Share-A-Drink

Github link: https://github.com/filip-svensson/DAT076.git

Introduction

Share-A-Drink is a forum website, developed for drink-enthusiasts to share their recipes. The site itself greets you with a landing page, with a button to take you to the forum. At the forum you can see what others have already posted and search for any preferred drink to see if there exists a recipe for it.

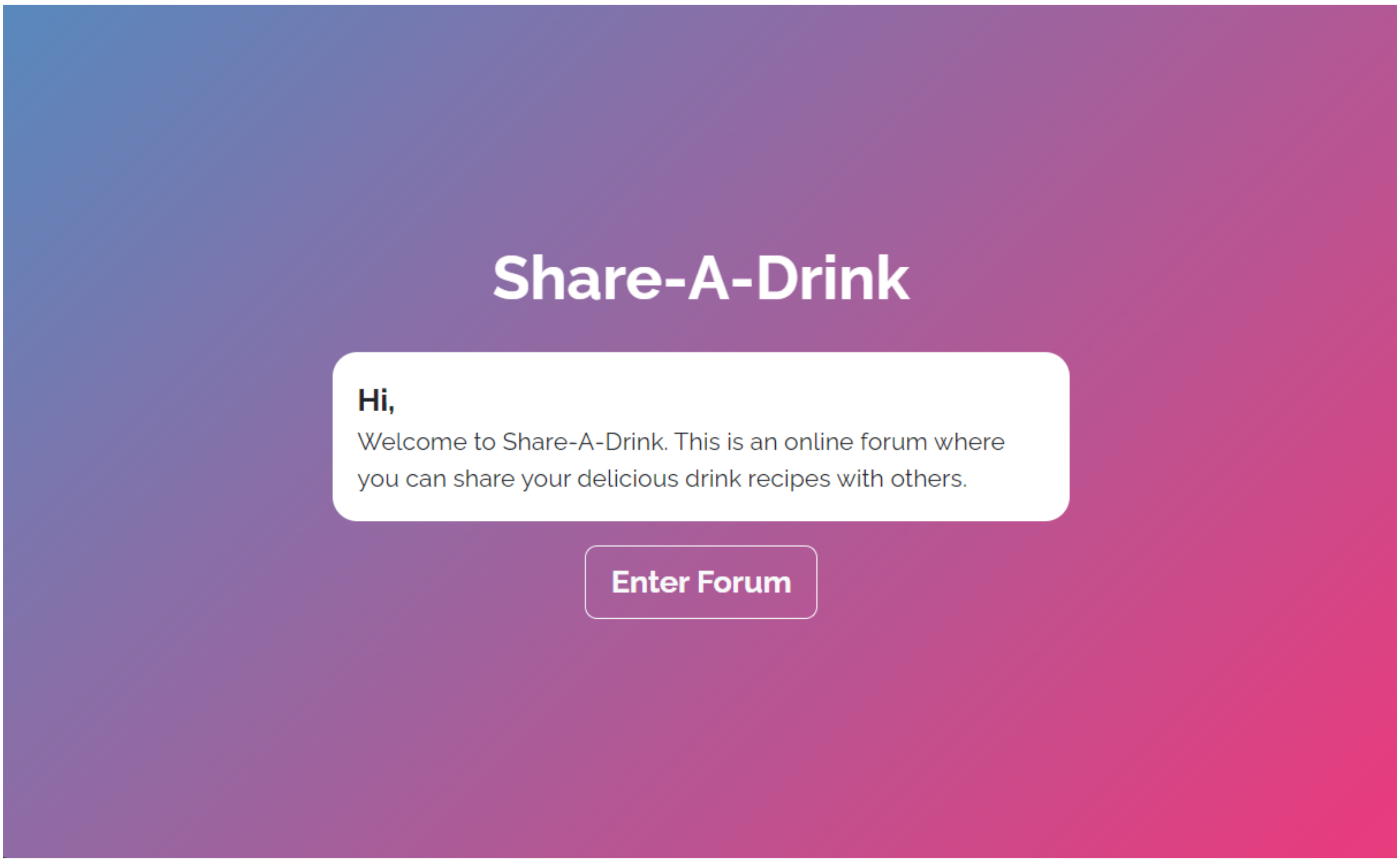
If you want to share a recipe of your own you can create a user and start sharing your drinks, your posts will be grouped under a separate tab for quick access. When you’ve created a user you can also rate and comment on other peoples posts, as well as add them to your favorites for quicker access.

A list of use cases

- Create a new account
- View all posts
- Search for posts
- Login to existing account
    - Create a post
        - Add a title (required)
        - Add a description (required)
        - Add multiple ingredients (at least 1 required)
    - See the posts created by the account you are logged into
    - See the posts favorited by the account you are logged into
    - Favorite a post
    - Unfavorite a favorited post
    - Create a review on a post
        - Add a rating (required)
        - Add a comment (optional)
    - Remove your review
- Logout

User manual

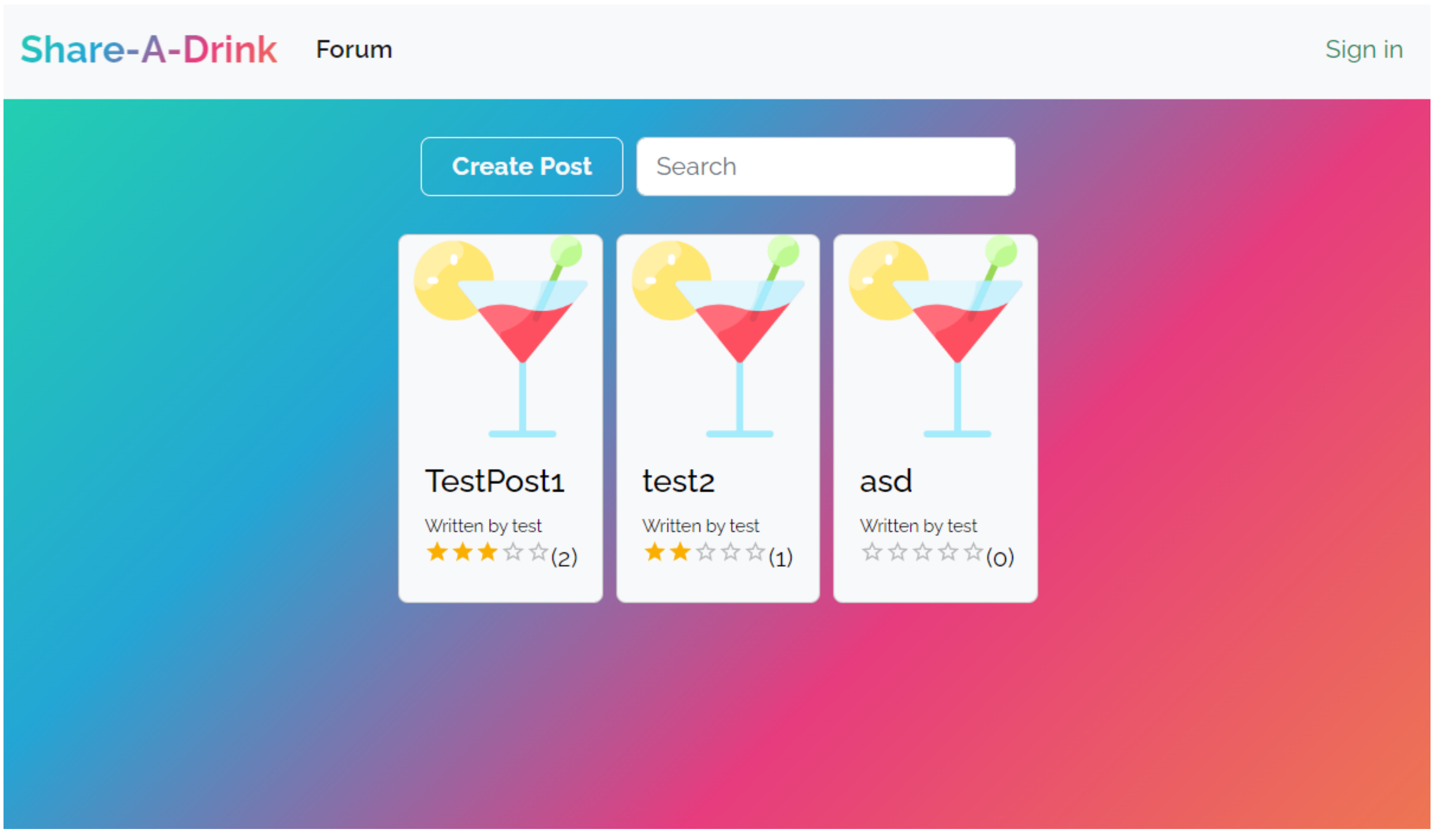
After pulling the code from github the first step is to run `npm run dev` in the server directory. After the server starts, open up a browser and enter localhost:8080, this should look like picture 1.



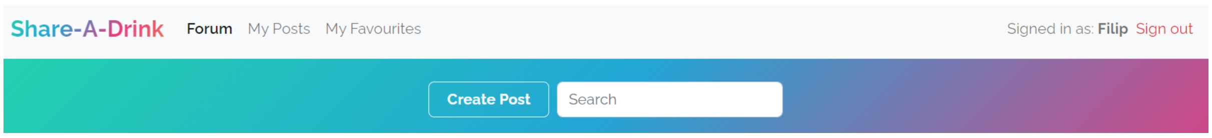
Hi,
Welcome to Share-A-Drink. This is an online forum where you can share your delicious drink recipes with others.

(picture 1, the landing page)

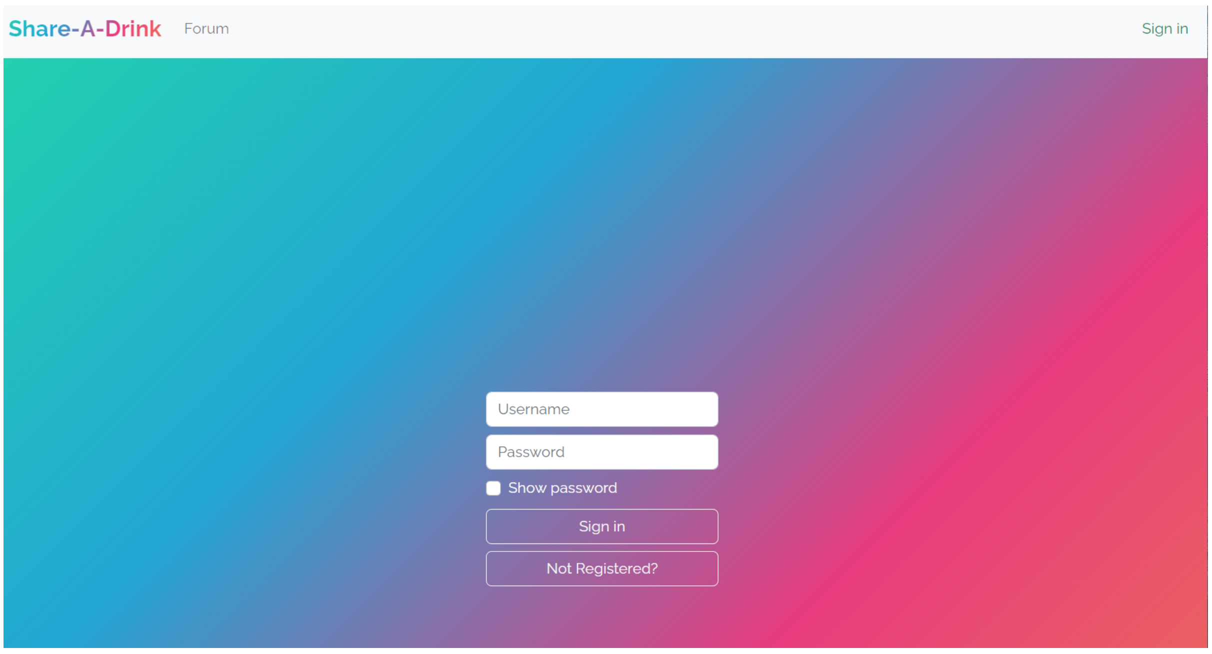
When the web-application is running, the user clicks “Enter Forum” to be taken to the main page as seen below in picture 2.



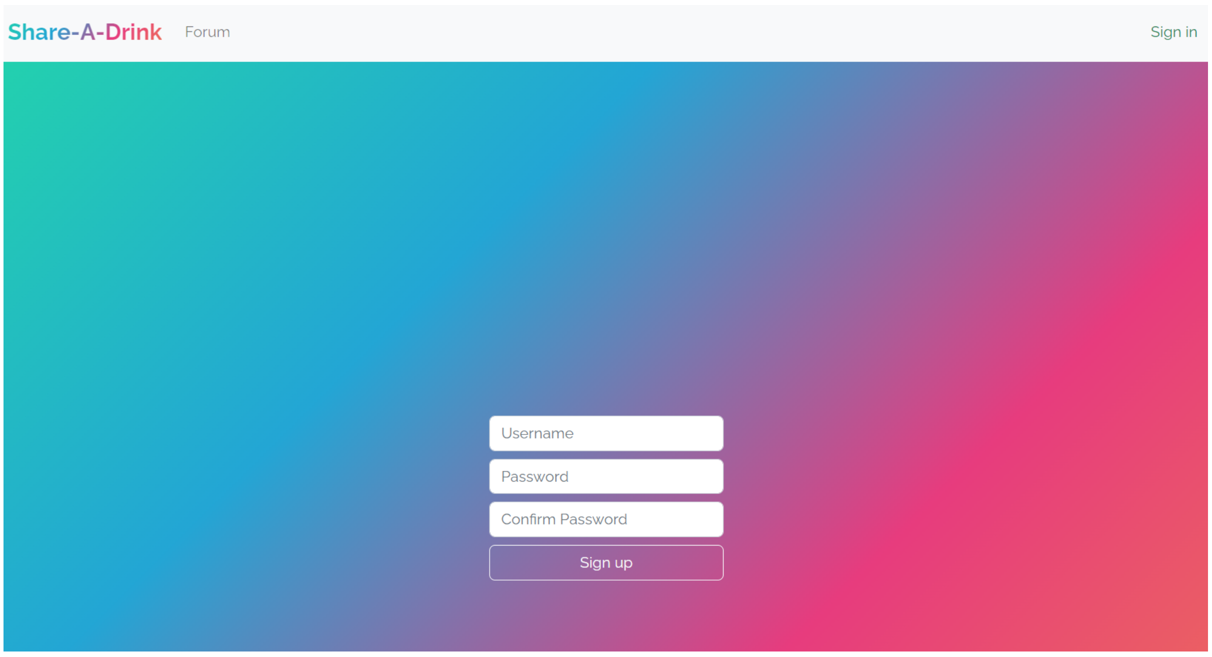
(picture 2, the main page)



(picture 2.1, the navbar as a signed-in user)



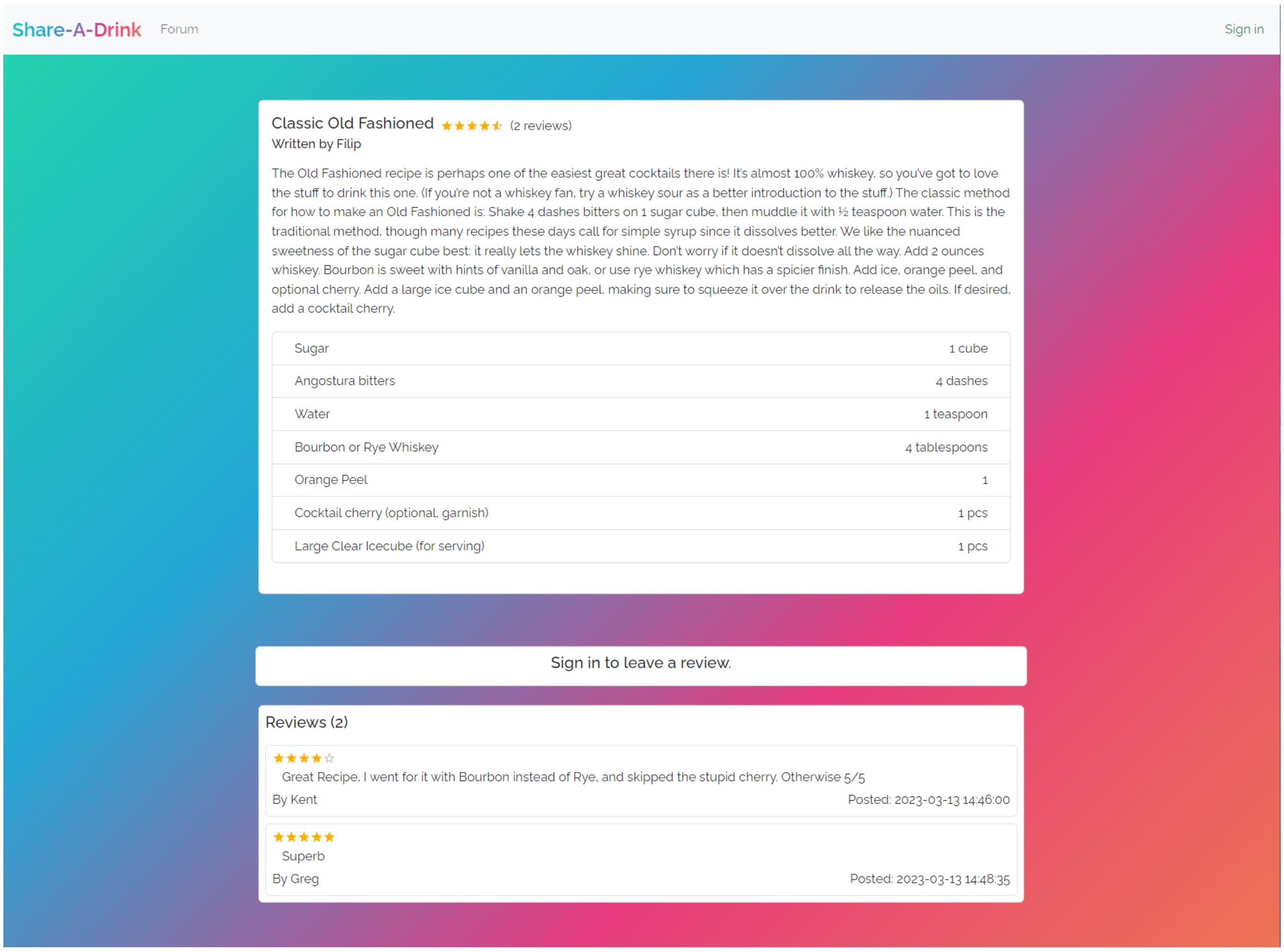
(picture 3, sign-in page)



(picture 3.1, the sign-up page)

At the main page the visitor can either look at recipes other users have already posted, but is unable to post anything until they've created a user. At the top of the page is a navigation bar. If you're not signed in you will only have the forum and the login options as in picture 2. Pressing "Login" will take you to the sign-in page in picture 3, where you have the option to click register and get to the page in picture 3.1. If you are signed in, you will have the additional tabs "My Posts" and "My Favourites" to easily navigate. To the right it states your username followed by a sign-out button.

On the forum page, click on one of the recipe cards to get to that posts page. You can also filter the posts by entering a search term in the search field. If you click on create post when not signed in, you will be redirected to the login page.



Classic Old Fashioned ★★★★⯪ (2 reviews)
Written by Filip

The Old Fashioned recipe is perhaps one of the easiest great cocktails there is! It's almost 100% whiskey, so you've got to love the stuff to drink this one. (If you're not a whiskey fan, try a whiskey sour as a better introduction to the stuff.) The classic method for how to make an Old Fashioned is: Shake 4 dashes bitters on 1 sugar cube, then muddle it with ½ teaspoon water. This is the traditional method, though many recipes these days call for simple syrup since it dissolves better. We like the nuanced sweetness of the sugar cube best: it really lets the whiskey shine. Don't worry if it doesn't dissolve all the way. Add 2 ounces whiskey. Bourbon is sweet with hints of vanilla and oak, or use rye whiskey which has a spicier finish. Add ice, orange peel, and optional cherry. Add a large ice cube and an orange peel, making sure to squeeze it over the drink to release the oils. If desired, add a cocktail cherry.

Sugar	1 cube
Angostura bitters	4 dashes
Water	1 teaspoon
Bourbon or Rye Whiskey	4 tablespoons
Orange Peel	1
Cocktail cherry (optional, garnish)	1 pcs
Large Clear Icecube (for serving)	1 pcs

Sign in to leave a review.

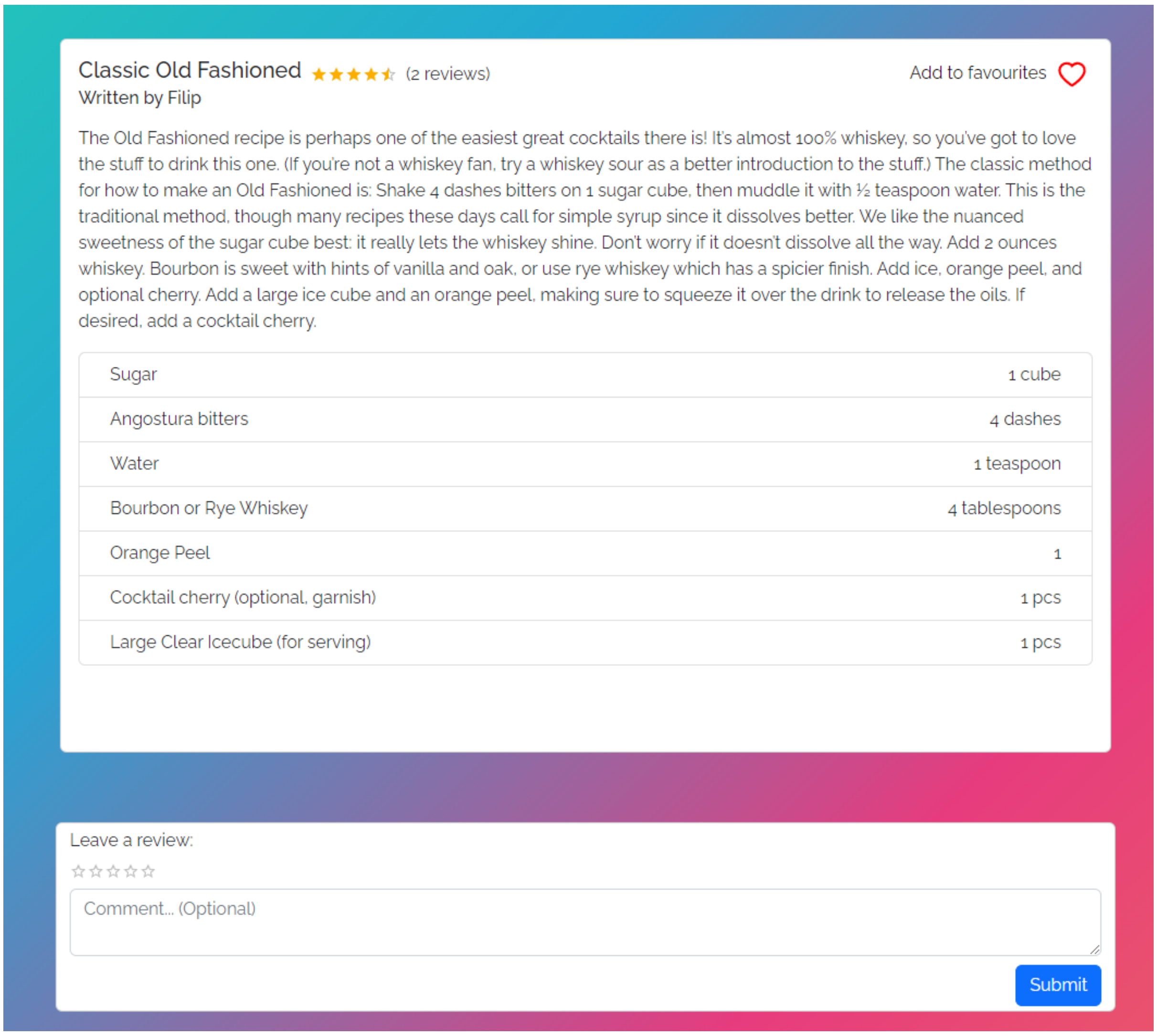
Reviews (2)

★★★★☆
Great Recipe, I went for it with Bourbon instead of Rye, and skipped the stupid cherry. Otherwise 5/5
By Kent — Posted: 2023-03-13 14:46:00

★★★★★
Superb
By Greg — Posted: 2023-03-13 14:48:35

(picture 4, a post's page viewed as guest)

At a post's page, you will see its description followed by the ingredient list. As a guest you will be able to read users' reviews. Following the post's title is the average rating given to this post, followed by the number of reviews made. The scale is 1-5. (See picture 4).


Classic Old Fashioned ★★★★☆ (2 reviews)

Add to favourites ♡

Written by Filip

The Old Fashioned recipe is perhaps one of the easiest great cocktails there is! It's almost 100% whiskey, so you've got to love the stuff to drink this one. (If you're not a whiskey fan, try a whiskey sour as a better introduction to the stuff.) The classic method for how to make an Old Fashioned is: Shake 4 dashes bitters on 1 sugar cube, then muddle it with ½ teaspoon water. This is the traditional method, though many recipes these days call for simple syrup since it dissolves better. We like the nuanced sweetness of the sugar cube best: it really lets the whiskey shine. Don't worry if it doesn't dissolve all the way. Add 2 ounces whiskey. Bourbon is sweet with hints of vanilla and oak, or use rye whiskey which has a spicier finish. Add ice, orange peel, and optional cherry. Add a large ice cube and an orange peel, making sure to squeeze it over the drink to release the oils. If desired, add a cocktail cherry.

Sugar	1 cube
Angostura bitters	4 dashes
Water	1 teaspoon
Bourbon or Rye Whiskey	4 tablespoons
Orange Peel	1
Cocktail cherry (optional, garnish)	1 pcs
Large Clear Icecube (for serving)	1 pcs

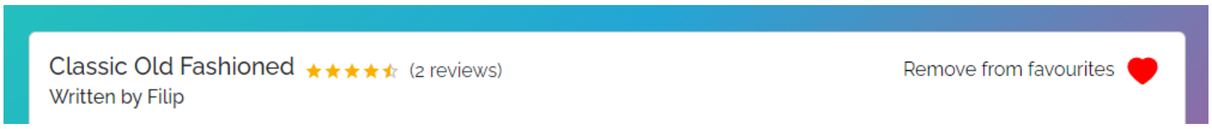
Leave a review:

☆☆☆☆☆

Comment... (Optional)

Submit

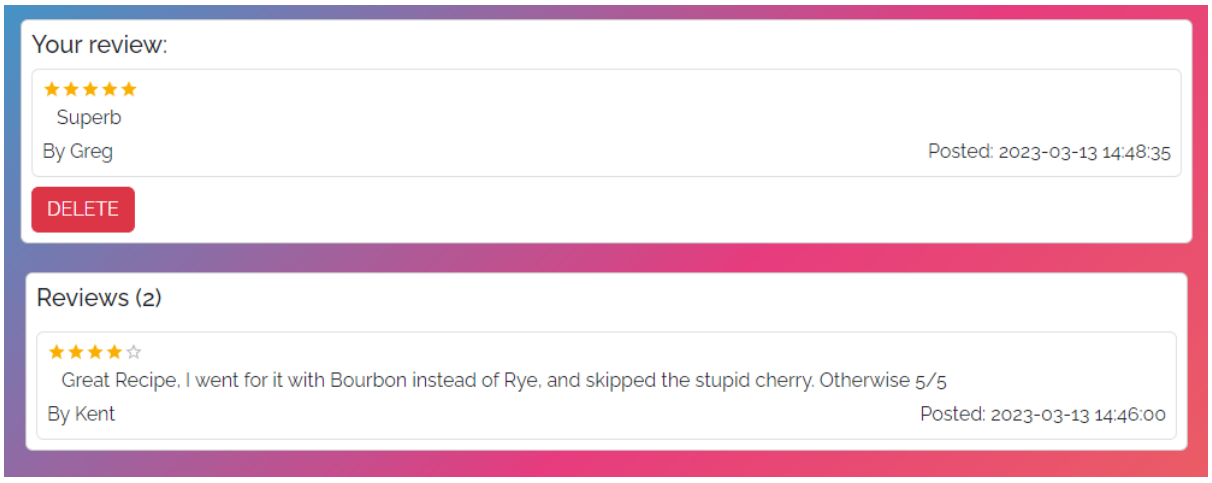
(picture 5, post viewed by a signed in user, no review made and not in their favourites)


Classic Old Fashioned ★★★★☆ (2 reviews)

Remove from favourites ♥

Written by Filip

(Picture 5.1, the user has favourited the post)


Your review:

★★★★★
Superb
By Greg
Posted: 2023-03-13 14:48:35

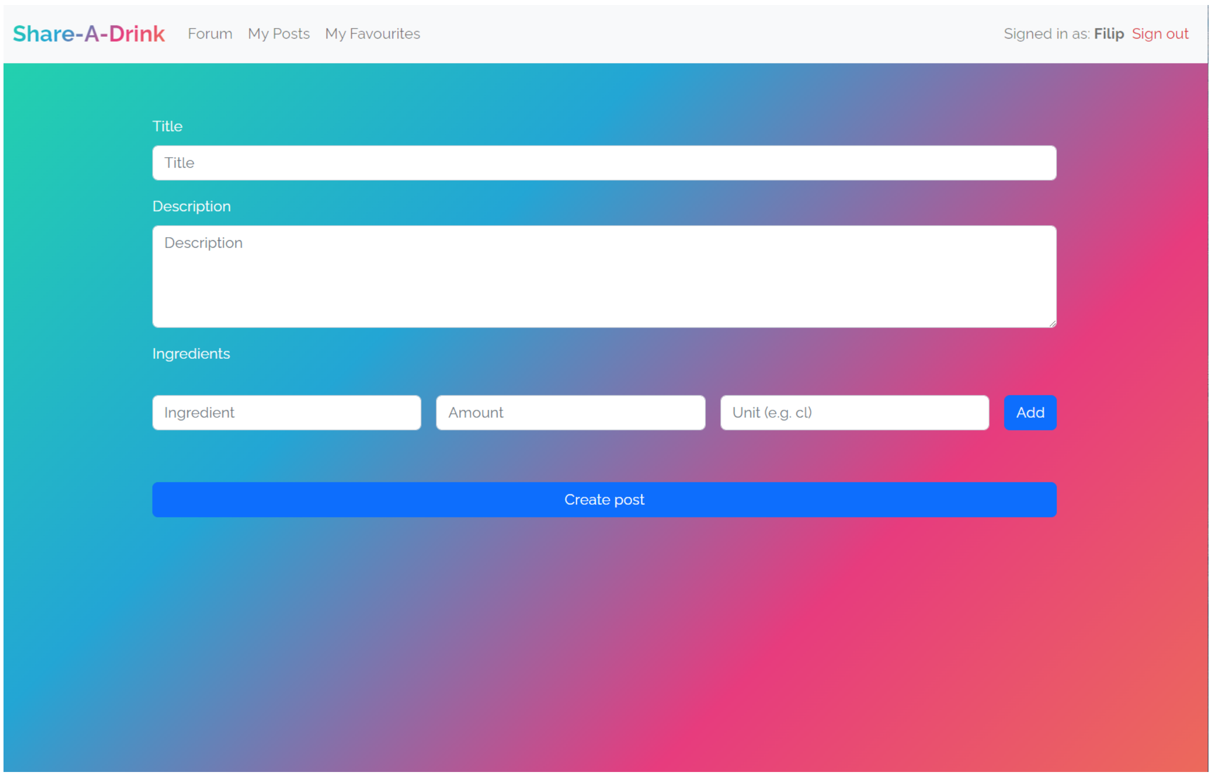
Reviews (2)

★★★★☆
Great Recipe, I went for it with Bourbon instead of Rye, and skipped the stupid cherry. Otherwise 5/5
By Kent
Posted: 2023-03-13 14:46:00

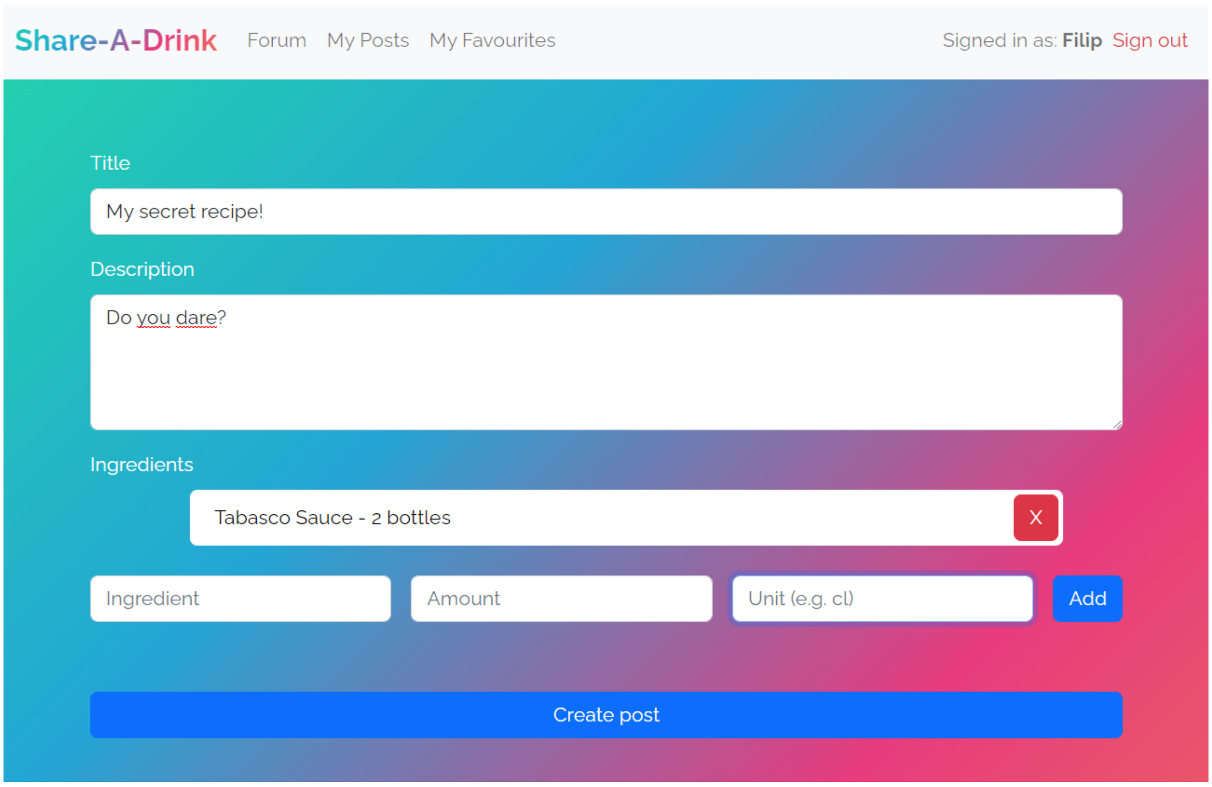
(Picture 5.2, the user has posted a review)

If the post is viewed as a signed in user you will have the option to leave a review if you haven't already (See picture 5). The top right corner will change into the one in picture 5.1 if you click either the "Add to favourites" label or the heart icon.

When you've posted a review, the option to leave a review will be replaced with your review from the list of reviews (See picture 5.1). Here you have the option to delete your review, which will bring you back to the state in picture 5.



(Picture 6, Create Post)

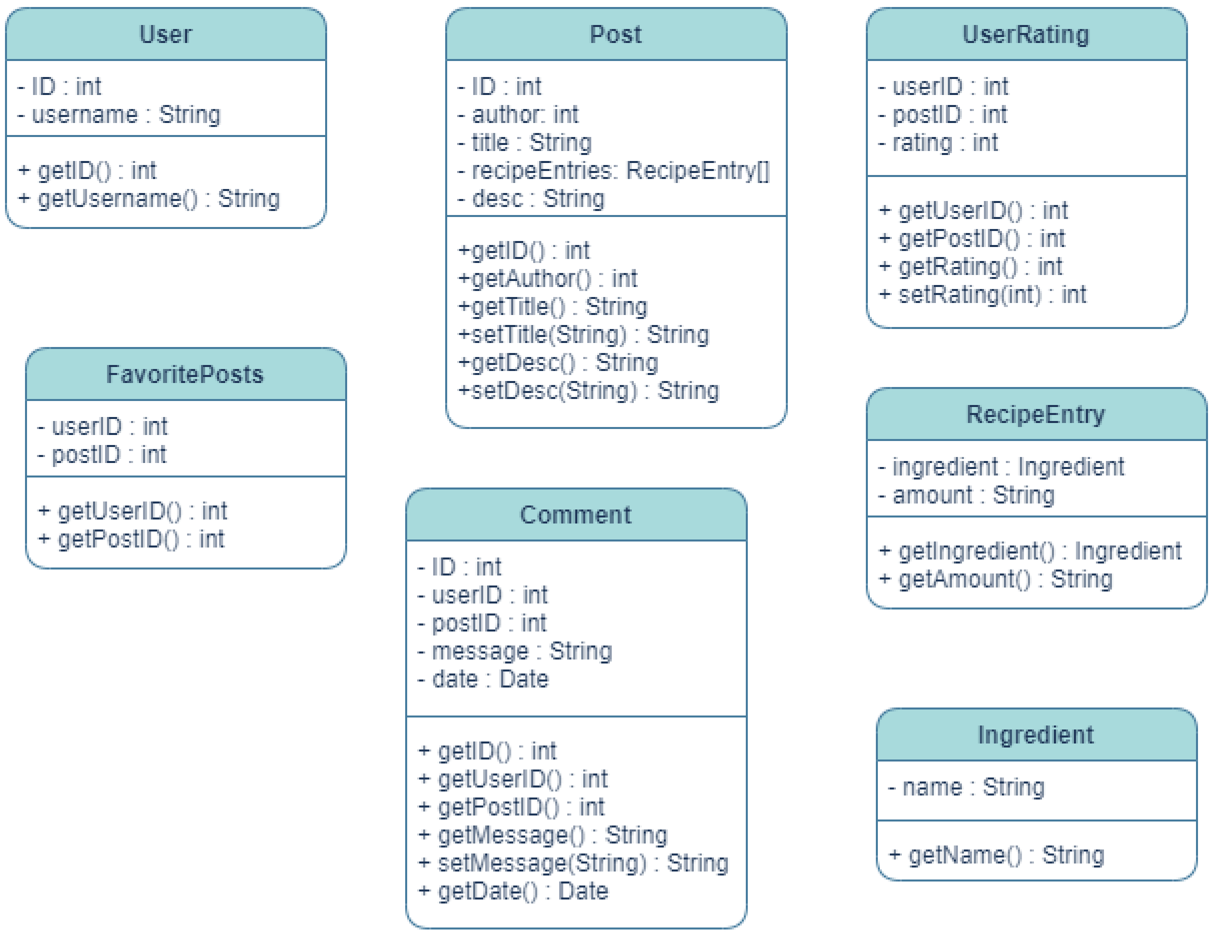


(Picture 6.1, Ingredient added)

When you are signed in, you may press the “Create Post” at the forum page (See picture 2). Picture 6 shows the page displayed, where you will be able to add a title, description as well as an ingredient list. To add an ingredient you must specify the name, amount and the unit of the amount. Press “Add” to add the entry to your recipe. This will display the ingredient under the Ingredients title, and there will be a red button with an “X” at the end of the row. (See picture 6.1). Any subsequent add of ingredients will create a new entry below the previous one and so on. If you type in the wrong ingredient, you can easily remove it from the list of entries by pressing the red button on the same row as the entry.

Once you feel ready, press the “Create Post” button to post your drink to the forum!

Design



Before our first supervision meeting we made a class diagram of what we thought the app would be like. Comparing the finished project with the original diagram, we can see that many of the things we sketched came to good use.

Both the server and client-side used Typescript for writing code. For the client-side we used HyperText Markup Language (HTML) and Cascading Style Sheets (CSS).

Server-side

- The server-side of the app is built using Node.js, with Express as the web framework.
- The *cors* library is used to handle Cross-origin Resource Sharing, which allows the client-side of the app to make requests to the server-side.
- The *express-session* library is used to manage user sessions on the server-side.
- MongoDB is used as the database for storing and retrieving data, for persistent data storage.

- The `mongoose` library is used as an Object-Document Mapper (ODM) to interact with the MongoDB database.
- Input validation is performed on the server-side using the *validator* library.
- Jest is used as the testing framework for server-side code.

Client-side

- The client-side of the app is built using React, with Emotion as the styling library and Material UI (MUI) as the component library.
- Axios is used to make HTTP requests to the server-side of the app to fetch and update data.
- The bootstrap library is used for styling, making the app responsive and providing a wide range of pre-designed UI components.
- Input validation is performed on the client-side using the *validator* library.
- Jest is used as the testing framework for the client-side code.

User related:

- **POST /user/** : Create a new user
    - Request parameters:
        - `username` (string): Username for the new user
        - `password` (string): Password for the new user
    - If the user is created successfully:
        - Status: 201 Created
        - Data: “User created”
    - If the request is invalid:
        - Status: 400 Bad Request
        - Data: <why bad request>
    - If the user already exists:
        - Status: 409 Conflict
        - Data: “User with username <username> already exists”
    - If there is an error:
        - Status: 500 Internal Server Error
        - Data: <Error message>
- **POST /user/login** : Log in as an existing user

- Request parameters:
    - `username` (string): Username for login
    - `password` (string): Password for login
- User successfully logged in:
    - Status: 200 Ok
    - Data: User Object
- If the request is invalid:
    - Status: 400 Bad Request
    - Data: <why bad request>
- The provided credentials are invalid:
    - Status: 401 Unauthorized
    - Data: “Bad username or password”
- If there is an error:
    - Status: 500 Internal Server Error
    - Data: <Error message>

- **POST /user/logout** : Log out the current user
    - Request parameters:
        - No request parameters needed
    - User successfully logged out:
        - Status: 200 Ok
        - Data: “Logged out”
    - User already logged out
        - Status: 204 No Content
        - Data: -
    - If there is an error:
        - Status: 500 Internal Server Error
        - Data: <Error message>
- **GET /user/username/:id** : Get the username associated with the user ID
    - Request parameters:
        - `id` (string): User ID to retrieve the username from
    - If the get request was successful:
        - Status: 200 Ok
        - Data: Username
    - If the request is invalid:
        - Status: 400 Bad Request
        - Data: <why bad request>
    - There was no user with given ID:

        - Status: 404 Not Found
        - Data: ”No user with id <id>”
    - If there is an error:
        - Status: 500 Internal Server Error
        - Data: <Error message>
- **POST /user/favourite** : Add a favourite post to the current user
    - Request parameters:
        - `postID` (string): Post ID to add as favourite
    - Post successfully added as a favourite to the user
        - Status: 200 Ok
        - Data: “PostID <postID> added as a favourite for user <user.username>”
    - If the request is invalid:
        - Status: 400 Bad Request
        - Data: <why bad request>
    - No user is currently logged in:
        - Status: 401 Unauthorized
        - Data: “Not logged in”
    - If there is an error:
        - Status: 500 Internal Server Error
        - Data: <Error message>
- **GET /user/favourite** : Get all the favourite posts from the current user
    - Request parameters:
        - No request parameters needed
    - If the get request was successful:
        - Status: 200 Ok
        - Data: List of Post Objects
    - No user is currently logged in:
        - Status: 401 Unauthorized
        - Data: “Not logged in”
    - Can not find favourite posts in user
        - Status: 404 Not Found
        - Data: “Bad GET call to <request.originalURL> — no favourite posts”
    - If there is an error:
        - Status: 500 Internal Server Error
        - Data: <Error message>

- **DELETE /user/favourite** : Delete a favourite post from the current user
    - Request parameters:
        - \`postID\` (string): Post ID to delete from favourites
    - Post successfully deleted from favourites in the user
        - Status: 200 Ok
        - Data: “PostID <postID> has been removed from <user.username>s favourites”
    - No user is currently logged in:
        - Status: 401 Unauthorized
        - Data: “Not logged in”
    - Can not find favourite posts in user
        - Status: 404 Not Found
        - Data: “Bad DELETE call to <request.originalURL> — no favourite post to delete”
    - If there is an error:
        - Status: 500 Internal Server Error
        - Data: <Error message>
- **GET /post/all/** : Returns all posts
    - If successful:
        - Status: 200 Ok
        - Data: <posts[]>
    - If there is an error:
        - Status: 500 Internal Server Error
        - Data: <Error message>
- **GET /post/all/user/:id/** : Returns all posts made by user with id :id, if no posts are made the returning array is empty.
    - Request parameter:
        - :id (string) : the id of the user
    - If successful:
        - Status: 200 Ok
        - Data: <posts[]>
    - If Id has wrong type:
        - Status: 400 Bad Request
        - Data: “Bad GET request to /post/all/user/<id> - ID has type <type>, should be string”
    - If there is an error:
        - Status: 500 Internal Server Error

        - Data: <Error message>
- **GET /post/:id/** : Returns the post with id :id
    - Request parameter:
        - :id (string) : the id of the post
    - If successful:
        - Status: 200 Ok
        - Data: <post>
    - If Id has wrong type:
        - Status: 400 Bad Request
        - Data: "Bad GET request to /post/<id> - ID has type <type>, should be string"
    - If post does not exists:
        - Status: 404 Not Found
        - Data: `No post with id <id>`
    - If there is an error:
        - Status: 500 Internal Server Error
        - Data: <Error message>

Post related:

- **POST /post/** : Create a post
    - Request parameters:
        - `title` (string) : Title of new post
        - `description` (string) : Description of new post
        - `recipeEntries` (recipeEntrie) : List of recipe entries for post
    - Post successfully created:
        - Status: 201 Created
        - Data: Post object
    - If there is no logged in user in session:
        - Status: 401 Unauthorized
        - Data: "Not logged in."
    - If any parameter has wrong type:
        - Status: 400 Bad Request
        - Data: "Bad POST call to /post/ - <parameter> has type <type>, should be <type>"
    - If there is an error:
        - Status: 500 Internal Server Error
        - Data: <Error message>

- **POST /post/review/** : Add review to a post
  - Request parameters:
    - `postID` (string) : The reviewed post's id
    - `comment` (string) : Review text
    - `rating` (number = {1,2,3,4,5}): Review rating
  - Review successfully added:
    - Status: 200 Ok
    - Data: "Review added to post <postID>"
  - If any parameter has wrong type:
    - Status: 400 Bad Request
    - Data: "Bad POST call to /post/review/ - <parameter> has type <type>, should be <type>"
  - If rating != {1,2,3,4,5}:
    - Status: 400 Bad Request
    - Data: "Bad POST call to /post/review/ - rating <rating> is not one of [1,2,3,4,5]"

  - If the user already made a review:
    - Status: 409 Conflict
    - Data: "Bad POST call to /post/review/ - user already reviewed this post
  - If no post with postID exists:
    - Status: 404 Not Found
    - Data: "No post with id <postID>"
  - If there is an error:
    - Status: 500 Internal Server Error
    - Data: <Error message>
- **DELETE /post/review/** : Delete a review
  - Request parameters:
    - `postID` (string) : The reviewed post's id
  - Deleted successfully:
    - Status: 204 No Content
    - Data: `Review by <Session.user> has been deleted successfully`
  - If there is no logged in user in session:
    - Status: 401 Unauthorized
    - Data: "Not logged in."

- If postID has wrong type:
    - Status: 400 Bad Request
    - Data: `Bad DELETE call to /post/review/` - postID has type <type>, should be string
- If user has no review posted:
    - Status: 409 Conflict
    - Data: `Bad DELETE call to /post/review/ - user has not reviewed this post`
- If review was not deleted:
    - Status: 500
    - Data: “Something went wrong trying to delete the review by <Session.user>”
- If there is an error:
    - Status: 500
    - Data: <Error message>

Responsibilities

In the beginning we all shared and did everything together. But we had to divide after lab 2 so Filip & Lukas took care of the use cases while Ernst & Casper made the tests. This report was a shared effort.