VILNIAUS UNIVERSITATAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

PROGRAMŲ SISTEMŲ INŽINERIJA II

TREČIASIS LABORATORINIS DARBAS
CHEAP EFFORT

Tedas Grinkevičius
Klaidas Sinkevičius
Pijus Zlatkus
Vadim Čeremisinov

Vilnius, 2023

Contents

1 First laboratory changes

1.1 Context

In our first laboratory work, we did a business analysis and provided our functional and non-functional requirements for a recipe-sharing platform called CheapEffort. With the detailed feedback that we got after evaluating our work, we decided to change and correct some of the already existing requirements while also adding some new functional and non-functional requirements and expanding the overall list.

1.2 Requirements

The requirement change doesn't include new requirements introduced later with the change request. Newly added requirements with this change **are highlighted with gray color**. In addition, the **white color** ones are edited.

1.2.1 Functional Requirements

Nr.	Requirements
FR 1	The system has authentication for users and chefs. When a user opens a website, there are a few options on how to use a website: register as a new user and log in to their account or use all functionalities as is. Only registered users have access to some functionalities like adding, editing and removing their recipes or subscribing to chefs.
FR 2	The system shows a pop-up warning window with information on what is wrong and a button to close it after a user makes a mistake while entering data and clicks proceed.
FR 3	The system displays all the newest recipes on the home page with a picture, which, when clicked on, redirects the user to the recipe's information page.
FR 4	The system allows a chef to create an infinite number of recipes.
FR 4.1	Every recipe must have a description from 50 to 500 words.
FR 4.2	Every recipe must have from 2 to 7 images.
FR 4.3	Every recipe must have a title and tags assigned to it.
FR 4.4	Every recipe must have a list of 1 to 50 required products.
FR 4.5	Every recipe must be rated on a scale of 5 stars determining how hard it is to make it.
FR 4.6	Every recipe could be marked as premium, and that way would be accessible by subscribing to the chef.
FR 5	On the home page, the system shows recipes as is or uses a search filter before displaying them. The recipe search filter should be at the top of the recipes page.
FR 6	The recipe filter provides different options on how user can filter recipes.
FR 6.1	The recipe filter should have an option to filter recipes by recipe's title.
FR 6.2	The recipe filter should have an option to filter recipes by chef's name.

FR 6.3	The recipe filter should have an option to filter recipes by products' titles.
FR 6.4	The recipe filter should have an option to filter recipes by recipe tags.
FR 7	The system submits the edited recipe information after a chef clicks on the button "Edit" on the recipe's page.
FR 8	The system deletes created recipe after a chef clicks on the button "Delete" on the recipe's page.
FR 9	The system adds a rating to a recipe after a user clicks one of the rating buttons on a recipe's information page.
FR 9.1	The system downvotes a recipe after a user clicks a downvote arrow button on a recipe's page.
FR 9.2	The system upvotes a recipe after a user clicks an upvote arrow button on a recipe's page.
FR 10	The system sorts recipes descending by popularity as a default. It also provides sort mechanism to sort recipes by upvote count or by creation date when a user chooses a different option on the recipe search filter.
FR 11	The system subscribes user to a chef when a user clicks "Subscribe" button on a chef's profile.
FR 11.1	Prior to subscribing to a chef, the system requests payment of the chef's set price.
FR 11.2	Subscription to a chef gives access to all chef's premium recipes.
FR 12	The system posts a review on a recipe when a user fills review form in recipe's page.
FR 13	The system collects Kudos points for every user and chef. Kudos points later can be exchanged for different premium-related content discounts.
FR 13.1	The system gives a 5% discount for one subscription to a user after the user with 400 Kudos points clicks on the "Claim" button on the user's profile page.
FR 13.2	The system gives a 10% discount for one subscription to a user after the user with 1000 Kudos points clicks on the "Claim" button on the user's profile page.
FR 13.3	The system gives a 15% discount for one subscription to a user after the user with 2000 Kudos points clicks on the "Claim" button on the user's profile page.
FR 14	The system allows users to get Kudos points for receiving subscribers or subscribing to another chef.
FR 14.1	The system adds 3 Kudos points to a user after a user subscribes to a chef.
FR 14.2	The system adds 5 Kudos points to a user after he or she receives one subscriber.
FR 15	The system removes Kudos points from a user if he or she acts against community standards.
FR 15.1	The system deducts 10 Kudos points from a user for posting fake and inappropriate recipes.
FR 15.2	The system deducts 5 Kudos points from a user for commenting or reviewing recipes inappropriately.

1.2.2 Non-functional Requirements

1.2.2.1 Performance

NFR 1	Every server request must be processed within 5 seconds.
NFR 2	Website must have a responsive design, meaning it should be optimized for all devices, e.g., phones, tablets, PC's
NFR 3	Database queries must be processed in less than 200 milliseconds.
NFR 4	The system should have separate third-party performance monitoring tools for identifying potential issues and scalability problems.
NFR 5	The server must work without crashing and solve issues automatically.

1.2.2.2 Scalability

NFR 6	The system must be designed according to SOLID principle.
NFR 7	Modules must be reused as much as possible.
NFR 8	The system must ensure that multiple users and chefs can use the system concurrently at the same time without experiencing errors or slowdowns.
NFR 9	The system should log error messages to the database when errors occur to help diagnose and resolve issues quickly.
NFR 10	The system should be updated and deployed continuously without experiencing any downtime.

1.2.2.3 Security

NFR 11	API must have JWT (JSON Web Token) token authorization.
NFR 12	All users' passwords must be encrypted with SHA512 hash algorithm.
NFR 13	The system should manage users' sessions and leave them logged in to the system unless a user logs off.
NFR 14	The system must have a data backup in case of a security attack.

1.2.2.4 Usability

NFR 15	All warnings and errors must be displayed in a pop-up window.
NFR 16	The website must be accessible with using only mouse and would require keyboard only for registration, reviews and adding a recipe.

2 Change Request

Our change request in this laboratory work is the admin role. Admin can approve or reject recipe requests, modify and delete existing recipes. In addition, the admin has the ability to oversee platform users and access metrics related to recipe activity. We created three implementation alternatives: NOP (No Operation), Simple and Correct.

3 Solution Alternatives

3.1 NOP (No Operation)

As NOP solution alternative suggests - no action would be taken. No admin role would be implemented. The platform runs without a dedicated admin, every recipe is approved, only recipe creators are responsible for the recipe requests, modification, deletion. The system would be decentralized, content could be inappropriate. It would not require any additional resources or costs, as the current status quo would be maintained without any changes. Additionally, the metrics related to recipe activity would be restricted and only visible to the users who have created the specific recipes. This no operation alternative may be perceived as a cost-effective solution that promotes decentralization, which some users may appreciate. However, it may also pose challenges as content cannot be effectively monitored, potentially resulting in the posting of undesirable or negative content.

3.2 Simple

As Simple solution alternative suggests - a straightforward and uncomplicated solution that is easy to understand should be implemented. This implementation should not require significant resources or effort. Its main goal is to be simple and create the basic functionality that could form the admin role.

We could start by creating a separate user account with admin role permissions. This account would have the basic functionalities for approving or rejecting recipe requests. It would be enough to monitor the content.

On the frontend side, admin would have access to a specific dashboard exclusively available to admins, where only requests for recipes would be awaiting review.

On the backend, we would implement simple authentication and authorization mechanisms to ensure that only users with admin role can access the admin functionalities. We would use a flag field to indicate whether the user has admin privileges. Also, we would create API endpoints for the admin functionalities, such as rejecting and approving.

In the database, we would add an additional field – flag, for indicating whether user is an admin.

3.3 Correct

As Correct solution alternative suggests – the most accurate and appropriate solution based on resources and time should be implemented. It may require specialized expertise, research or other resources. Its main goal is to create a complete admin role implementation in our platform.

We could start by creating a separate user account with admin role permissions. This account would have the necessary functionalities for approving or rejecting recipe requests, modifying or deleting recipes and overseeing platform users.

On the frontend side, admin would have access to a specific dashboard exclusively available only to admins, where recipe requests and metrics would be listed. A user supervision dashboard would also be created for overseeing users and their metrics.

On the backend, we would implement authentication and authorization mechanisms to ensure that only users with admin role can access the admin functionalities. We would use standard authentication protocol such as JWT (JSON Web Tokens). In addition, we would create API endpoints for the admin functionalities, such as rejecting, approving, deleting or modifying recipes.

In the database, we would add additional fields to track metrics related to recipes and ensure that user credentials and permissions are stored securely.

3.4 Solution Evaluation Matrix

Solution Evaluation Matrix			
	NOP	Simple	Correct
Cost	3	2	2
Time	3	2	1
Accuracy	1	2	3
Scalability	3	3	3
Reliability	1	2	3
Functionality	1	3	3

1 Fig. Solution evaluation matrix

In this matrix we utilized an evaluation system ranging from 1 to 3, with 3 representing the highest score and being the best and 1 being the worst. The best solution with 15 points is Correct. In addition, based on our specific application requirements, Correct is the most suitable to implement, even though it may not have scored the highest in time and cost, but considering our resources, these benchmarks are not the most critical factors for our decision. For example, other benchmarks, such as accuracy and reliability, are of greater importance to us on our platform.

4 ICONIX

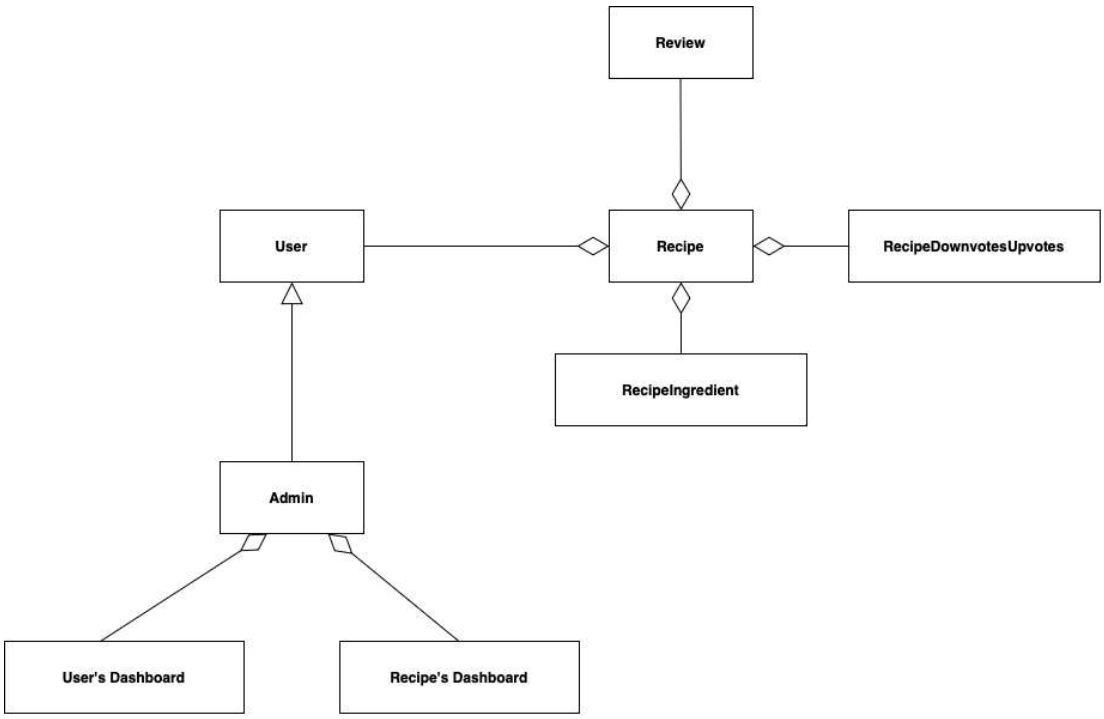
4.1 Context

The goal of our program is to create a convenient and easy to use platform for sharing recipes or following favorite chefs. Users can filter the recipes by most important information, like ingredients, chefs or tags. In pursuit of making our platform even more professional and user friendly, we decided to implement an admin role, which will manage recipes and have access to different metrics. Admins will ensure that the platform is free from trolls or inappropriate information in recipes. The newly created recipe becomes accessible for everyone only after admins approve it. Also, admin has permission to edit the recipe and change this information: recipe's title, ingredients, instructions, picture, number of steps, the time it takes, date, amount of upvotes and downvotes.

4.2 Static Model

4.2.1 Domain Model

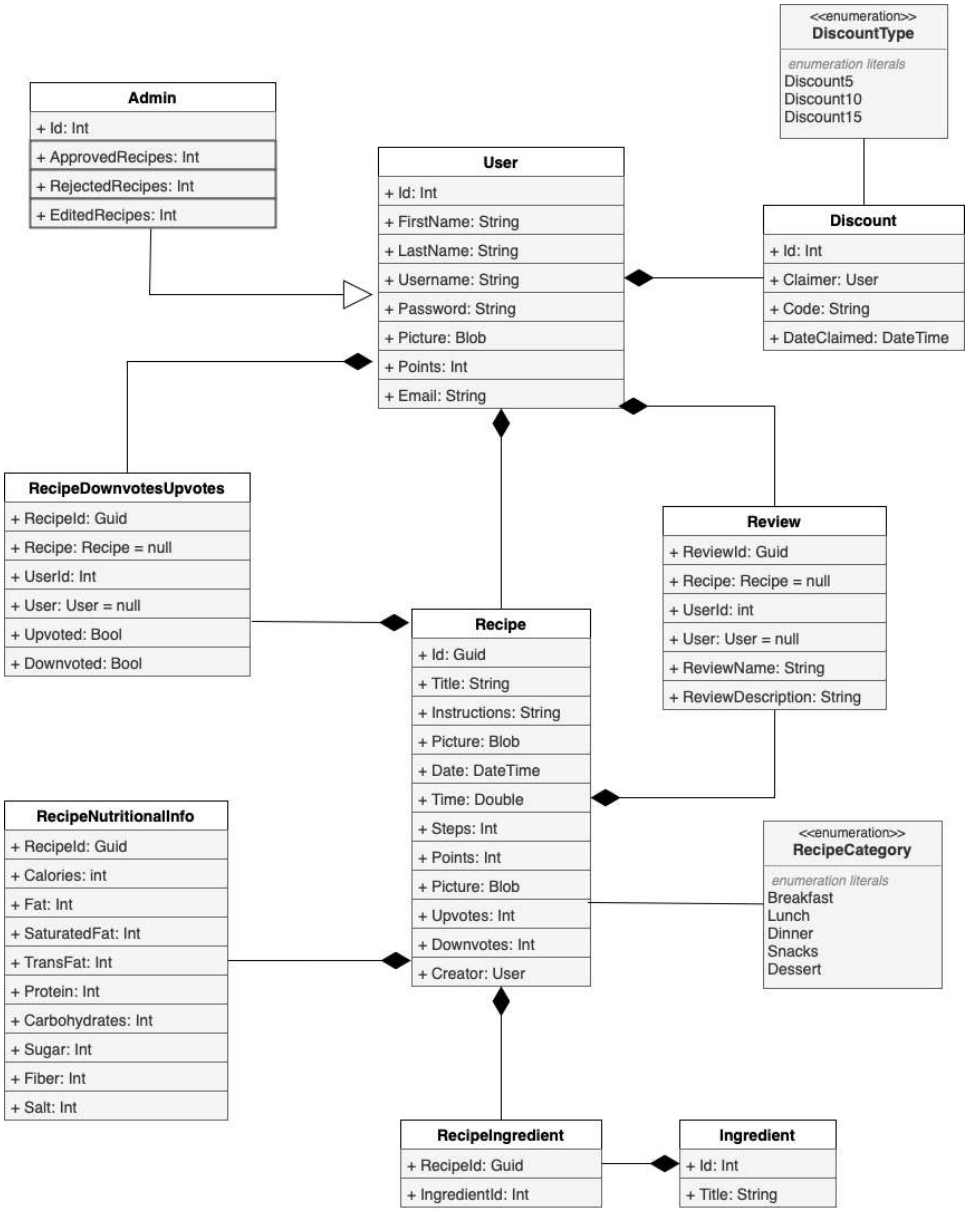
In this domain model, a user creates recipes and adds ingredients. Reviews and recipe downvotes or upvotes are then added by other users later on. For admin to access the recipe's information, it inherits from User. This way it becomes a user with more rights: to access other users' overseeing pages and recipes' administration pages. These are separate classes, that cannot exist without admin.



2 Fig. Change request's domain model diagram

4.2.2 Class Diagram

This class diagram presents the main classes and their attributes. In this iteration, Admin class was added to earlier class diagram to show the relationship between Admin and User. Admin inherits from User, this way, he or she is a regular user, however with additional rights. Extra attributes were added to keep track of information which is used to keep track of admin's activities: ApprovedRecipes, RejectedRecipes and EditedRecipes. All of these attributes can be seen by other admins when navigated to the admin's page.



3 Fig. Change request's class model diagram

4.3 Requirements

4.3.1 Functional Requirements

Nr.	Requirements
FR 1	System has an authentication for the admin role. When a user logs into his/her account, only the user that has admin rights should be granted access to admin role's functionalities.
FR 2	Access to the admin role can be granted manually by the organization.
FR 3	Recipes must be approved by the system before appearing on public recipes list and chef's profile. When an admin is looking at a recipe, it can be approved by clicking "Approve" button, located in recipe's admin action section.
FR 4	Recipes can be rejected by the system. When an admin is looking at a recipe, it can be rejected by clicking "Reject" button, located in recipe's admin action section. When the recipe is rejected, it should be removed from the database.
FR 5	Recipes can be modified by the system. When an admin is looking at a recipe, it can be edited by clicking "Edit" button, located in recipe's admin action section. After a request is received by the system, the admin should be redirected to recipe's edit page and allow to edit and save the recipe.
FR 6	Recipes can be removed from the system. When an admin is looking at a recipe, it can be approved by clicking "Remove" button, located in recipe's admin action section. After a request is got from admin, the recipe should be removed from database by the system.
FR 7	The system should provide metrics related to recipe activity exclusively to an admin. When any recipe information page is opened by an admin, a dashboard with metrics related to the recipe should be provided.
FR 7.1	Information about recipe's request history should be provided by recipe's dashboard: date and time when it was added, edited, approved or rejected.
FR 7.2	Information about recipe's rating activity should be provided by recipe's dashboard: when and who downvoted or upvoted that recipe.
FR 7.3	Information about recipe's activity of how many users have viewed that recipe should be provided by recipe's dashboard.
FR 8	The system should provide a user supervision dashboard related to the user's activity exclusively to an admin. When any user's profile page is opened by an admin, a dashboard with metrics related to the user should be provided.
FR 8.1	The user's dashboard should contain details regarding their activity history, such as the recipes they added, removed, edited, upvoted, downvoted and commented on, along with the corresponding dates and times.
FR 8.2	If a user has an admin role, the user dashboard should include supplementary details regarding management activity history. This information should contain the recipes that the admin has approved, rejected, edited, or removed, along with the corresponding dates and times.

4.3.2 Non-functional Requirements

4.3.2.1 Performance

NFR 1	All user management and metric dashboards must be actively updated within 10 seconds.
NFR 2	Dashboards must have a responsive design for optimized usage for different devices e.g.: phones, tablets, PC's.
NFR 3	The system should respond to admin requests within 5 seconds to ensure that administrators can perform tasks quickly and efficiently.
NFR 4	The system must ensure that multiple administrators can use the system concurrently at the same time without experiencing race conditions or slowdowns.

4.3.2.2 Security

NFR 5	Admin authentication must use JWT (JSON Web Tokens) for secure login process.
NFR 6	Admin password must be encrypted with SHA512 hash algorithm.
NFR 7	The system should manage admin sessions and give admins 10 minutes between actions before active session time out.
NFR 8	Admin should update the password at least once every 3 months.

4.3.2.3 Usability

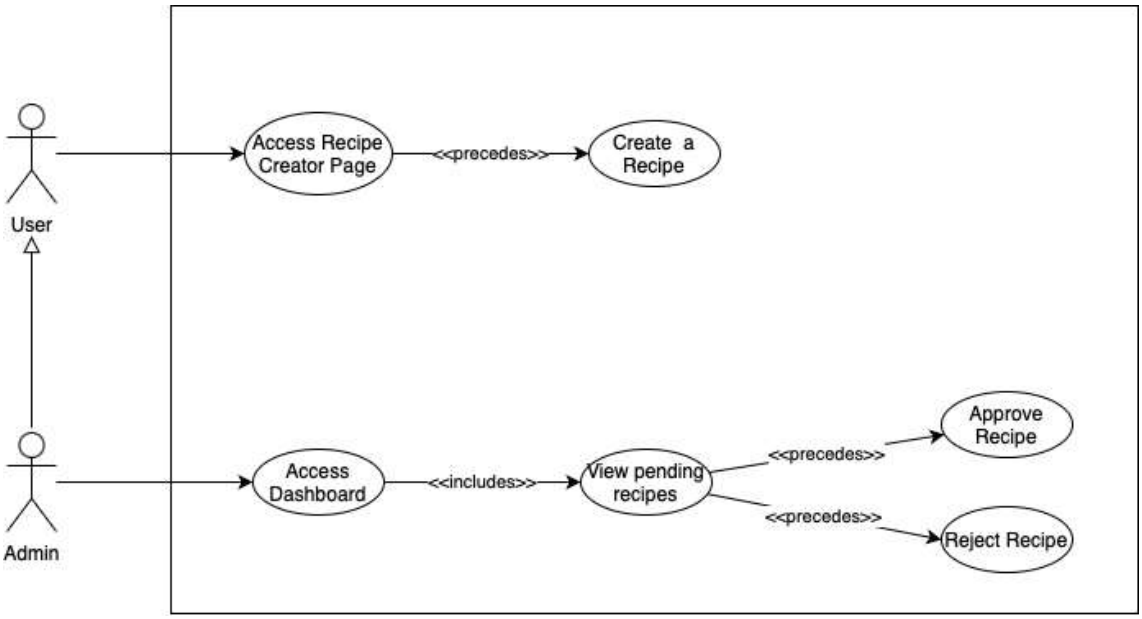
NFR 9	All warnings about session time out must be displayed in a pop-up window.
NFR 10	The system must inform admin about expiring password every session when 1 week is left before expiring. Warning must be displayed in a pop-up window.
NFR 11	All admin's actions must be accessible using only mouse and would require keyboard only for registration or editing recipes.

4.4 Use Cases

Actors: User, Admin.

Admin inherits from User, since an admin will be a user with additional abilities. The role of becoming admin is given by the administration of the website.

4.4.1 Recipe's approval and rejection

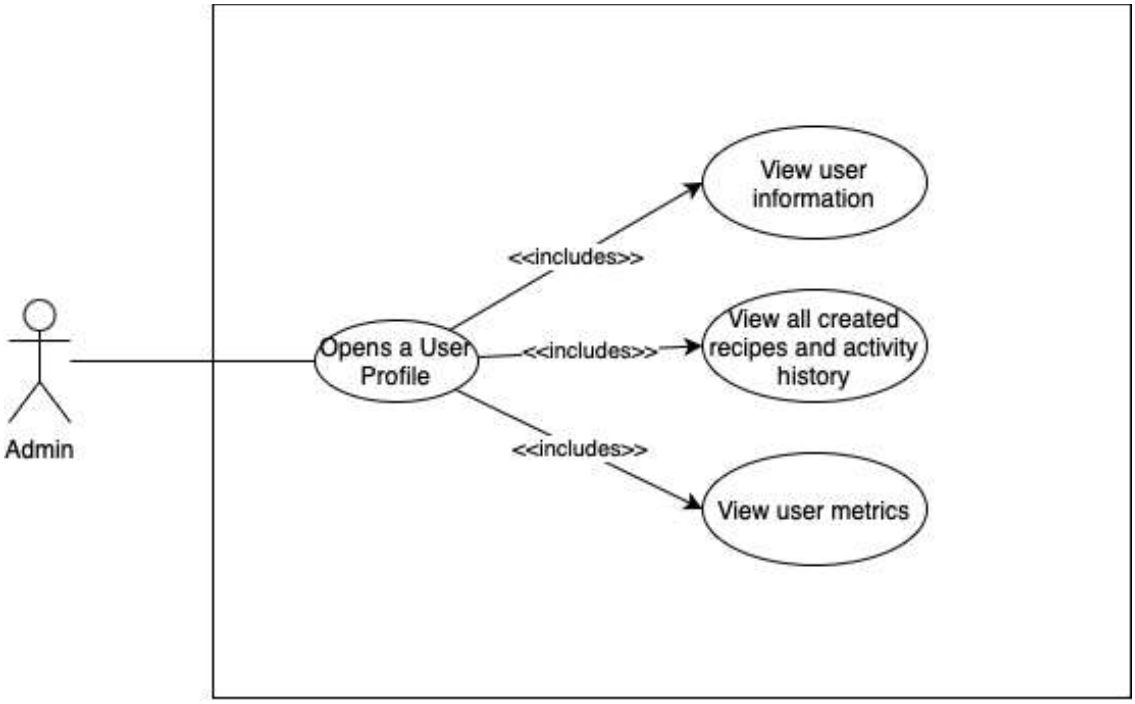


4 Fig. Recipe's approval and rejection use case diagram

This diagram represents the Admin and User behavior when creating a recipe. After a user opens the recipe creator page, fills in all the required information and clicks a button "Create", the recipe is then put into a pending state.

Whenever an admin opens the pending recipes dashboard, the newly created recipes will be displayed there in a list. Then, an admin has two choices: to accept the recipe by clicking button "Approve", or decline it, by clicking "Reject". If admin clicks "Approve" button, the recipe automatically becomes public, and everyone can access it. On the other hand, if admin decides to reject the recipe, user can see that the recipe has not passed administration and is rejected. He can try editing it and trying again or deleting it from the list.

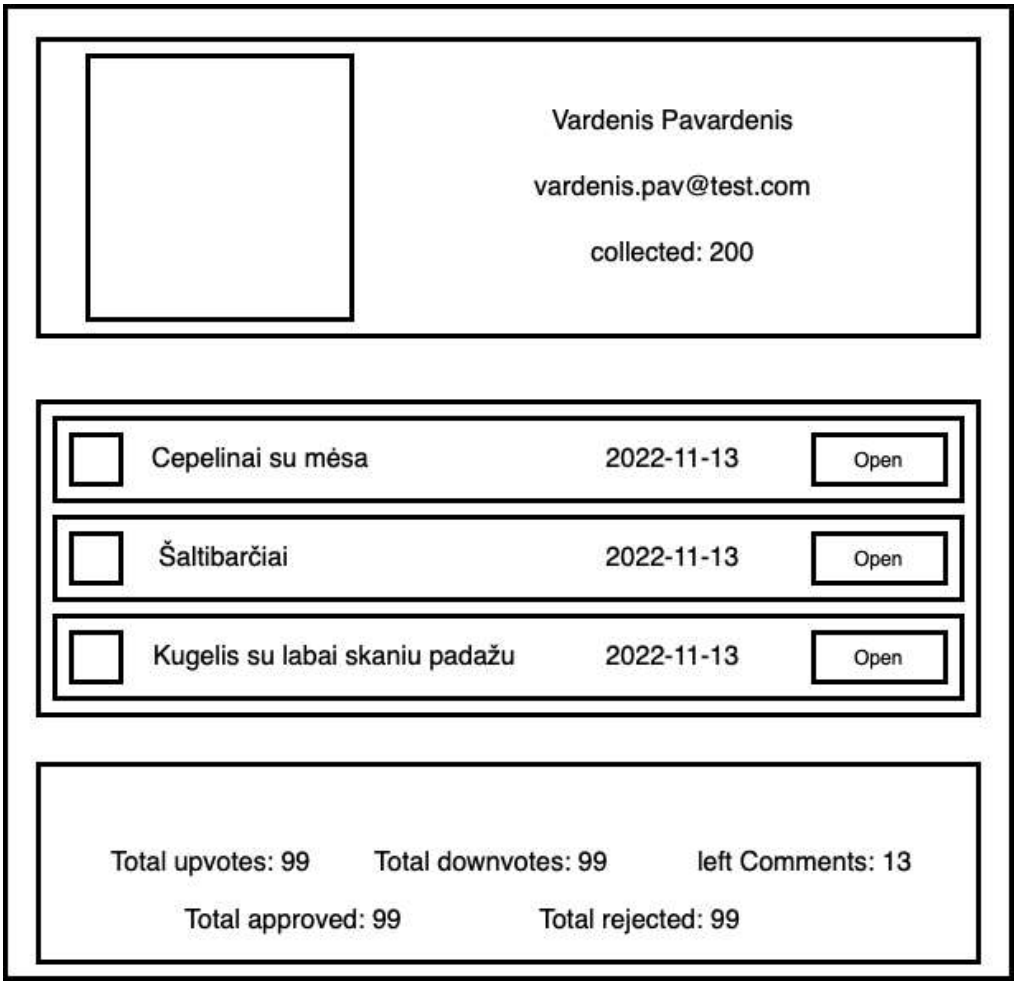
4.4.2 Admin's access to user information and metrics



5 Fig. Admin's access to user information and metrics use case diagram

This diagram represents what information admin can see when a user profile page is opened. After clicking on a name or on a picture of a specific user, that user's profile page is opened. That page is only accessible for admins, so if a user clicks on another user, nothing happens.

In the top of newly opened page, the admin can access Users profile information: first name, last name, username, picture and email. Below this information, a list of his or her recipes is shown. Every recipe has an image, title and these metrics: how many upvotes the recipe has, how many downvotes and how many reviews. After the list, the admin can also see overall user metrics, which are: how many recipes the user has created, how many recipes were deleted, numbers for how much upvotes or downvotes the user gave to others and lastly how many reviews he or her has left on other recipes.



6 Fig. Admin's view after opening user's profile page

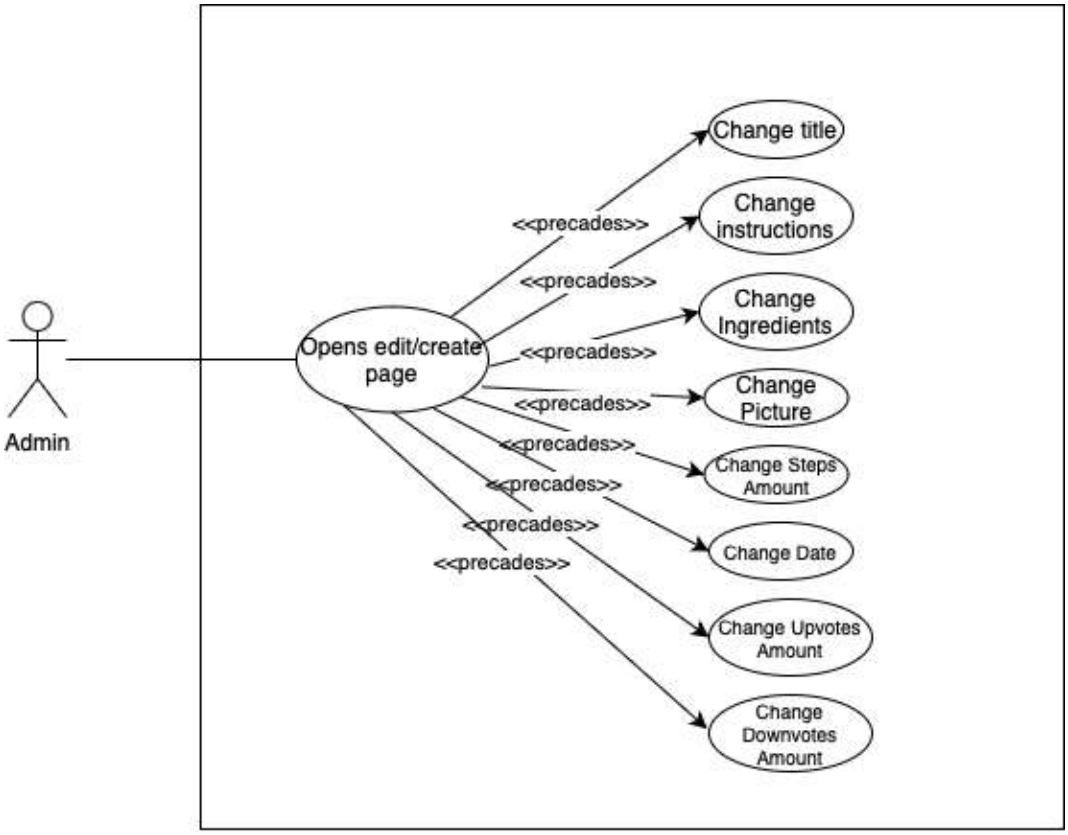
In this image, we created a wireframe for a user’s overseeing page, which is only accessible for users with admin role. It is opened after clicking on an image or username of a user. A regular user should not have any way of entering other’s overseeing pages.

The first section represents the base user information: an image, name, surname, email and how many Kudos points he or she has collected. When creating this feature, we came to a decision to not let admins edit user details, since it could open possibilities for inappropriate admins to troll users.

Below that, a section where all user’s activity history can be seen. A list, in which every recipe that a user has created, edited, deleted, upvoted, downvoted or commented is mapped. Every table element has a button “Open”, which opens a recipe management page. The recipe row is built out of these elements: an image, title of the recipe, date, and the “Edit” button, which is the only clickable component.

The last section in the page is overall user’s metrics. A number of total upvotes, total downvotes and left comments can be seen. All of these values are fixed, which means that admin cannot change them. The last two metrics, which are total approved and total rejected can only be seen in dashboards of users with admin roles. The idea behind these is that other admins can track their colleagues' work, this way ensuring that admins are active and monitoring the flow of recipes.

4.4.3 Recipe's edit

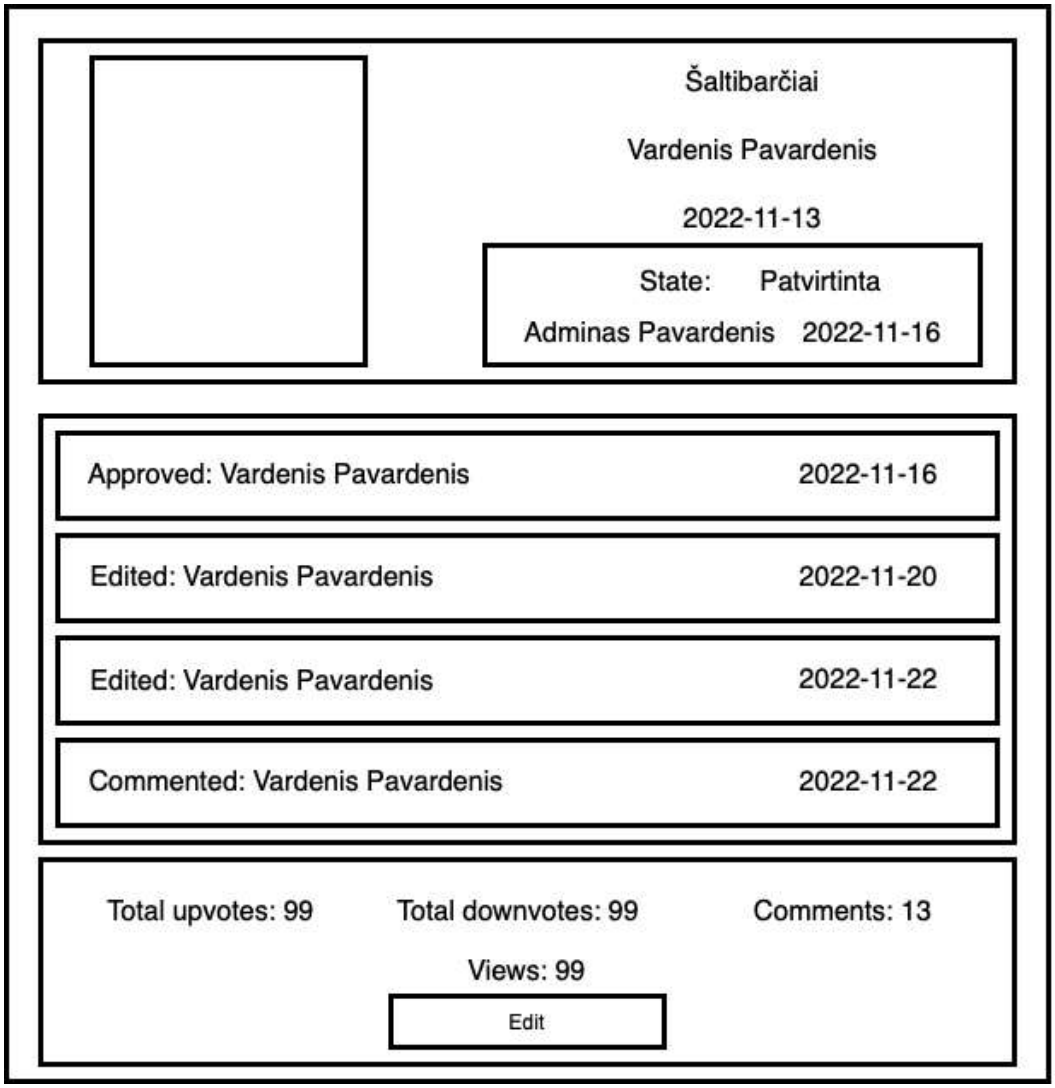


7 Fig. Admin's recipe edit option use case diagram

This diagram represents all the available edit choices the admin has when edit/create page is opened. The page can be reached via recipe's administration page, when clicked on "Edit" button. These use cases are only available for admins, which means that users cannot use this functionality.

When the page is opened, it looks the same as it is when creating a recipe, however, the fields are now prefilled with recipe's information. Admin can change whatever information is needed and must click "Save" to save the changes. Edited recipes are instantly accessible to all users, meaning it does not need approval from another admin.

Admin has these choices: to change amount of downvotes and upvotes, the date when it was created, amount of steps it requires, picture, ingredients and the title. Not all fields are required to be edited, the system will automatically detect changes and save them.



8 Fig. Admin's view after opening recipe's page

In the first section, image, title and date created will be displayed, alongside with the recipe's current state information: whether it was approved or rejected, by whom, and date when. Nothing is changeable, all information is only for gathering information.

Below that, a list of all the recipe's activities can be found. A table, in which each element represents one of these activities: Approved, Declined, Upvoted, Downvoted or Commented. Also the date of activity is shown. All this information is also not changeable.

The last section is overall metrics of the recipe. Amount of total upvotes, total downvotes, comments and views are displayed. In the bottom, "Edit" button can be seen. On click, it opens the same page as users use to create recipes, but this time, admin can change its information.

The recipe administration page has no editable fields. Its main purpose is to display all the main information, with also a link to create/edit page, which is already created.

4.5 Traceability matrix

Requirements	Use cases					
	Recipe's approval	Recipe's rejection	View user information	View activity history	View user metrics	Open edit/create page
FR 1	■	■	■	■	■	■
FR 2	■	■	■	■	■	■
FR 3	■					
FR 4		■				
FR 5						■
FR 6						■
FR 7			■	■	■	■
FR 7.1						■
FR 7.2						■
FR 7.3						■
FR 8			■	■	■	■
FR 8.1				■		
FR 8.2			■			

5 Project Plan

5.1 Project Goal

Implement new functionality: introduce administrator's feature who has the possibility to manage recipe content posted on the platform. In addition, the administrator will be able to manage platform users and see different metrics.

5.2 Project Scope

The functionality of the platform after implementing the new feature:

1) **Admin can approve and reject new recipes.**

When a chef creates a new recipe, the recipe is initially in a pending state. The admin will be able to view a list of pending recipes and take action on them, either approving or rejecting them.

2) **Admin can modify recipes.**

Admins should be able to modify existing recipes, such as correcting errors or adding additional information.

3) **Admin can delete recipes.**

Admins should be able to delete existing recipes from the platform.

4) **Admin can manage users and see metrics.**

Admin should be able to manage the users of the platform and see metrics that are related to recipe requests.

5.3 Project Stages

5.3.1 Admin Authorization/Authentication

- Create an Admin table in the database that contains admin credentials, such as username and password, and a boolean value 'isAdmin' to indicate if the user is an admin.
- Use JSON Web Tokens to authenticate the admin when they log in. The JWT should contain the 'isAdmin' boolean value to verify their admin status.
- Create an Admin middleware that will validate the JWT and check if the user is an admin before allowing access to the admin-only functionalities.

5.3.2 Admin Approval/Rejection

- Add a new API endpoint for retrieving a list of pending recipe requests.
- Create a new database table to store the recipe request status (Pending/Approved/Rejected).
- Add a new API endpoint for updating the recipe request status.
- Implement logic to check the JWT token of the admin and ensure they are authorized to approve or reject recipe requests.
- Add a new field in the recipe table to store the admin's approval status.
- Update the API endpoint for retrieving recipes to only return approved recipes.

5.3.3 Admin Recipe Modification

- Add a new API endpoint for retrieving a single recipe by its ID.
- Add a new API endpoint for updating an existing recipe.
- Implement logic to check the JWT token of the admin and ensure they are authorized to modify recipes.
- Add validation to the API endpoint for updating a recipe to ensure that the data is valid.
- Update the API endpoint for retrieving recipes to include a "canModify" field that indicates whether the logged-in user is an admin and can modify the recipe.
- Update the recipe table to include a timestamp for the last modification time.

5.3.4 Admin Recipe Deletion

- Add a new API endpoint for deleting a recipe by its ID.
- Implement logic to check the JWT token of the admin and ensure they are authorized to delete recipes.
- Add a new API endpoint for retrieving a list of deleted recipes.
- Update the recipe table to include a timestamp for the deletion time and a flag indicating whether the recipe is deleted.

5.3.5 Other steps

5.3.5.1 Frontend development

The Frontend development plan for this project will focus on creating a user-friendly interface for the "Admin role" features, including the admin login page, approval, modification, deletion pages and error handling. The plan will also include the development of TypeScript React components that use the backend API endpoints to ensure smooth data flow and user experience. Lastly, proper testing and deployment strategies will be employed to ensure the reliability and efficiency of the Frontend components.

5.3.5.2 User Management implementation

The development plan for the User Management will include designing and implementing a user interface for admins to manage user accounts. This includes functionality to create, update, and delete user accounts, as well as the ability to grant and revoke admin privileges.

5.3.5.3 Metrics Implementation

The implementation of Metrics will include designing and implementing a logging and monitoring system that tracks user interactions and performance metrics related to recipe requests. The data collected will be stored in the database and then analyzed to create reports and visualizations. Finally, the reports and visualizations will be presented to stakeholders in a meaningful way to help them make informed decisions about the platform.

5.3.5.4 Error Handling

It is important to handle errors properly to provide good user experience. The backend API should return appropriate error codes and error messages if an error occurs during a request. The Frontend should display error messages to the user if an API request fails. (Warnings that server is down or other permission related warnings).

5.3.5.5 Performance

The recipe platform is expected to handle a large number of users and recipes, so it is important to consider performance when implementing the "Admin role" feature. It is important to optimize database queries and API requests to minimize the load on the server. Caching can also be used to reduce the number of database queries.

5.3.5.6 Testing

It is important to thoroughly test the "Admin role" feature before deploying it to production. Automated unit tests can be used to test the backend API endpoints, while manual testing can be used to test the Frontend. It is important to test different scenarios, such as approving, rejecting, modifying, and deleting recipes, as well as handling errors and access control.

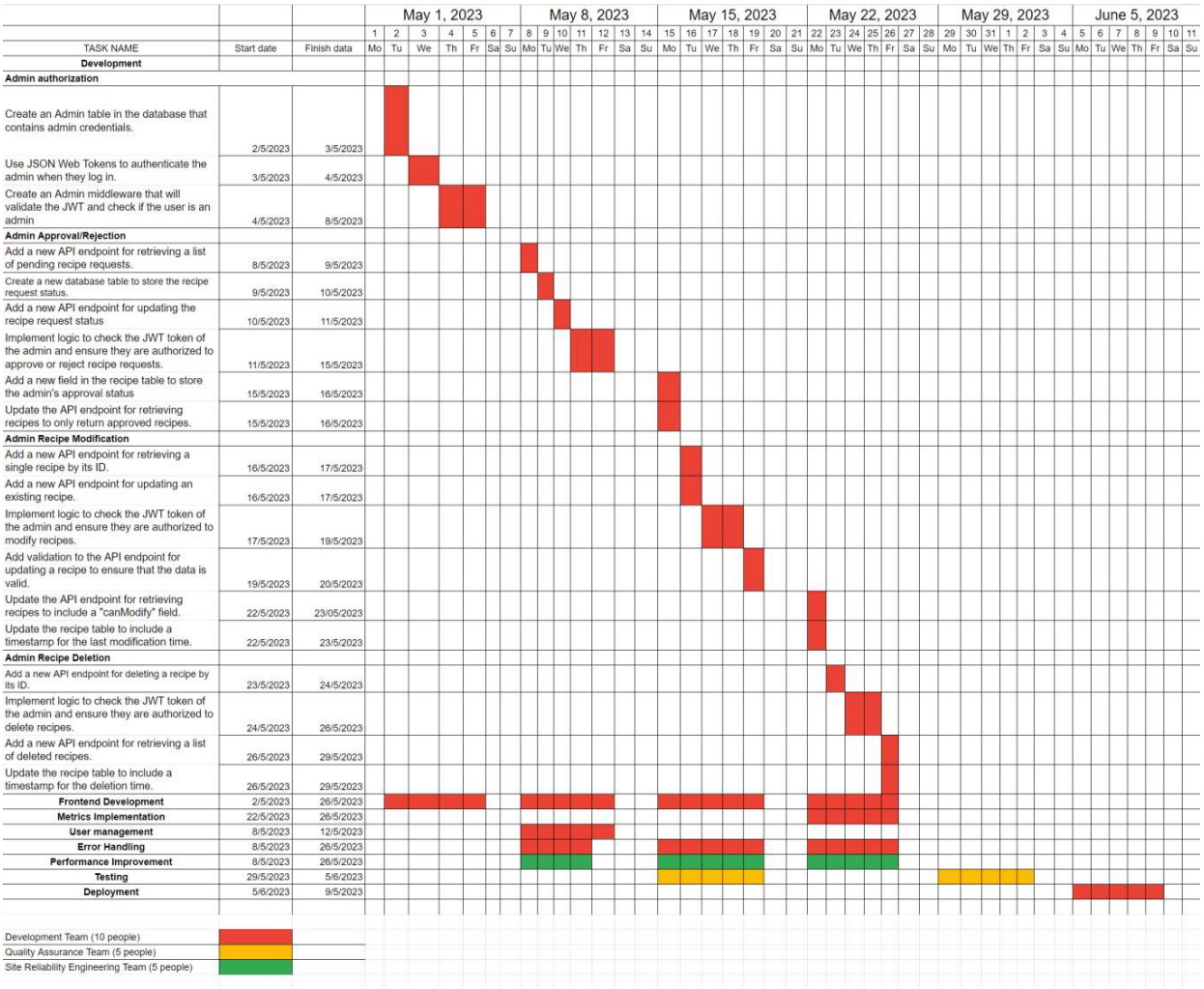
5.3.5.7 Deployment

The deployment of the feature will focus on setting up a production environment that is capable of handling the expected load of users and recipes. This includes configuring the server and database, deploying the Frontend and Backend components, and monitoring the environment for any errors or performance issues. Proper security measures and rollback plans will also be implemented to ensure the safety of user data and the reliability of the deployed features.

5.4 Gantt Chart

Gantt chart planning and synchronizing the implementation of the admin role's functionality.

.



9 Fig. Change request's Gantt chart