

Web Applications A.Y. 2023-2024
Homework 1 – Server-side Design and Development

Master Degree in Computer Engineering
Master Degree in Cybersecurity
Master Degree in ICT for Internet and Multimedia

Deadline: 29 April, 2024

Group Acronym	eagtf	
Last Name	First Name	Badge Number
Antolini	Gianluca	2080960
Bergamasco	Tommaso	2052409
Felline	Andrea	2090597
Frigato	Francesco	2086524
Risi	Emilio	2122841

1 Objectives

Dinner Dilemma is a web app designed to help users make the most of the ingredients they have at home. If you are staring at your open fridge or pantry wondering what to make for dinner, Dinner Dilemma has got you covered: by simply inserting the ingredients you have on hand, the app will give you a list of suggested recipes that use those exact ingredients.

2 Main Functionalities

The focal purpose of the website is to show recipes so all the main functionalities revolve around the creation and visualization of them.

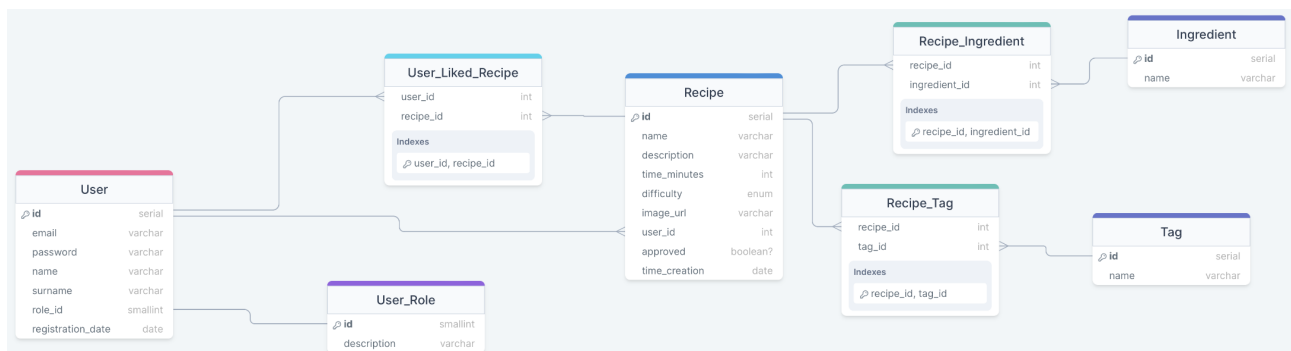
Users can search recipes by inputting some ingredients and they will receive a list of recipes that are made using (also) those ingredients. There is also a list of similar recipes to the searched ones, a button to search a random recipe and finally a list of top liked recipes.

Users can in fact register and use their account to manage their informations, create recipes and like other recipes to always keep them at hand.

There is another type of account that is of type "admin" and that is the one responsible for approving/disapproving recipes created by users (when they create a recipe, they have to wait for it to be approved to be visible to other people) and eventually ban users who violate the guidelines.

3 Data Logic Layer

3.1 Entity-Relationship Schema



The following is the description of the ER Schema of the database (PostgreSQL) for the web app.

- **User:** Represents users of the website.
 - id (primary key): INTEGER - the serial id associated to the user
 - email: VARCHAR(255) - user's email
 - password: VARCHAR(255) - user's password, hashed using SHA256
 - name: VARCHAR(255) - user's name
 - surname: VARCHAR(255) - user's surname
 - role_id (foreign key referencing User_Role): SMALLINT - the role of the user
 - registration_date: DATE - user's registration date
- **User_Role:** Describes the 5 defined roles that users can have: "admin", "user", "banned", "unverifiedUser" and "unverifiedAdmin"

- id (primary key): SMALLINT - id of the role
- description: VARCHAR(255) - title of the role
- **Recipe:** Stores information about recipes.
 - id (primary key): SERIAL - serial id of the recipe
 - name: VARCHAR(255) - recipe's title
 - description: VARCHAR(255) - description/preparation of the recipe
 - time_minutes: INTEGER - time needed for the preparation of the recipe (in minutes)
 - difficulty: VARCHAR(255) - enumerator for the difficulty of the recipe: "easy", "medium", "hard"
 - image_url: VARCHAR(255) - url for a thumbnail image of the recipe
 - user_id (foreign key referencing User): INTEGER - id of the user that created the recipe
 - approved: BOOLEAN - value that represents the status of the recipe: "NULL" = to be approved, "true" = approved, "false" = not approved
 - time_creation: DATE - date of creation of the recipe

We chose to keep the URL of the image in the database instead of directly memorizing the image to balance the memory load of the site (for which we are billed by the provider of the site) and to allow an easier portability, reconstruction and modification of the schema. This does not create any issues since, even if users input harmful or not appropriate data in a Recipe, it will not get approved and eventually will get deleted.

- **Ingredient:** Contains information about ingredients.
 - id (primary key): SERIAL - serial id of the ingredient
 - name: VARCHAR(255) - name of the ingredient
- **Recipe_Ingredient:** Represents the ingredients used in recipes.
 - recipe_id (foreign key referencing Recipe): INTEGER - id of the recipe
 - ingredient_id (foreign key referencing Ingredient): INTEGER - id of the ingredient

This table is used for the many-to-many relationship between recipes and ingredients, indicating which ingredients are used in each recipe.

- **Tag:** Contains information about tags that can be associated with recipes.
 - id (primary key): SERIAL - id of the tag
 - name: VARCHAR(255) - name of the tag
- **Recipe_Tag:** Links recipes to tags.
 - recipe_id (foreign key referencing Recipe): INTEGER - id of the recipe
 - tag_id (foreign key referencing Tag): INTEGER - id of the tag

This table is used to associate multiple tags with multiple recipes.

- **User_Liked_Recipe:** Tracks which recipes users have liked.
 - user_id (foreign key referencing User): INTEGER - id of the user
 - recipe_id (foreign key referencing Recipe): INTEGER - id of the recipe

This table facilitates the many-to-many relationship between users and recipes specifying which recipes each user has liked.

Foreign key constraints ensure referential integrity between related tables and primary keys uniquely identify records within each table.

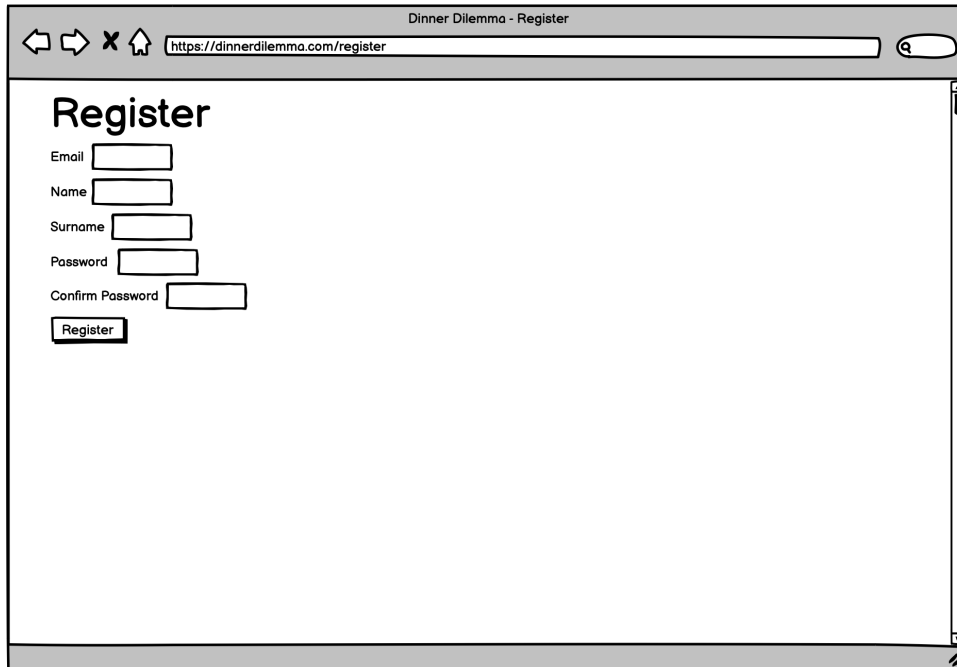
3.2 Other Information

To facilitate the initialization of the web app's database and fill in some starting data: we downloaded Food.com - Recipes and Reviews dataset, cleaned the data and adapted it to our database. We finally created a script to insert the data about some test users (also with role "admin") and those recipes. A guide on how to set up the database is found in the "src/main/database/psql/set_up_guide.txt" file.

4 Presentation Logic Layer

The website is simple but effective and can handle authenticated and non-authenticated users and, for the authenticated ones, the different roles. Each user can visualize only the pages and perform only the actions that he is authorized to do.

4.1 Register



The screenshot shows a web browser window titled "Dinner Dilemma - Register". The address bar contains the URL "https://dinnerdilemma.com/register". The page content is titled "Register" and features a registration form with the following fields and controls:

- Email:
- Name:
- Surname:
- Password:
- Confirm Password:
- Register:

Page where a user can register by inputting e-mail, name, surname and password (there is also a text field to confirm the password). All checks on the input data are done to catch eventual errors and avoid duplicate emails. The id of the user is generated and the password gets hashed using SHA256. With this registration form only users of type "user" can be registered, while "admin" users are created manually by the owners of the website. In both cases an email is sent to the registered email with a code to verify the account and the user is prompted a text field in which he needs to insert the just received code. Without this step the registration process is not completed and the user cannot login. At the moment in this page, for test purposes only, there is a dropdown menu to choose the role of the user, so it is possible to create an admin.

4.2 Login

The image shows a web browser window with the title "Dinner Dilemma - Login". The address bar contains the URL "https://dinnerdilemma.com/login". The main content area of the browser displays a login form. At the top of the form is the heading "Login". Below this heading are two input fields: "Email" and "Password". Under the "Password" field is a blue hyperlink labeled "Forgot Password?". At the bottom of the form is a button labeled "Login". The browser window also shows standard navigation icons (back, forward, home, etc.) and a scrollbar on the right side.

Simple login form where the user inputs his email and his password and, if correct, logs in and gets redirected to the home page. If the user has been banned, a popup will inform the user of his ban and he will not be able to login.

An HTTP session is created so the status of the logged user is kept during the navigation on the website. There is also an option called "Forgot Password?" that, similarly to the registration flow, sends an email to the user with a code that he needs to verify to enter a page in which he can change its password.

4.3 Verify Code

Dinner Dilemma - Login

https://dinnerdilemma.com/verify_code

Verify Code

Insert the code sent to the email user@email.com

Verify

The just mentioned page used to verify the code received via email during registration or during the forgot password process.

4.4 Home

Dinner Dilemma - Recipe Approval

https://dinnerdilemma.com/recipe_approval

Dinner Dilemma

Welcome User

[Logout](#) [User Control Panel](#) [Sign Up](#) [Login](#)

Authenticated user
Non-authenticated user

Search Recipe

Ingredient Tag Search Random Recipe

Recipe

- Recipe 52
- Recipe 61
- Recipe 42

Suggested Recipes

Recipe

- Recipe 77
- Recipe 91
- Recipe 4

Most Liked Recipes

Recipe	Likes
Recipe 11	134
Recipe 22	99
Recipe 33	82

[About Us](#)

The home page of the website: here the user can find links to login and registration (if not already logged in) and logout (if already logged in).
If the user is logged in there is also a link for the user control panel page.

In the middle of the page the main functionality of the web app is presented (it can be accessed by everyone, even non-authenticated users): with a dropdown menu the user can choose from a list of ingredients and also from a list of tags and search recipes that use them. The result of the search is a list of recipes presented below the dropdown menu and below some suggested recipes with similar ingredients appear.

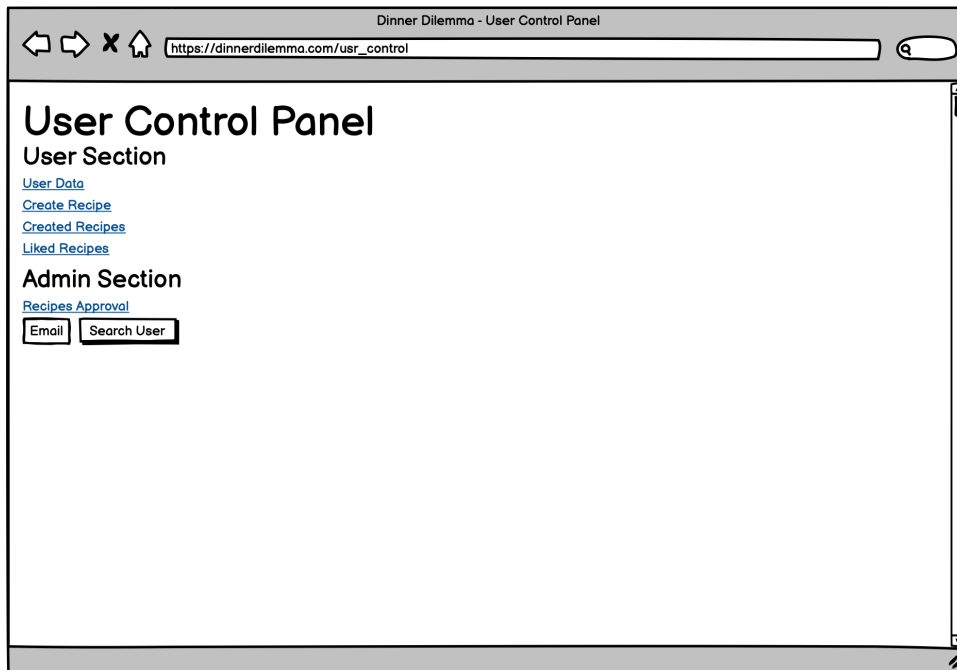
On the right there is also a list with the most liked recipes of the website.

For both the two lists, the user can click on each recipe and get redirected to the related recipe details page.

Finally, a button inspired by Google's "I'm Feeling Lucky" that opens the details page of a random recipe is presented.

In the footer of the page, there is a link to the "About Us" page.

4.5 User Control Panel



The user control panel, accessible only by logged users, contains the link for the other pages: user data, create recipe, created recipes and liked recipes. For users of type admins, there are two links for recipes approval and recover/remove page and a text input where the admin can insert a user email and go to the search user page. At the bottom of the page, at the moment, for test purposes only, there is a link to the log of the web app.

4.6 User Data

Dinner Dilemma - User Data

https://dinnerdilemma.com/usr_data

User Data

Email

Name

Surname

Password ☐ Show Password

Registration date: 2024-01-01

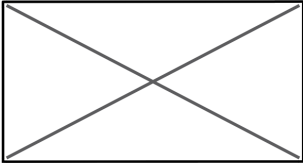
This page shows the logged user its informations inserted at registration. Here the user can change its name and surname.

4.7 Recipe

Dinner Dilemma - Recipe

https://dinnerdilemma.com/recipe?id=1

Recipe Name Like



Instructions

Creation Date
2024-12-12

Difficulty
Medium

Time (minutes)
60

Ingredients

- ☐ Ingredient 1
- ☐ Ingredient 2
- ☐ Ingredient 3

Tags

- ☐ Tag 1
- ☐ Tag 2
- ☐ Tag 3

The page that shows the details of a specific recipe: the thumbnail image, the name, the description/preparation, the time required (in minutes), the name and surname of the user that inserted it, the difficulty, the ingredients, the tags and the number of likes on the recipe. Only for logged users, there is a button to like/unlike the recipe.

4.8 Create Recipe

Dinner Dilemma - Create Recipe

https://dinnerdilemma.com/create_recipe

Create Recipe

Name

Ingredients

Tags

Image URL

Time in minutes

Difficulty

Instructions

This form, only accessible by logged users, is used to create a recipe that once submitted, is sent to be approved by an admin. The page contains inputs for name, description, url of the thumbnail image, difficulty level and time for preparation (in minutes), a drop down menu to add the ingredients and one for the tags.

4.9 Created Recipes

Dinner Dilemma - Created Recipes

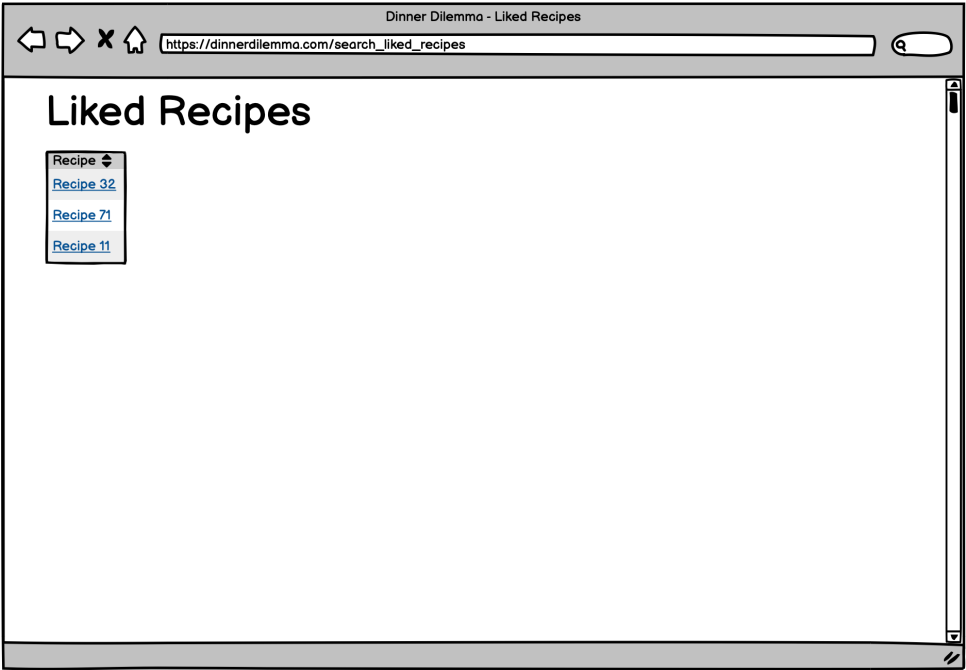
https://dinnerdilemma.com/search_created_recipes

Created Recipes

Recipe	Status
Recipe 1	Approved
Recipe 2	Not Approved
Recipe 3	To be approved

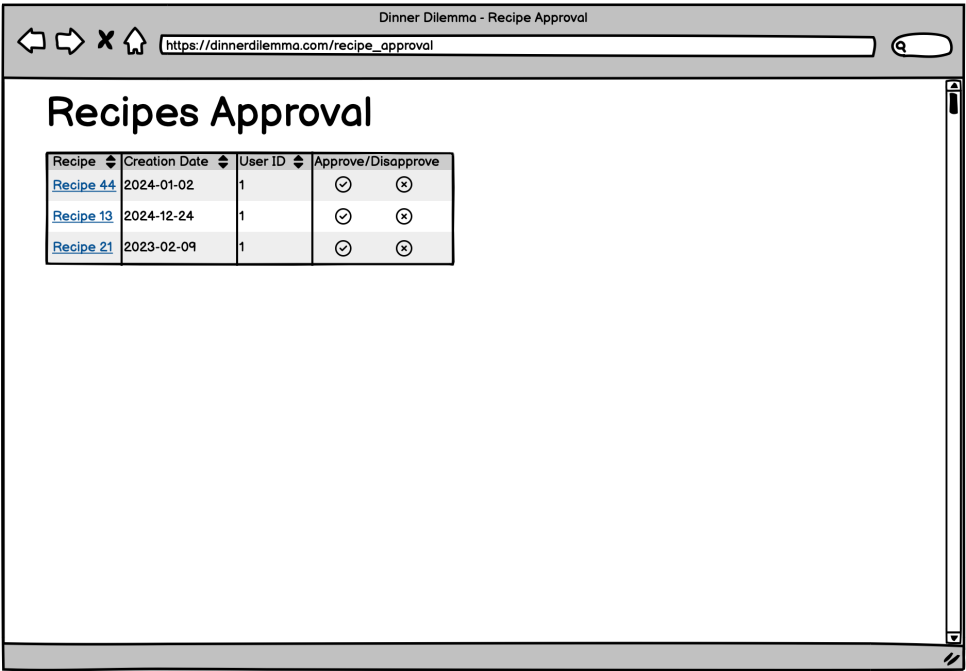
In this page the logged user can see the list of the recipes he created, along with the status of each one to find out if they have been approved or not.

4.10 Liked Recipes



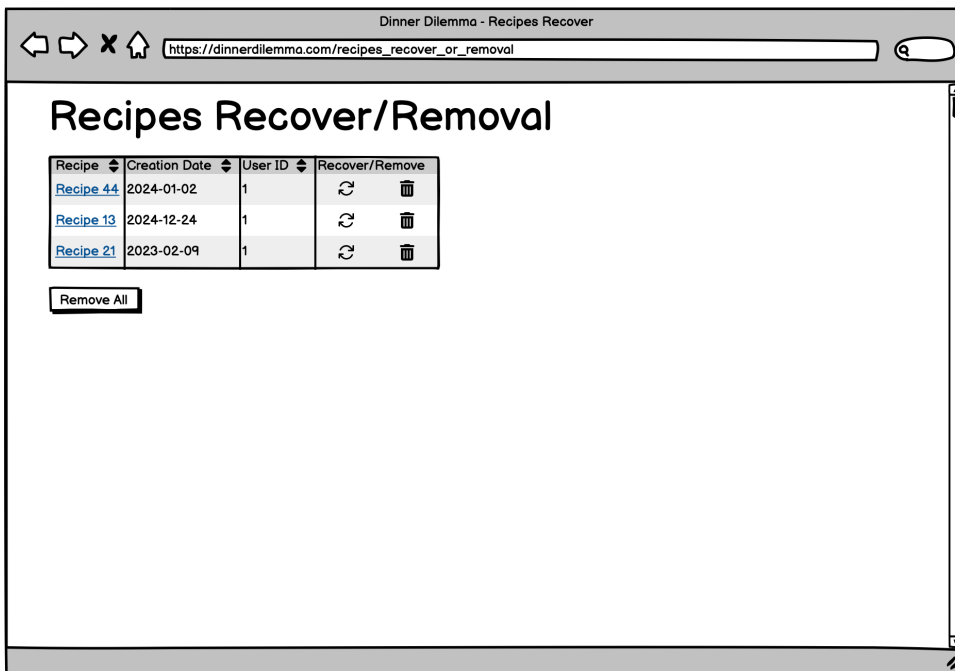
A list of the recipes liked by the logged user.

4.11 Recipes Approval



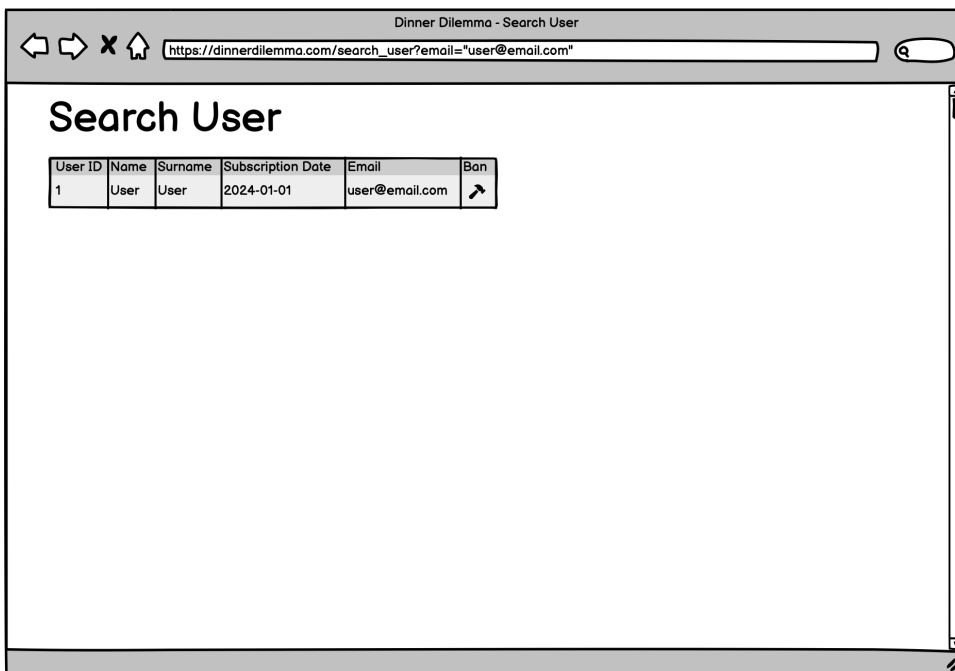
A page where the logged admin can see the recipes that are waiting to be approved. He has the possibility to approve or disapprove them.

4.12 Recipes Recover Or Removal



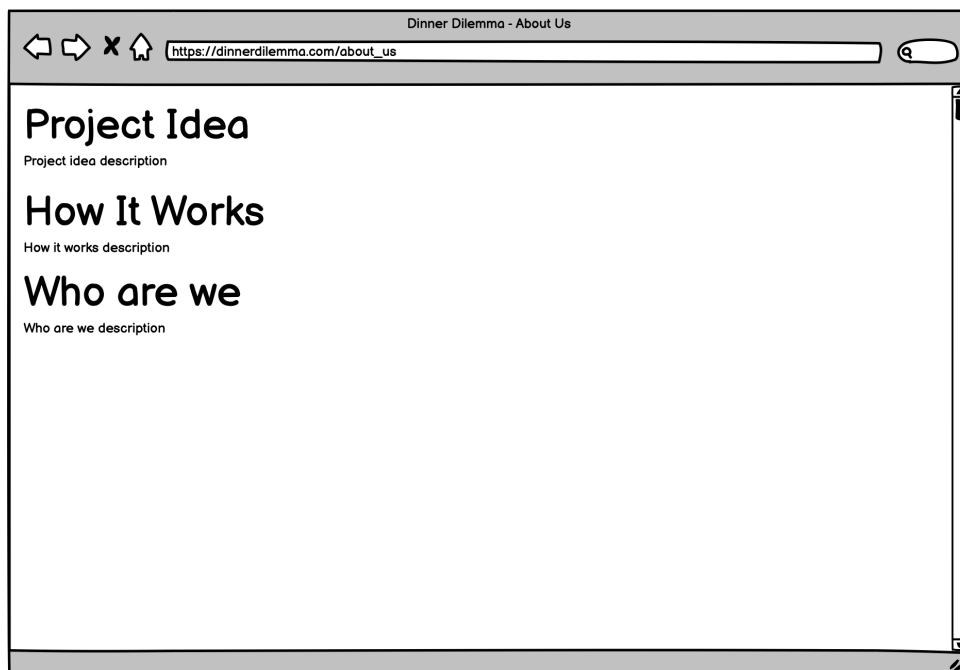
Contrary to the last cited page, here the logged admin can see the recipes that have been rejected and can decide if recover them or completely delete them from the database. There is also a button to remove all the rejected recipes.

4.13 Search User



In this page a logged admin can see the information of the user associated with the specific email inputted in the user control page. Along with this he has the possibility to ban the user.

4.14 About Us



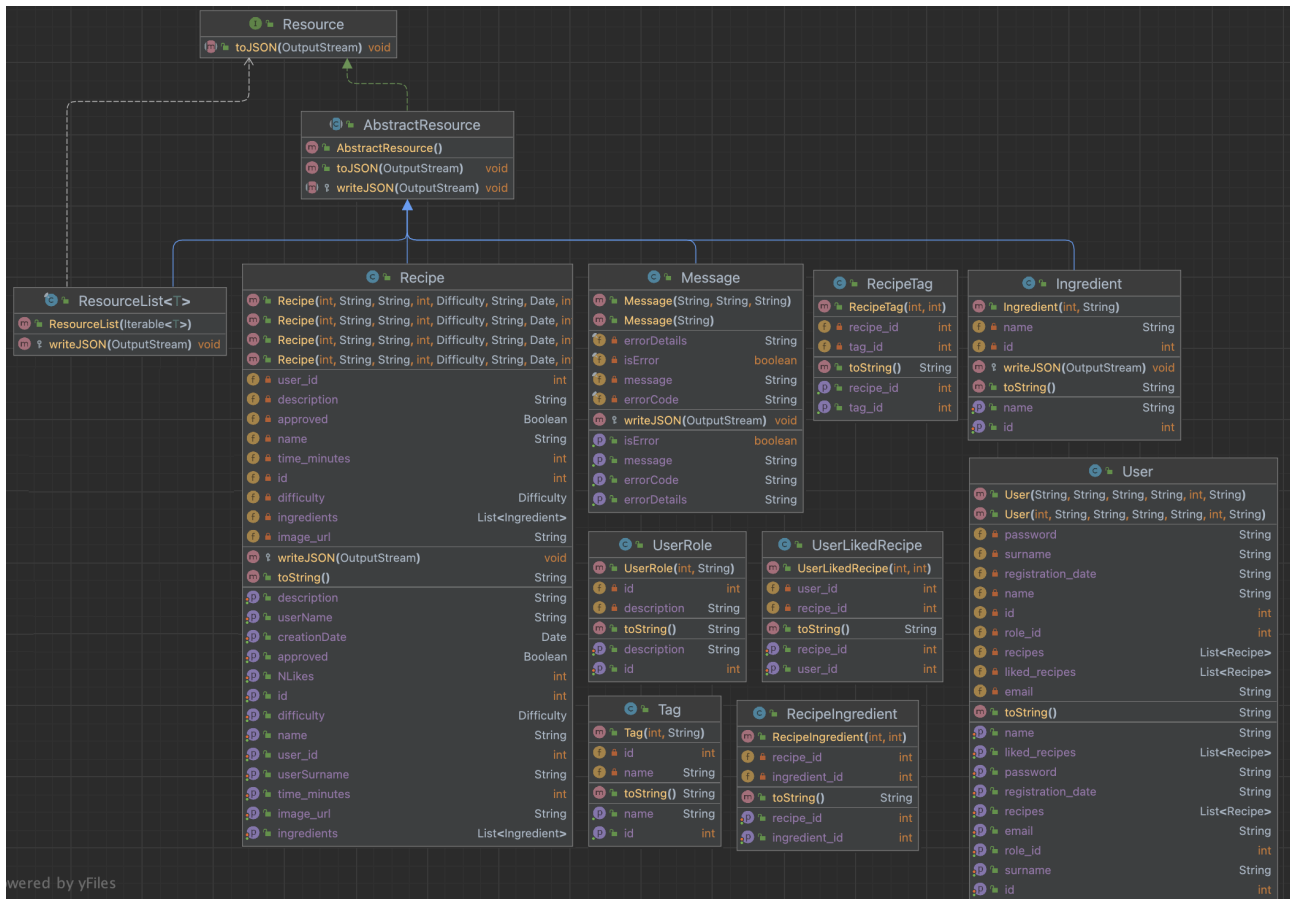
A simple page with the description of Dinner Dilemma and a guide on how the website works. There is also a short section to list the creators of the web app.

5 Business Logic Layer

5.1 Class Diagram

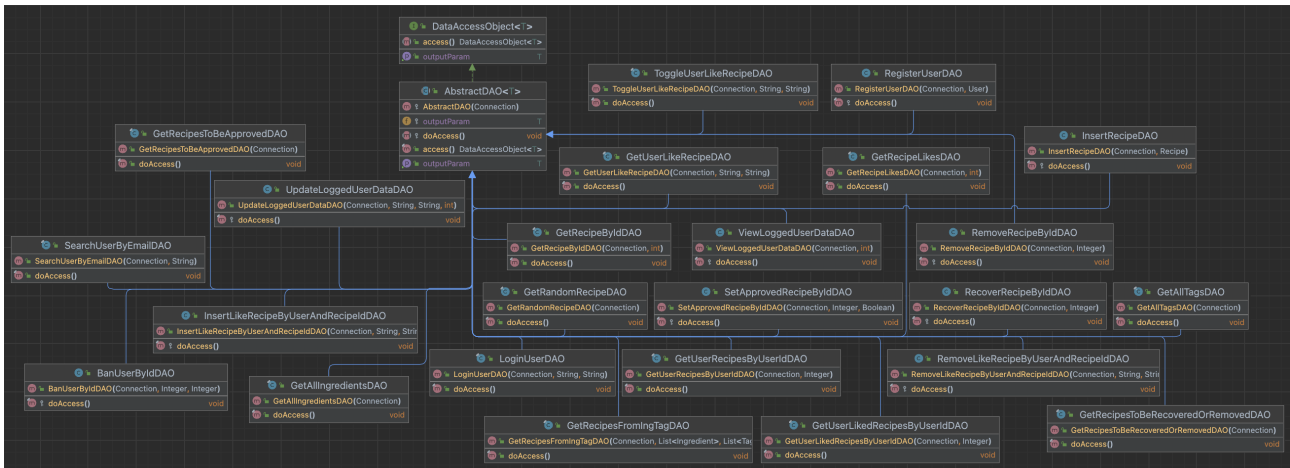
The following is the class diagram of the project, divided in the main five parts.

5.1.1 Resources



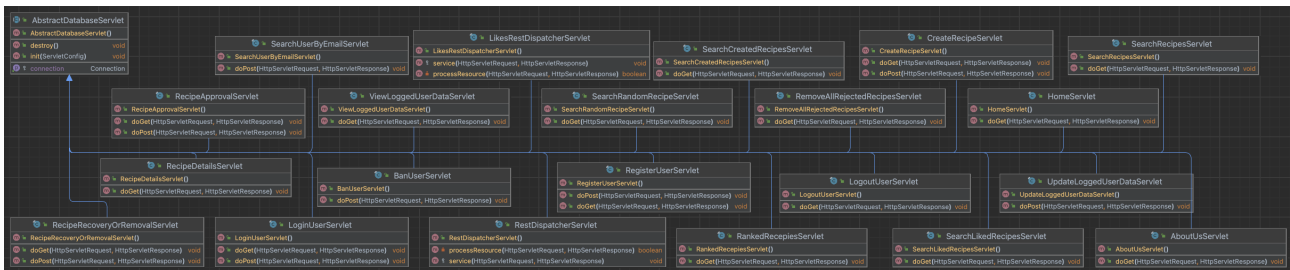
These are the classes that represent the objects mapped from/to the database tables, also the relation tables. For example, the class "Recipe" reflects the "Recipe" table, having the same fields plus methods. They all are subclasses of the class "Resource", that implements the definition of the "toJSON" method.

5.1.2 DAOs



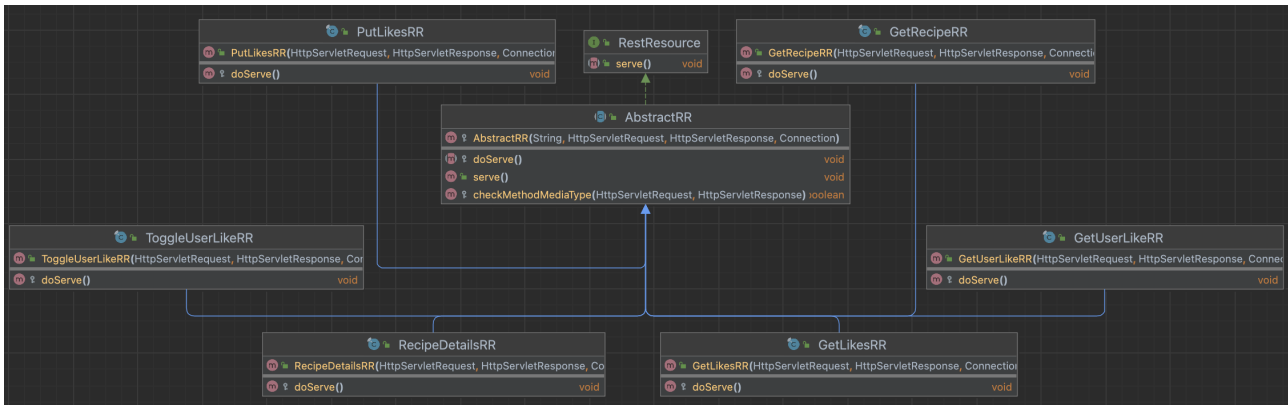
The DAOs are the classes used to access the database and perform an atomic action. For example the action of creating a recipe is done in the "InsertRecipeDAO", that takes an input a "Recipe" and a "Connection" and saves it on the database. Necessary checks for errors are of course done. All DAOs are subclasses of "AbstractDAO", which defines the "doAccess" method, that handles the access to the database with a "Connection" and the subsequent closure.

5.1.3 Servlets



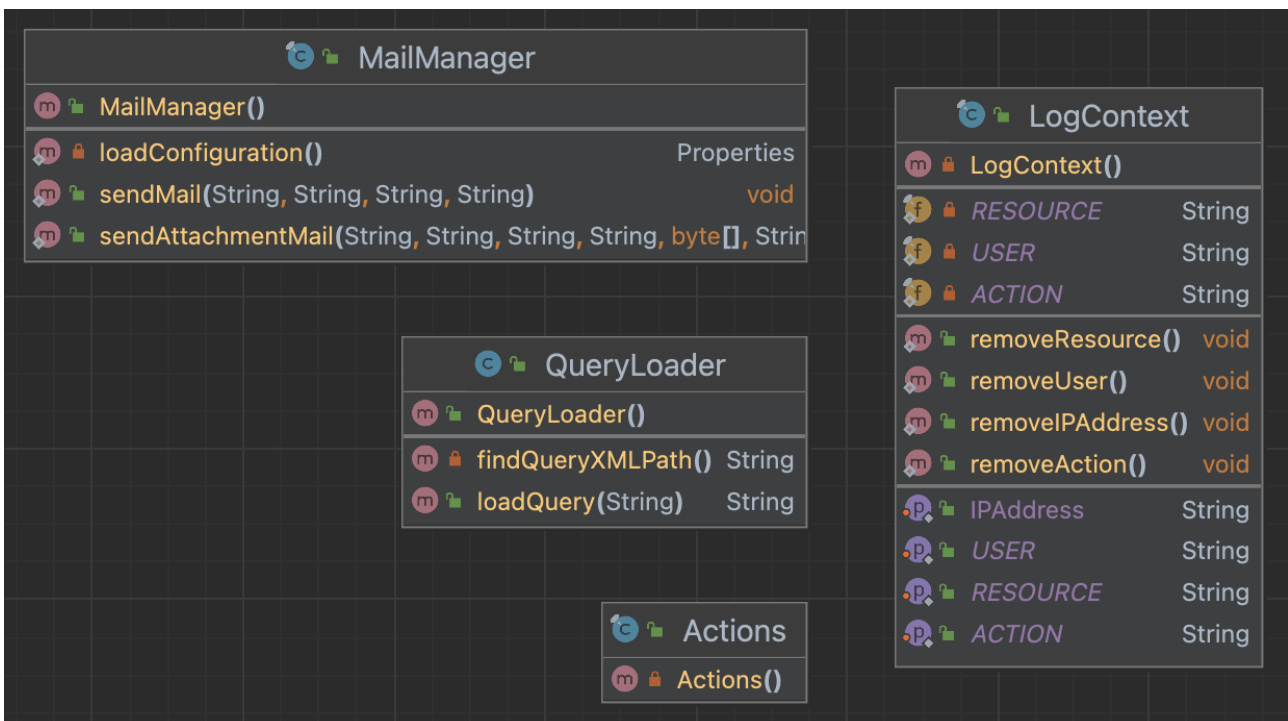
Servlets are used as the intermediary between the client and the server by handling HTTP requests and generating appropriate HTTP responses. Each servlet corresponds to a particular URL pattern (all configured in "WEB.xml" file) and performs specific tasks based on the request parameters, also handling POST requests in the "doPost" method and the GET ones in the "doGet" method. For instance, the "SearchRandomRecipeServlet" manages requests of type GET with the appropriate URL and redirects the user to the page of a random recipe.

5.1.4 REST



The classes used to manage the REST APIs, discussed later.

5.1.5 Other

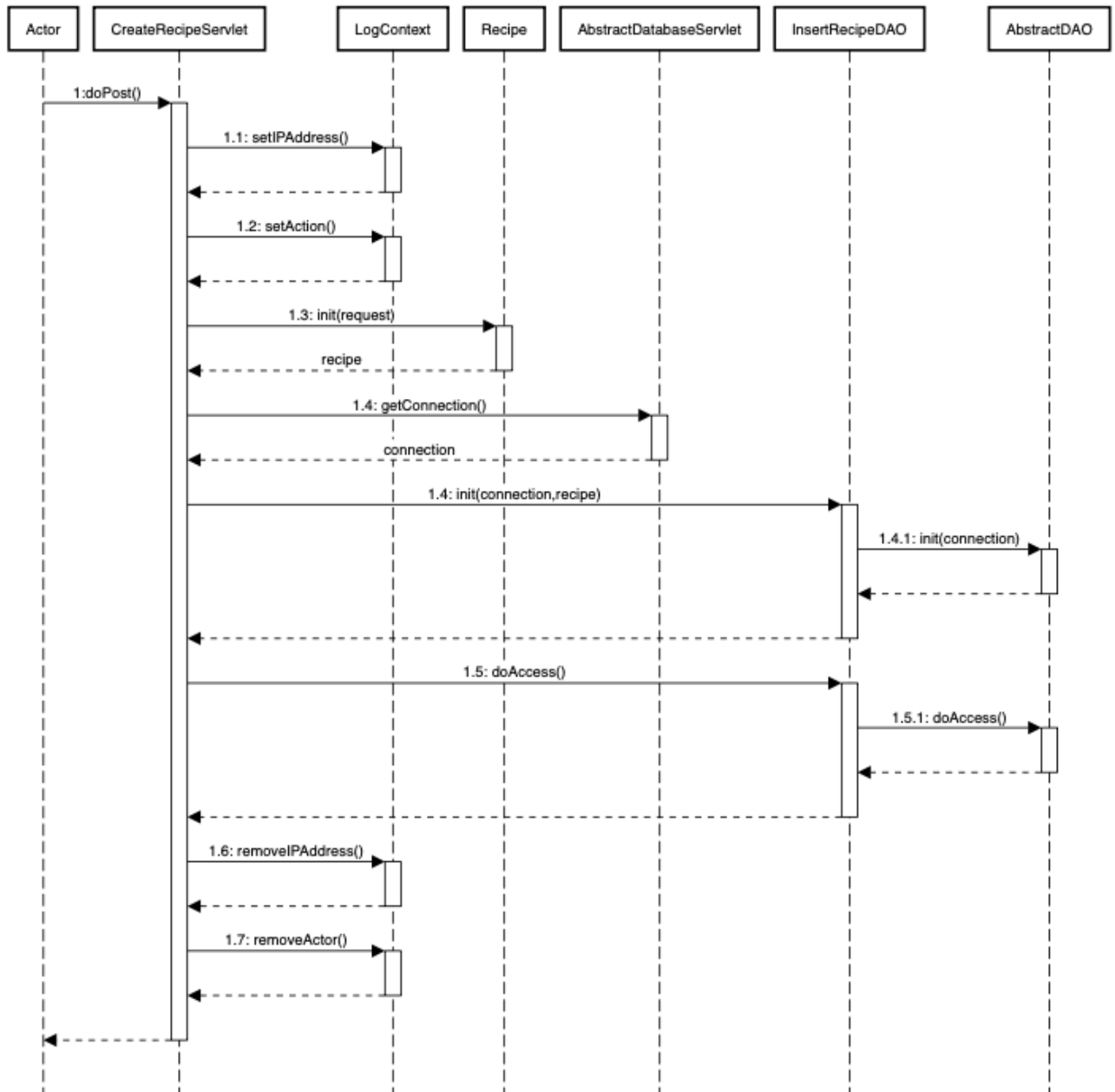


These are other tools/utils that have been used in the project.

- LogContext: used to store some user information when the user is trying to do a specific action, to be later logged into the web app logs. For example when a user logs in, we log the ip address, the time and other information of the user.
- Actions: class that defines a series of actions (strings) that are used in "LogContext" to explicitly log what action the user is actually doing.
- QueryLoader: Queries are stored in an external file. This class serves the DAOs and loads the specific query they need.
- MailManager: class that manages the sending of emails to users.

5.2 Sequence Diagram

The following is the sequence diagram for the action of "creating a new recipe" performed by a user.



5.3 REST API Summary

The following is a summary of the REST API used in the webapp.

Only the "Like" resource has been implemented with REST, while the others are being currently developed. This also holds for REST Error Codes.

More details are presented later.

URI	Method	Description	Filter
rest/like/{recipe_id}	POST	Inserts the like to recipe with id "recipe_id" by user with id "user_id" found in the HTTP Session	
rest/like/{recipe_id}	DELETE	Removes the like to recipe with id "recipe_id" by user with id "user_id" found in the HTTP Session	
rest/like/{recipe_id}	GET	Retrieves whether recipe with id "recipe_id" has been liked by user with id "user_id" found in the HTTP Session	
rest/user	GET	Retrieves the user with id "user_id" found in the HTTP Session	
rest/user	PUT	Updates the data of the user with id "user_id" found in the HTTP Session. Used by the user when he is changing its information or by the admin when he is banning a user	
rest/user/login	GET	Login of a user, email and password are found in the body	
rest/recipe/{recipe_id}	POST	Inserts a new recipe with id "recipe_id" and the rest of the data is found in the JSON in the body	
rest/recipe/{recipe_id}	DELETE	Deletes the recipe with id "recipe_id"	
rest/recipe/{recipe_id}	PUT	Updates the recipe with id "recipe_id". Used to update the status of the recipe by an admin	
rest/recipe/{recipe_id}	GET	Returns the recipe with id "recipe_id"	
rest/recipe/search_recipe	GET	Returns the result of the search of recipes. Ingredients and tags for the research are found in the body of the request	
rest/recipe/ranked_recipe	GET	Returns the list of most liked recipes	
rest/recipe/suggested_recipe	GET	Returns the list of suggested recipes based on the searched recipes	

Table 2: REST API Summary

5.4 REST Error Codes

Error Code	HTTP Status Code	Description
200	OK	No error
201	CREATED	Resource created
400	BAD REQUEST	Request not correctly formatted
404	NOT FOUND	Resource not found
405	METHOD NOT-ALLOWED	HTTP method for this resource not available/allowed

405	NOT ACCEPTABLE	The action of the request doesn't conform with the current status of the database
500	INTERNAL SERVER-ERROR	Internal error

Table 3: REST Error Codes

5.5 REST API Details

The following are the details of the REST APIs implemented for the "Like" resource.

Like - GET

Returns whether a user has liked a recipe or not.

- URL: `rest/like/{recipe_id}`
- Method: GET
- URL Parameters:
 - `recipe_id`: the id of the recipe
- Data Parameters: none
- Success Response:
 - Code: 200
 - Content: `{"like": "true"}`
- Error Response:
 - Code: 500
 - Reason: internal server error

Like - DELETE

Removes the like that a user has on a recipe.

- URL: `rest/like/{recipe_id}`
- Method: DELETE
- URL Parameters:
 - `recipe_id`: the id of the recipe
- Data Parameters: none
- Success Response:
 - Code: 200
- Error Response:
 - * Code: 500
 - * Reason: internal server error
 - * Code: 405
 - * Reason: like is not present in the database

Like - POST

Inserts the like on a recipe by a user.

- URL: `rest/like/{recipe_id}`
- Method: DELETE
- URL Parameters:
 - `recipe_id`: the id of the recipe
- Data Parameters: none
- Success Response:
 - Code: 200
- Error Response:
 - Code: 500
 - Reason: internal server error
- – Code: 405
- Reason: like already present in the database

6 Group Members Contribution

Gianluca Antolini Worked on the setup of the project, the creation of the database ER schema and the relative translation to SQL code. Created DAOs (also with Abstract DAO pattern) and some Servlets and JSPs. Worked on pages: login, create recipe, liked recipes and created recipes. Worked on report.

Tommaso Bergamasco Worked on the "Create Recipe" servlet and wrote the queries responsible for inserting new ingredients into the database.

Andrea Feline Written (and fixed multiple times) the Python script for cleaning and formatting the CSV dataset of recipes. During this process, I ensured that the dataset adhered to PostgreSQL requirements for data formatting. I developed the largest queries, such as searching for recipes based on ingredients and tags, suggesting recipes to compensate for empty or poor search results, and retrieving recipes along with their ingredients, creator information, and number of likes. I've written the About Us JSP page and servlet, as well as the JSP page, servlet, and DAO for displaying recipe details. I also created the like REST service with the collaboration of Francesco Frigato. I developed the verify-user JSP page and servlet using the mail manager, and implemented the change password functionality, including the respective page, servlet, and DAO. Finally, I updated the login and registration processes to utilize the verification servlet when needed.

Francesco Frigato Worked on the setup of the project and development of the database. Created and managed the admin section of the WA, including the ones for managing recipes approval, removal or recovery, and for managing the users of the WA. And created the servlet for managing the logged user data. Helped fixing minor issues in other servlets, jsps and htmls.

Emilio Risi Wrote on the maven configuration of the project and the scripts for a fast load and reload of the Postgres database. Wrote the Query loader and format of queries.xml to be used in the daos. Worked on the Login and Register servlets and Created the Homepage servlet and the annex servlets to obtain the the recepies searched by the user, the ranked list and the suggested recipes, their daos and the dinamic SQL code (based on the queries of Andrea Feline) and their jsp code. Working on the recipe part of the rest Api and writing the dispatcher for it .