2.1 Web Application Component

It is the application dedicated to three roles, including farmer, agronomist and policy maker, which allows them to keep access data under control and possibly modify the store information.

2.2.2 Web Server Component

A *Web Server* is required to provide *Web Application* for all users. This component receives **HTTPS** requests from users' browsers, forwards them to the *Application Server*, and generates the dynamic web pages based on the response from the *Application Server*.

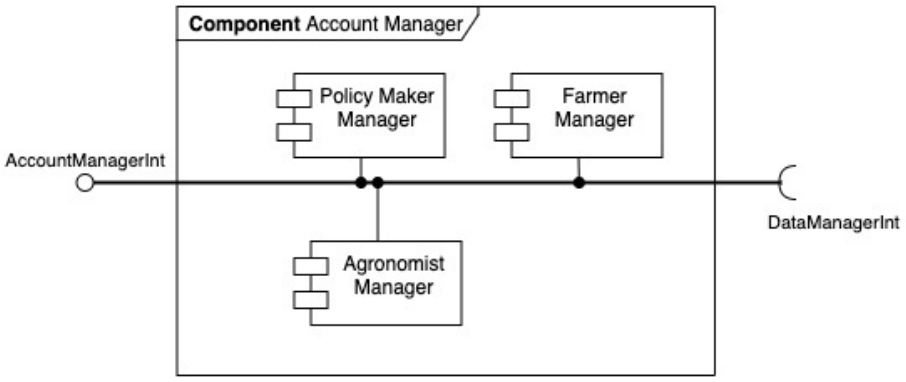
2.2.3 Application Server Component

The *Application Server* is responsible for the business logic which means that it needs to integrate all the needed data and coordinate the flow of information between application layer and the data layer.

As shown in the component view, the *Application Server Component* consists of the following sub-elements:

- **Account Manager**

This component handles all the account operations related to Farmer, Agronomist and Policy Maker which includes account creation and authentication. It communicates with *Data Manager* to verify, access and store account information.



Sub-component Diagram: Account Manager

It consists of the following subcomponents:

- **Policy Maker Manager:** it handles the registration process of a new policy maker and the authentication process by checking the credentials.
- **Agronomist Manager**: it handles the registration process of a new agronomist and the authentication process by checking the credentials.
- **Farmer Manager:** it handles the registration process of a new farmer and the authentication process by checking the credentials. Besides, it is responsible for the performance update process.

- **Production Report Manager**

This component is responsible for all the operations related to production report. It interacts with *Data Manger* to store and access production data.

- **Daily Plan Manager**

This component handles all the operations related to daily plan. It communicates with *Data Manager* to access, store and update information of daily plan.

- **Problem Manager**

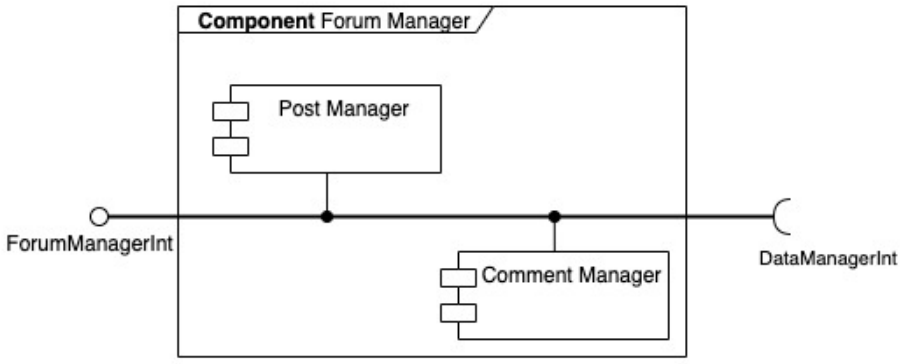
This component manages the problem-raising and handling processes, which access through *Data Manager* to retrieve, store and update information related to problem. Besides, once a new problem is created, it adds a notification to Agronomist. On the other hand, if a new problem is read, it removes the notification.

- **Search Manager**

This component is responsible for weather forecasting and production suggestions retrieving based on the farmer's input. It communicates with *Geospatial Data Manager* to access weather information and production suggestions.

- **Forum Manager**

This component handles all the operations related to discussion forum, which accesses the *Data Manager* to retrieve and store the related information.



Sub-component Diagram: Forum Manager

It consists of the following subcomponents:

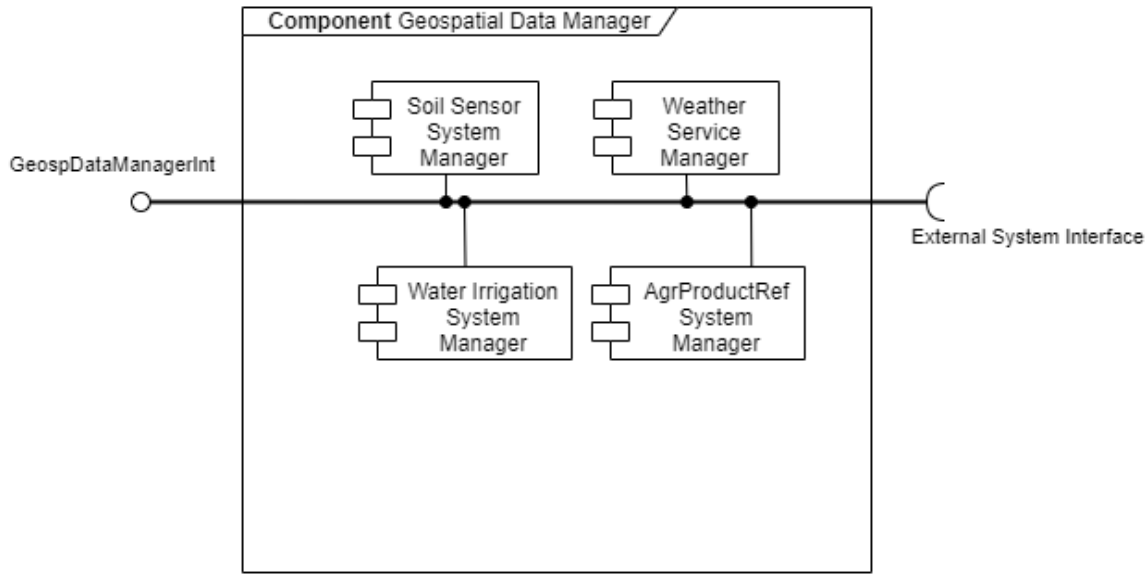
- **Post Manager**: it supports post creation and retrieval operations
- **Comment Manager**: it is responsible the comment process related to specific post.

- **Data Manager**

This component receives requests coming from the other components for retrieving and storing data in the database by interacting with the data layer.

- **Geospatial Data Manager**

The main function of this component is interacting with external systems and services and offering the processed data.



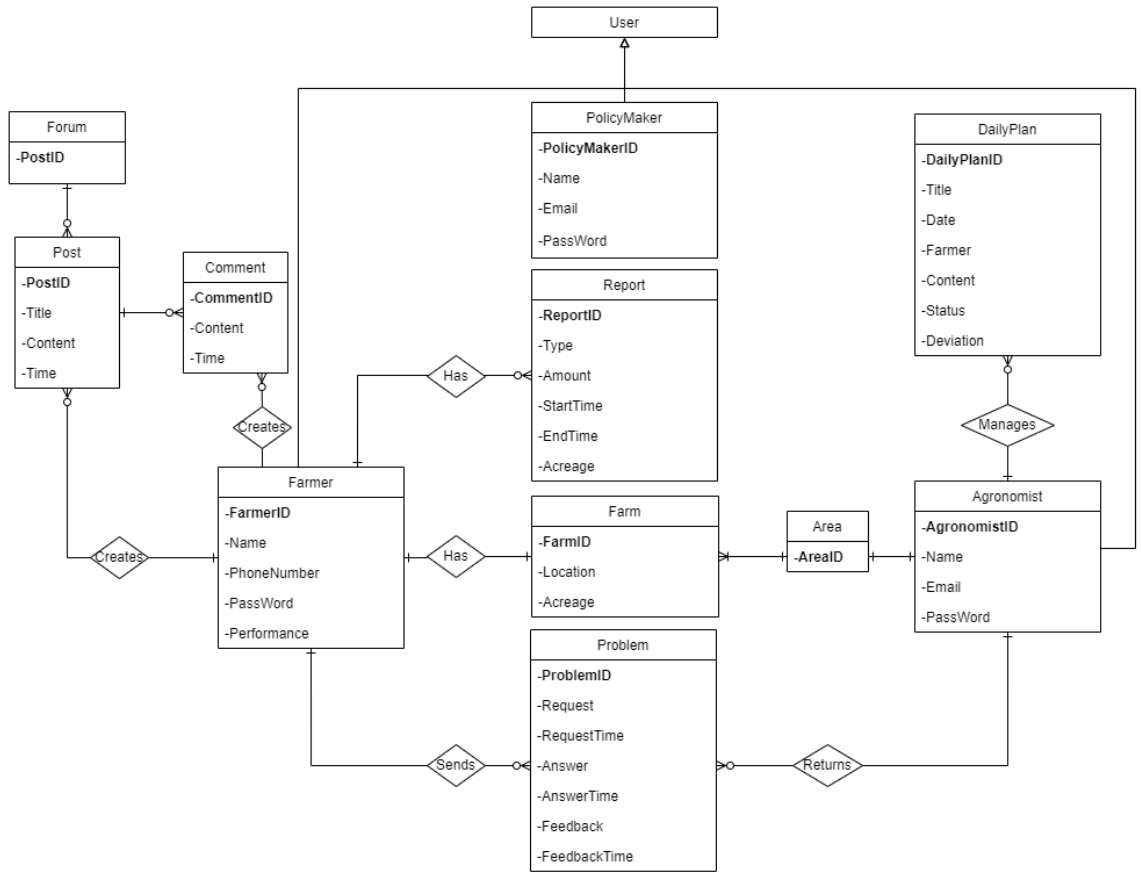
Sub-component Diagram: Geospatial Data Manager

It consists of the following subcomponents:

- **Soil Sensor System Manager**: it processes the data of soil humidity from *Soil Sensor System*.
- **Weather Service Manager**: it handles the data of weather from *Weather Service*.
- **Water Irrigation System Manager**: it handles the data of water usage from *Water Irrigation System*.
- **AgrProductRef System Manager**: it handles the data of agricultural product suggestion from *Agricultural Product Reference System*.
- **Area Manager**: it handles the area information.

2.2.4 Data Components

The image below represents the Entity-Relationship (ER) diagram of the database.



Diagonal 1: Entity Relationship

From the previous ER schema, it is possible to derive the following logical model:

Farmer(FarmeID, Name, PhoneNumber, Password, Performance)

PolicyMaker(PolicyMakerID, Name, Email, Password)

Agronomist(AgronomistID, Name, Email, Password, Area)

Farm(FarmID, Location, Acreage, Farmer, Area)

Problem(ProblemID, Request, RequestTime, Answer, AnswerTime, Feedback, FeedbackTime, Famer, Agronomist)

Report(ReportID, Type, Amount, Starttime, Endtime, Acreage, Famer)

DailyPlan(DailyPlanID, Title, Date, Farmer, Content, Status, Deviation, Agronomist)

Post(PostID, Title, Content, Time, Farmer, Forum, comment)

Comment(CommentID, Content, Time, Farmer, Post)

Area(AreaID)

Forum(ForumID)

2.2.5 External System

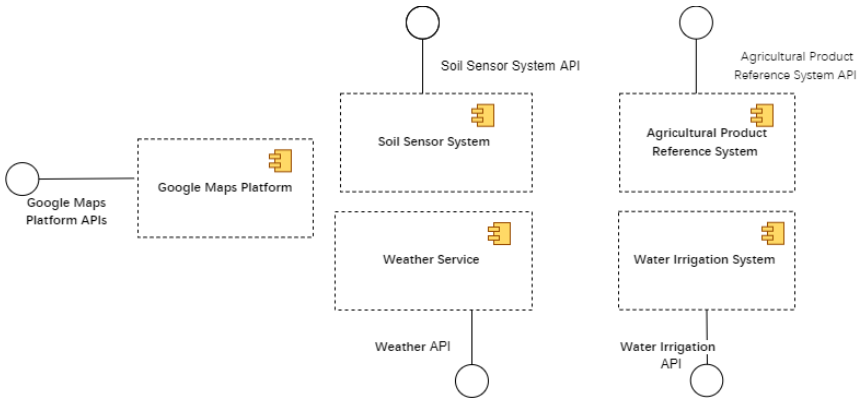
- **Google Maps Platform**
  This external system communicates with the Web Application via APIs. In particular, it provides an interface to acquire the geolocation based on WiFi or IP address.

- **Weather Service**
  This external system communicates with the Application Server via HTTP page. By using these APIs and thus being able to automatically access the information of the weather forecast, which supproted temperature, humidity, precipitation, cloud, wind direction, sunlight duration and so on.

- **Water Irrigation System**
  This external system communicates with the Application Server via APIs and supports the water usage information with regard to the farmer as an extra reference information, which supproted the water consumption per each area, Irrigation time and so on.

- **Soil Sensor System**
  This external system communicates with the Application Server via APIs and supports the soil humidity information regarding the farmer as an extra reference information, which supproted the soil temperature, soil modist, soil fertility and so on.

- **Agricultural Product Reference System**
  This external system communicates with the Application Server via APIs , which supprtes crop properity, crop type, crop, planting advice base on the planting time.

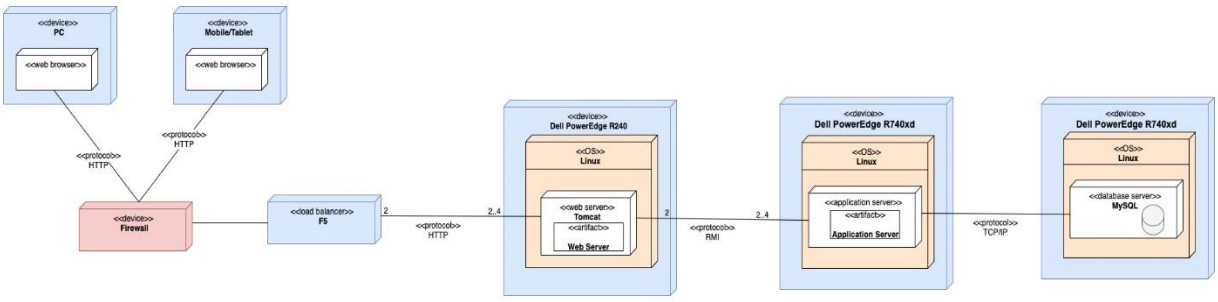


External Components

2.3 Deployment view

This section describes the topology of DREAM System's hardware and the components distribution which is shown below.

- *Firewalls* and *load balancers* manage the data flow from clients to servers.
    - *Firewalls* filter the packets from the Internet for security.
    - *Load balancers* are used to distribute the workload among available resources to achieve availability and reliability.
- *Web Server* handles the HTTP requests and communicates with the *Application Server* through the public interfaces defined in section 2.5 using RMI.
- *Application Server* is responsible for the business logic by interacting between *J2EE Application* instance and the *Database Server* through MySQL public API of the database system.



Deployment Diagram

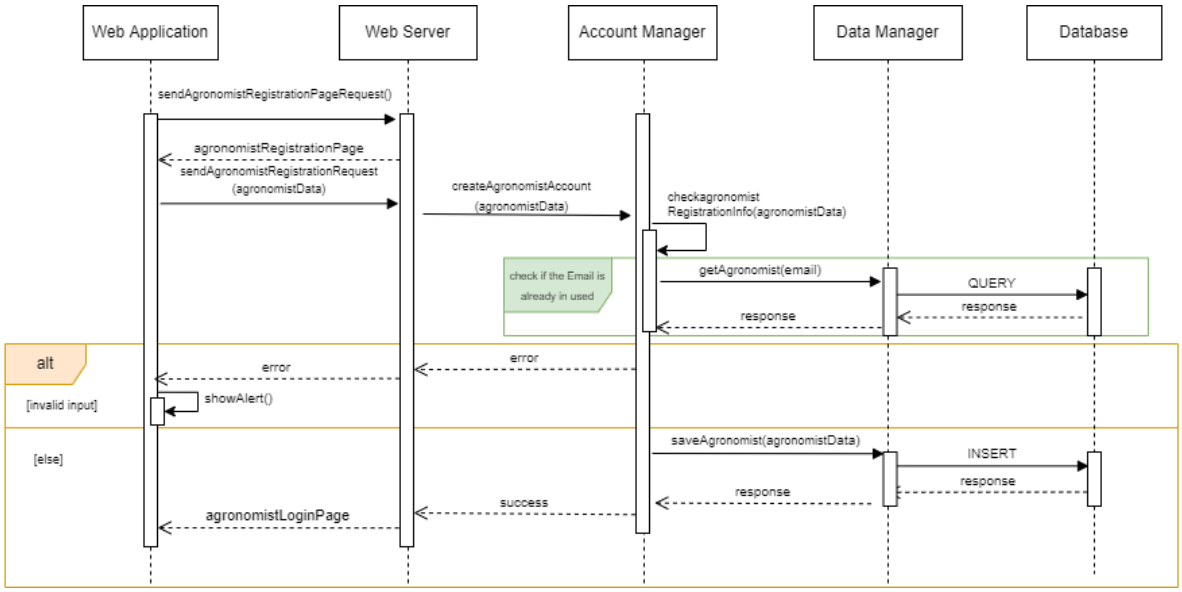
2.4 Runtime view

The runtime view describes concrete behavior and interactions of the system's building blocks in form of scenarios from the following areas:

- Important use cases or features
- Interactions at critical external interfaces
- Operation and administration: launch, start-up, stop
- Error and exception scenarios

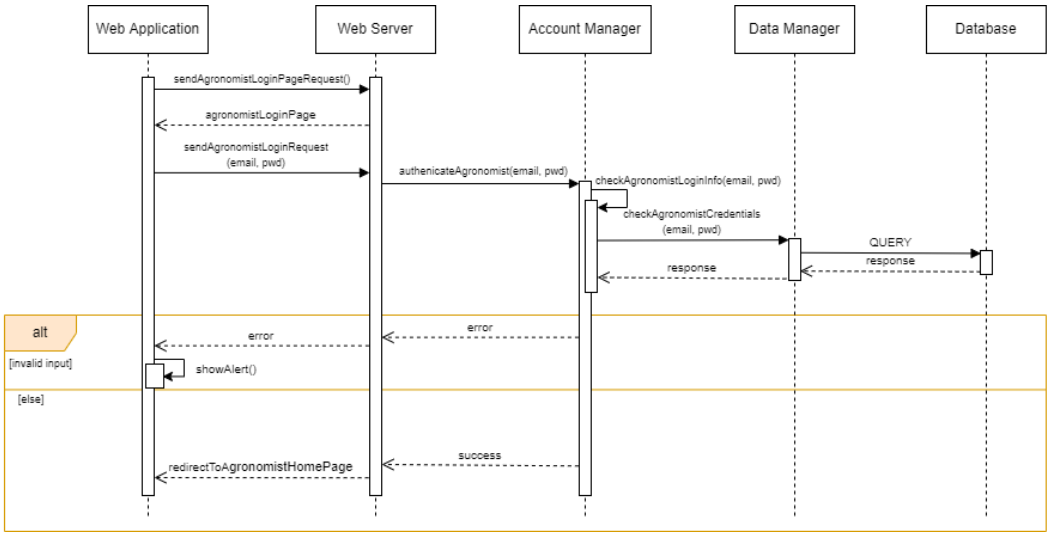
Further details about the component interfaces can be found in section 2.5.

- **Agronomist Registers**



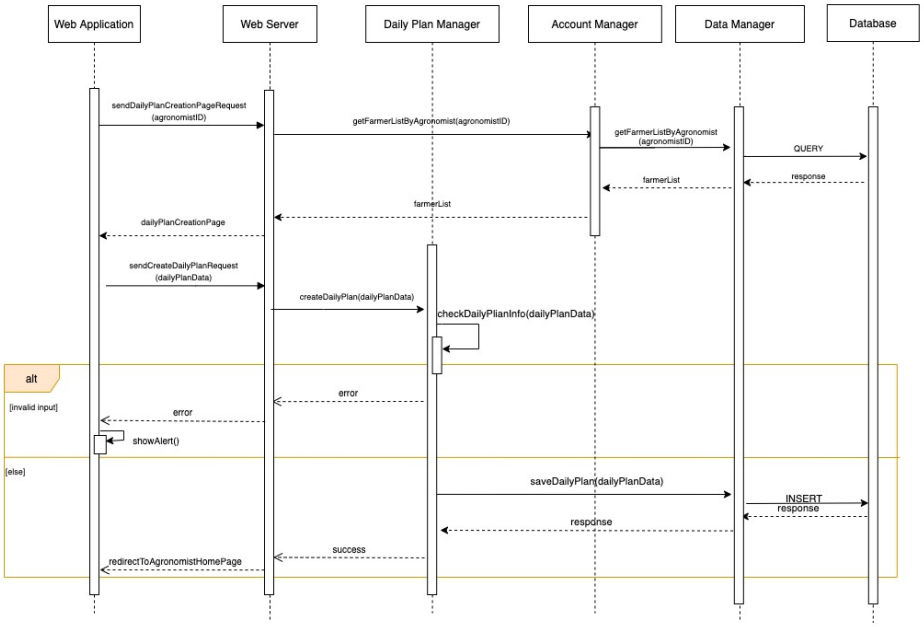
Runtime View Diagram 1: Agronomist Registers

- **Agronomist Logins**



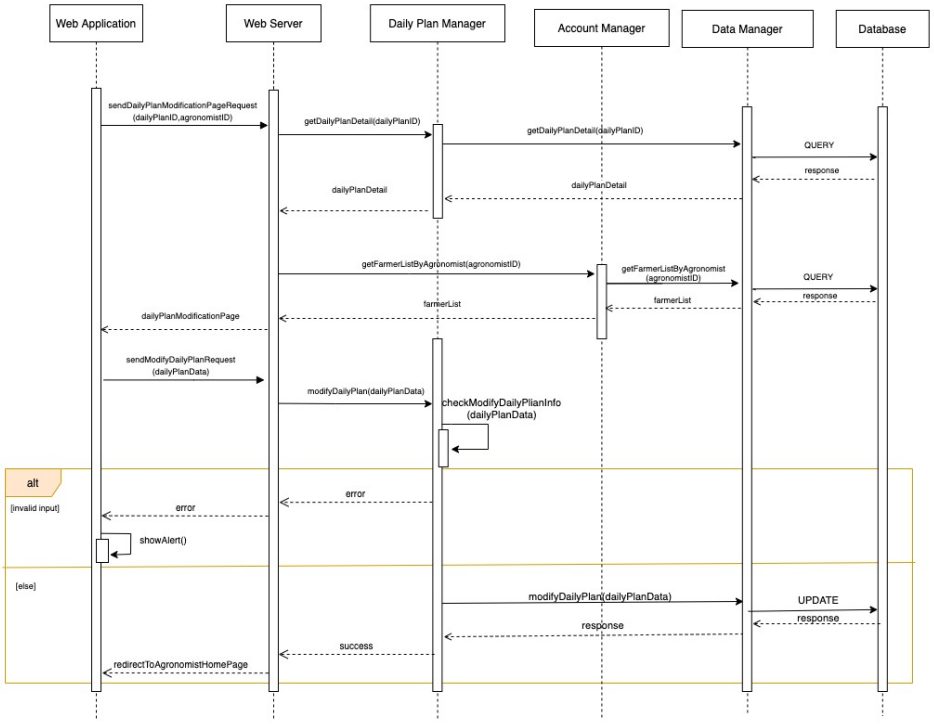
Runtime View Diagram 2: Agronomist Logins

- **Agronomist creates the daily plan**



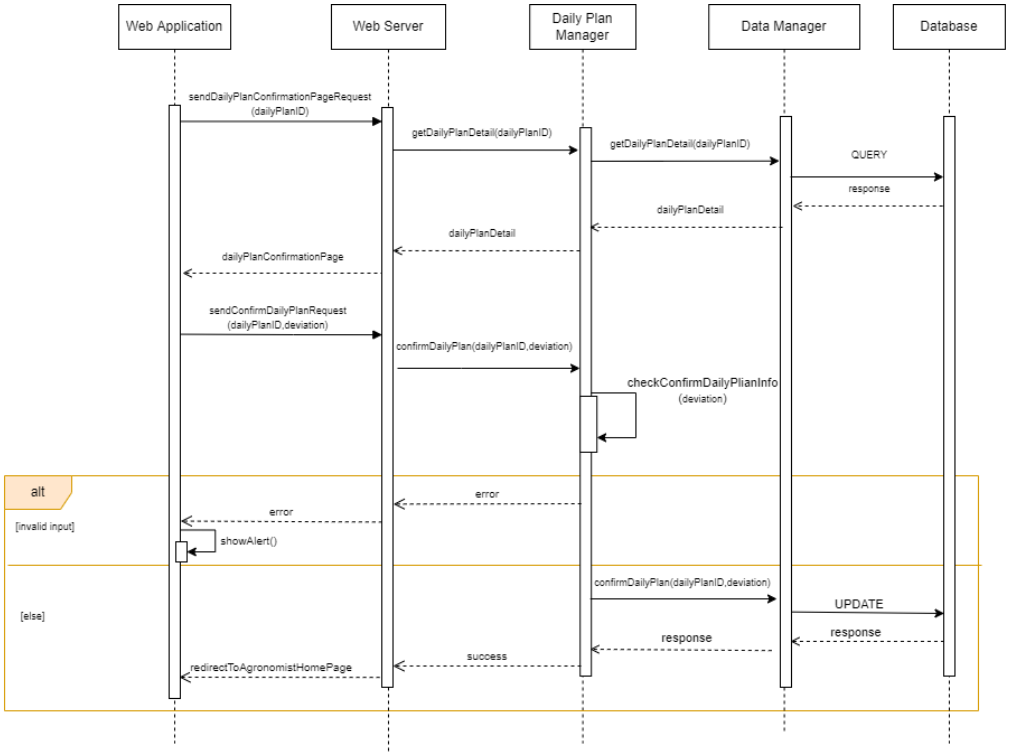
Runtime View Diagram 3: Agronomist creates the daily plan

- **Agronomist modifies the daily plan**



- *Runtime View Diagram 4: Agronomist modifies the daily plan*

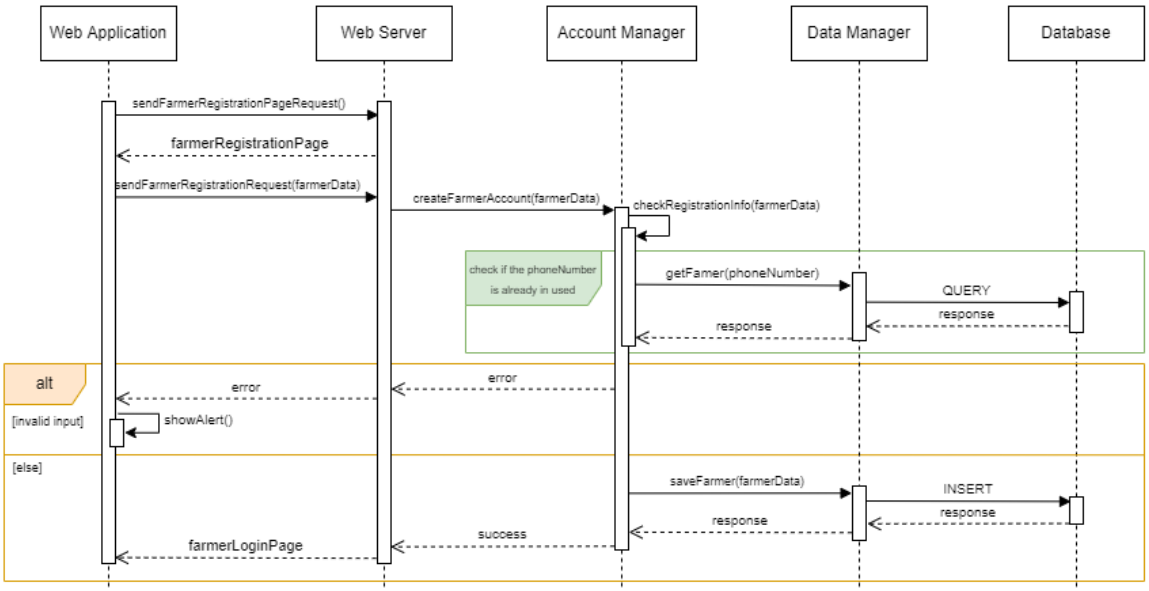
- **Agronomist confirms the daily plan**



Runtime View Diagram 5: Agronomist confirms the daily plan

- **Farmer Registers**

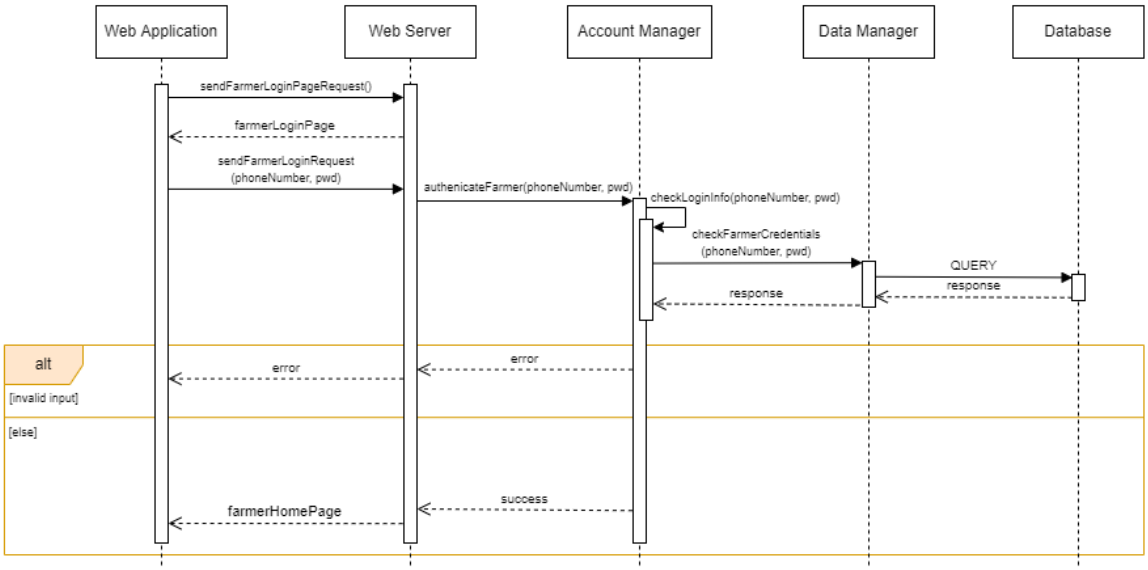
The following diagram represents workflow that Farmer registers in DREAM system. When unregistered user enters his/her data to register, System will check whether the input data meets the requirement. If so, it proceeds by forwarding the account creation request to the Application Server. Here the Account Manager checks the data entered by the Customer and in particular verifies if the phoneNumber is not already in use. If the checks pass, it proceeds by storing this data.



Runtime View Diagram 6: Farmer Registers

- **Farmer Logins**

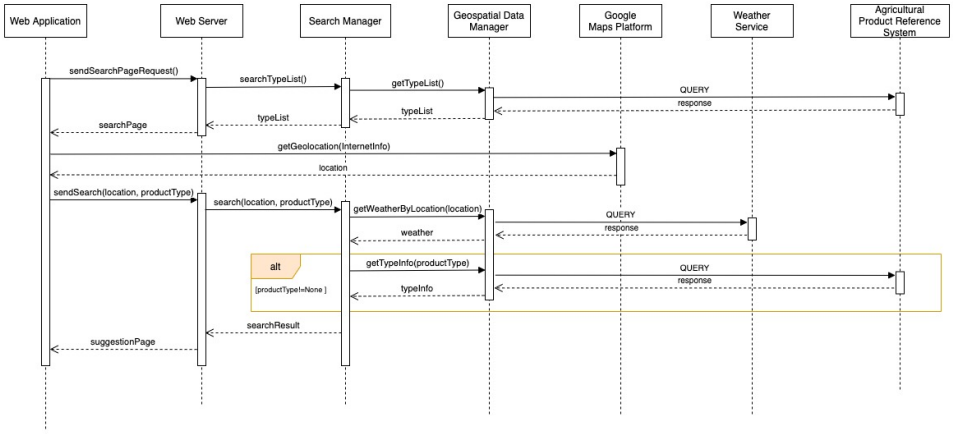
The following diagram represents workflow that Farmer Logins in DREAM system. After the Farmer Login Data has been submitted, the Web Application sends the request to the Web Server which forwards it to the Account Manager. This component verifies the credentials by interacting with the Data Manager. If it is correct, the Web Server redirects the Farmer Homepage to the homepage otherwise it shows an error.



Runtime View Diagram 7: Farmer Logins

- **Farmer Searches Information**

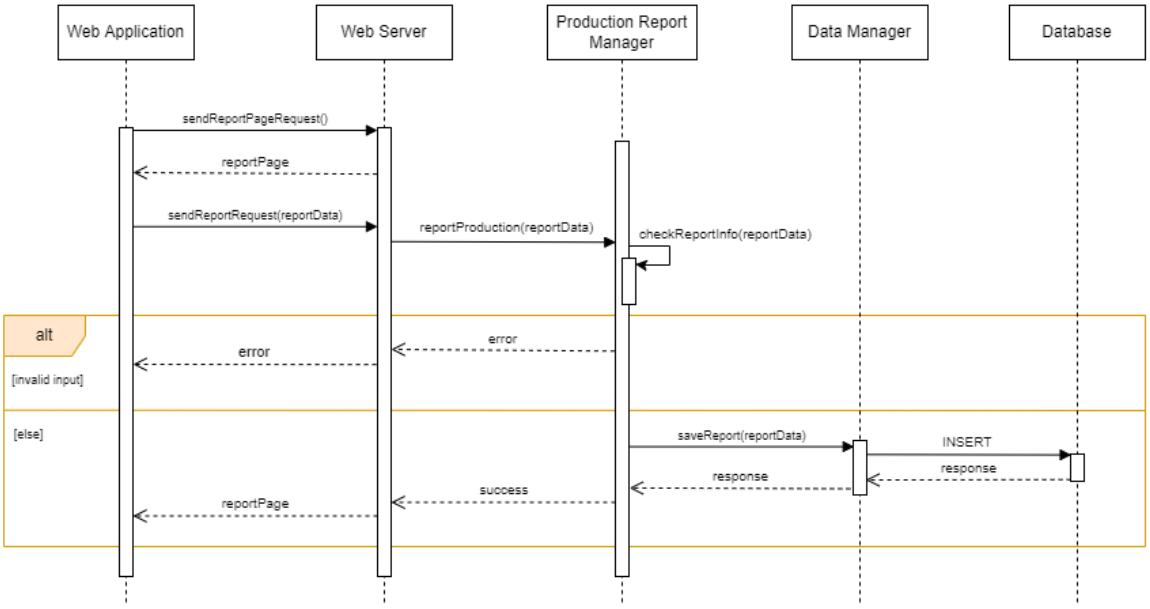
The following diagram represents workflow that Farmer Search Information in DREAM system. First, When Farmer sends Request to get Search Page, Search Manager will get production Type through Geospatial Data Manager and external Agricultural Product Reference System, then back Type List to Farmer. Meanwhile, Web Application send its Internet information, like IP and WIFI, to Google Maps System to get Geolocation of Famer. After the Farmer Search Data has been submitted, the Web Application sends the request to the Web Server which forwards it to the Search Manager. If only location is submitted, it returns only weather information. Otherwise, it returns both of weather forecast and type information.



Runtime View Diagram 8: Farmer Searches Information

- **Farmer Reports production**

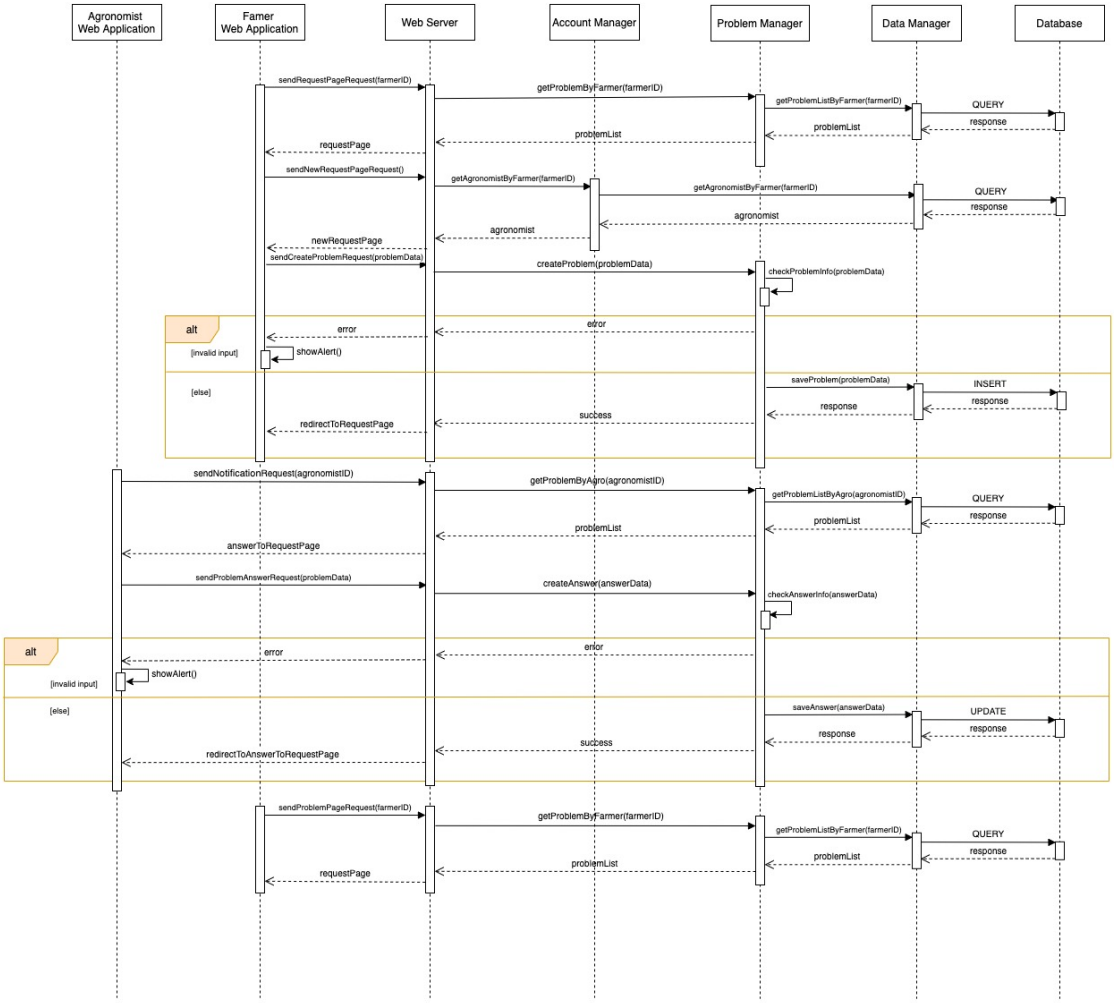
The following diagram represents workflow that Farmer Reports production in DREAM system. After the Farmer Search Data has been submitted, the Web Application sends the request to the Web Server which forwards it to the Production Report Manager. Production Report Manger checks whether the input data is valid and non-repeating. If so, it proceeds by forwarding the save request to Data Base and save the report. Then, Farmer is redirected to Report Page. Otherwise, it will show an error.



Runtime View Diagram 9: Farmer Reports production

- **Solve A Problem**

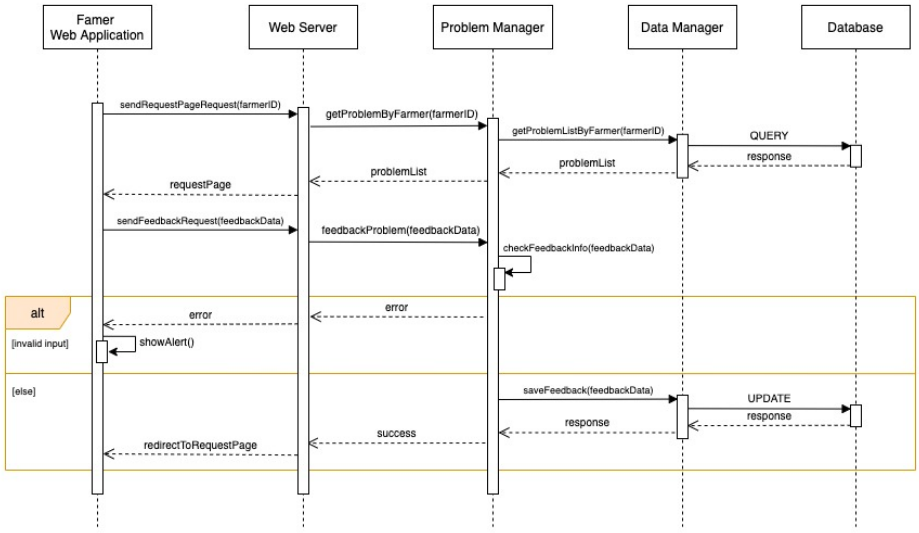
The following diagram represents workflow that Solve A Problem in DREAM system.



Runtime View Diagram 10: Solve A Problem

- **Farmer Gives Feedback**

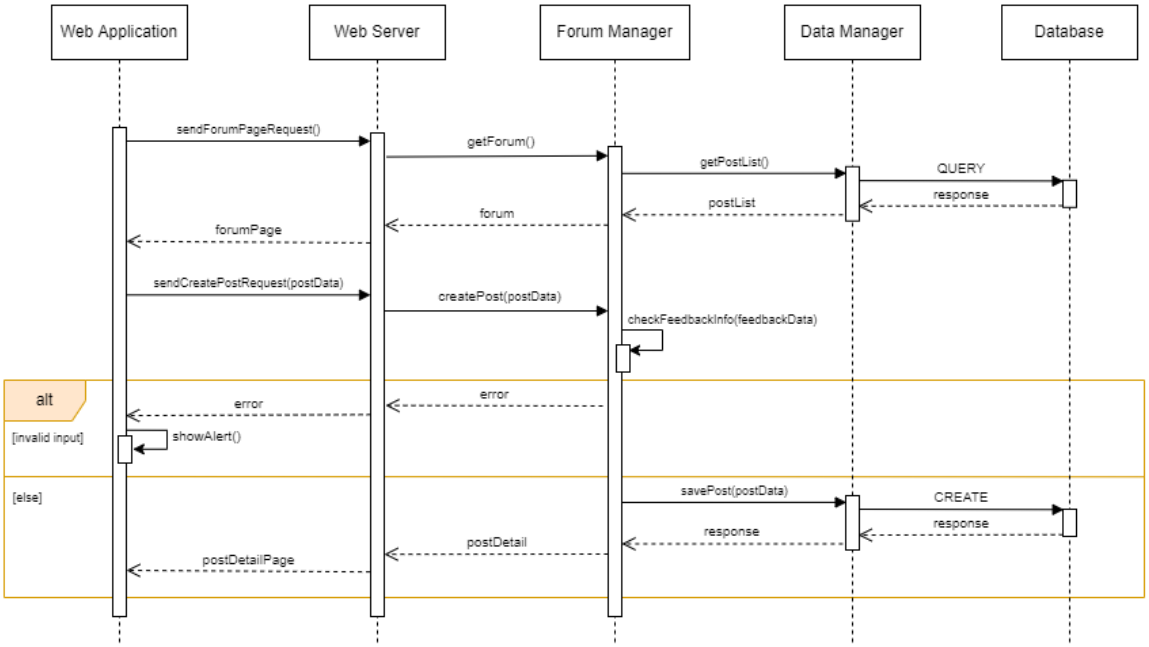
The following diagram represents workflow that Farmer Gives Feedback in DREAM system



Runtime View Diagram 11: Farmer Gives Feedback

- **Farmer Creates A Post**

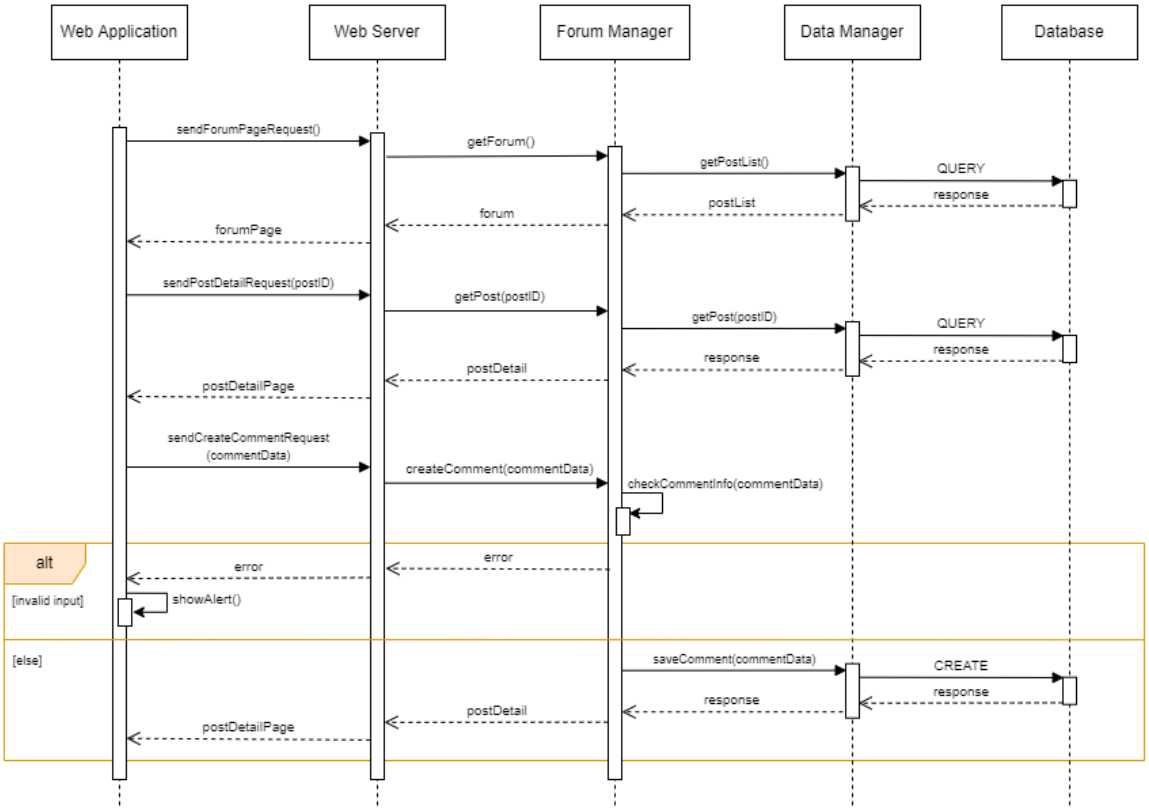
The following diagram represents workflow that Farmer Creates A Post in DREAM system



Runtime View Diagram 12: Farmer Creates A Post

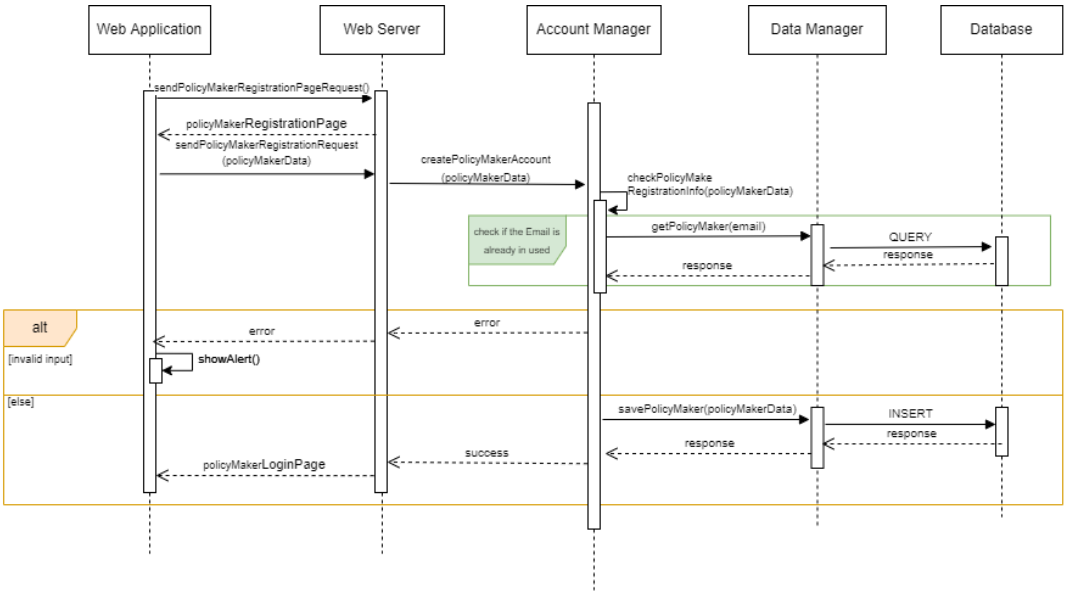
- **Farmer Leaves A Comment**

The following diagram represents workflow that Farmer Leaves A Comment in DREAM system



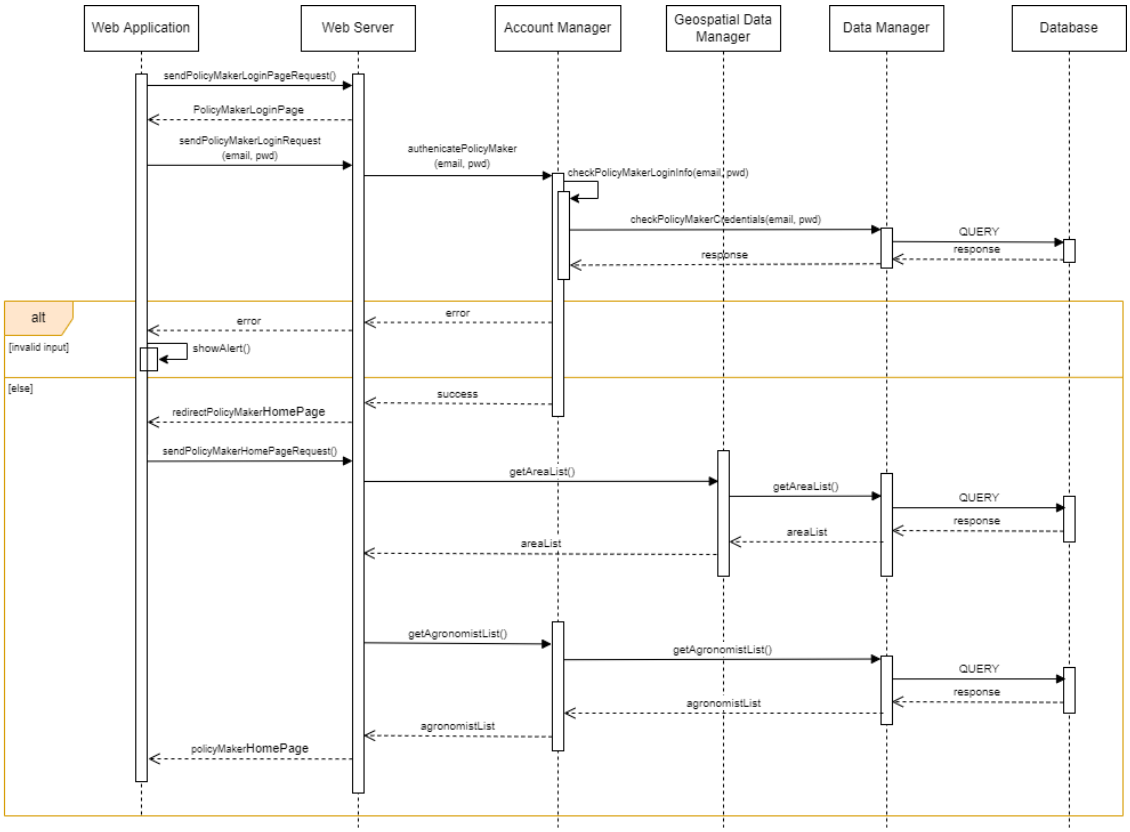
Runtime View Diagram 13: Farmer Leaves A Comment

- **Policy Maker Registers**



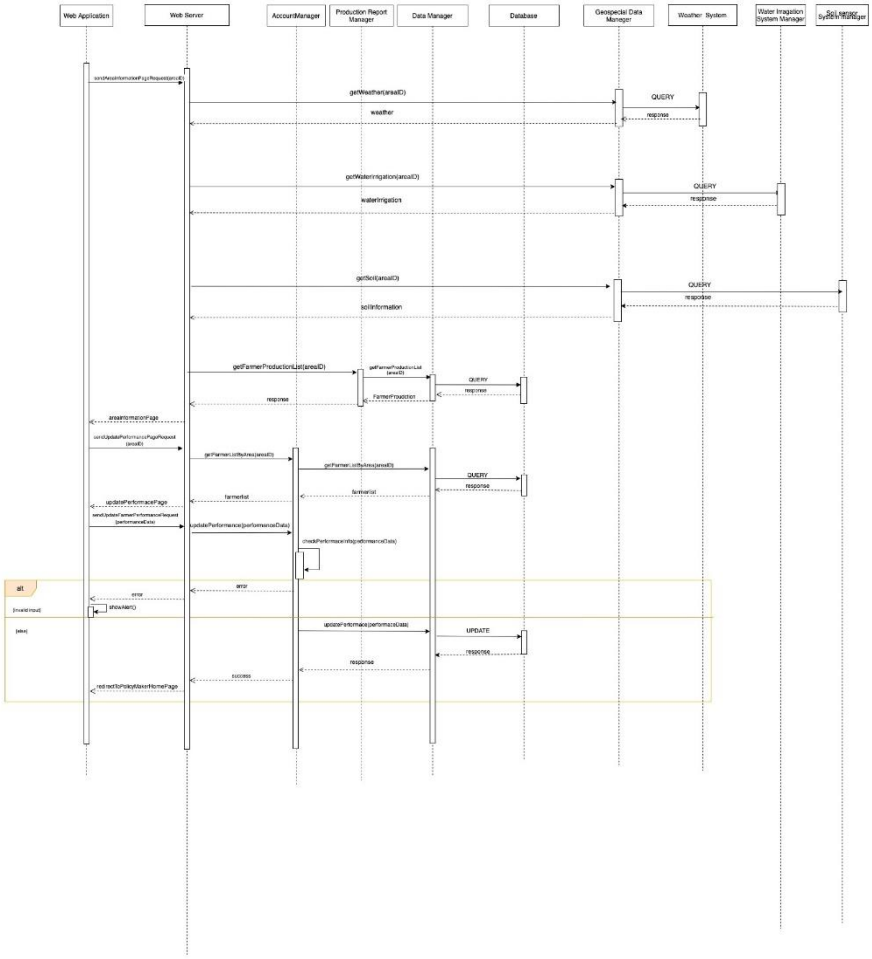
Runtime View Diagram 14: Policy Maker Registers

- **Policy Maker Logins**



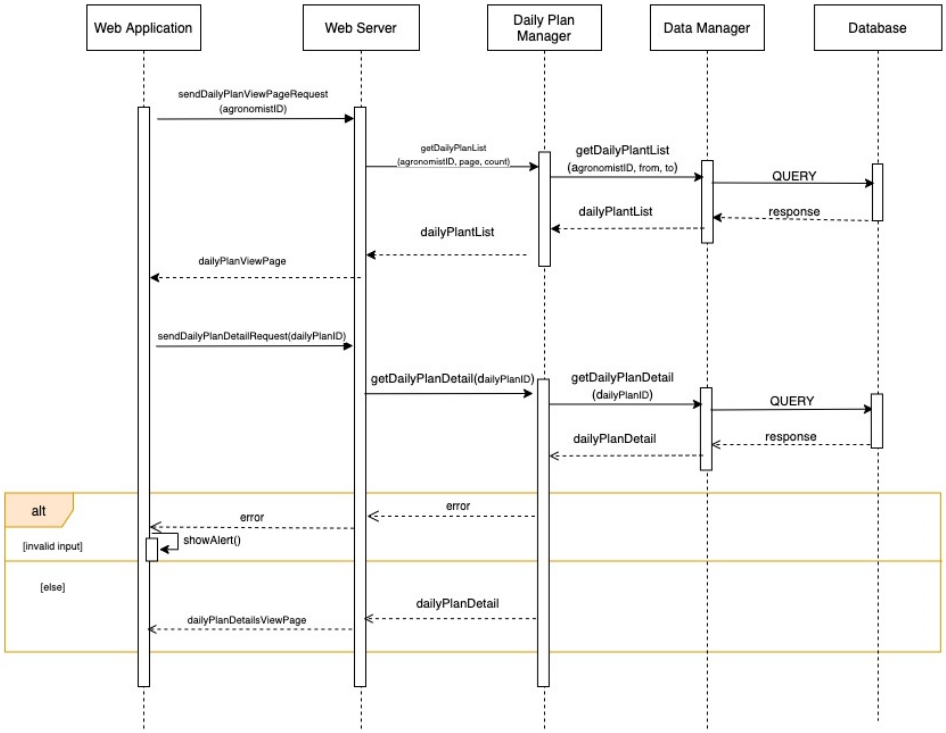
Runtime View Diagram 15: Policy Maker Logins

- **Policy Maker identifies the performance of farmers**



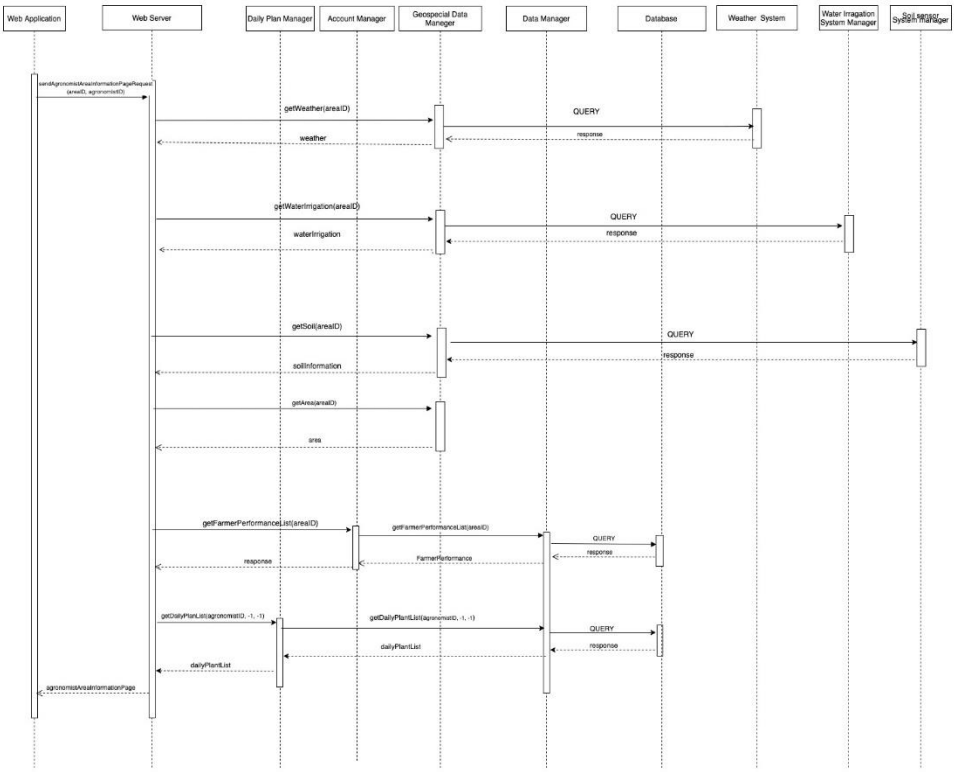
Runtime View Diagram 16: Policy Maker identifies the performance of farmers

- **Policy Maker understand Agronomists' work**



Runtime View Diagram 17: Policy Maker understand Agronomists' work

- **Agronomist visualizes information**



Runtime View Diagram 18: Agronomist visualizes information

2.5 Component interfaces

2.5.1 Account Manager Interfaces

The *Account Manager* component exposes external interfaces that are accessible from the *Web Server*. Its most important methods are listed below.

- **createAgronomistAccount(agronomistData)**
  Create a new account for an Agronomist if the provided data are correct, otherwise, return errors accordingly. The element '*agronomistData*' includes *'username', 'pwd', 'email'*, and *'area'*.

- **authenticateAgronomist(email, pwd)**
  It handles the login request of an Agronomist.

- **getFarmerListByAgronomist(agronomistID)**
  Return a list of farmers who belong to the area the specific agronomist is responsible for.

- **createFarmerAccount(farmerData)**
  Create a new account for a Farmer if the provided data are correct, otherwise, return errors accordingly. The elements '*farmerData'* includes *'username', 'pwd' and 'phoneNumber'*.

- **authenticateFarmer(phonenumber, pwd)**
  It handles the login request of a Farmer.

- **getAgronomistByFarmer(farmerID)**
  Return the agronomist who is responsible for the area where the specific farmer belongs to.

- **createPolicyMakerAccount(policyMakerData)**
  Create a new account for a Policy Maker if the provided data are correct, otherwise, return errors accordingly. The element '*policyMakerData*' includes *'username', 'pwd', and 'email'*.

- **authenticatePolicymaker(email, pwd)**
  It handles the login request of a Policy Maker.

- **getAgronomistList()**
  Return a list of agronomists in Telegana.

- **getFarmerListByArea(areaID)**
  Return a list of farmers who belong to the specific area.

- **updatePerformance(performanceData)**
  It handles the performance update request from a Policy Maker. The element "performanceData" includes a list of farmers with performance.

- **getFarmerPerformanceList(areaID)**
  Return a list of farmer's performance who belong to the specific area.

2.5.2 Production Report Manager Interfaces

The *Production Report Manager* component exposes external interfaces that are accessible from the *Web Server*. Its most important methods are listed below.

- **reportProduction(reportData)**
  Create a production report if the provided data is correct. Otherwise, return errors accordingly. The element *'reportData'* includes '*farmerID', 'type', 'amount', 'acreage', 'startTime'* and *'endTime'.*

- **getFarmerProductionList(areaID)**
  It returns a list of production reports for the given area.

2.5.3 Daily Plan Manager Interfaces

The *Daily Plan Manager* component exposes external interfaces that are accessible from the *Web Server*. Its most important methods are listed below.

- **createDailyPlan(dailyPlanData)**
  Create a daily plan for a specific Agronomist if the provided data is correct. Otherwise, return errors accordingly. The element *'dailyPlanData'* includes *'agronomistID', 'title', 'date', 'farmerlist'* and *'content'*

- **modifyDailyPlan(dailyPlanData)**
  Update a specific daily plan if the provided data is correct. Otherwise, return errors accordingly. The element *'dailyPlanData'* includes *'dailyPlanID', 'title', 'date', 'farmer'* and *'content'.*

- **getDailyPlanDetail(dailyPlanID)**

Return detail information for a specific daily plan.

- **confirmDailyPlan(dailyPlanID, deviation)**
  Change the status of a specific daily plan to be completed and add deviation to it.

- **getDailyPlanList(agronomistID, page, count)**
  It returns a list of daily plans of the given agronomist. The size of the list is computed based on the given parameters, and the list will be sorted by time in descending order (latest entries listed first). If "page" and "count" are both set to "-1", it will return all daily plans for the given agronomist.

2.5.4 Problem Manager Interfaces

The *Problem Manager* component exposes external interfaces that are accessible from the *Web Server*. Its most important methods are listed below.

- **getProblemByFarmer(farmerID)**
  It returns a list of specific farmer's problems, including 'problemID' 'request' 'answer' 'farmer' 'agronomist'. Besides, the returning list will be sorted by time by default.

- **createProblem(problemData)**
  Create a problem with request only if the provided data is correct. Otherwise, return errors accordingly. The element *'problemData'* includes *'request'*, *'famer'* and *'agronomist'*.

- **getProblemByAgro(agronomistID)**
  It returns a list of problems requested by the farmers who belongs to the area that the specific agronomist is responsible for. Besides, the returning list will be sorted by time by default.

- **createAnswer(answerData)**
  Add an answer to a specific problem if the provided data is correct. Otherwise, return errors accordingly. The element *'answerData'* includes *'problemID' and 'answer'*.

- **feedbackProblem(feedbackData)**
  Add feedback to a specific problem if the provided data is correct. Otherwise, return errors accordingly. The element *'feedbackData'* includes *'problemID' and 'feedback'*.

2.5.5 Search Manager Interfaces

The *Search Manager* component exposes external interfaces that are accessible from the *Web Server*. Its most important methods are listed below.

- **searchTypeList()**
  It returns a list of production types existing in external component.
- **search(location, productType)**

  It returns weather forecast for the given location. If specify the product type, it will return the suggestion for the specific production.

2.5.6 Forum Manager Interfaces

The *Forum Manager* component exposes external interfaces that are accessible from the *Web Server*. Its most important methods are listed below.

- **getForum()**
  It returns a forum with a list of posts, including comment number, title, time, post creator of each post. Besides, the returning list will be sorted by time by default.
- **createPost(postData)**
  Create a post for a given Farmer if the provided data is correct. Otherwise, return errors accordingly. The element '*postData*' includes '*title', 'content', 'time*' and '*farmerID*'.
- **getPost(postID)**
  It returns information about a specific post, including 'title', 'content', 'post owner' and 'time'.
- **createComment(commentData)**
  It adds a comment to a specific post. The element "commentData" includes 'postID', 'content', 'time', and 'farmer'

2.5.7 Data Manager Interfaces

The Data Manager component exposes external interfaces that are used by all other Application Server components. Its most important methods are listed below.

- **getAgronomist(email)**
  Return agronomist with sepcific email

- **saveAgronomist(agronomistData)**

It creates a new agronomist in *Database*. The element “agronomistData” includes “username”, “email”, “pwd” and “area”.

- **checkAgronomistCredentials(email, pwd)**

Return the agronomist with specific email and password (identified by “pwd”).

- **getFarmerListByAgronomist(agronomistID)**

Return a list of farmer who are in the area which the specific agronomist is responsible for.

- **saveDailyPlan(dailyPlanData)**

Create a new daily plan in *Database*. The element “dailyPlanData” includes “title”, “date”, “farmer”, “content”, “status” and “agronomist”.

- **modifyDailyPlan(dailyPlanData)**

Modify a given daily plan. The element “dailyPlanData” includes “dailyPlanID”, “title”, “date”, “farmer” and “content”.

- **getDailyPlanDetail(dailyPlanID)**

Return a daily plan specified by dailyPlanID.

- **confirmDailyPlan(dailyPlanID, devaition)**

Set the status of specific dailyplan to “completed” and update the deviation accordingly.

- **getFarmer(phoneNumber)**

Return farmer with specific phone number.

- **saveFarmer(farmerData)**

It creates a new farmer in Database. The element “farmerData” includes “username”, “phonenumber”, and “pwd”.

- **checkFarmerCredentials(phoneNubmer, pwd)**

Return the famer with specific phone number and password (identified by “pwd”).

- **saveReport(reportData)**

Create a new production report in Database. The element “reportData” includes “type”, “amount”, “starttime”, “endtime”, “acreage” and “farmer”.

- **getProblemListByFarmer(farmerID)**

Return a list of problems for a specific farmer.

- **getAgronomistByFarmer(farmerID)**

Return the agronomist who is responsible for the area where the specific farmer belongs to.

- **saveProblem(problemData)**

Create a new problem in Database. The element “problemData” includes “request”, “farmer” and “agronomist”.

- **getProblemListByAgro(agronomistID)**

Return a list of problems associated to the specific agronomist.

- **saveAnswer(answerData)**

Update the answer for a specific problem. The element “answerData” includes “problemID” and “answer”.

- **saveFeedback(feedbackData)**

Update the feedback for a specific problem. The element “answerData” includes “problemID” and “feedback”.

- **getPostList()**

Return a list of posts.

- **savePost(postData)**

Create a new post in Database. The element “postData” includes “title”, “content”, “time” and “farmer”.

- **getPostDetail(postID)**

Return a specific post with associated comments.

- **saveComment(commentData)**

Create a new comment for a specific post in Database. The element “commentData” includes “content”, “time”, “farmer” and “postID”.

- **getPolicyMaker(email)**

Return a policy maker with specific email.

- **savePolicyMaker(policyMakerData)**

It creates a new policy maker in Database. The element “policyMakerData” includes “username”, “email” and “pwd”.

- **checkPolicyMakerCredentials(email, pwd)**

Return the policy maker with specific email and password (identified by “pwd”)

- **getAreaList()**

Return a list of area information for Telegana.

- **getAgronomistList()**
Return a list of agronomists in Telegana.

- **getFarmerProductionList(areaID)**
Return a list of farmer production information for specific area.

- **getFarmerListByArea(areaID)**
Return a list of farmer information in specific area.

- **updatePerformance(performanceData)**
Update a list of farmer performance information. The element “performanceData” includes a list of mapping between farmers and performance.

- **getDailyPlanList(agronomistID)**
Return a specific list of daily plan depending on agronomist.

2.5.8 Geospatial Data Manager Interfaces

The *Geospatial Data Manager* component provides various APIs which allow access to external data. Its most important methods are listed below.

- **getSoil(areaID)**
  Return the data of soil humidity for a specific area.

- **getWeather(areaID)**
  Return the weather for a specific area.

- **getWeatherByLocation(location)**
  It transfers the location to the area that the location belongs to and return the weather in that area.

- **getWaterIrrigation(areaID)**
  Return the data of water usage for a specific area.

- **getTypeInfo(productType)**
  Return the product suggestion for a specific product type.

- **getTypeList()**
  Return a list of product type supported by *Agricultural Product Reference System*.

- **getAreaList()**
  Return a list of areas in Telegana.

- **getArea(areaID)**
  Return a specific area in Telegana.

2.5.9 Geolocation API

The Geolocation API of Google Maps Platform is an external interface that is used by the Web Browser of the user to get their location.

- **getGeolocation(InternetInfo)**
  Send Internet information, including IP address and WIFI, to Google Maps Platform. It returns location of web application according to Google Data Base.

2.5.10 Web Server Interfaces

The Web Server component exposes an external interface that is used by the Web Browser of the users to access the Web Application.

- **sendAgronomistRegistrationPageRequest()**
  It handles opening Agronomist registration page request. And it returns Agronomist Registration Page.

- **sendAgronomistRegistrationRequest(agronomistData)**
  It handles the Agronomist's registration request. The element '*agronomistData*' includes *'username', 'pwd', 'email'*, and *'area'*. It redirects him to either agronomist login page or an error page depending on the outcome of the request.

- **sendAgronomistLoginPageRequest()**
  It handles opening Agronomist login page request. And it returns Agronomist Login Page.

- **sendAgronomistLoginRequest(email, pwd)**
  It handles the Agronomist's login request, including agronomist's '*email'* and '*password'*. It redirects him to agronomist home page or an error page depending on the outcome of the request.

- **sendDailyPlanCreationPageRequest()**

It handles opening Daily Plan Creation Page request. And it returns Daily Plan Creation Page.

- **sendCreateDailyPlanRequest(dailyPlanData)**
  It handles the Creating Daily Plan request. The element *'dailyPlanData'* includes *'agronomistID'*, *'title'*, *'date'*, *'farmer'* and *'content'*. It redirects him to agronomist home page or an error page depending on the outcome of the request.

- **sendDailyPlanModificationPageRequest(dailyPlanID, agronomistID)**
  It handles opening a specific daily plan modification Page request. And it returns Daily Plan Modification Page depending on dailyPlanID.

- **sendModifyDailyPlanRequest(dailyPlanData)**
  It handles the modifying Daily Plan request. The element *'dailyPlanData'* includes *'agronomistID'*, *'title'*, *'date'*, *'farmer'* and *'content'*. It redirects him to agronomist home page or an error page depending on the outcome of the request.

- **sendDailyPlanConfirmationPageRequest(dailyPlanID)**
  It handles opening Daily Plan Confirmation Page request. And it returns a specific Daily Plan Confirmation Page depending on dailyPlanID.

- **sendFarmerRegistrationPageRequest()**
  It handles opening farmer registration page request. And it returns Farmer Registration Page.

- **sendFarmerRegistrationRequest(farmerData)**
  It handles the farmer's registration request. The element '*farmerData*' includes *'username'*, *'pwd'*, *'email'*, and *'farm'*. It redirects him to either farmer login page or an error page depending on the outcome of the request.

- **sendFarmerLoginPageRequest()**
  It handles opening farmer login page request. And it returns Farmer Login Page.

- **sendFarmerLoginRequest(phoneNumber, pwd)**
  It handles the farmer's login request, including farmer's '*phoneNumber'* and '*password*'. It redirects him to farmer home page or an error page depending on the outcome of the request.

- **sendSearchPageRequest()**
  It handles opening farmer's search page request. And it returns Search Page.

- **sendSearchRequest(location, productType)**

It handles the farmer's search request, including '*location'* and '*productType'*. It redirects him to farmer home page or an error page depending on the outcome of the request.

- **sendReportPageRequest()**
  It handles opening farmer's report page request. And it returns Search Page.

- **sendReportRequest(reportData)**
  It handles the farmer's report request, including '*type'* , *'amount', 'starttime',* 'endtime' 'acreage' and '*farmer'*. It redirects him to report page or an error page depending on the outcome of the request.

- **sendRequestPageRequest(farmerID)**
  It handles opening farmer's request page request. And it returns a personal Request Page denpending on farmerID.

- **sendNewRequestPageRequest(farmerID)**
  It handles opening farmer's new request page request. And it returns a personal New Request Page denpending on farmerID.

- **sendCreateProblemRequest(problemData)**
  It handles opening farmer's creating a new problem request, including 'request', 'requestTime' and 'farmer'. It redirects him to request page or an error page depending on the outcome of the request.

- **sendNotificationRequest(agronomistID)**
  It handles opening agronomist's notification page request. And it returns a personal Answer to Request Page denpending on agronomistID.

- **sendProblemAnwserRequest(problemData)**
  It handles opening agronomist's answer to problem request, including 'problemID', 'answerTime', 'answer', 'farmer' and 'agronomist'. It redirects him to request page or an error page depending on the outcome of the request.

- **sendFeedbackRequest(feedbackData)**
  It handles opening farmer's feedback request, including 'problemID', 'feedbackTime', 'feedback', 'farmer' and 'agronomist'. It redirects him to request page or an error page depending on the outcome of the request.

- **sendForumPageRequest()**
  It handles opening forum's post page request. It returns forum page denpending and posts are sorted by time by default.

- **sendCreatePostRequest(postData)**

It handles sending creating post request, including 'title', 'content', 'time' and 'farmer'. It redirects him to post detail page depending on postID or an error page depending on the outcome of the request.

- **sendPostDetailPageRequest(postID)**
  It handles opening forum's post page request. It returns forum page denpending and posts are sorted by time by default.

- **sendCeateCommentRequest(commentData)**
  It handles sending creating comment request, including 'post', 'content', 'time' and 'farmer'. It redirects him to post detail page depending on postID or an error page depending on the outcome of the request.

- **sendPolicyMakerRegistrationPageRequest()**
  It handles opening policy maker registration page request. And it returns policy maker registration page.

- **sendPolicyMakerRegistrationRequest(policyMakerData)**
  It handles the policy maker's registration request. The element '*policyMakerData*' includes *'username', 'pwd' and 'email'*. It redirects him to either policy maker login page or an error page depending on the outcome of the request.

- **sendPolicyMakerLoginPageRequest()**
  It handles opening policy maker login page request. It returns policy maker login Page.

- **sendPolicyMakerLoginRequest(email, pwd)**
  It handles the policy maker login request, including policy maker's '*email'* and '*password'*. It redirects him to policy maker home page or an error page depending on the outcome of the request.

- **sendPolicyMakerHomePageRequest()**
  It handles opening policy maker home page request. It returns policy maker home page, including area and agronomist list.

- **sendAreaInformationPageRequest(areaID)**
  It handles opening policy maker's area information page request. It returns a specific area information page depending on areaID.

- **sendUpdatePerformancePageRequest(areaID)**
  It handles opening update performance page request. It returns policy maker home page, including area and agronomist list.

- **sendUpdateFarmerPerformanceRequest(performanceData)**

It handles the policy maker update farmer performance request, including '*performance*' and 'farmer'. It redirects him to policy maker home page or an error page depending on the outcome of the request.

- **sendDailyPlanViewPageRequest(agronomistID)**
  It handles opening daily plan view page request. It returns daily plan view page depending on agronomistID.

- **sendDailyPlanDetailRequest(dailyPlanID)**
  It handles opening daily plan detail page request. It returns daily plan detail page depending on dailyPlanID.

- **sendAgronomistAreaInformationPageRequest(areaID, agronomistID)**
  It handles opening agronomist's area information page request. It returns agronomist area information page.

2.6 Selected architectural styles and patterns

2.6.1 Four tiers client-server

The System adopts the most common architectural pattern for web applications: four-tier client-server architecture. As shown above, the architecture consists of the following four layers:

- **Client Tier** components run on the client machine and the client could be a Web client or an application client. When it was a Web client, it consisted of dynamic Web pages generated by Web components in **Web Tier**, and a Web browser which is used to render the pages.
  - **Thin Client**: it does not do heavyweight operations such as query database or execute complex business logic. Such operations are off-loaded to other tiers.
- **Web Tier** components are responsible for handling the requests from the **Client Tier**, generating dynamic Web pages based on the interface call results of the **Business Tier**. On the other hand, it needs to maintain the state of the sessions.
- **Business Tier** encapsulates the main business logic rules. It interacts with the **Data Tier**, fetches data and processes it if necessary.
- **Data Tier** or **Enterprise information system (EIS) Tier** contains the database server which is accessed by the components on the **Business Tier**.

This architecture is used to provide maintainability, flexibility and scalability. It allows to decouple the complexity of the System and enable parallel development. If a developer wants

to modify the business logic, he/she only needs to modify the components in business tier without disrupting the entire application.

2.6.2 stateless components

The Application Server component don't contain any status or information in the requests but maintain all the information needed for the service provision, and this property helps the System improve the performance by removing the server load caused by retaining session information.

2.7 Other design decisions

2.7.1 Relational Database

The data management of the system will be handled by an Relational DBMS. Indeed, an relational database is transactional and the so-called ACID properties are guaranteed. No partial execution is allowed, every state is consistent after a transaction execution and each transaction is isolated. Even in case of failures, changes in the database persist.

2.7.2 Load static data when starting the web application

For decreasing the interactions between web components and database, area data will be loaded by once because the size of data is small and would not be changed very often. If there are any operations need area information on runtime, the Application Server will find them in the loaded data.

2.7.3 Password Storing and User Authentication

The Dream needs to acquire a personal identifier and a password for guaranteeing and login to the system. For Policy Maker, the id coincides with the email address entered the system when they register. For Farmer, the Id coincides with the phone number entered during registration. For an Agronomist, the id corresponds to the email address entered.

for each user, the password is first hashed using SHA512 algorithm and stored in the Database, and there is no password stored in the database without text.

3. USER INTERFACE DESIGN

All the interfaces have already been presented in the RASD. Therefore, in the next sections, flow graphs of the application are provided.

In the following graphs, rectangles represent the screens, referenced by figure number in the RASD, diamonds represent conditions, and arrows represent buttons or decisions. And the information in the "[ ]" reprensent the operations.

3.1 Web Application Flow Graph

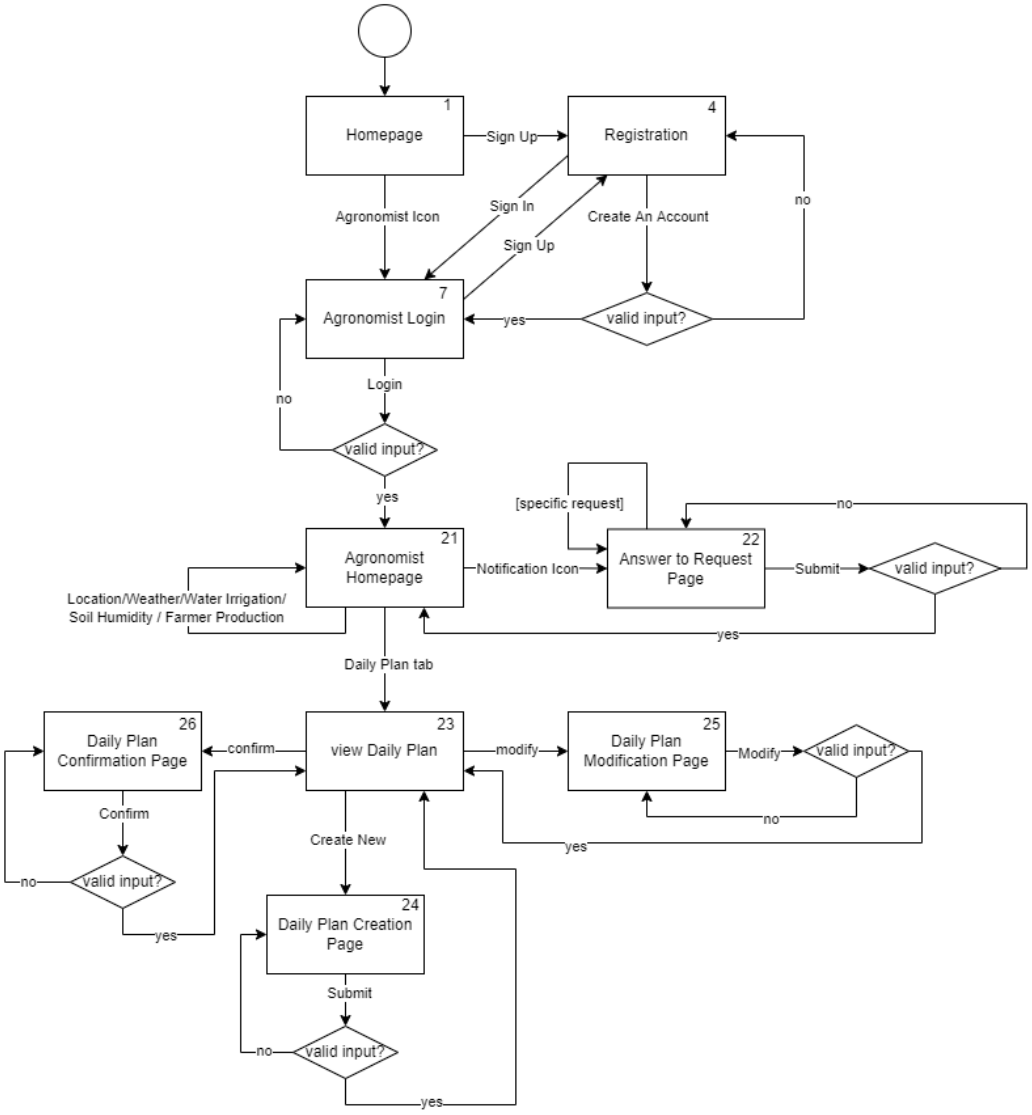


Diagram 1: Web application flow for Agronomist

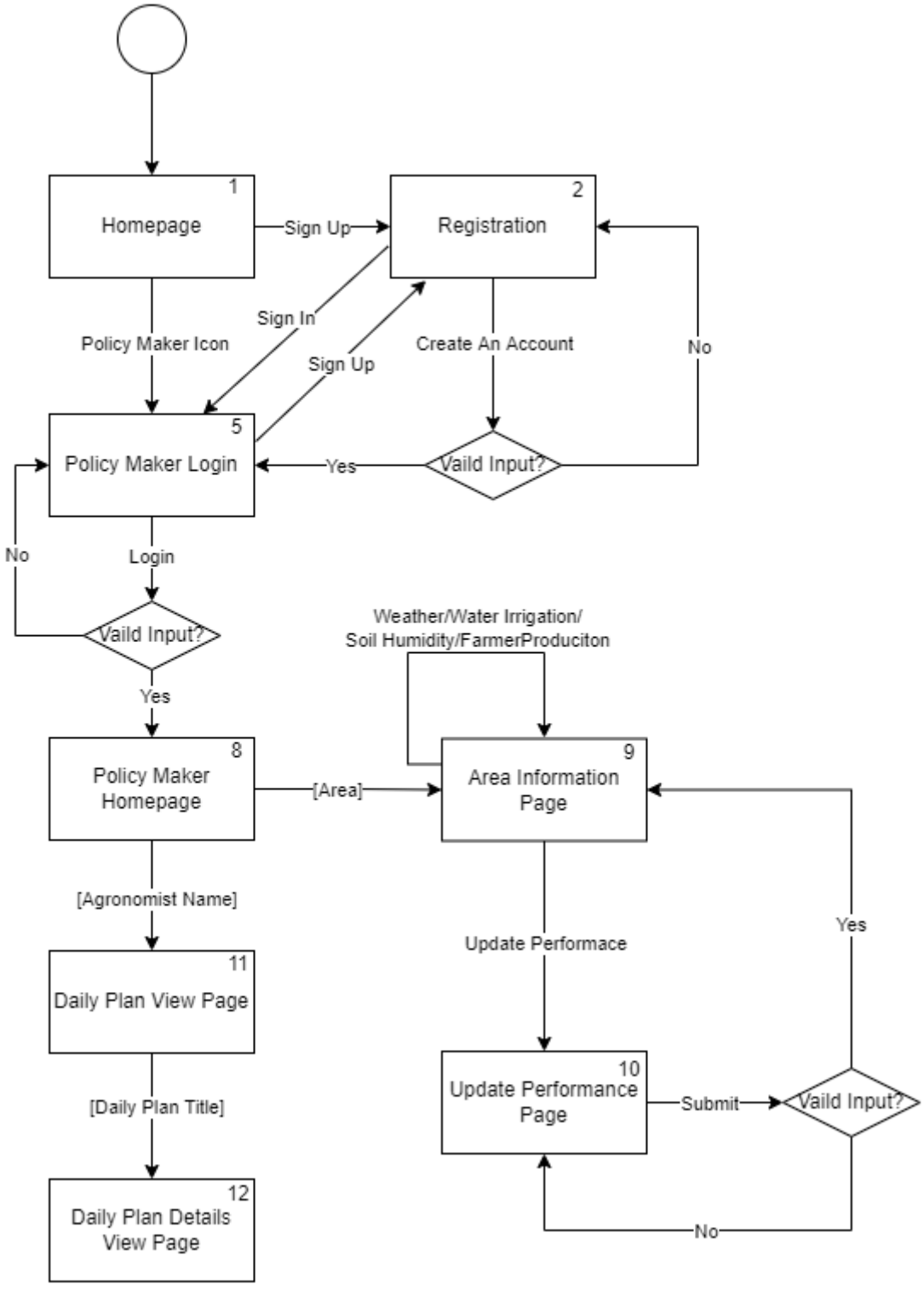


Diagram 2: Web application flow for Policy Maker

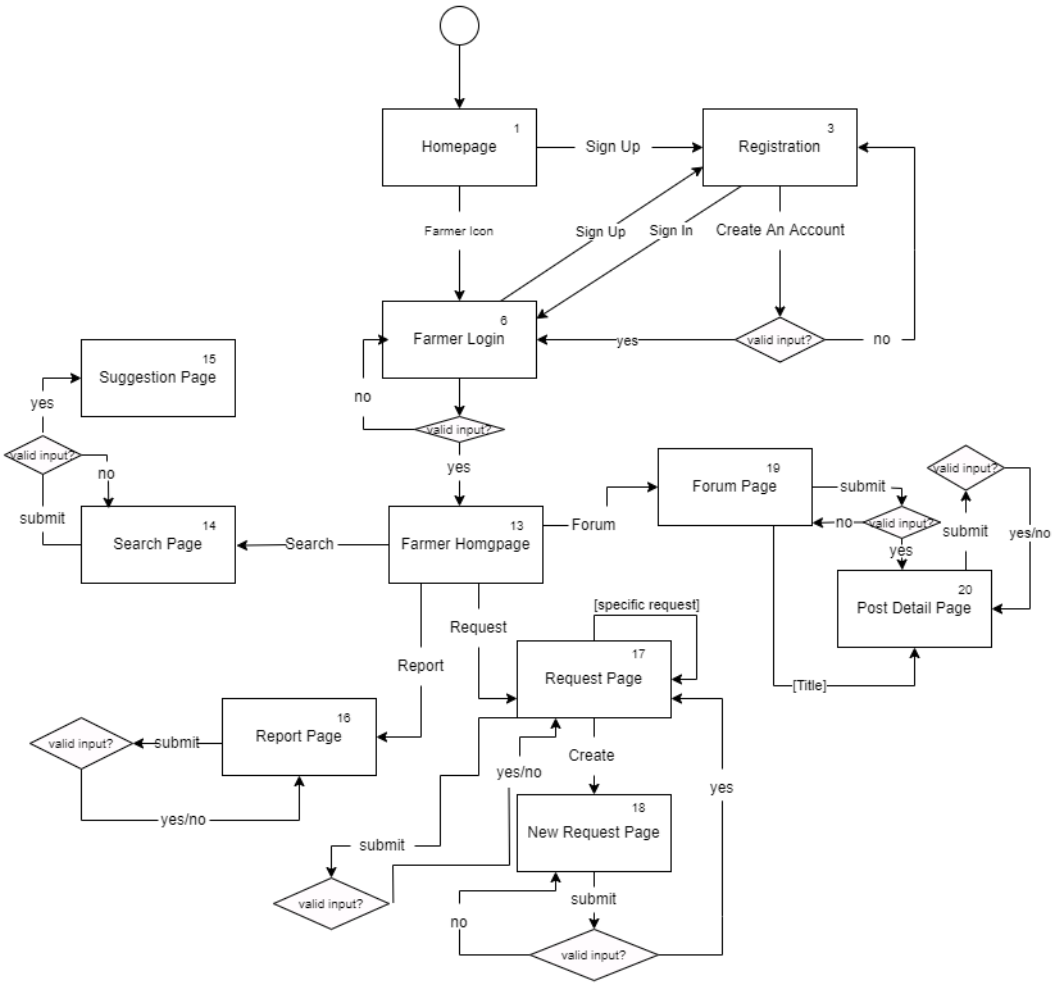


Diagram 3: Web application flow for Farmer

4. REQUIREMENTS TRACEABILITY

5. IMPLEMENTATION, INTEGRATION AND TEST PLAN

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

5.1 Development Process

// dependency diagram: https://www.uml-diagrams.org/dependency.html

5.2 Implementation Plan

//

5.3 Integration Sequence

5.4 System Testing

Once all system components have been integrated, System Testing begins. This process aims at verifying functional and non-functional requirements and must take place in a testing environment which is as close as possible to the production environment. Specifically, DREAM System will be subjected to the following tests:

- Functional testing.
- Performance testing.
- Load testing.
- Stress testing.

6. EFFORT SPENT

TimeLine	Comment	Xu	Zhang	Hu
27/12/2021	Overview and arragement	3h		
28/12/2021	section 1	0.5h		
30/12/2021	arrangement section2	1h	1h	2h
31/12/2021-05/01/2022	Section2	8h	12h	4h
06/01/2022	Section2 review part1	3h		
07/01/2022	Section2 review part2	3h		
07/01/2022	Section3&4	4h	4h	5h
08/01/2022	Section2 review part3	4h	4h	6h
08/01/2022	Section5	3h	1h	1h

7. REFERENCES

- E. Di Nitto. Lecture Slides. Politecnico di Milano
- E. Di Nitto. Project Assignment AY 2021-2022. Politecnico di Milano