POLITECNICO
MILANO 1863

DREAM - Data-dRiven PrEdictive FArMing in Telangana

Careddu Gianmario
La Greca Michele Carlo
Zoccheddu Sara

Prof. Elisabetta Di Nitto

Design Document

Version 1.0

8th January 2022

Deliverable: DD

Title: Design Document

Authors: Careddu Gianmario
La Greca Michele Carlo
Zoccheddu Sara

Version: 1.0

Date: 8th January 2022

Download page: https://github.com/gccianmario/Zoccheddu-LaGreca-Careddu

---

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Purpose

DREAM (Data-dRiven prEdictive fArMing) is an elaboration of an ongoing initiative promoted by Telangana's government which is the 11th largest state in India, with about 35 million residents living in 33 districts.
Telangana's economy, as the Indian economy, is mainly based on agriculture for the majority conducted by smallholder farmers with less than 2 hectares of farmland.
More than a fifth of the smallholder farm households are below poverty and in the following years the situation is going to be worse due to climate change. A 4%-26% loss in net farm income is expected, together with an increase in the food demand as a consequence of the forecasted population growth.
What Telangana wants is to prevent hitches on their food supply chain that are likely to occur during critical situations (COVID-19 pandemic is a clear example) by adopting data-driven policy making and optimizing as much as possible the productivity of their fragmented agricultural production.
In order to achieve its goal the Telangana's government needs to assemble as much meaningful data as possible and to be ready to help farmers in distress with ad-hoc initiatives, possibly making farmers aware about forthcoming issues and teaching them how to tackle them.

1.2. Goals

DREAM is meant to satisfy the needs of farmers, agronomists and policy makers of Telangana, reaching the following 5 goals:

G1: Sharing of knowledge among farmers

G2: Tracking farmers' production

G3: Allowing farmers to get help

G4: Supporting agronomist work

G5: Providing data-driven decision tools to agronomists and policy makers

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Definitions

District: portion of Telangana equivalent to the political district area. There are 33 districts in total.

Area: sub portion of the district. They are decided by the government stakeholders outside the system.

Help request to farmers: text request sent by a farmer to one or possibly many neighbor farmers.

Help request to agronomist: text request sent by a farmer to the agronomist who is responsible for the farmer's area, it is classified using a category.

Help request messages: text messages associated with a specific help request that are exchanged between the receiver and the sender.

Tip request: request sent by a policy maker to a farmer, for asking to write a tip.

Tip request messages: text messages associated with a specific tip request from a policy maker to a farmer.

Post: content that can be published in the forum. It can be of the following types:

- Tip: useful advice. It has no answers.
- Question: question that can be answered by other forum users. The answer can be voted.

Vote: either a like or a dislike received by a tip or an answer.

Score: any votable item in the forum has a score obtained by:
(number of likes) - (number of dislikes)

Star tips: forum's tips highlighted by the policy maker, they are displayed in the relevant information section of farmers in the same area of the tip.

Visit: event of an agronomist visiting a farmer. It has a creation date (timestamp), a proposed date, a time length and a status (scheduled, done, finalized).

Visit plan: it is a calendar where all the visits that an agronomist has to do are saved.

Visit messages: text messages associated with a specific visit that allow the agronomist and the farmer to have a direct contact point in case of need (i.e. the farmer wants to postpone the visit).

Visit-report: predefined questionnaire where the agronomist assesses the farmer's work and the observed situation. It is divided in 3 sections:

- *Basic*: mandatory part of the report containing the farmer's assessment;
- *Problem*: optional part to describe issues that the farmer is facing or has faced;
- *Weather*: optional part used to report adverse weather conditions or events.

Harvest-report: predefined questionnaire where farmers add all the information about the harvest of crops. It is divided in 3 sections:

- *Basic*: mandatory part of the report containing quantitative information and description of the harvest.
- *Problem*: optional part to describe issues related to that specific harvest.
- *Weather*: optional part used to report weather issues related to that specific harvest

Super agronomist: agronomist who has decided to be available to perform visits to farmers in the whole district.

Metric: it is a numerical value computed by the system for every farmer. It is based on features extracted from specific data referring to a specific period (i.e. a metric can be computed every 2 months based on the farmer production and the number of HRs sent). There can be many metrics, and each metric is computed for every farmer.

Key performance indicator: it is a numerical value computed by the system combining one or possibly multiple metrics associated with a farmer. Their objective is to be a performance indicator. High values of the KPI correspond to a good performance, and vice versa.

Well-performing farmer: a farmer with KPI values higher than average.

Under-performing farmer: a farmer with KPI values below than average**.**

Authorization code: alpha-numeric string used by users to register to DREAM. Regarding agronomists and policy makers, this code is given to them by a government admin outside the system. Regarding the farmer, this code corresponds to the VAT number.

Crop Category: each crop is categorized in one of the nine categories:
Cereals, Seeds, Pulses, Oil Seeds, Fruits, Vegetables, Spices, Fodder Crops, Commercial crops [1].

1.3.2. Acronyms

HR: Help Request
TR: Tip Request
FAQ: Frequently Asked Questions
KPI: Key Performance Indicator
VAT: Value Added Tax
CMP: Component
ORM: Object-relational mapping
HTTP: Hypertext Transfer Protocol
API: Application Program Interface
REST: Representational State Transfer

1.3.3. Abbreviations

Auth code: authorization code
HR-message: help request message
TR-message: tip request message

1.4. Revision history

- 8 January 2022, version 1.0
    - First DD implementation

1.5. Document Structure

- **Chapter 1** gives an introduction about the purpose of the document and the goals, with its corresponding specifications such as the definitions and acronyms. Also the scope of the project is described, as well as the world and shared phenomena. Finally, the revision history of the document and the references are also available in this chapter.

- **Chapter 2** describes the architecture of the system. In particular, this section offers an overview of the system, then it deals more in-depth with descriptions of the components of the system and how they are used in the application. Finally, architecture and design decisions are better explained and justified.

- **Chapter 3** shows how the system appears to the users (mockups) using laptop or mobile size screens and how the navigation inside the system is organized.

- **Chapter 4** has the purpose of connecting the RASD and the DD document. It includes a mapping between the requirements and the design components that are used to fulfill them.

- **Chapter 5** provides implementation, integration and testing details.

- **Chapter 6** shows the effort spent by each member of the group.

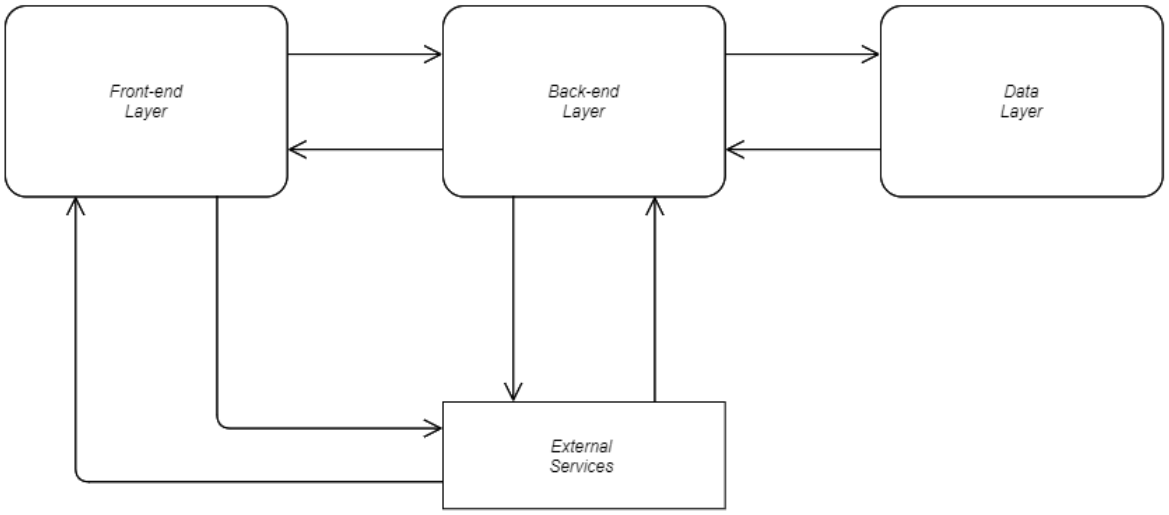
- **Chapter 7** includes the reference documents.

2. Architectural Design

2.1. Overview

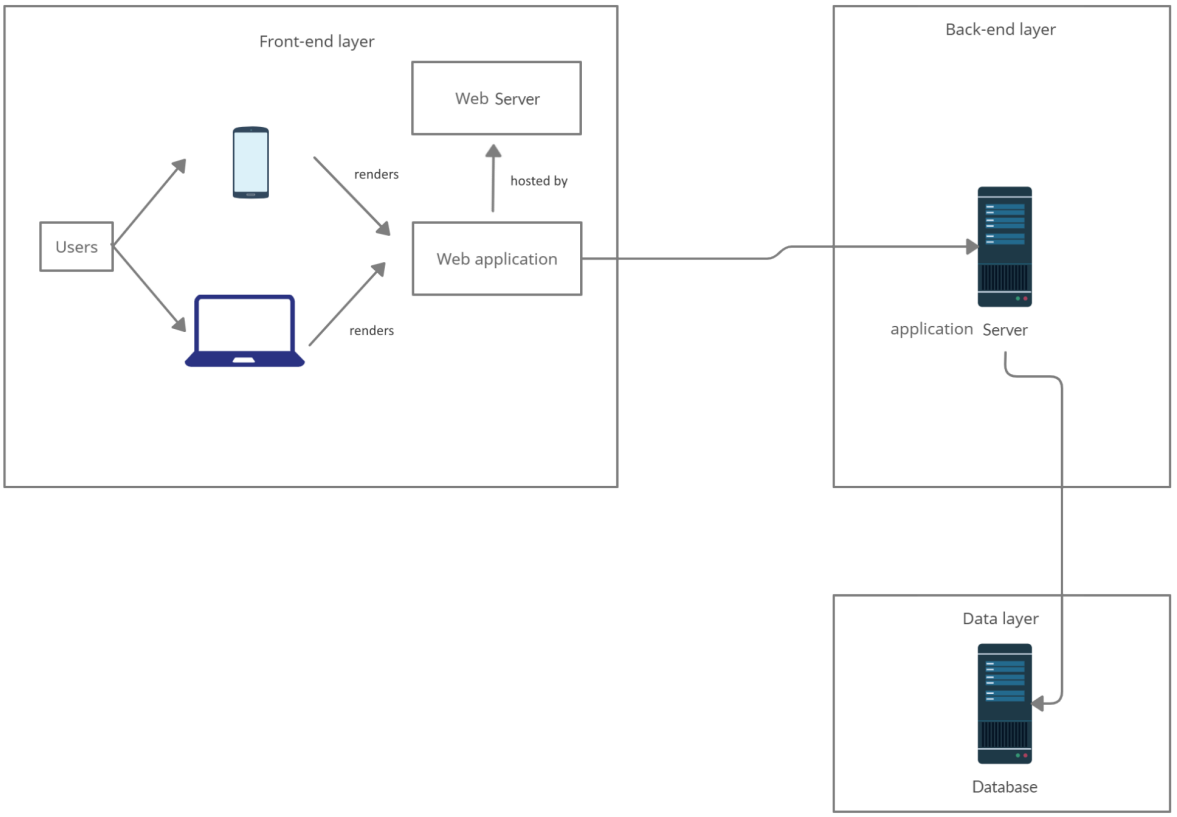
DREAM is a system that, from an architectural point of view, works and follows the client-server paradigm.
The client is defined as a *thin* client, because it is totally dependent on the server, due to the fact that the logic of the system is implemented on it. The client is only responsible for managing the interaction between system and users.
The server is defined as a *fat* server, because it is responsible for the data management and the logic of the system, as well as the interaction with external services.

Below, it is possible to take a look at a high-level architecture of the system, based on three layers: front-end layer, back-end layer, and data layer.



- **Front-end layer:** this layer comprises the UI of the system and the interaction between the user and the system. It communicates with the back-end layer, since all the business logic and data management is there. The front-end communicates also with external services, such as location APIs and weather APIs.
- **Back-end layer:** in short, this layer comprises the application logic, it is in the middle between the frontend and the data layer allowing the flow of data between them with a specific logic. It also communicates with external services, such as weather and government APIs. Another important function of this layer is the interaction with the data layer through DB APIs, allowing the data management, using an ORM technique.
- **Data layer:** this layer comprises a way to store all the data of the system inside a DB. It provides APIs to allow the back-end layer to retrieve and store data.

A more detailed view of each layer is shown below.



In the front-end layer, users can access the system using several devices through a web application, where all the front-end functionalities are located. The web application communicates with a web server, whose function is providing web pages. The web app interacts with the server located in the back-end layer. This communication is handled by accessing REST API endpoints that the back-end provides to the front-end layer. The communication between front-end and back-end is stateless, according to the RESTful standard definitions. The server in the back-end layer communicates with the database located in the data layer. This is made by using ORM techniques for generating and handling database models.

All the components will be described in depth in the following sections.

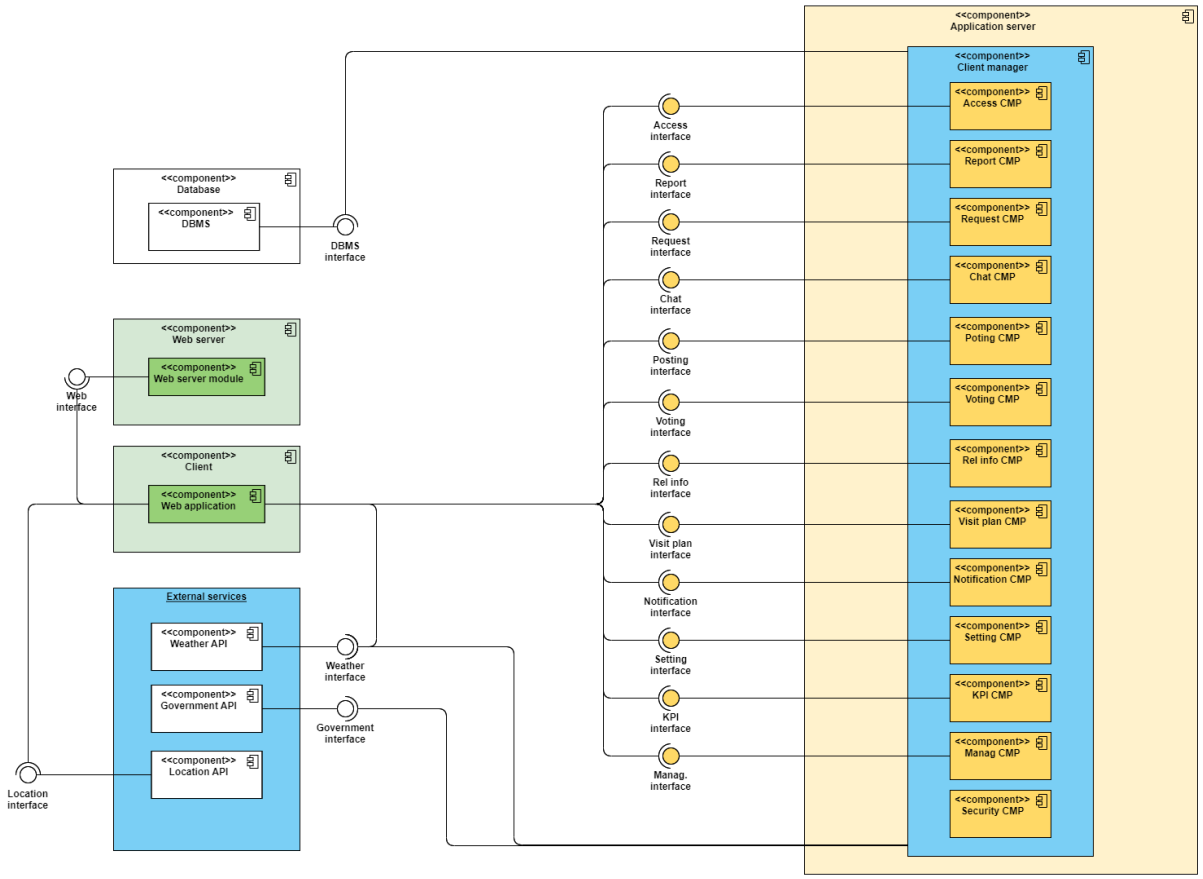
2.2. Component View

Below, the components of the layers in the previous section are more closely examined, as well as the external services. Particularly, each main subsystem (web application, web server, application server, database) of every layer will be described in terms of the subcomponents that they contain, together with the interfaces that are used to connect subcomponents of different subsystems.

It is important to point out some notations, made for the ease of reading.

The term *interface* means the use of several methods to allow the components to communicate with other components; these methods will be implemented in the subsequent sections and in the implementation phase.

The web application will communicate with the interfaces through HTTP requests, using some endpoints provided by the application server. Moreover, all the methods that each interface uses will be located in the controllers which actually implement the interfaces (they will be described in the following sections of the document). In addition to this, the components in the application server communicate with each other through other methods that are inside the components. This aspect is not fully described in this document because it strongly depends on the implementation characteristics of the application server, as well as the technologies that will be used.



2.2.1. External services

- **Weather API**
  It is the component that deals with the meteorological forecasts, a service that provides data to the front-end and back-end layer through an interface.

- **Government API**
  It is the component that deals with sensors deployed on the territory and measuring the humidity of soil and water irrigation systems, which are services that provide data to the back-end layer through an interface.

- **Location API**
  It is the component that provides the precise location of a device, and it is used by the front-end layer through an interface to allow the system to save information regarding the location of a user.

2.2.2. Front-end components

- **Web server**
  It is a server containing all the presentation logic of the application, it is responsible for providing to clients web pages with all the code needed to interact with the backend, render and adapt the pages based on the user's actions and what data is received from the backend, it is important to notice that the web server has not any connection with the backend.

- **Web application**
  It is the component of the front-end layer that is responsible for the interaction between users and the system. The web application is obtained from the web server and executed on the client browser. Through an interface, this component obtains data from the location API component, for providing data about location to users, as well as data from a weather API component, for providing data about meteorological forecasts. Through several interfaces, the web application obtains/provides data from/to the back-end components, which will be described in the following section. Finally, the web application is not directly connected with any interface to the data layer.

2.2.3. Back-end components

- **Access component**
  It is the component of the back-end layer that deals with the sign-up and log-in procedure of the users, in particular it is responsible for the management of access tokens. This component provides an interface to the front-end layer.

- **Report component**
  It is the component of the back-end layer that deals with the filling and uploading of the harvest reports and visit reports, as well as all the automated procedures that derive from them. This component provides an interface to the front-end layer.

- **Request component**
  It is the component of the back-end layer that deals with all the requests sent between users: HR and TR. This component provides an interface to the front-end layer.

- **Chat component**
  It is the component of the back-end layer that deals with the messages exchanged by users: HR-messages, visit messages, TR-messages. This component provides an interface to the front-end layer.

- **Posting component**
  It is the component of the back-end layer that deals with the elements that can be posted inside the forum by the users: tips, questions and answers. This component provides an interface to the front-end layer.

- **Voting component**
  It is the component of the back-end layer that deals with the voting procedure which allows users to modify the score of votable elements in the forum. This component provides an interface to the front-end layer.

- **Relevant info component**
  It is the component of the back-end layer that deals with the relevant information section of the users, handling the visualization of important and helpful information, customized for each user. This component provides an interface to the front-end layer.

- **Visit plan component**
  It is the component of the back-end layer that deals with the visits of the agronomists. It manages the initialization of the visit plan, the insertion, the updating and the deleting of visits, the change of the visit states, the delegation of a visit to a super agronomist, as well as all the automated procedures that derive from them (for example notification of unfilled visit reports) .This component provides an interface to the front-end layer.

- **Notification component**
  It is the component of the back-end layer that deals with all the notifications received by the users. This component provides an interface to the front-end layer.

- **Setting component**
  It is the component of the back-end layer that deals with the settings of the elements: promoting a tip to a star tip, giving availability to be a super agronomist and posting FAQs. This component provides an interface to the front-end layer.

- **Metric and KPI component**
  It is the component of the back-end layer that deals with the metrics and the KPIs. This component retrieves data from the government and weather API component of the external services, obtains data about harvest reports, visit

reports, visits and HR from the data layer, combining this data according to the definition of these metrics and KPI. This component also deals with the routine that periodically updates metrics and KPIs. Finally, this component provides an interface to the front-end layer, allowing users to retrieve metrics and KPIs.

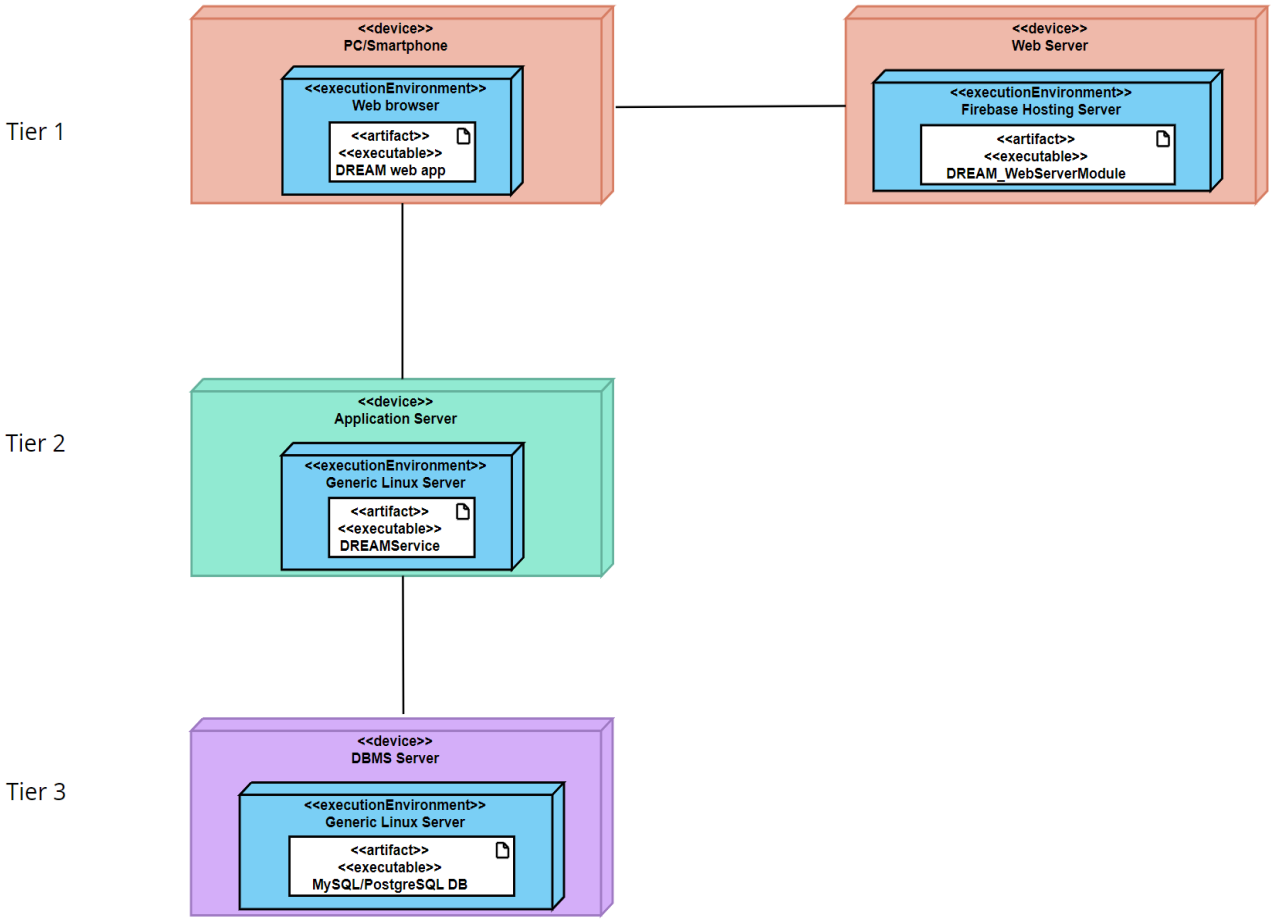
- **Management component**
  It is the component of the back-end layer that deals with the management of the authorization codes for sign-up, the procedures useful for delete items and reset user profiles. This component provides an interface to the front-end layer.

- **Security component**
  It is the component of the back-end layer that deals with all the controls and exceptions that the system must enforce to permit a correct behavior of all the system.

2.2.4. Data components

- **DBMS**
  It is the component of the back-end layer that deals with storage of the data inside the data layer. This component provides an interface to the back-end layer, allowing the management of the data.

2.3. Deployment View



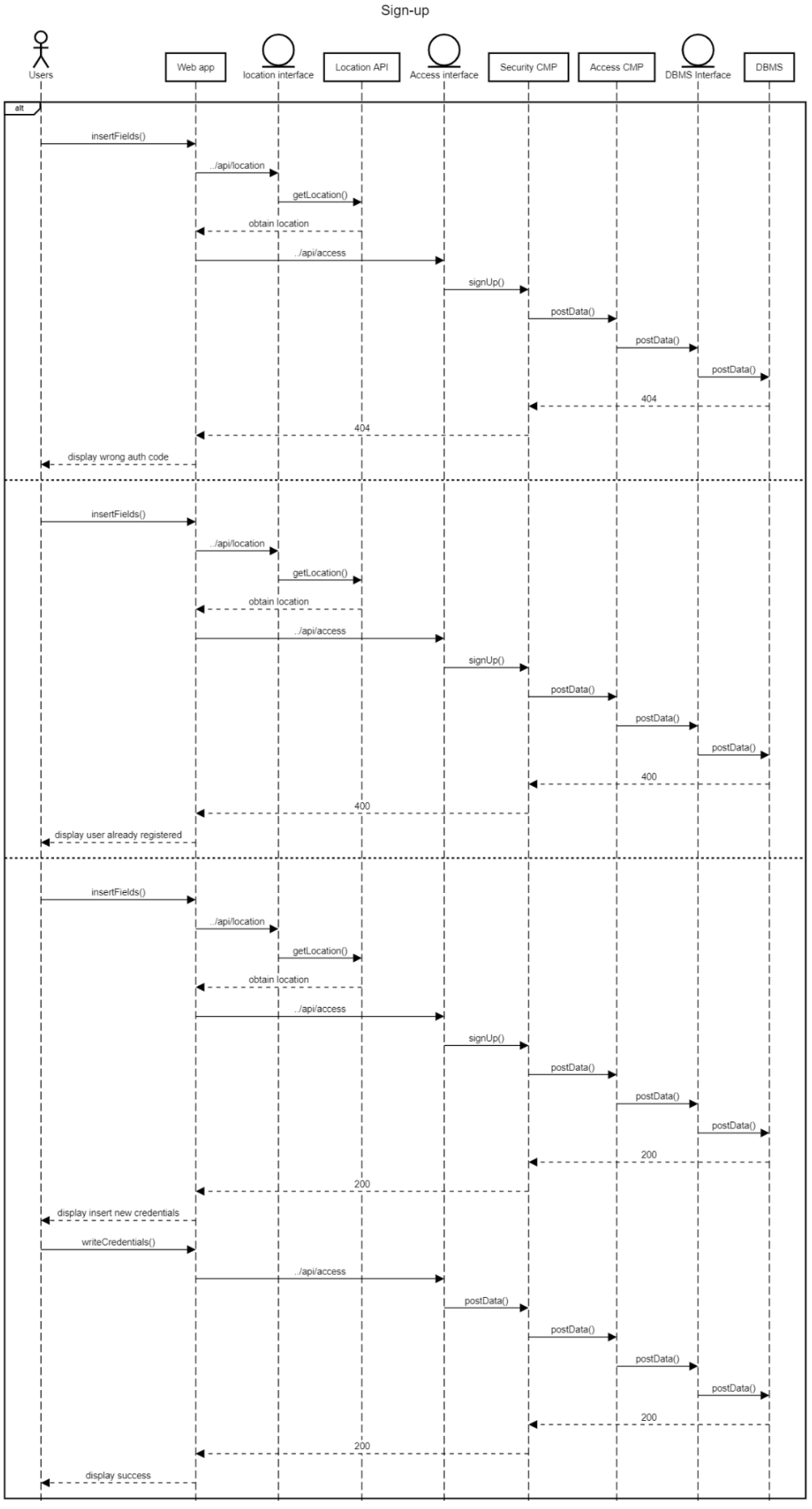
The system is deployed through a three-tier architecture, as shown in the diagram. Each tier has its own infrastructure and corresponds to a specific layer.

1. The **first (presentation) tier** is composed of the client machine (computer or a mobile device) and of the web server hosting the dream web app. The client obtains a bare-bone HTML document with dynamic (Javascript) code ready to be executed on the browser. The web pages provided by the web server are artifacts able to send/receive raw data to/from the application server and present it in the proper way.
2. The **second (logic) tier** is represented by the application servers. It corresponds to the back-end layer. This layer is responsible for all the business logic of the system providing through a RESTful API some endpoints allowing the flow of information between the first and the second tier.
3. The **third (data) tier** is the data tier and it is composed of the DBMS servers where the data is stored. This tier corresponds to the data layer.

2.4. Runtime View

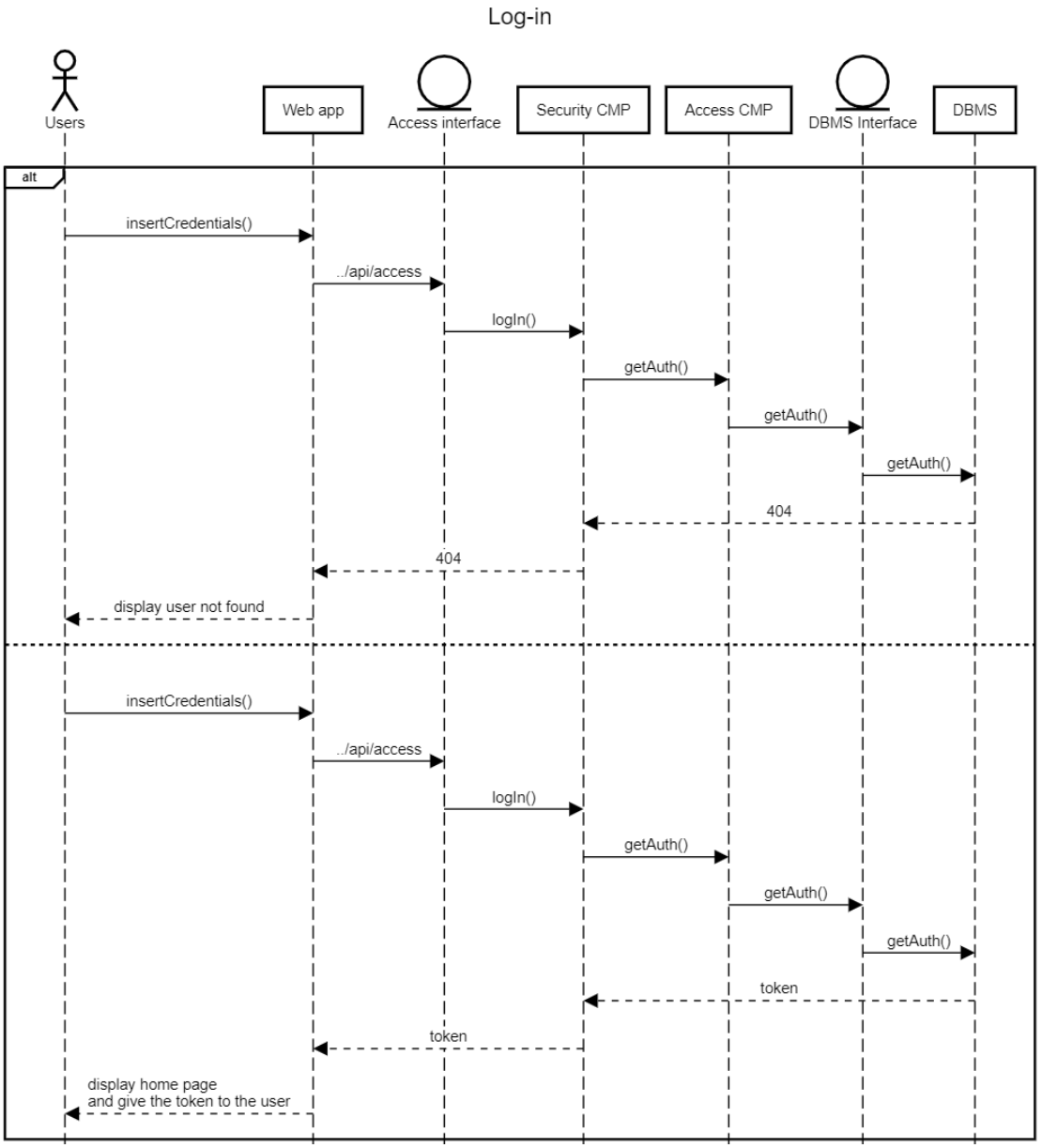
In this section the most important characteristics of the runtime operations of the system will be described, using sequence diagrams.

2.4.1. Sign-up



In this sequence there are the procedures that allow users to sign up. They retrieve a location through the location API, and then, using the access interface, it is possible to post the data in the DBMS. The security component is used to check the validity of the data. The DBMS can return some codes, according to the success/failure of the operation.

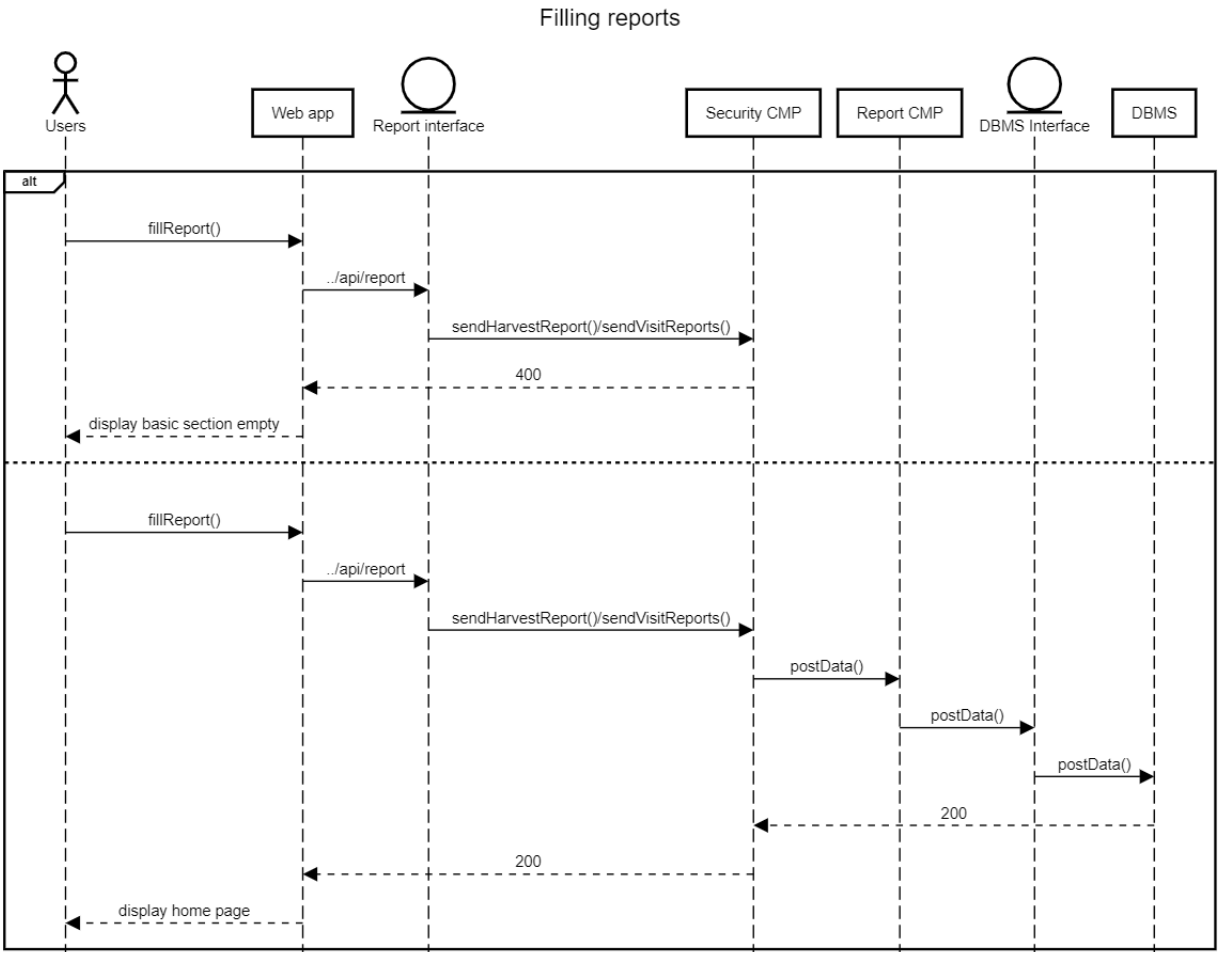
2.4.2. Log-in



This sequence deals with the login procedure of users. Using the access interface, it is possible to post the credentials inserted by the user to the DBMS. The DBMS will return a token that will allow users to prove their identity each time they interact with the system. **This will be done implicitly in every following sequence diagram, by the check of this token in the security component,** any request to the backend that is trying to access data without the correct permission will receive as response a 401 http

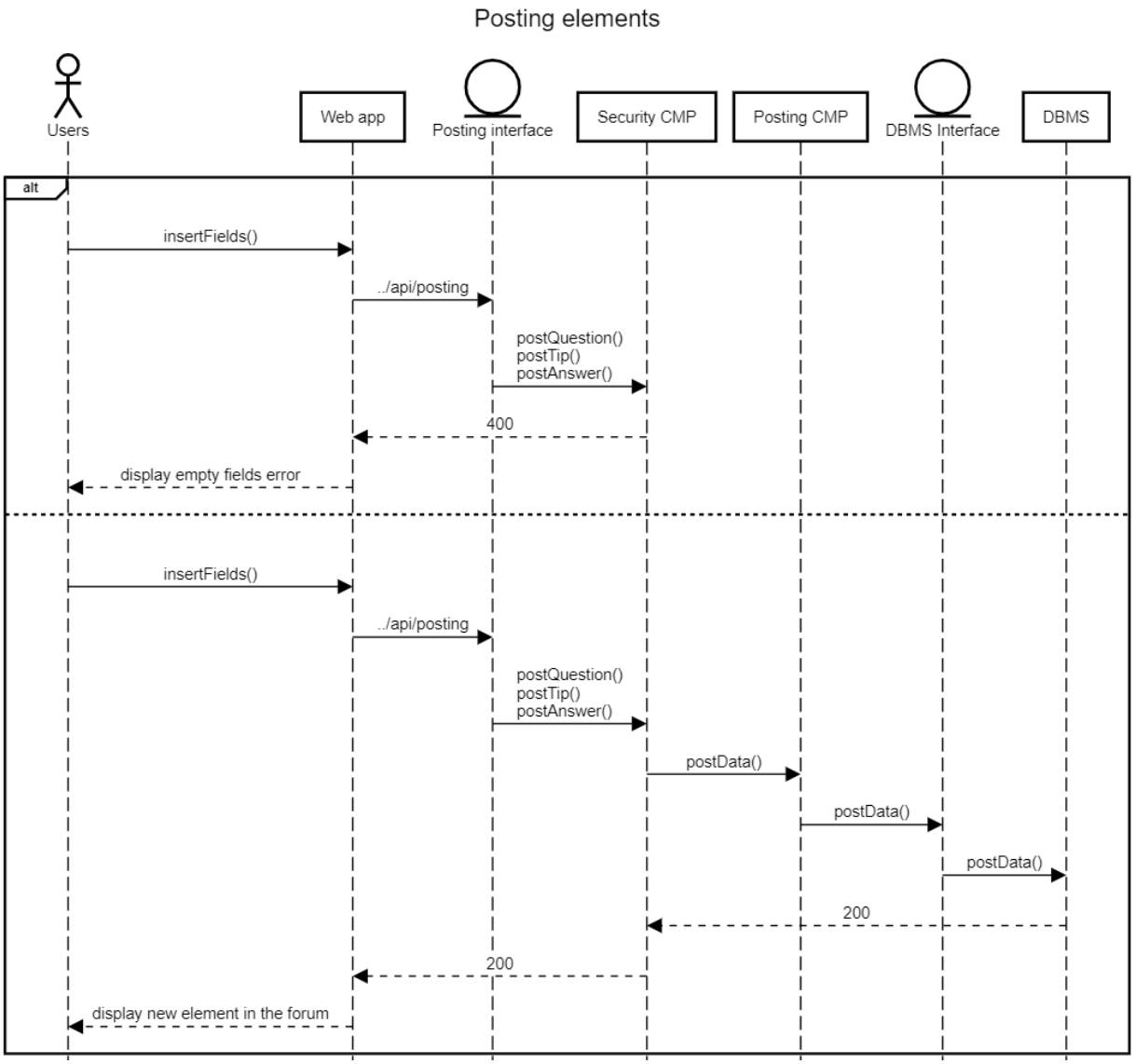
error. As a result, the communication created is stateless, since the application server does not open a session (virtual channel with the user) but everytime a new http request is received, the associated token is analyzed: the fact to possess a valid token makes users logged in. When the user logs out, the token associated with the user is dismissed and no longer accepted.

2.4.3. Filling reports



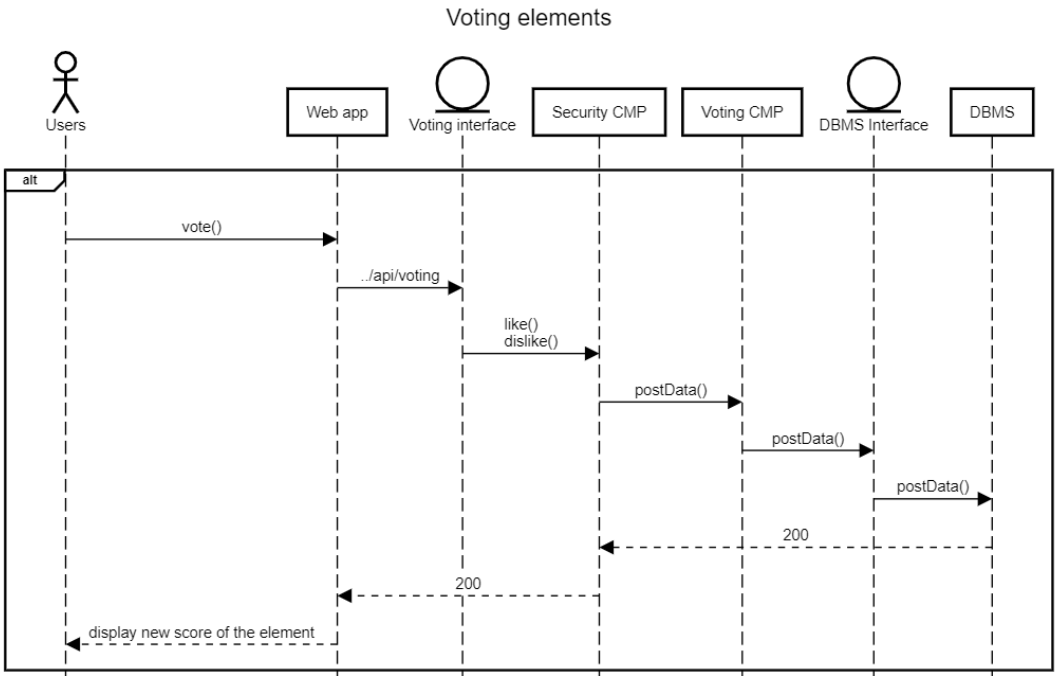
This sequence deals with the uploading of the harvest-reports and visit-reports in the database. Through the report interface, it is possible to post data that will pass firstly through the security component, that checks the correctness of the data. If some mandatory fields are empty, the security component will return a bad request error code 400, otherwise the data will be sent to the DBMS. The DBMS will return some codes depending on the result of the procedure. These codes are fetched by the security component that handles the response to be given to the user.

2.4.4. Posting elements



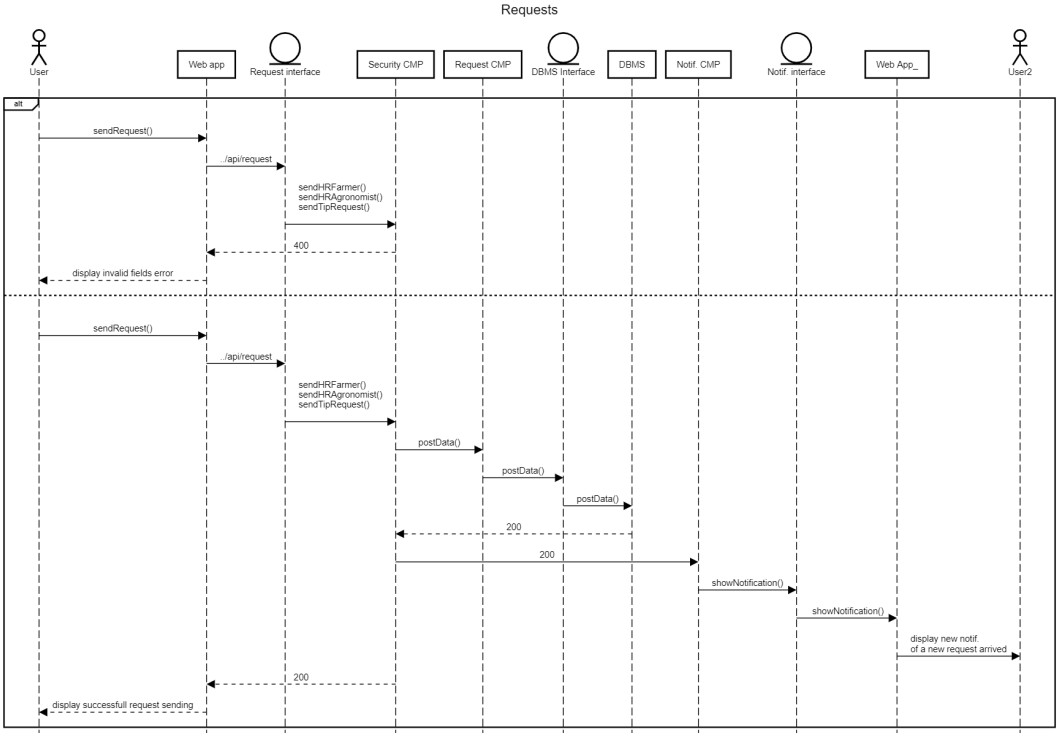
This sequence deals with the process of posting elements (tips, questions, and answers) in the forum. Through the posting interface it is possible to post data that will pass firstly through the security component that checks the correctness of the data. If some mandatory fields are empty, the security component will return a bad request error code 400, otherwise, the data will be sent to the DBMS. The DBMS will return some codes depending on the result of the procedure. These codes are fetched by the security component that handles the response to be given to the user.

2.4.5. Voting elements



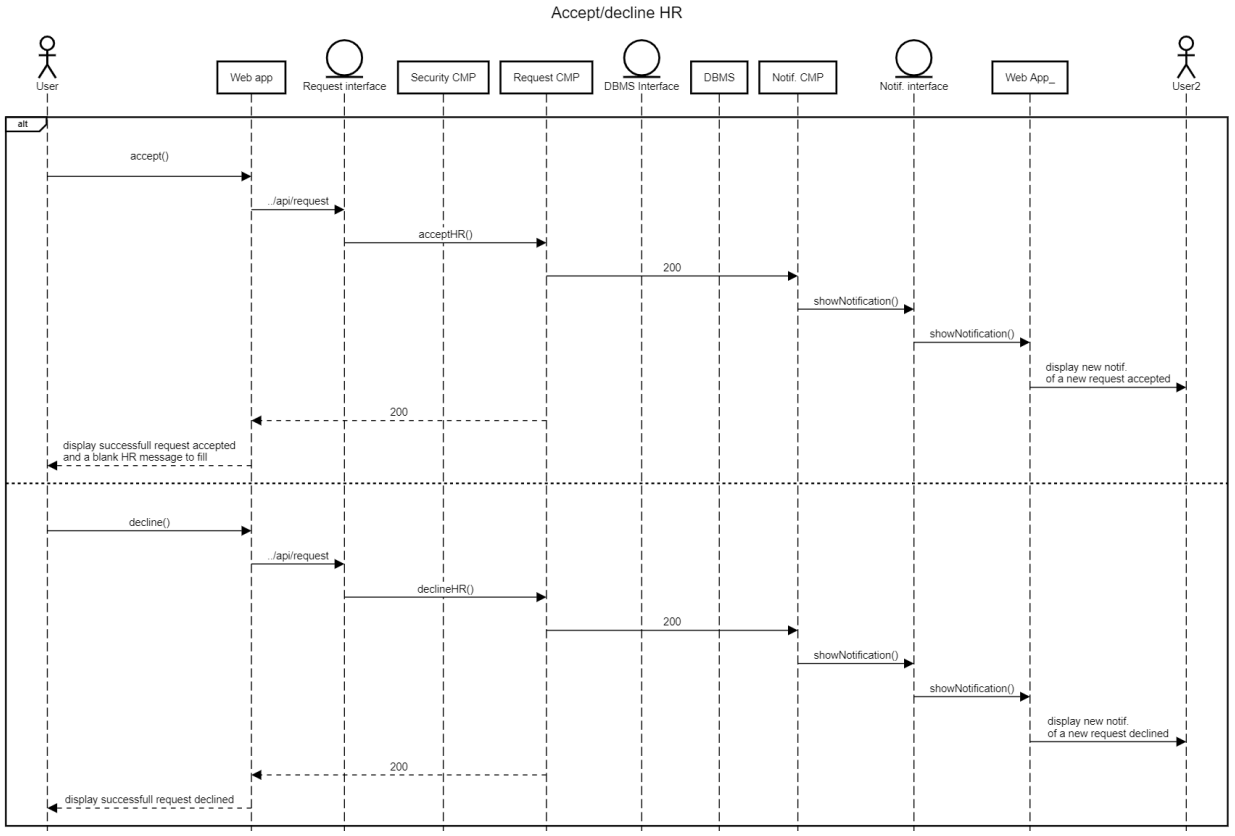
This sequence deals with the process of voting a forum element. Through the voting interface, it is possible to send the votation of the user to the DBMS and update the score of the voted element. Since there are no exceptions in this flow of operations, the DBMS will return 200.

2.4.6. Send requests



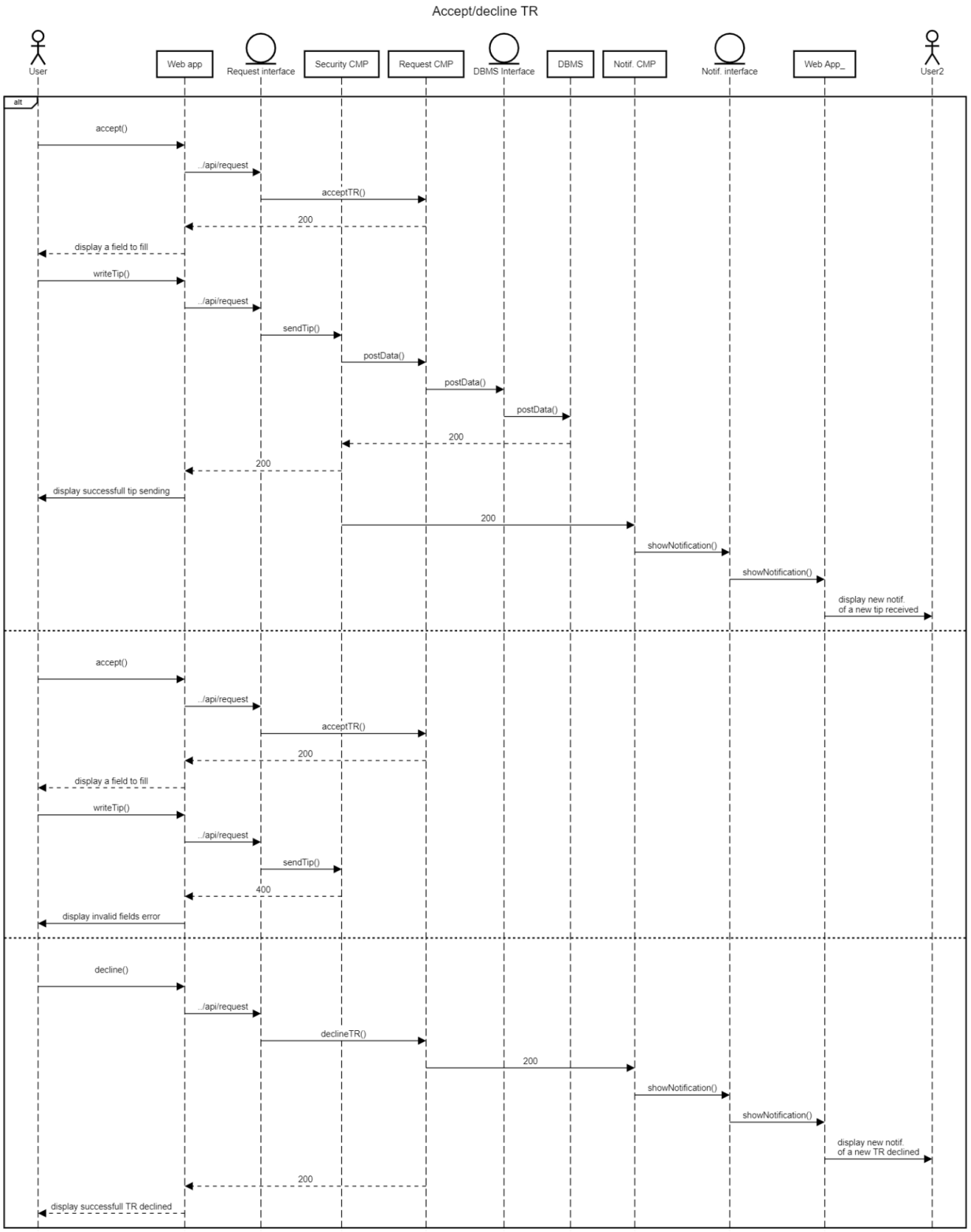
This sequence deals with all the procedures that involve the sending of requests, such as HR and TR. The request interface will be used to forward the request from the web app to the security component, which will forward the data to the DBMS unless some fields are not valid. Once the data is posted in the DBMS, it returns a 200 code that, once it is received from the security component, will both display a success message to the sender, and send a notification to the receiver, using the notification interface.

2.4.7. Accept/decline HR



This sequence deals with the procedures of accepting an HR received by a farmer. Through the request interface, it is possible to communicate from the web app, through the notification component and to the web app of the sender if the receiver has decided to accept an HR or not. The only difference between accepting or declining an HR is what is displayed in the receiver web app.

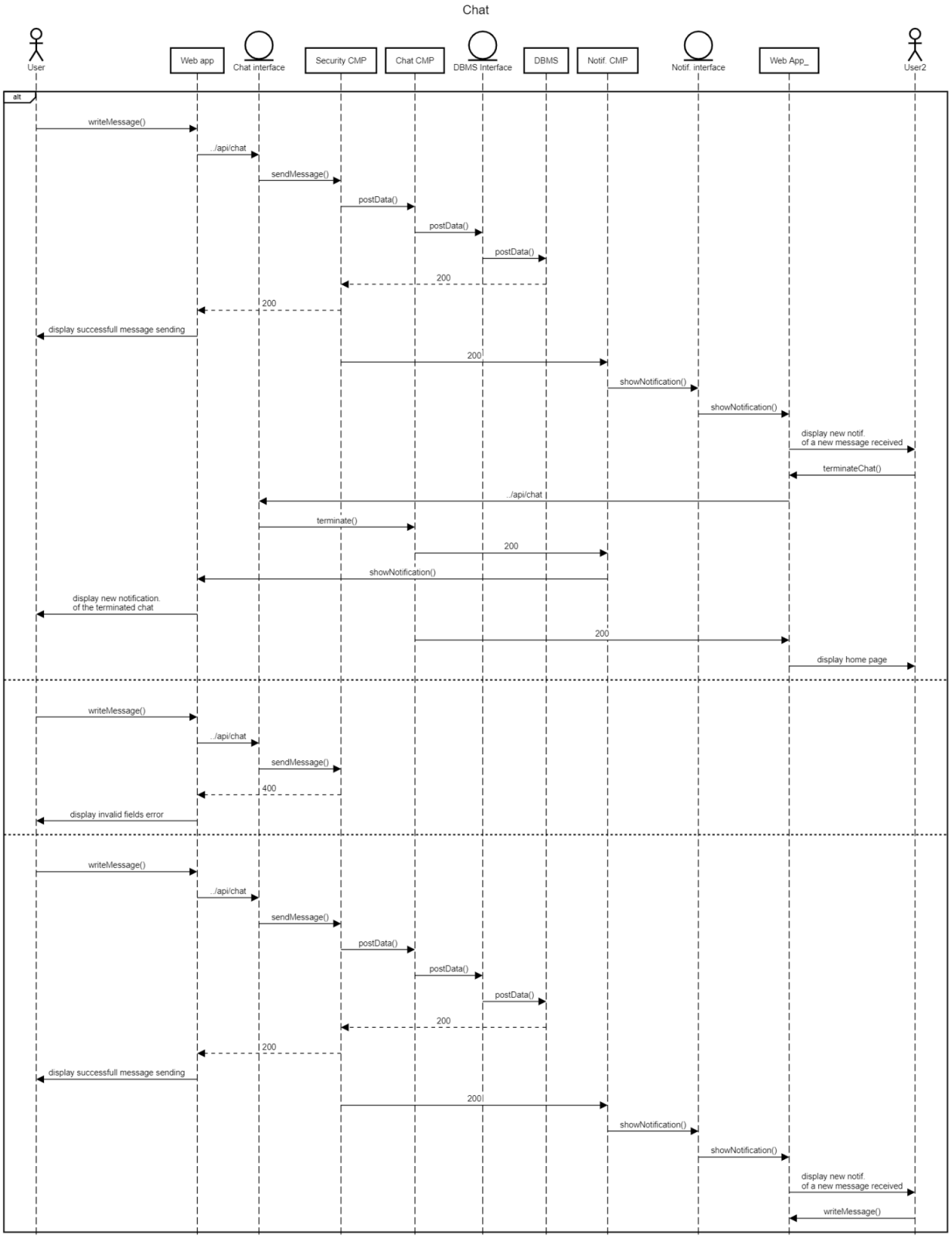
2.4.8. Accept/decline TR



This sequence deals with the procedure of accepting a TR by a farmer. Through the request interface it is possible to forward what the farmer decides to do after receiving a TR. After having accepted a TR, the request component displays to the user a field to be filled, that corresponds to the draft tip that will be sent to the policy maker. If this draft tip is blank, the security component will return a 400 code to the web app. Otherwise, the draft tip will be posted in the DBMS, which returns 200 to the security component. After

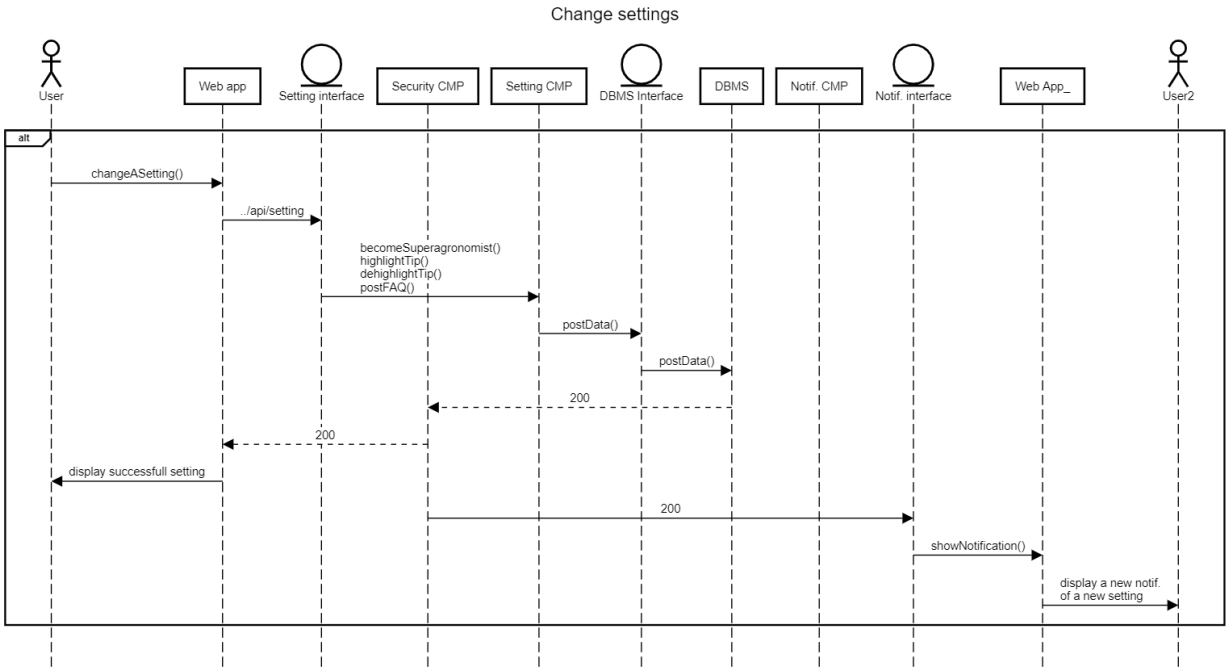
that, the security component will both send 200 codes to the web app and to the notification component, displaying a successful sending to the farmer, and a notification to the policy maker.

2.4.9. Chat



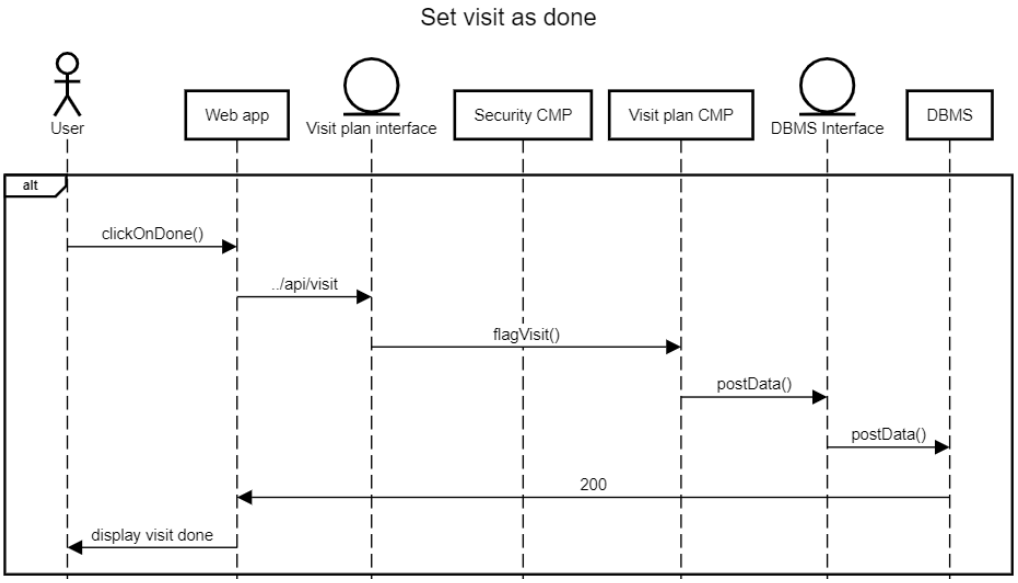
This sequence deals with the procedure of exchanging messages. Through the chat interface it is possible to send a message from a web app to another web app. Each message will be passed through the security component to check if it is blank, otherwise it will be saved in the DBMS, which will return 200 after the storing. This code will be again sent to the security component that will forward to both sender and receiver respectively a message of successful sending and a notification. The receiver can decide to send another message, or to terminate the chat. The chat will be used when two users have to communicate after the accepting or receiving of an HR, to change the visit date of a visit, and to allow the policy maker to ask for a modification of a draft tip.

2.4.10. Change settings



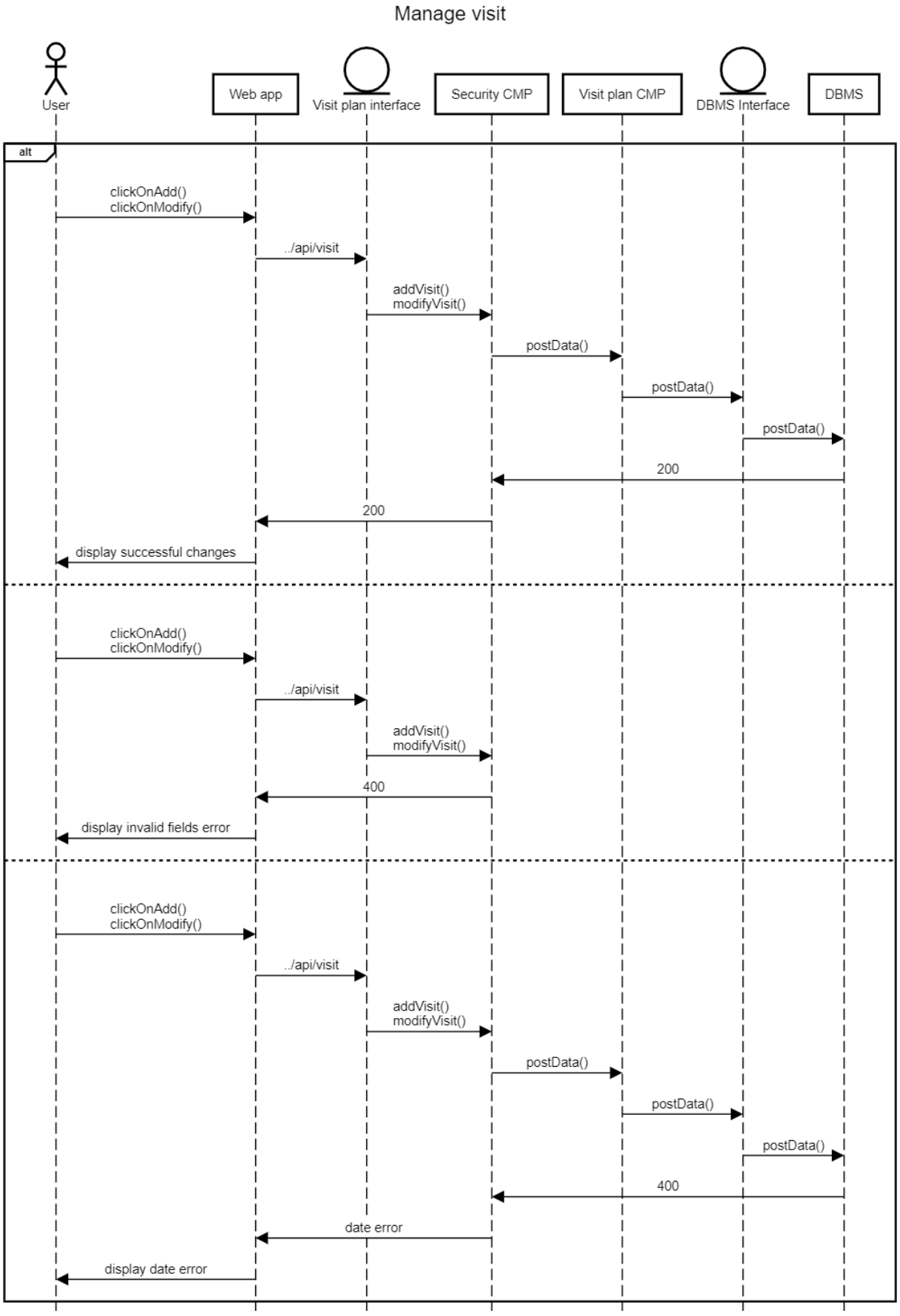
This sequence deals with the procedure of changing the most important settings of the system. Through the setting interface it is possible to forward the change that has been made by the user to the DBMS, saving it. After a change of setting, the web app of the user that made that change will receive a message of successful change of setting. Moreover, this change will be notified through the notification interface to users that are involved in that change.

2.4.11. Set a visit as done



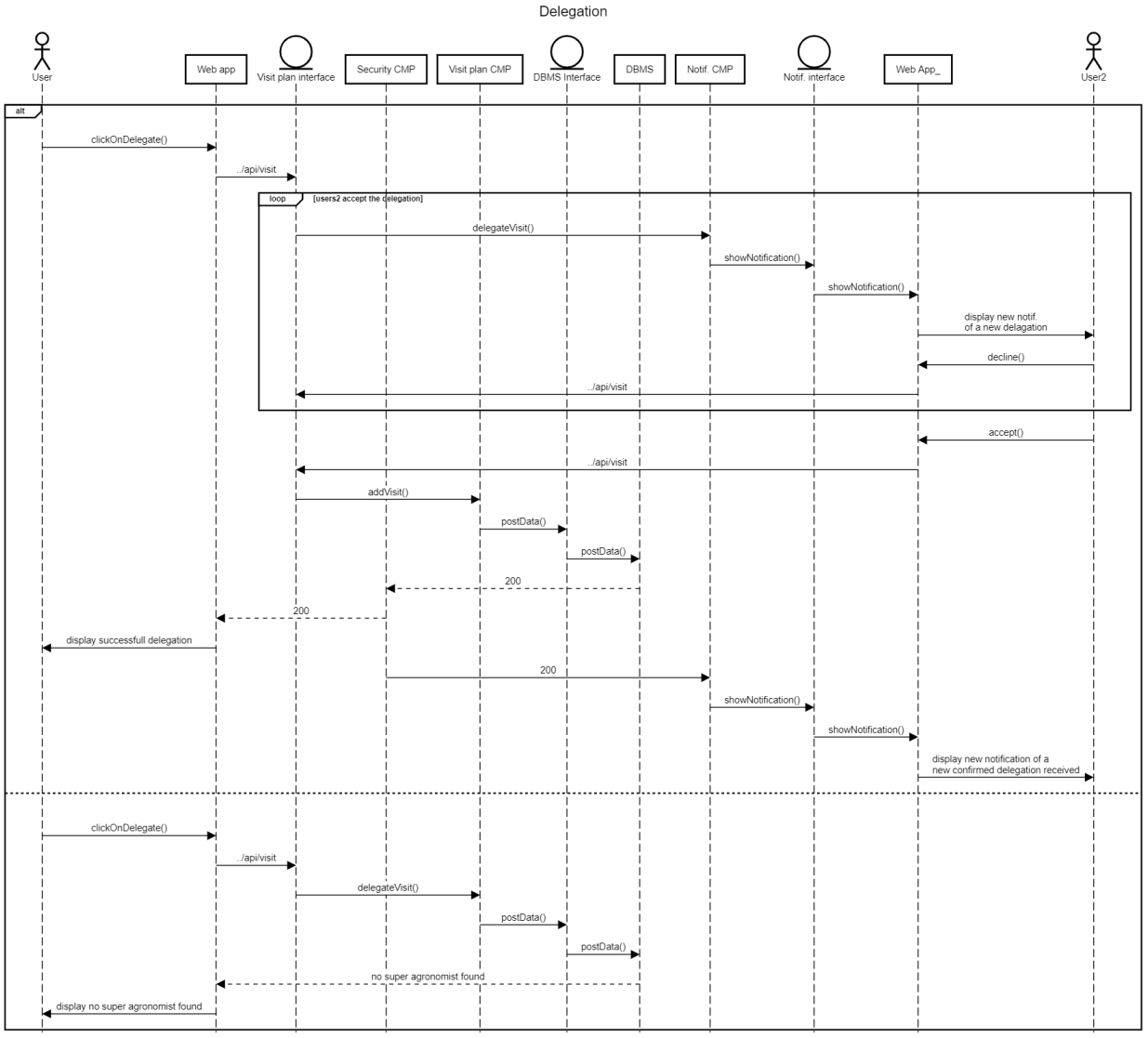
This sequence deals with the process of changing the state of a visit from scheduled to done. Through the visit plan interface, the action of the user is sent to the DBMS, which saves the new data. After that, the web app will display the successful setting, and the system will automatically do some operations in the background, such as making available the visit-report to the agronomist.

2.4.12. Manage visit



This sequence deals with the process of adding or updating a visit by the agronomist. Through the visit plan interface, the data about a visit is forwarded from the web app to the DBMS, through the security component that checks for the correctness of the data inserted: if the data is not blank, it is posted in the DBMS, otherwise the security component will return a 400 to the web app. In the DBMS, a check of the date of the visit will be made, and if this check is positive, the DBMS will return a 200 code, otherwise it will return a 400 code that will be fetched by the web app, displaying an inconsistency of the data inserted.

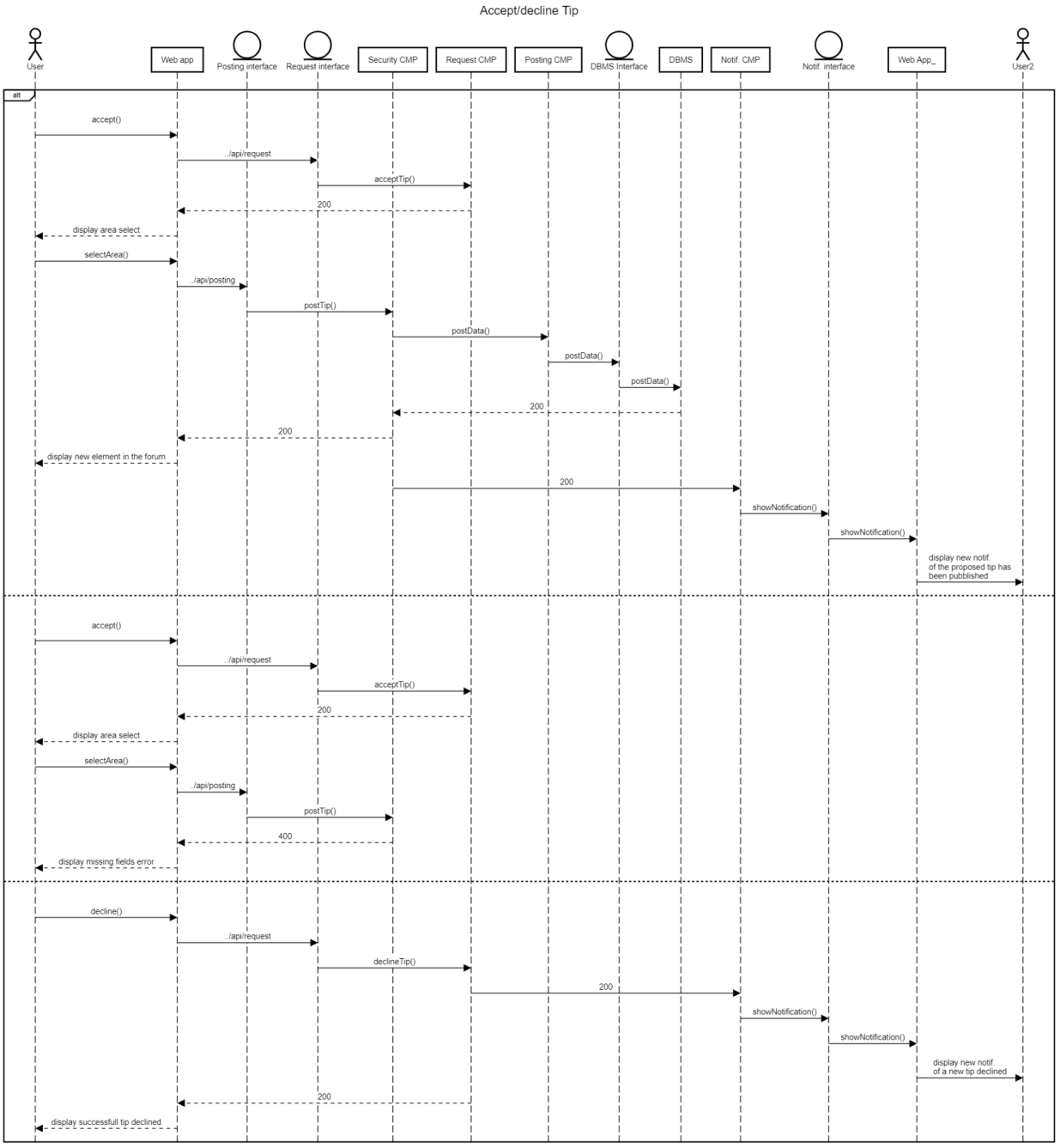
2.4.12. Delegate a visit



This sequence deals with the procedure of delegating a visit to a super agronomist. Through the visit plan interface, the choice to delegate a visit made by an agronomist is forwarded to the system. The visit plan component forwards this operation to the notification component, which sends a notification to the web app of the super

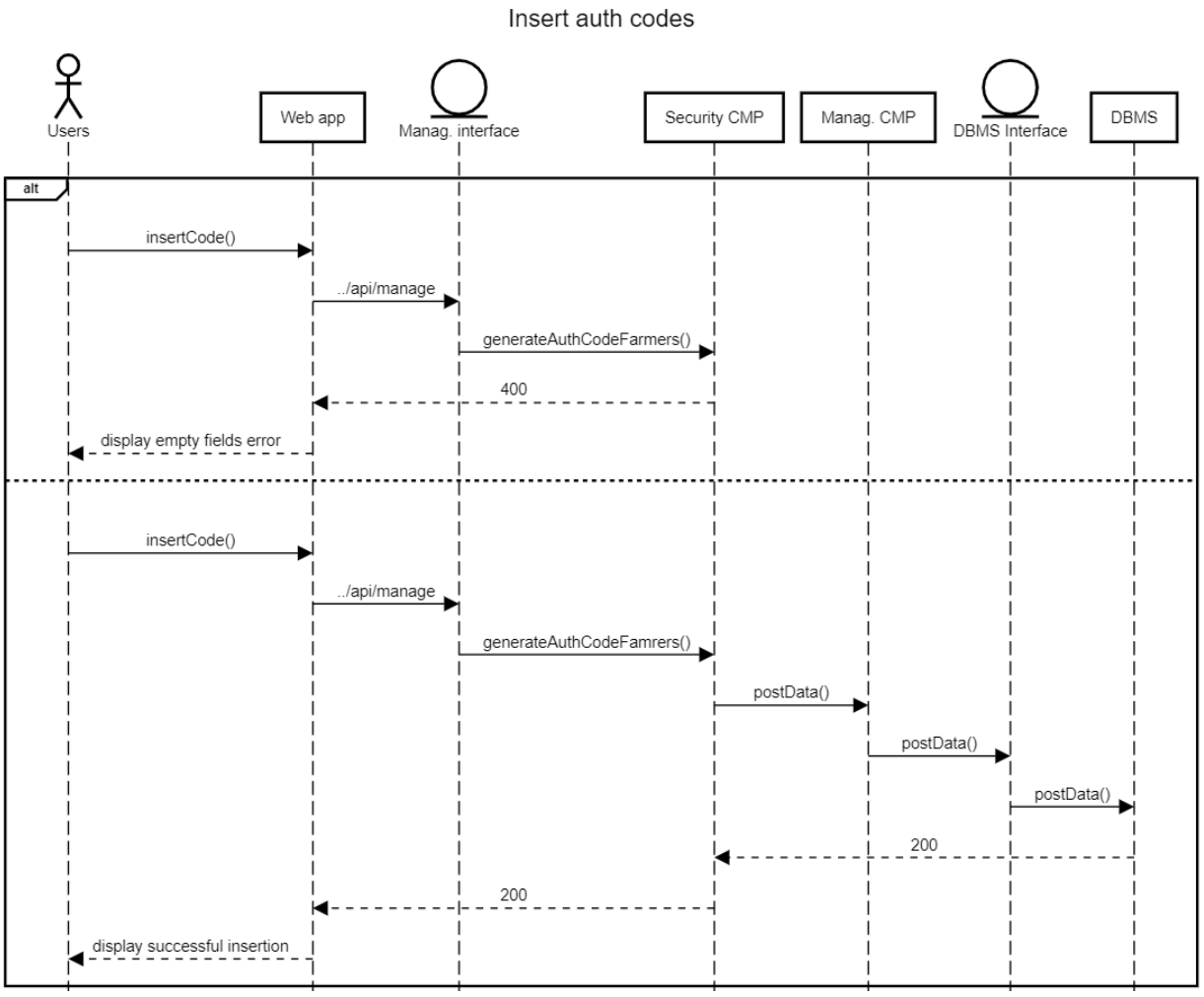
agronomist receiver. The super agronomist can decide to decline the delegation or accept it. If it is declined, the scenario is repeated, and again the visit plan component will send a notification to another super agronomist and so on, until either the delegation is accepted or there is nobody available that accepts it. If the delegation is accepted, the web app makes a request to the visit plan interface, which adds a visit as it happened in the previous sequence diagram *Manage a visit.* If the delegation process fails then the visit remains in the visit plan of the requesting agronomist.

2.4.13. Accept/decline tip



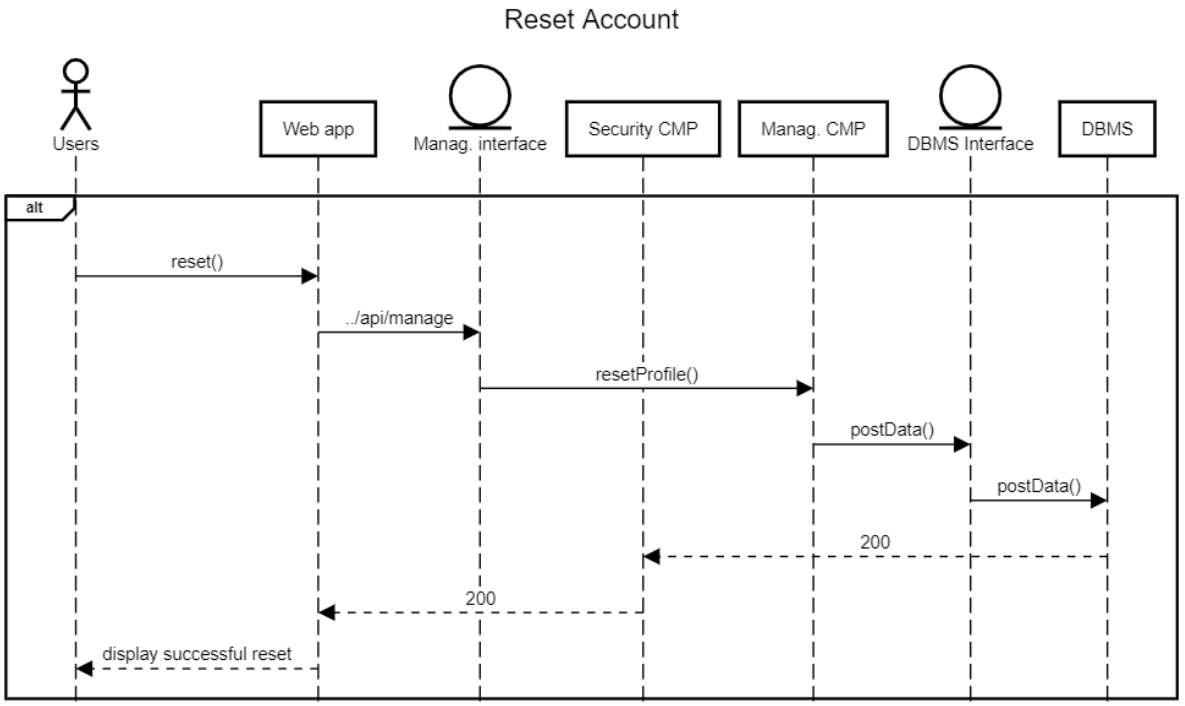
This sequence deals with the procedures that allow a policy maker to accept or decline a draft tip sent by the farmer. If the choice of the policy maker is to accept, it is forwarded using the request interface to the request component, which gives the possibility to the policy maker to insert the area which the tip will be flagged with. If this information is blank, the security component will send a 400 code to the web app, otherwise, the tip will be posted according to the sequence already described in the *Posting elements* sequence. If the policy maker decides to decline the draft tip, a notification will be sent to the farmer and the policy maker will see a message of a successful declining tip.

2.4.14. Insert auth codes



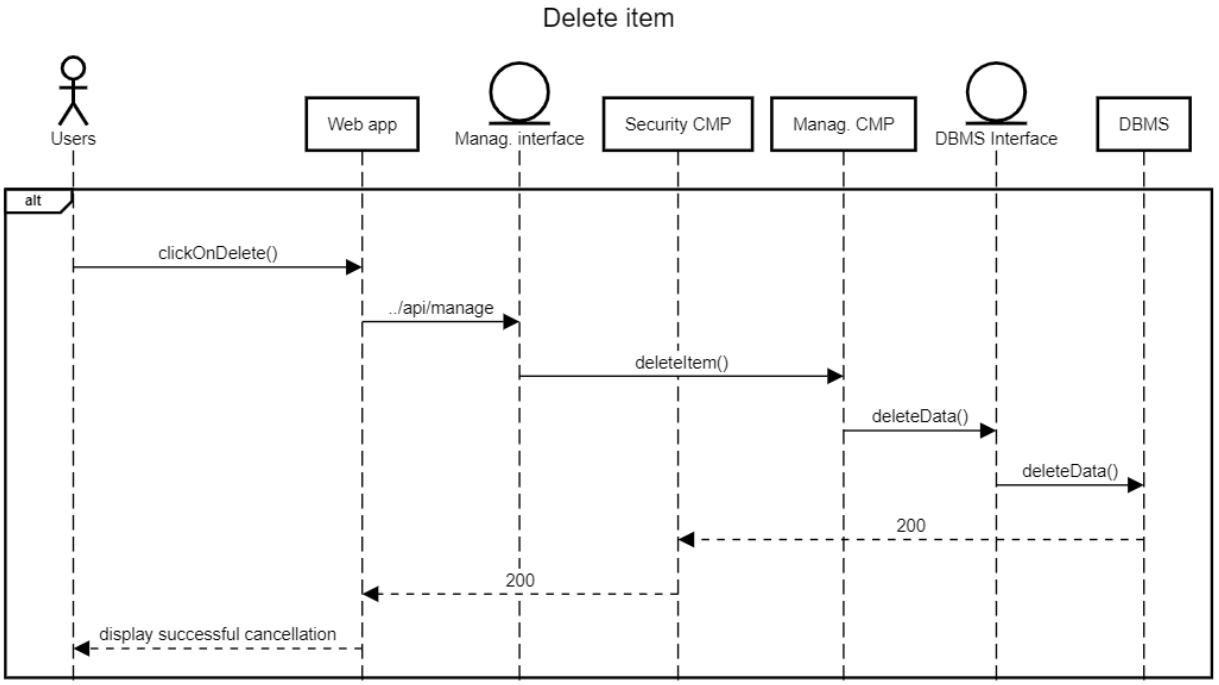
This is the sequence that deals with the procedures used to generate new auth codes allowing farmers to register to the system. Through the managing interface, the field inserted by the user is forwarded to the security component, which returns a 400 code if the code is blank. Otherwise, new data about a new auth code generated starting from the inserted text is posted in the DBMS. It returns a code 200 to the web app, allowing the user to receive a confirmation message about the operation just done.

2.4.15. Reset account



This sequence deals with the procedures that allow the resetting of an agronomist's account or policy maker's account. Through the management interface, it is possible to forward to the management component the account that needs to be resetted. The choice is posted in the DBMS and saved, and a 200 code is sent to the web app that shows to the user the successful reset operation.

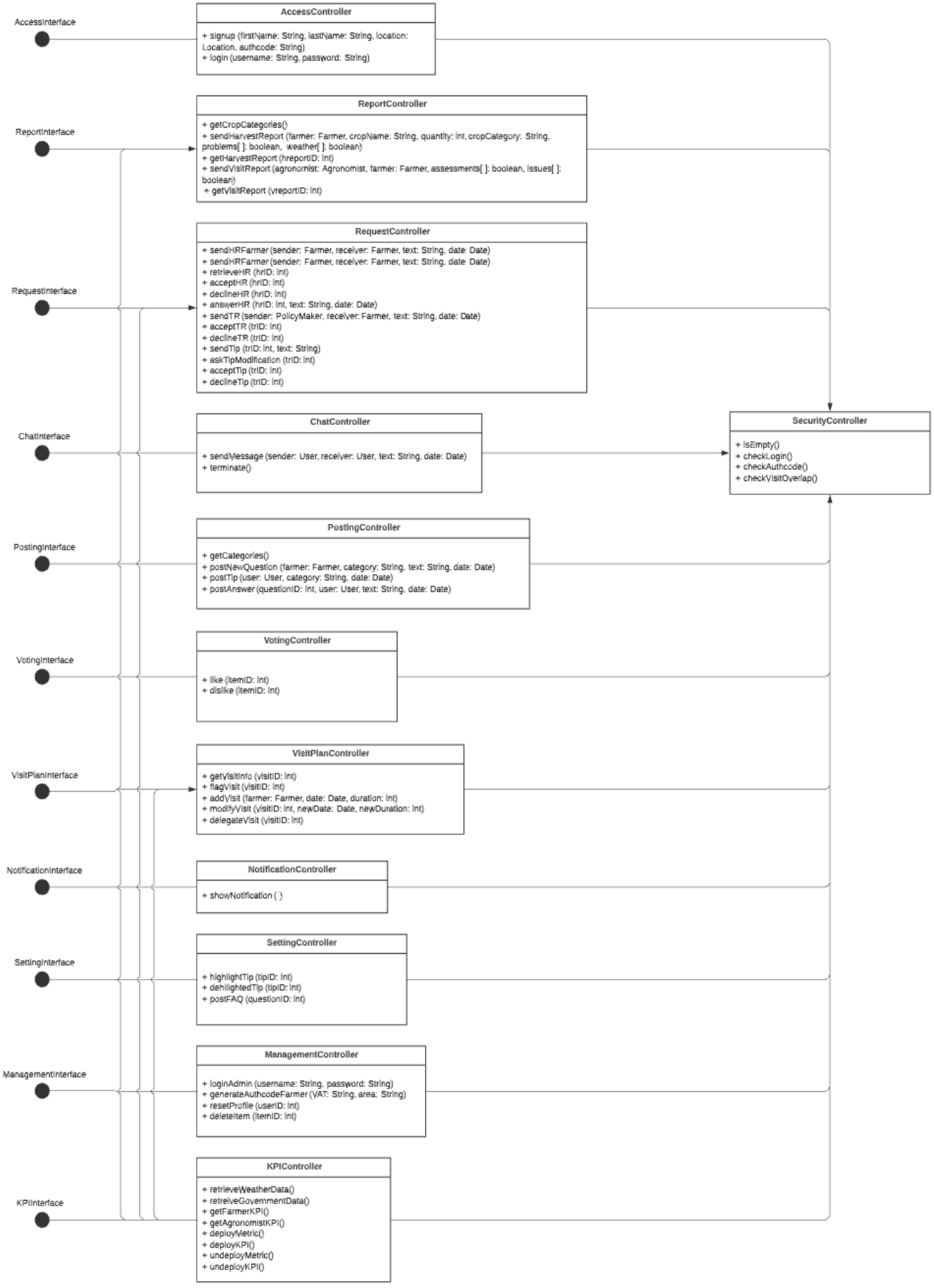
2.4.16. Delete an item



This sequence deals with the procedure of deleting an item from the system. Through the management interface, it is possible to forward to the management component which element has to be deleted. The data is forwarded to the DBMS that cancels that element and a 200 code is sent to the web app that shows to the user the successful deleting operation.

2.5. Component Interfaces

The diagram below shows the main interfaces that allow the correct execution of DREAM.
Each interface is implemented through a controller, all the most important methods are presented. The diagram also shows dependencies among controllers.



- The **AccessInterface** allows the user to sign-up and log-in. It is realized through the **AccessController,** that contains the signup and login methods.

- The **ReportInterface** is necessary for handling the activities related to both the harvest report and the visit report. It is implemented through the **ReportController**, whose methods allow a farmer to send the harvest report, and allow an agronomist to send a visit report. Through the methods *getHarvestReport (hreportID: int)* and *getVisitReport (vreportID: int)* it is also possible to retrieve them.

- The **RequestInterface** deals with the HRs and the TRs, and it is implemented through the **RequestController**. This controller contains all the methods that allow sending, accepting and declining HRs and TRs, and other methods useful to handle these functionalities.

- The **ChatInterface** is needed for the exchange of messages between users. It is realized through the **ChatController** whose methods are necessary to send messages and terminate the chat.

- The **PostingInterface** allows to achieve the most important forum functionalities. It is implemented through the **PostingController**, which contains methods necessary to post questions, answers and tips.

- The **VotingInterface** is needed for voting inside the forum. It is realized through the **VotingController**, whose methods *like (itemID:int)* and *dislike (itemID:int)* respectively allow adding a like or a dislike to a post.

- The **VisitPlanInterface** allows an agronomist to manage their visit plan. It is realized through the **VisitPlanController**, whose methods allows an agronomist to add or modify visits and delegate them.

- The **KPIInterface** provides the computed KPIs, obtained combining the data collected from other controllers and calculated according to the metrics and KPIs definition. It also allows government admin to deploy/undeploy metrics and KPIs. It is realized through the **KPIController**.

- The **NotificationInterface**, through the **NotificationController**, shows the notifications to the user.

- The **SettingInterface** deals with settings such as (de)highlighting a tip, becoming a super agronomist and posting FAQs. It is realized through the **SettingController**.

- The **ManagementInterface** allows the government admin to manage the system. It is realized through the **ManagementController**. The method *generateFarmerAuthcode(VAT: String, area: String)* allows a farmer's VAT to be used as auth code (by inserting the new VAT to the valid VAT list) and to associate the farmer to a specific area. The other methods are used to reset a

profile, by generating new auth codes, from agronomist and policy makers and to delete items.

Finally, interfaces are also implemented through the **SecurityController**. Here some security checks are handled through the corresponding methods. This controller is needed to authenticate users and validate any data coming from clients. In particular, the *checkLogin()* function verifies that the user sending the http request has a valid token. The other functions are responsible to check that there are no missing mandatory fields in the write requests and that all the allowed operations on the data are meaningful, in the sense that all the data types are respected and more high level consistency checks among inserted data. Also, overlapping between visits is handled with the method *checkVisitOverlap()*.

2.6. Selected Architectural Styles and Patterns

2.6.1. Three-tier architecture

The architecture chosen for DREAM is a three-tier architecture.
The main advantage of this kind of architecture is that every tier is implemented separately in its own infrastructure, therefore updates or modifications on one of the tiers do not affect the other tiers. Furthermore, since every tier can be developed from different teams, the whole development process is speeded up. Scalability is a key point: each tier is scalable independently of the other tiers. Also, the reliability is improved, since tiers are independent one from another, and if an interruption occurs in one tier it doesn't directly affect the others.
Very importantly, this kind of architecture provides an improved security with respect to the two-tier architecture: in fact, the client cannot directly access the database and therefore it is more difficult to obtain unauthorized data; also, if well developed, the application tier could work as an internal firewall.

2.6.2. RESTful architecture

The backend is handled by using a REST API.
This architecture makes use of the HTTP methods through which data is delivered.
A RESTful API must respect some requirements:

- A client-server architecture is used, and requests are handled with HTTP methods.
- The stateless principle must be respected: client's data is not memorized in the GET requests and the server does not contain any information about the state of the client.
- Data is cacheable, in order to optimize client-server interaction.

- Component interfaces are uniform, in order to have a common approach for accessing the resources
- The system is layered in order to organize different kinds of servers.

An important property of this architecture is the *code on demand*, that is the possibility to send executable code from the server to the client.

In conclusion, the choice of a REST API allows to improve scalability and speed, and it is simple to use.

2.6.3. Model View Controller (MVC)

Model–view–controller (MVC) is a software design pattern commonly used for developing user interfaces that divide the related program logic into three interconnected elements. This is done to separate internal representations of information from the ways information is presented to and accepted from the user [6].
The components of the MVC are:

- **Model**
  The model is the central component of the pattern. It is the application's dynamic data structure, independent of the user interface.It directly manages the data, logic and rules of the application.
- **View**
  The view is any representation of information such as a chart, diagram or table. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.
- **Controller**
  The controller accepts input and converts it to commands for the model or view.

2.7. Other Design Decisions

2.7.1. Thin client

Thin client is software that is primarily designed to communicate with a server. The client is responsible only for the presentation logic, the business logic of the application is contained in the application server. In a thin client, functions mostly don’t work when offline. Furthermore, a thin client generally consumes few local resources such as disk, computing power and memory; functionality may depend on a fast network connection [5].

2.7.2. Client side rendering

It has been used since single web pages are highly interactive.

Client-side rendering (CSR) allows the website to be updated in the browser when navigating to different pages the content is rendered by the client-side (browser). This method requires an initial download hit and effort to load every resource as it's doing a lot of round trips to the server, therefore the first visit to the webpage can be slower even though after the first load, this turns faster, also the load of traffic between client and backend is reduces since only the raw data need to be fetched and not the full web page [4].

2.7.3. Mobile First

Mobile First Approach refers to the practice of designing and/or developing an online experience for mobile before designing for desktop web or any other device. Taking a Mobile First approach aims to reverse the workflow of designing for desktop and scaling down the design for mobile afterwards [2].

2.7.4. Dry

"Don't repeat yourself" (DRY, or sometimes "do not repeat yourself") is a principle of software development aimed at reducing repetition of software patterns, replacing it with abstractions or using data normalization to avoid redundancy [3].

2.7.5. HTTP response status codes

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

- Informational responses (100–199)
- Successful responses (200–299)
- Redirection messages (300–399)
- Client error responses (400–499)
- Server error responses (500–599)

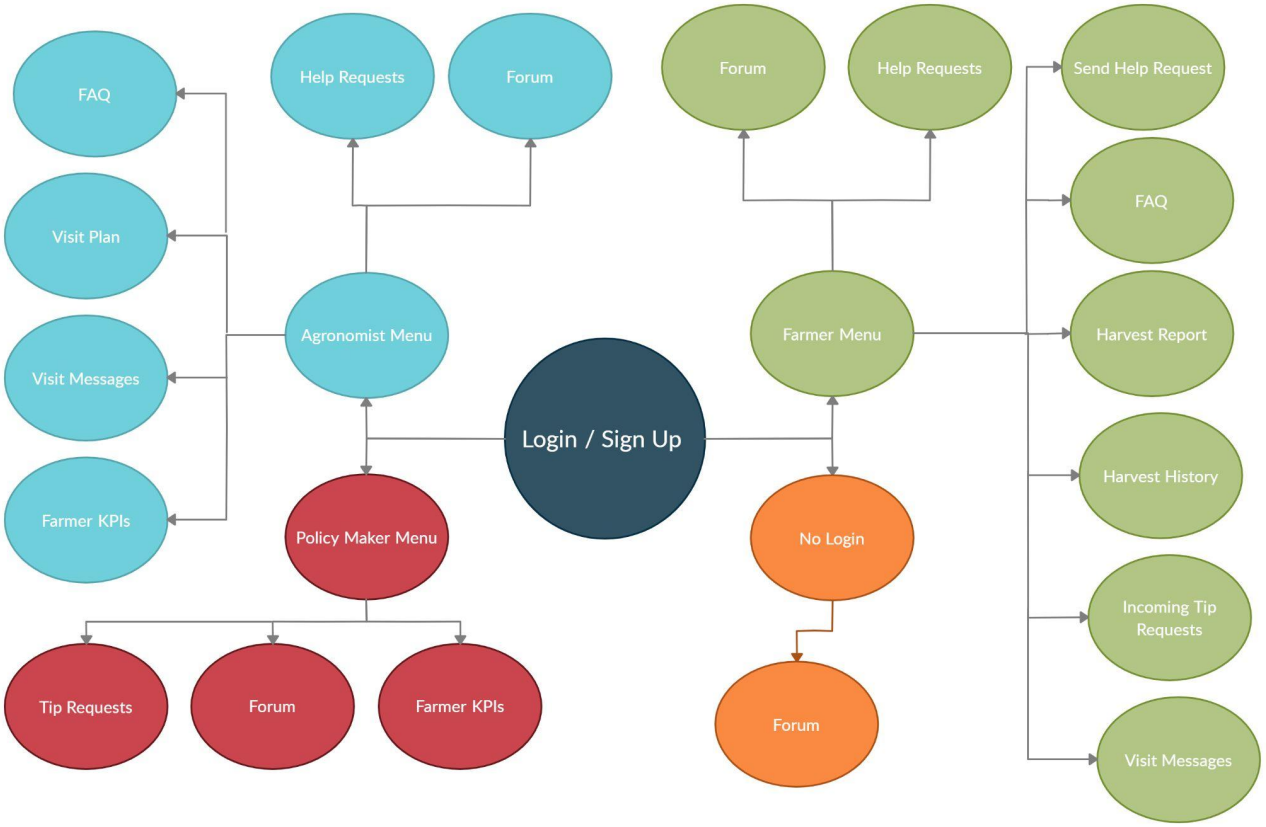
If a response that is not in this list is received, it is a non-standard response, possibly custom to the server's software [7].

3. USER INTERFACE DESIGN

The user interface has to be as simple as possible, all the DREAM services offered to each user have to be reachable following only one path.
The UI has to be adaptable to mobile phone screens ensuring all the functionality of the system with the same ease of use as if opened with a laptop: to achieve such a result the development team is strongly advised to start first designing all the front-end pages for mobile screens, then adapting them to bigger screens (tablet, laptop, monitors).

3.1. Site map

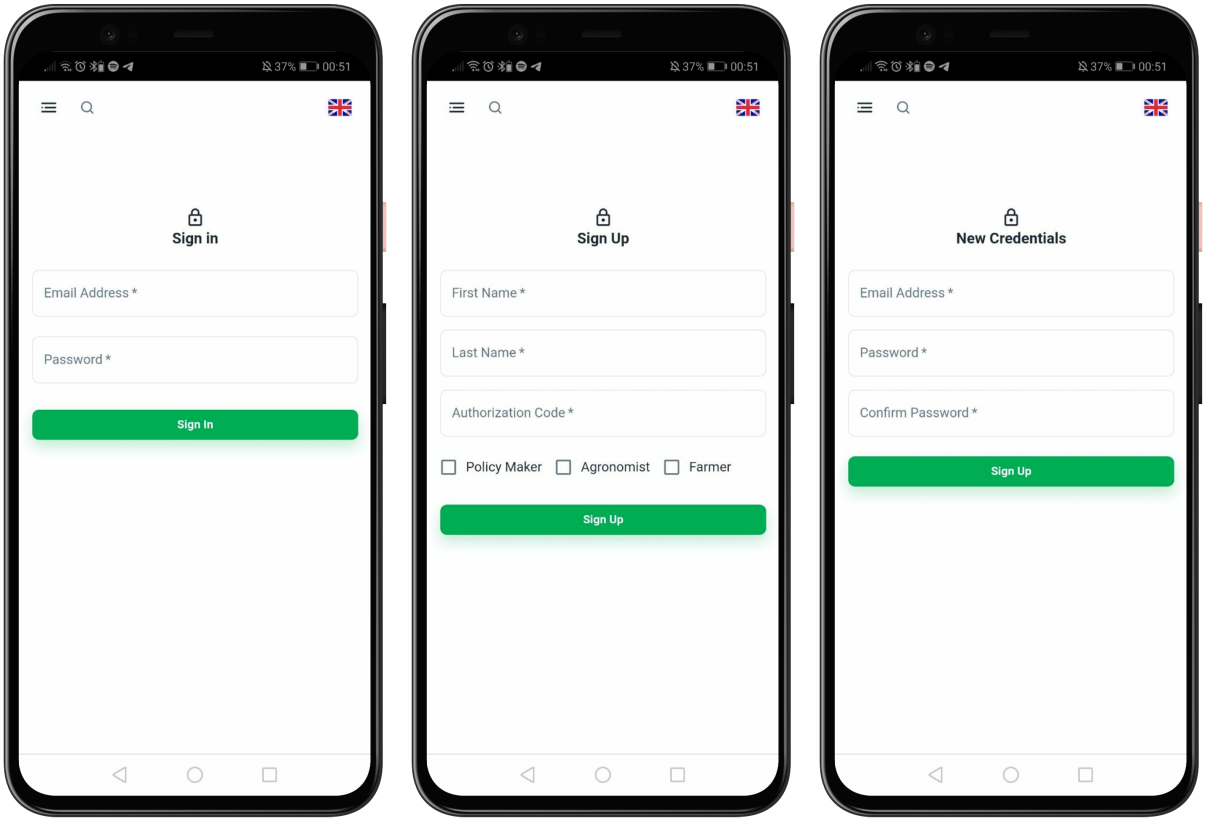
All the explorable pages are available to each user from the menu after logging in. The navigation can be divided in 3 levels; all the pages at the end of a path (last level) are dynamic pages that adapt their layout to what the user is doing and they handle the write/read requests to the REST API.



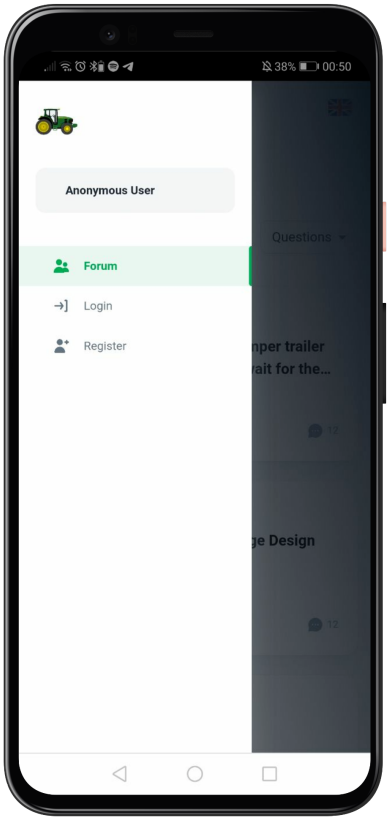
3.2. Mockups

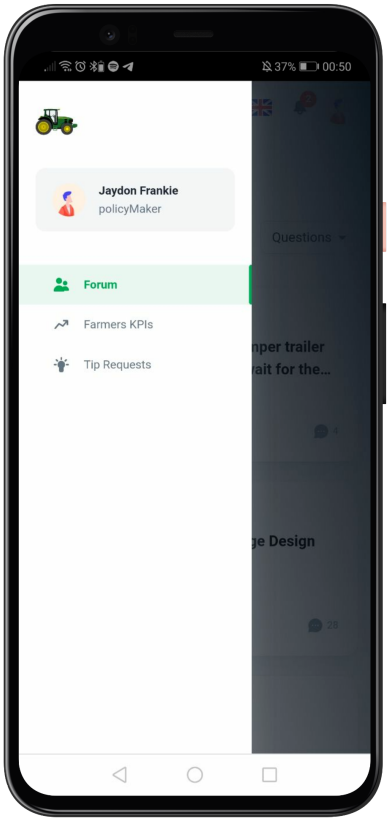
In the following paragraph the design of all the key components is presented. More attention is paid to mobile screens. The mockups are designed following Material Design guidance.
The development team should use them as a guideline for the front-end development, however some screens that are responsible only for data display, such as list of items or calendars, are not presented. The development team is free to choose how to design the unpresented details as long as they use material UI components.

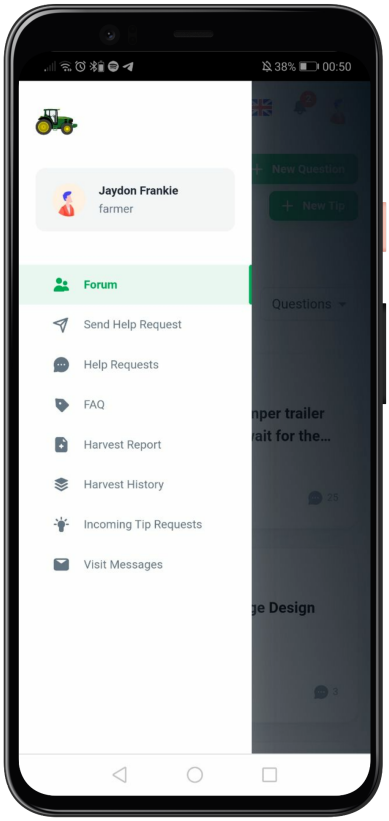
- Log in and sign up; if skipped, the user is anonymous (Level 0).

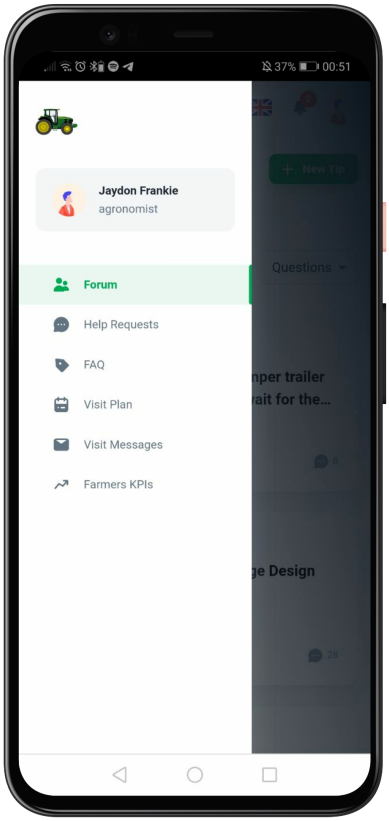


- Navigation sidebar for every user (Level 1).

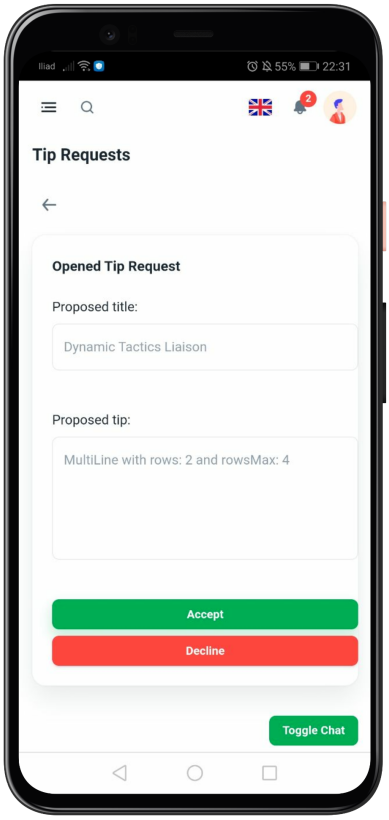


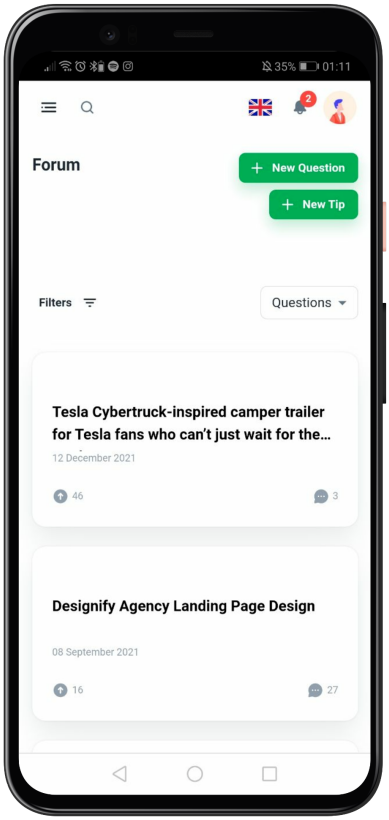


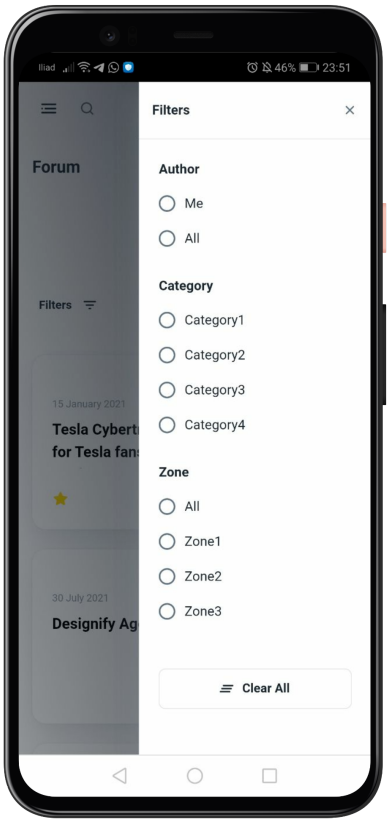


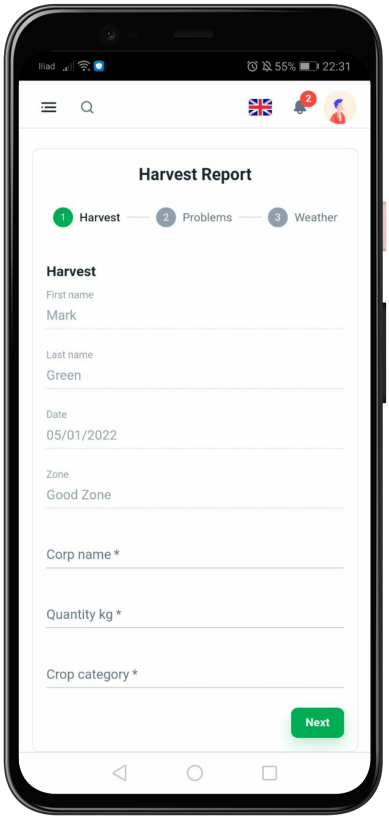


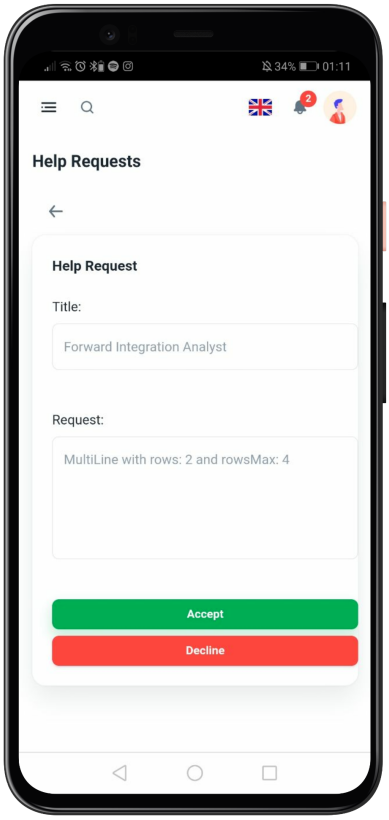
- Main pages for Farmers (Level 3).

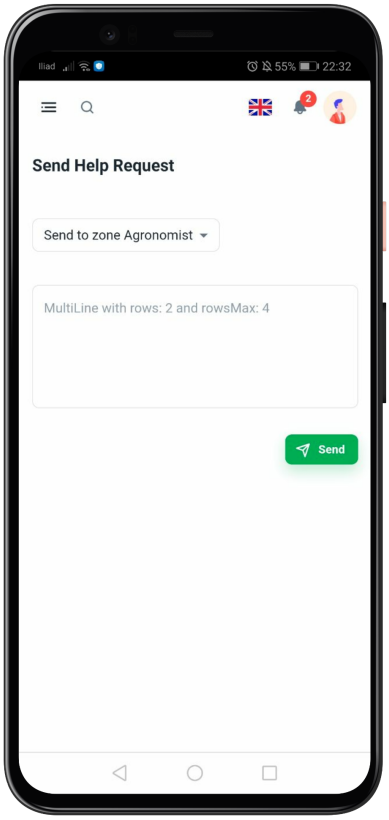




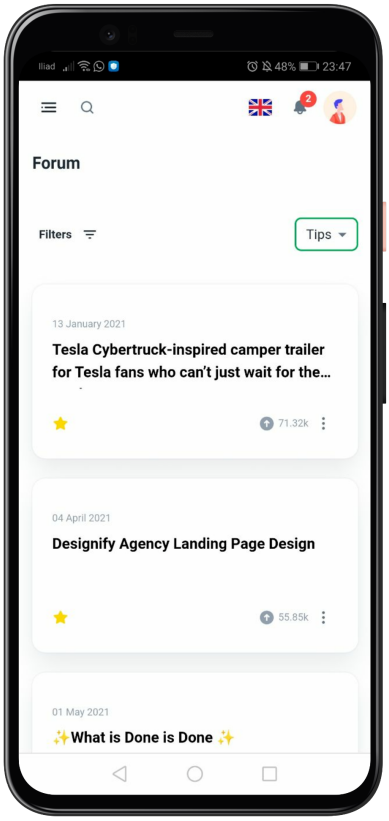


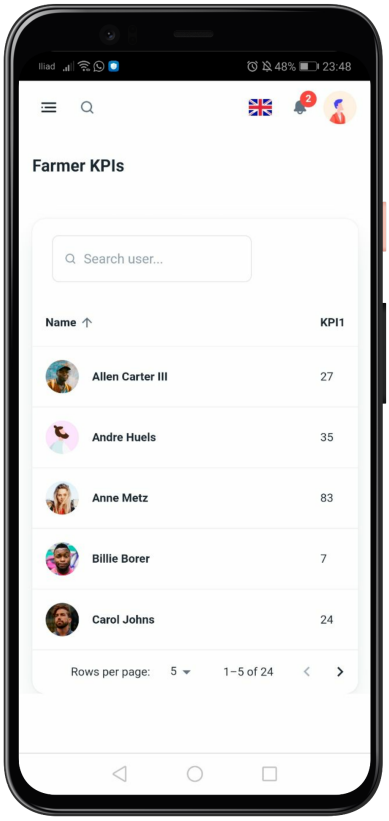




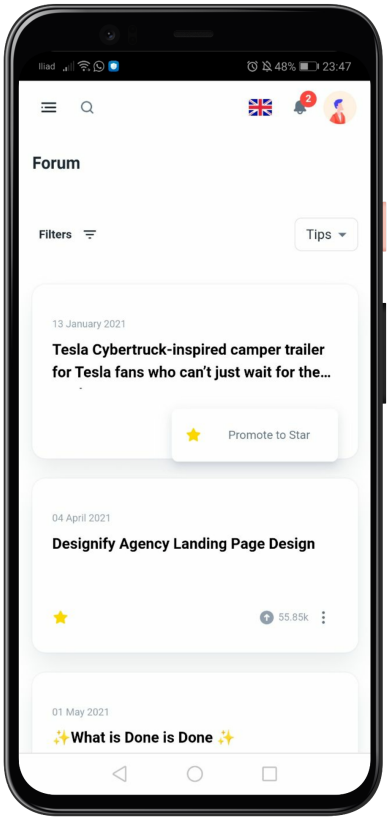


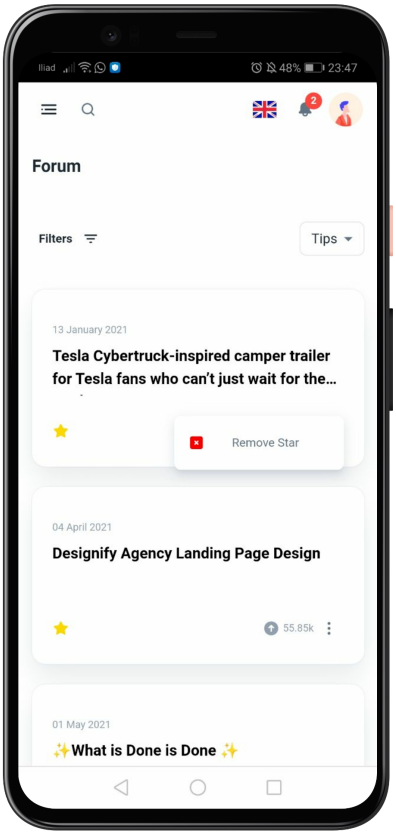
- Main pages for Policy Makers (Level 3).



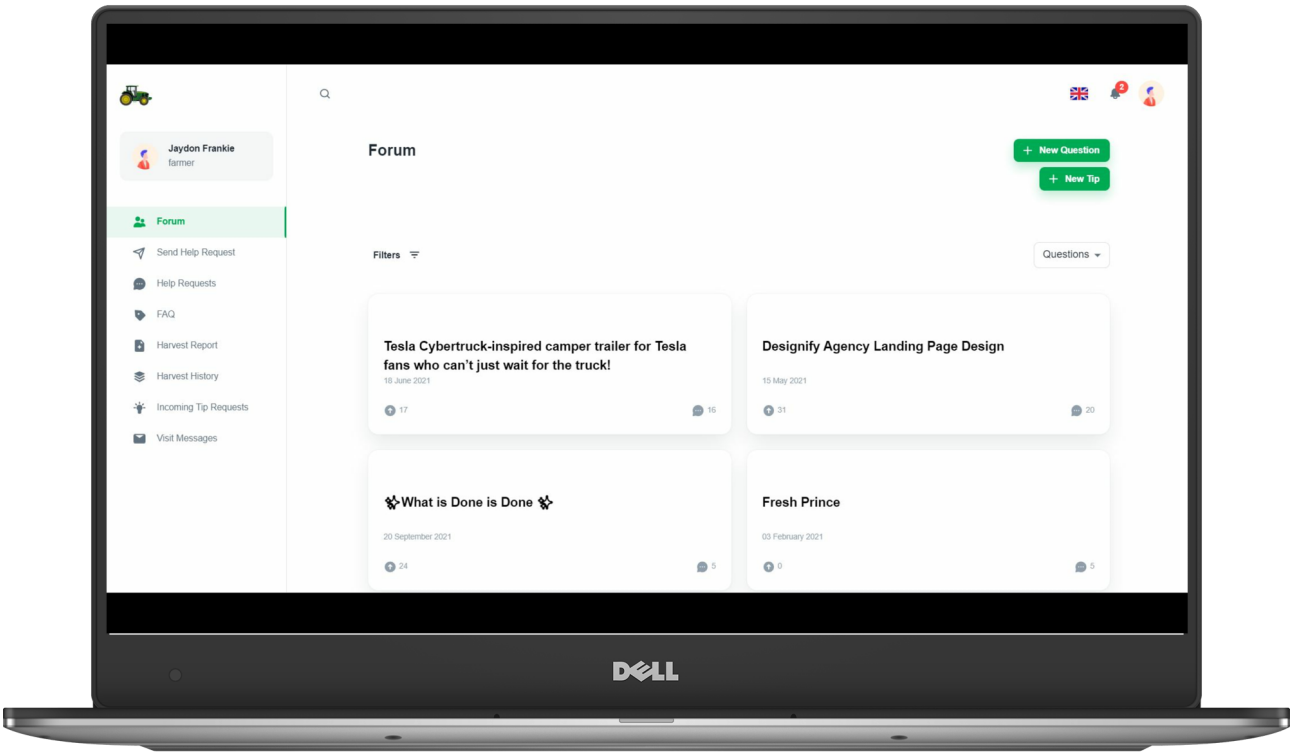


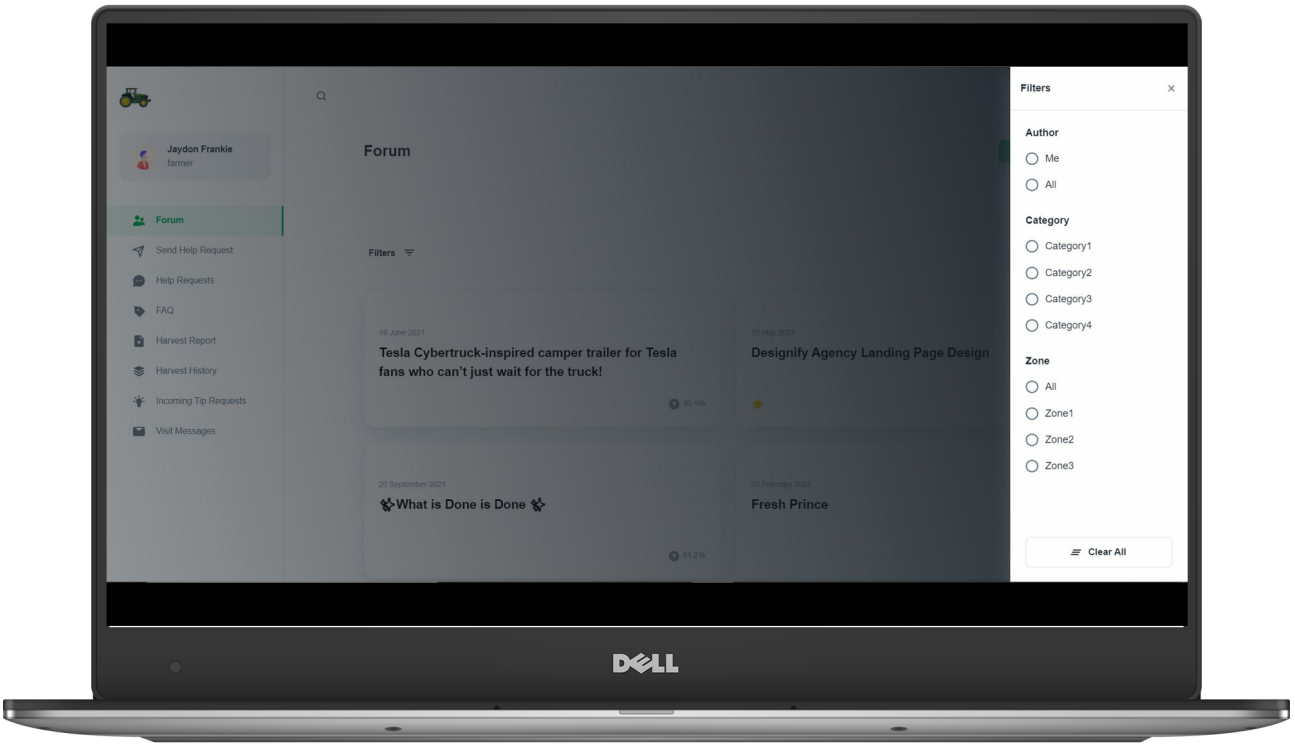
r

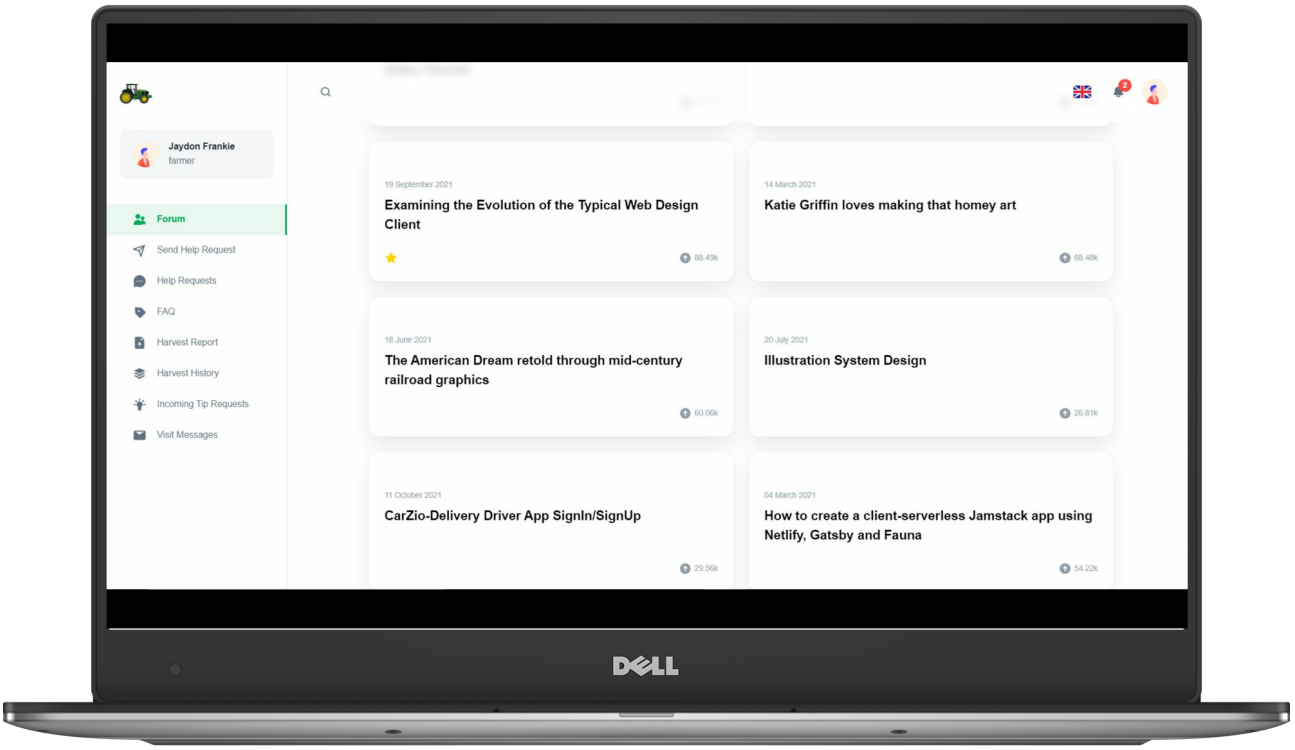


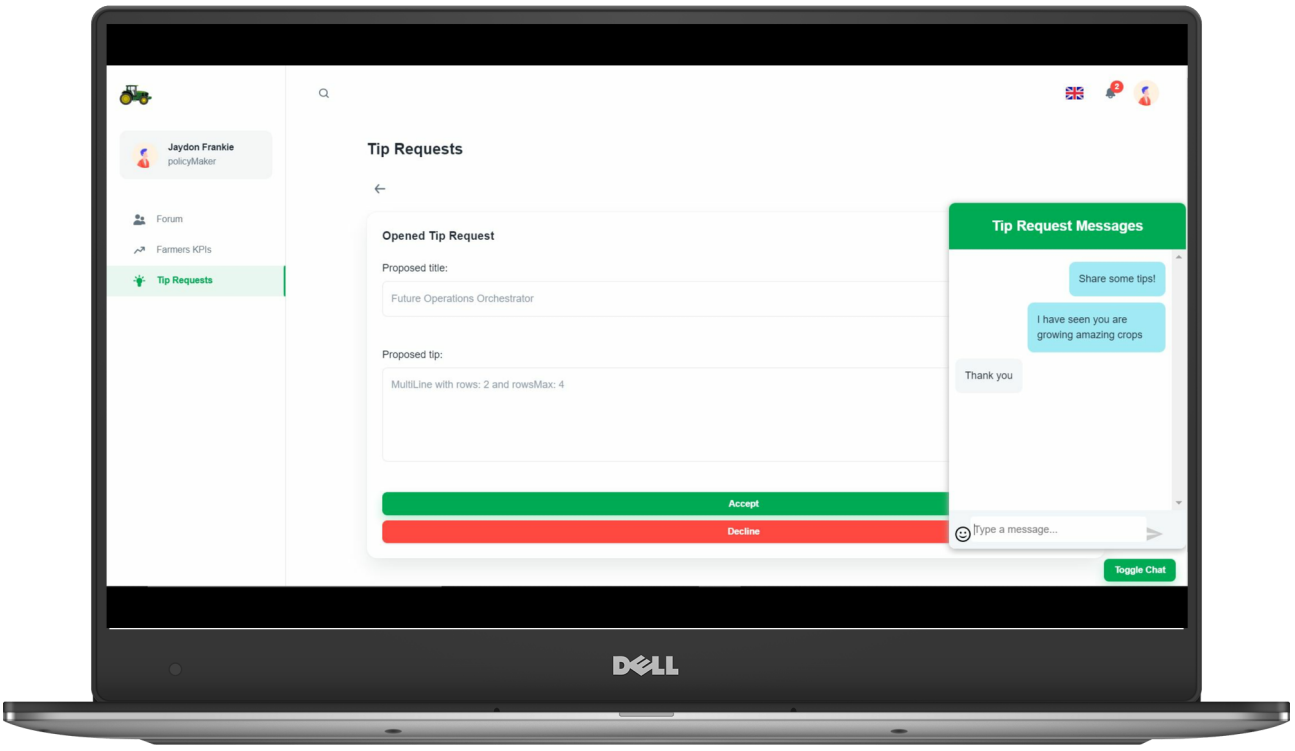


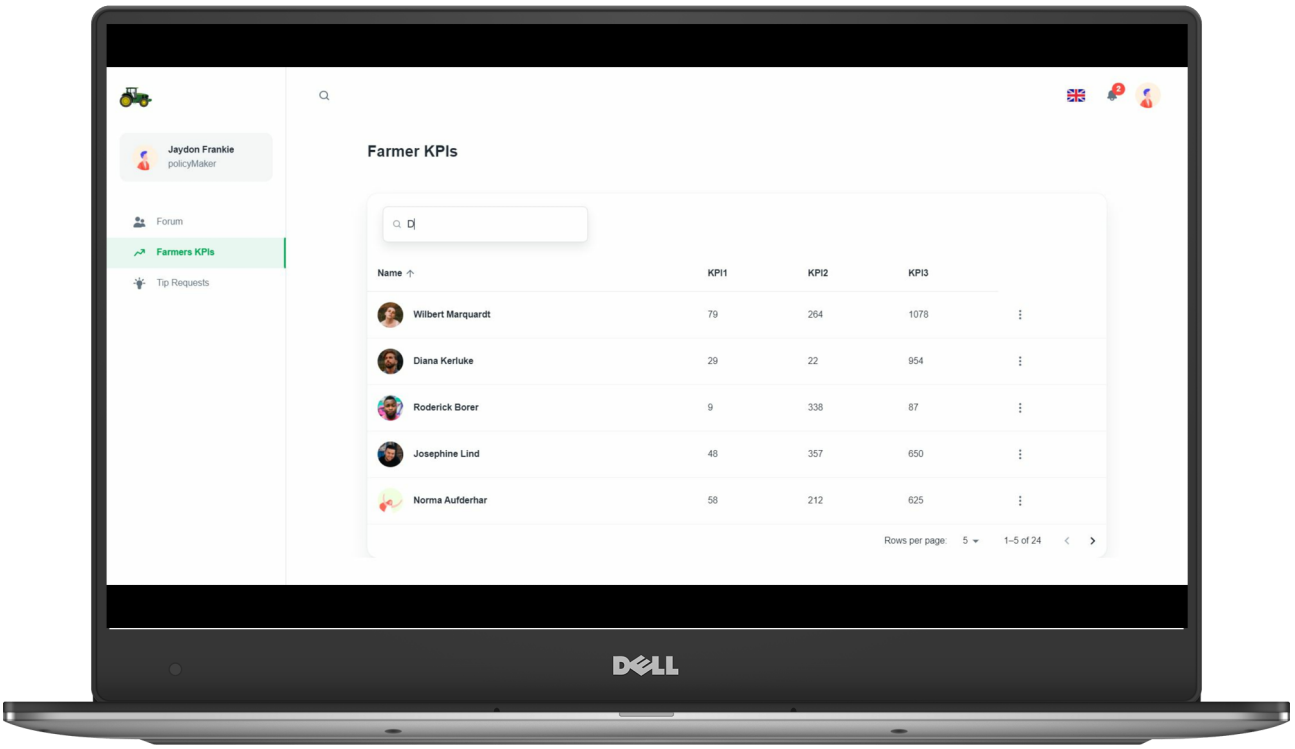
- Some examples on how the design can be generalized for laptop screens.

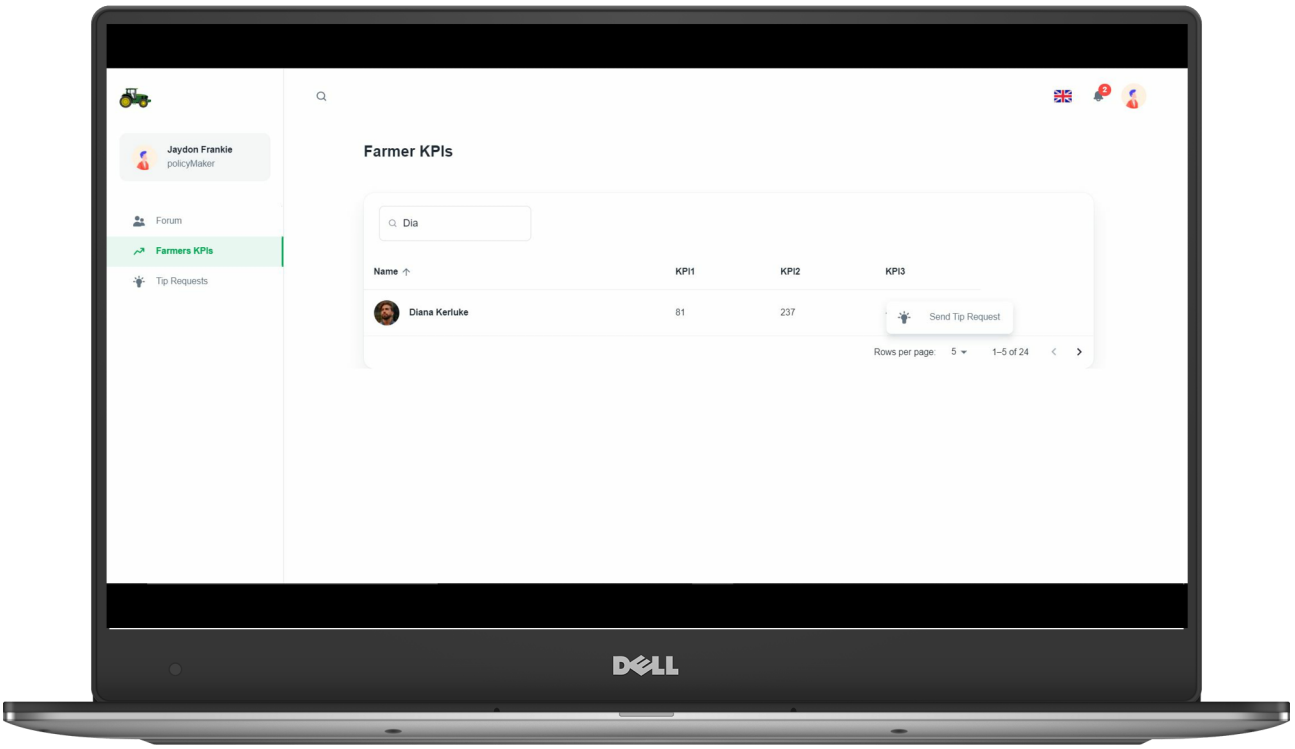












4. REQUIREMENTS TRACEABILITY

4.1. Requirements traceability

The aim of this section is to show the traceability between requirements and the design components previously described in section 2.2.

- **R1**: the system allows farmers to register using an authorization code (VAT).
  - Access CMP: in order to sign up.
  - Security CMP: in order to validate input data.

- **R2**: the system allows farmers to log in.
  - Access CMP: in order to log in.
  - Security CMP: in order to validate input data.

- **R3**: the system allows agronomists to register using an authorization code.
  - Access CMP: in order to sign up.
  - Security CMP: in order to validate input data.

- **R4**: the system allows agronomists to log in.
  - Access CMP: in order to log in.
  - Security CMP: in order to validate input data.

- **R5**: the system allows farmers and agronomists to publish tips in the forum.
  - Posting CMP: in order to publish in the forum.
  - Security CMP: in order to check the access token; in order to validate input data.

- **R6**: the system allows farmers to publish a question in the forum.
  - Posting CMP: in order to publish in the forum.
  - Security CMP: in order to check the access token; in order to validate input data.

- **R7**: the system allows farmers and agronomists to answer a question in the forum.
  - Posting CMP: in order to publish in the forum.
  - Security CMP: in order to check the access token; in order to validate input data.

- **R8**: the system allows farmers and agronomists to vote tips and answers in the forum.
  - Voting CMP: in order to vote in the forum.
  - Security CMP: in order to check the access token; in order to validate input data.

- **R9**: the system allows farmers to visualize relevant information regarding weather and tips.
    - Rel. Info CMP: in order to visualize relevant information.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R10**: the system allows farmers to fill in the harvest report.
    - Report CMP: in order to send the harvest report.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R11**: the system allows farmers to send an HR to farmers.
    - Request CMP: in order to send the request.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R12**: the system allows farmers to send an HR to his area's agronomist.
    - Request CMP: in order to send the request.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R13**: the system allows farmers to accept or decline an HR coming from another farmer.
    - Request CMP: in order to accept or decline the request.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R14**: the system allows farmers and agronomists to use HR messages.
    - Chat CMP: in order to exchange messages.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R15**: the system allows agronomists to add a new visit.
    - VisitPlan CMP: in order to add a new visit.
    - Security CMP: in order to check the access token; in order to check against the overlapping of two visits; in order to validate input data.

- **R16**: the system allows agronomists to update an existing visit.
    - VisitPlan CMP: in order to modify an existing visit.
    - Security CMP: in order to check the access token; in order to check against the overlapping of two visits; in order to validate input data.

- **R17:** the system allows farmers and agronomists to use visit messages.
    - Chat CMP: in order to exchange messages.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R18**: the system allows farmers and agronomists to terminate visit messages.

    - Chat CMP: in order to terminate a chat.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R19**: the system allows agronomists to publish HR-messages as FAQ.
    - Setting CMP: in order to mark HRs as FAQ.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R20**: the system allows agronomists to fill in and confirm a visit report.
    - Report CMP: in order to send a visit report.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R21**: the system allows agronomists to change the status of a visit.
    - Visit CMP: in order to mark a visit as done.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R22**: the system allows agronomists to visualize farmers KPIs.
    - KPI CMP: in order to visualize KPIs.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R23**: the system allows agronomists to delegate a visit to super agronomists.
    - VisitPlan CMP: in order to delegate a visit.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R24**: the system allows policy makers to register using an authorization code.
    - Access CMP: in order to sign up.
    - Security CMP: in order to validate input data.

- **R25**: the system allows policy makers to log in.
    - Access CMP: in order to log in.
    - Security CMP: in order to validate input data.

- **R26**: the system allows policy makers to highlight a tip as a star tip.
    - Settings CMP: in order to highlight a tip.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R27**: the system allows policy makers to send a tip request.
    - Request CMP: in order to send a request.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R28**: the system allows policy makers and farmers to use tip request messages.

    - Chat CMP: in order to exchange messages.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R29**:. the system allows policy makers to de-highlight a tip.
    - Settings CMP: in order to de-highlight a tip.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R30**: the system adds the two mandatory visits to an agronomist's visit plan when a new farmer signs up.
    - VisitPlan CMP: in order to add the new visits.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R31:** the system allows government admin to log-in.
    - Acces CMP: in order to login.
    - Security CMP: in order to validate input data.

- **R32:** the system allows policy makers to visualize farmers KPIs.
    - KPI CMP: in order to visualize KPIs.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R33**: the system allows agronomists to become a super agronomist.
    - Setting CMP: in order to select the option to be a super agronomist.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R34:** the system allows policy makers to visualize agronomist related KPIs.
    - KPI CMP: in order to visualize KPIs.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R35:** the system allows agronomists to visualize agronomist related KPIs.
    - KPI CMP: in order to visualize KPIs.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R36**: the system allows government admins to manage authorization codes.
    - Management CMP: in order to create auth codes.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R37**: the system allows government admins to reset an agronomist or policy maker's account.
    - Management CMP: in order to reset an account.

    - Security CMP: in order to check the access token; in order to validate input data.

- **R38**: the system allows government admins to delete a farmer's account.
    - Management CMP: in order to delete an account.
    - Security CMP: in order to check the access token; in order to validate input data.

- **R39**: the system allows government admins to delete posts or answers.
    - Management CMP: in order to delete an item.
    - Security CMP: in order to check the access token; in order to validate input data.

4.2. Other attributes

4.2.1. Reliability

The system must be highly reliable in order to detect runtime problems, therefore exceptions need to be handled very carefully.

4.2.2 Availability

High availability is required, it can be obtained by scaling the system horizontally, so adding more machines to support the workload, and using one or many redirect servers to route the traffic based on the area from where the request comes from. The database needs to be partitioned and replicated. With adding machines working in parallel and enlarging the replication factor of the data, the availability can be tuned as much as needed.

4.2.3. Security

The security can be guaranteed using the HTTPS (Hypertext Transfer Protocol Secure), which is a version of HTTP where encrypted HTTP data is transferred over a secure connection. By using secure connections the privacy and integrity of data are maintained.

4.2.4. Maintainability

The maintainability of the system is guaranteed through the action and the functionalities of the government admin, as well as the action of an IT support team of developers. The government admin maintains the system in terms of managing the software functionalities, moderating the forum and handling the access of users, as well as checking if new technologies can be integrated with the already implemented software.

The IT support team will be needed during all the life-cycle, periodically checking and testing if new hardware components might be integrated with the pre-existing architectures, making sure that the system is capable of integrating them. Moreover, deprecated code should be replaced as quickly as possible.

4.2.5. Portability and Usability

Portability is achieved with the web app hosted into a web server, so available using a browser and not directly associated with an operating system distribution.
Both portability and usability are reached thanks to the “mobile first” approach, which consists in designing a product for a mobile device first, then expanding its features in order to create a desktop version of the product.
The system guarantees usability achieved through a simple interface that allows to reach every functionality by following only one path, therefore the navigation is made very fluent. Furthermore, following the Material Design guidance helps to develop an usable system since the components are very common and widespread.

5. IMPLEMENTATION, INTEGRATION AND TEST PLAN

5.1. Implementation

The nature of the problem is suited for a bottom up approach for development, since there are many parts of the system that are similar to others and this approach can maximize the reusability of already written code, therefore speeding up the process.
DREAM is a system that needs to be widespread among farmers of Telangana, this process needs time and, to facilitate its implementation, it will be carried out in 3 cycles, that will produce a working version of the system with iteratively more and more functionalities.
After the first release the farmer can start to get familiar with DREAM. After the second release agronomists can start to work with the farmer who is already using DREAM, while the policy makers will start using the system after the third release, when some data is ready to be processed and presented.
To achieve what just described, the development team has to be focused on the functionality of the system and not on the single building components, because all the components will go through multiple cycles of development.

In the following, all the product functions (available in details at *RASD section 2.2*) are associated with a development cycle, since most of the product functions are available to more than one user; when a new cycle begins, all the previously developed functions have to be extended to the new user functionality considered in the new cycle if needed, since each cycle adds a new user type in the system.

Cycle 1 (Farmer and Government-admin cycle)

- Signing up
- Logging in
- Filling in the harvest-report
- Sending a HR to farmers
- Receiving a HR by a farmer
- Using HR-messages
- Posting a tip
- Posting a question
- Voting inside the forum
- Managing authorization codes
- Deleting a farmer account.

Cycle 2 (Agronomist cycle)

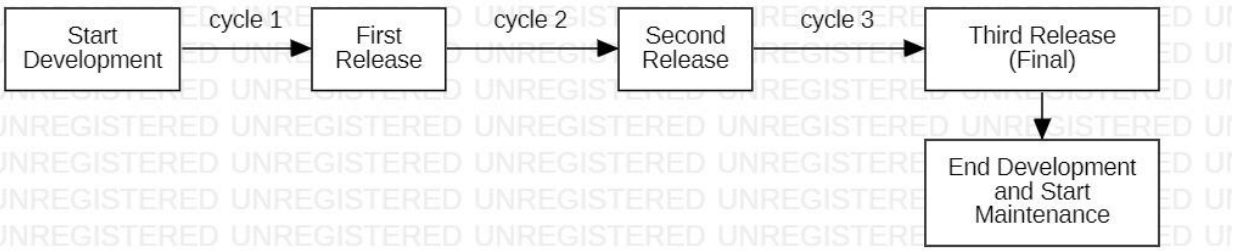
- Receiving a HR by an agronomist

- Creating visits for a new farmer account
- Posting a FAQ
- Posting an answer to a question
- Sending a HR to agronomist
- Checking relevant information
- Initialization of a visit plan
- Giving a contact point for a visit
- Adding a visit
- Changing a visit state
- Filling the visit-report

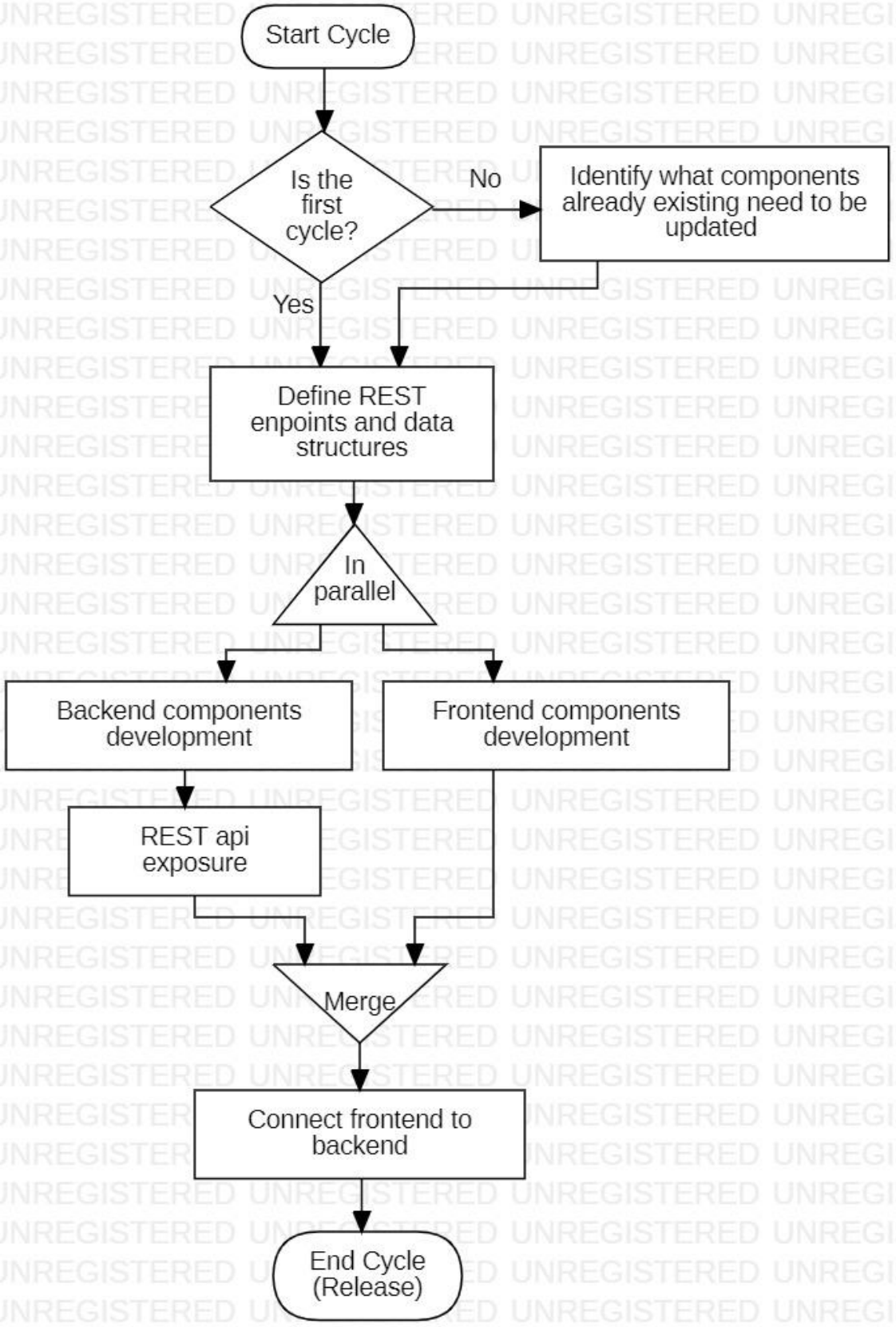
Cycle 3 (Policy Maker cycle)

- Giving availability to be a super agronomist
- Delegation of a visit to a super agronomist
- Monitoring of farmers' performance
- Requesting a tip
- Promoting a tip to star tip
- Monitoring of agronomists' performance
- Resetting a policy maker or agronomist's account

State chart of the project flow



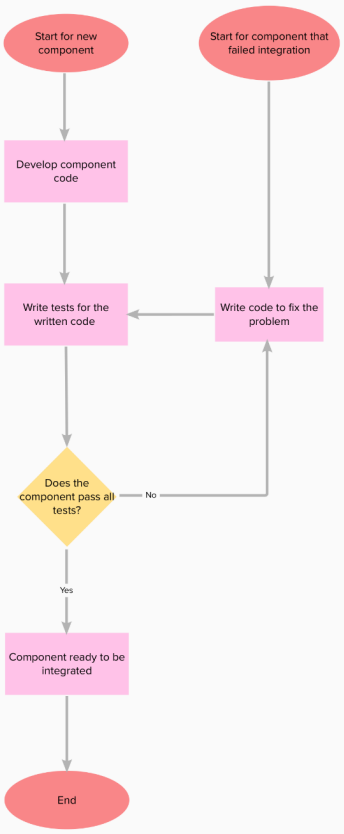
Cycle flow chart



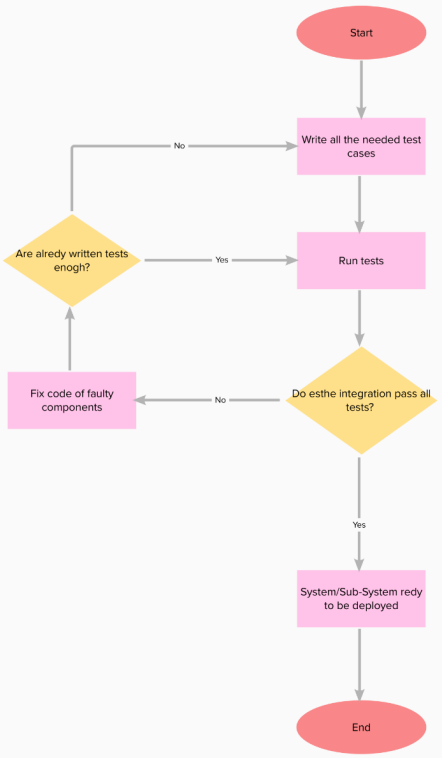
5.2. Testing and Integration

Since all the components will possibly go through multiple iterations of development, the testing has to be carried out just after the code is written. A component cannot go into production with non tested parts and, in addition, anytime modifications/fixes are applied to the code the tests have to be updated.
After all the components are ready, they can be integrated and then tested again with integration tests. The development team can choose to divide the system in an arbitrary number of subsystems and iteratively repeat the integration process, firstly for components and then for subsystems. It is important to notice that if a component need to be modified/fixed after the integration tests fail, the component tests have to be updated and only after the integration tests can be repeated.

DEVELOPMENT AND TESTING FOR COMPONENTS



COMPONENTS INTEGRATION



6. EFFORT SPENT

	S2	**S3**	**S4**	**S5**	**Meetings**
Careddu Gianmario	1h 30min	28h 30min	1h	5h	6h
La Greca Michele Carlo	27h 30min	1h	1h	1h	6h
Zoccheddu Sara	26h	1h	2h 30min	1h	6h

7. REFERENCES

1. https://thefactfactor.com/facts/pure_science/biology/crops/2082/
2. https://smartbear.com/learn/performance-monitoring/what-is-mobile-first/
3. https://en.wikipedia.org/wiki/Don%27t_repeat_yourself
4. https://developers.google.com/web/updates/2019/02/rendering-on-the-web?hl=zh-cn
5. https://simplicable.com/new/thin-client-vs-thick-client
6. https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
7. https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

