AFK CookBook Implementation Documentation

Filip Korus, Aleksandra Łapczuk, Krzysztof Wdowczyk

January 2024

Contents

1 Introduction **4**

2 Project scope **4**

3 Project structure **4**

4 Project architecture **4**

5 Directory: frontend **5**
5.1 Used libraries . . . 5
5.2 Subdirectory: public . . . 6
5.3 Subdirectory: src . . . 6
5.3.1 api . . . 6
5.3.2 assets . . . 7
5.3.3 components . . . 7
5.3.4 config . . . 8
5.3.5 context . . . 8
5.3.6 hooks . . . 9
5.3.7 pages . . . 9
5.3.8 styles . . . 10
5.3.9 theme . . . 10
5.3.10 types . . . 10
5.3.11 utils . . . 10
5.3.12 App.tsx . . . 11
5.3.13 main.tsx . . . 11
5.4 index.html . . . 11

6 Directory: backend **11**
6.1 Used libraries . . . 11
6.2 config . . . 12
6.3 prisma . . . 12
6.4 src . . . 12
6.4.1 server.ts . . . 12
6.4.2 initdb.ts . . . 12
6.4.3 index.ts . . . 13
6.4.4 middlewares . . . 13
6.4.5 routes . . . 13
6.4.6 utils . . . 14

7 Directory: nginx **14**
7.1 Dockerfile . . . 14
7.2 nginx.conf . . . 14

8 HTTP REST API **15**
8.1 POST /auth/login . . . 15
8.1.1 Input parameters . . . 15
8.1.2 Success output . . . 15
8.1.3 Status codes . . . 15
8.1.4 Additional description . . . 15
8.2 POST /auth/refresh . . . 15
8.2.1 Input parameters . . . 15
8.2.2 Authorization . . . 15
8.2.3 Success output . . . 16
8.2.4 Status codes . . . 16
8.2.5 Additional description . . . 16
8.3 GET /auth/logout . . . 16
8.3.1 Input parameters . . . 16
8.3.2 Authorization . . . 16
8.3.3 Success output . . . 16
8.3.4 Status codes . . . 16
8.3.5 Additional description . . . 16
8.4 GET /user . . . 16
8.4.1 Input parameters . . . 16
8.4.2 Authorization . . . 16
8.4.3 Success output . . . 16
8.4.4 Status codes . . . 16
8.4.5 Additional description . . . 17
8.5 GET /user/{id} . . . 17
8.5.1 Input parameters . . . 17
8.5.2 Authorization . . . 17
8.5.3 Success output . . . 17
8.5.4 Status codes . . . 17
8.5.5 Additional description . . . 17
8.6 GET /recipe/{id} . . . 17
8.6.1 Input parameters . . . 17
8.6.2 Authorization . . . 17
8.6.3 Success output . . . 17
8.6.4 Status codes . . . 17
8.6.5 Additional description . . . 17
8.7 GET /recipe . . . 18
8.7.1 Input parameters . . . 18
8.7.2 Authorization . . . 18
8.7.3 Success output . . . 18
8.7.4 Status codes . . . 18
8.7.5 Additional description . . . 18
8.8 POST /recipe . . . 19
8.8.1 Input parameters . . . 19
8.8.2 Authorization . . . 19
8.8.3 Success output . . . 19
8.8.4 Status codes . . . 19
8.8.5 Additional description . . . 19
8.9 PUT /recipe/{id} . . . 19
8.9.1 Input parameters . . . 19
8.9.2 Authorization . . . 19
8.9.3 Success output . . . 20
8.9.4 Status codes . . . 20
8.9.5 Additional description . . . 20
8.10 DELETE /recipe/{id} . . . 20
8.10.1 Input parameters . . . 20

8.10.2 Authorization . . . 20
8.10.3 Success output . . . 20
8.10.4 Status codes . . . 20
8.10.5 Additional description . . . 20
8.11 POST /recipe/review/{recipeId} . . . 20
8.11.1 Input parameters . . . 20
8.11.2 Authorization . . . 20
8.11.3 Success output . . . 21
8.11.4 Status codes . . . 21
8.11.5 Additional description . . . 21
8.12 GET /recipe/review/{recipeId} . . . 21
8.12.1 Input parameters . . . 21
8.12.2 Authorization . . . 21
8.12.3 Success output . . . 21
8.12.4 Status codes . . . 21
8.12.5 Additional description . . . 22
8.13 PUT /recipe/review/{reviewId} . . . 22
8.13.1 Input parameters . . . 22
8.13.2 Authorization . . . 22
8.13.3 Success output . . . 22
8.13.4 Status codes . . . 22
8.13.5 Additional description . . . 22
8.14 DELETE /recipe/review/{reviewId} . . . 22
8.14.1 Input parameters . . . 22
8.14.2 Authorization . . . 22
8.14.3 Success output . . . 22
8.14.4 Status codes . . . 22
8.14.5 Additional description . . . 23
8.15 GET /recipe/review/stars/{recipeId} . . . 23
8.15.1 Input parameters . . . 23
8.15.2 Authorization . . . 23
8.15.3 Success output . . . 23
8.15.4 Status codes . . . 23
8.15.5 Additional description . . . 23
8.16 GET /recipe/ingredient/{name} . . . 23
8.16.1 Input parameters . . . 23
8.16.2 Authorization . . . 23
8.16.3 Success output . . . 23
8.16.4 Status codes . . . 24
8.16.5 Additional description . . . 24
8.17 GET /recipe/category/{name} . . . 24
8.17.1 Input parameters . . . 24
8.17.2 Authorization . . . 24
8.17.3 Success output . . . 24
8.17.4 Status codes . . . 24
8.17.5 Additional description . . . 24
8.18 GET /recipe/ingredients/{commaSeparatedNames} . . . 25
8.18.1 Input parameters . . . 25
8.18.2 Authorization . . . 25
8.18.3 Success output . . . 25
8.18.4 Status codes . . . 25
8.18.5 Additional description . . . 25
8.19 GET /recipe/categories/{commaSeparatedNames} . . . 25
8.19.1 Input parameters . . . 25
8.19.2 Authorization . . . 26
8.19.3 Success output . . . 26

8.19.4 Status codes 26
8.19.5 Additional description 26

9 Tests **26**
9.1 Functional requirements 26
9.1.1 Manual tests 26
9.1.2 Automated tests 28
9.2 Automated Tests 28
9.3 Non-Functional requirements 30
9.3.1 Manual tests 30

1 Introduction

The purpose of this implementation documentation is to provide a comprehensive overview of the structure, features, and key technical aspects of the AFK CookBook project. The documentation is organized into sections, starting with an overview of the system architecture, followed by the implementation of specific features, and concluding with testing. Each section provides detailed insights into the implementation, including decisions and challenges faced during the process.

2 Project scope

AFK CookBook is a culinary application designed to provide users with a platform to share and discover recipes. The project includes key features such as Google OAuth2 login, user account management, recipe creation, browsing and reviewing. Additionally, it allows users to search for recipes based on available ingredients. The AFK CookBook system architecture is built on a client-server model. The client side consists of a web application accessible through standard web browsers, while the server side hosts the application logic, user data, and database.

3 Project structure

AFK CookBook is organized into a structured directory layout to enhance maintainability, scalability, and code readability. The project structure consists of three main directories named frontend, backend, and nginx. The project is primarily written in TypeScript, both on the frontend and backend, to ensure consistency throughout the application. The frontend uses popular libraries such as React.js, while the backend leverages Node.js and Express.js. MySQL is employed as the database technology for data storage.

4 Project architecture

The architecture consists of five Docker containers: frontend, backend, PHPMyAdmin, MySQL and nginx container. They can be built by docker compose command. Nginx container acts as proxy server. Backend uses MySQL database located in MySQL container and PHPMyAdmin is used to manage data from this database, which makes them dependent on MySQL container. The diagram is presented on Figure 1.

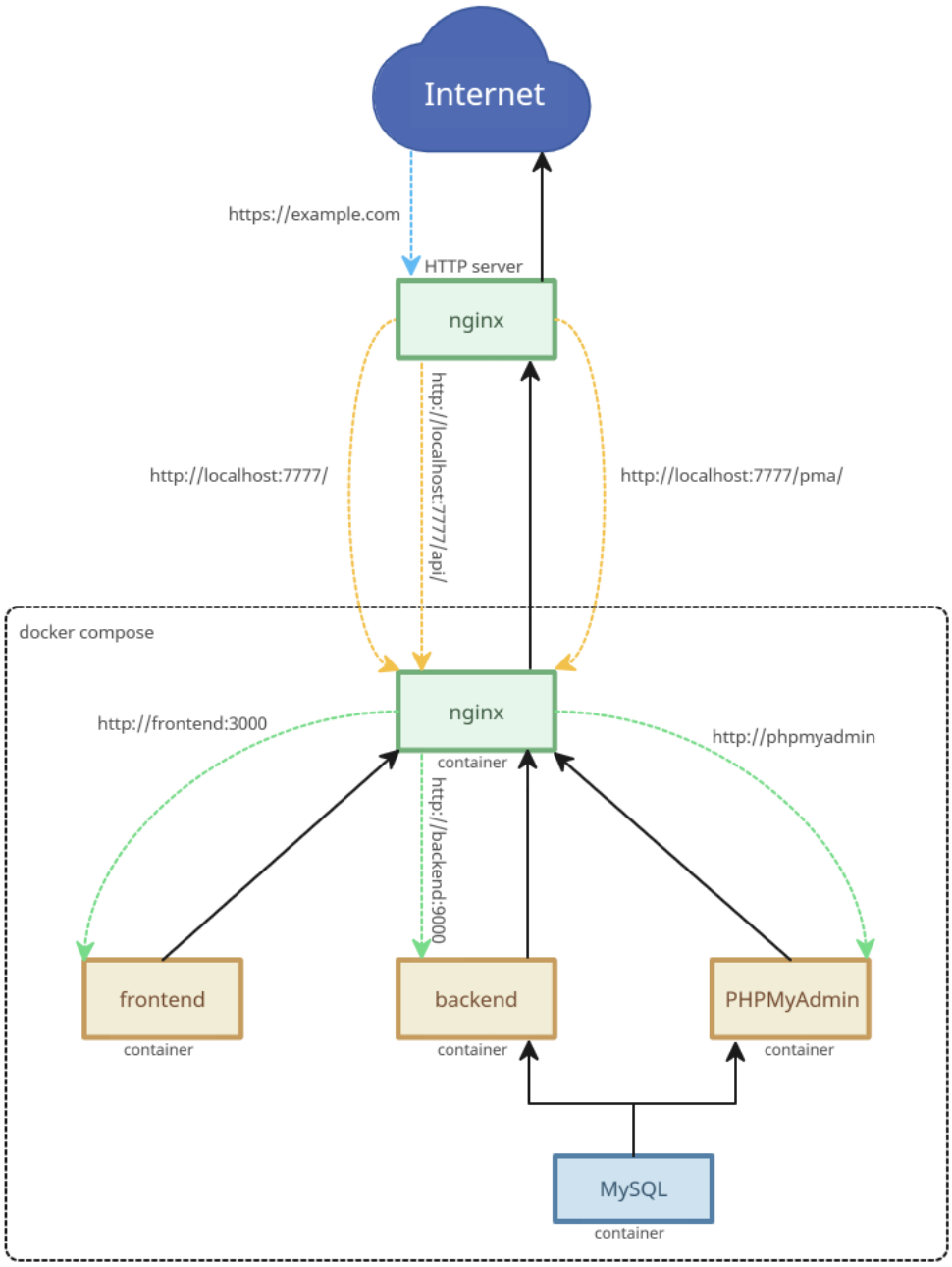

Figure 1: Architecture diagram

5 Directory: frontend

5.1 Used libraries

- axios (version 1.5.0) - A library for making HTTP requests
- emotion/react (version 11.11.1) - A library for writing styles with JavaScript, providing styling solutions for React applications
- emotion/styled (version 11.11.0) - A utility for styling React components with the '@emotion/react' library
- mui/icons-material (version 5.14.9) - Material-UI icons for React applications

- mui/joy (version 5.0.0-beta.19) - an open-source React component library that implements MUI's own design principles. It's comprehensive and can be used in production out of the box
- mui/material (version 5.14.10) - Material-UI core components and styling for React applications
- react-oauth/google (version 0.12.1) - A library for integrating Google OAuth authentication into React applications
- react (version 18.2.0) - A JavaScript library for building user interfaces
- react-dom (version 18.2.0) - A package for rendering React components into the DOM
- react-router-dom (version 6.16.0) - A library for declarative routing in React applications
- uuidv4 (version 6.2.13) - A library for generating UUIDs (Universally Unique Identifiers) according to the RFC4122 standard.
- zod (version 3.22.4) - A library for data validation and schema definition in JavaScript/TypeScript.

5.2 Subdirectory: public

This directory contains the AFK logo in .ico format, which is then implemented in the index.html file as an icon.

5.3 Subdirectory: src

5.3.1 api

- auth.ts - responsible for handling authentication-related API requests. File consists of 3 functions: login, logout and refreshToken.
  The login function sends a POST request to the /auth/login endpoint with the provided credential data.
  The logout function sends a GET request to the /auth/logout endpoint.
  The refreshToken function sends a POST request to the /auth/refresh endpoint. It does not provide any additional data in the request body, but it includes the withCredentials: true option.

- index.ts - configures an Axios instance for making HTTP requests with token-based authentication.
  The axios.create method is employed to establish the Axios instance, configuring it with a base URL specified in CONFIG.API_URL. Subsequently, an interceptor is incorporated into the instance through api.interceptors.response.use. This interceptor actively monitors for a 401 response status.
  In the event of encountering a 401 and ensuring the refresh flag remains unset, it endeavors to refresh the token via the refreshToken function sourced from auth.ts. Upon a successful token refresh, the interceptor updates the headers with the new token and initiates a retry of the original request.

- recipe.ts - exports a set of functions that interact with the recipe-related endpoints of the API: createRecipe, getRecipeById, getRecipes, getRecipesByUserId and type RecipeToAdd.
  RecipeToAdd specifies the structure of an object used for creating a new recipe.
  createRecipe function is designed to handle the creation of a new recipe by sending a POST request to the /recipe endpoint.
  getRecipeById retrieves a recipe by its unique identifier (ID) by making a GET request to the /recipe/id endpoint.
  getRecipes function is designed to retrieve a list of recipes from the API with optional pagination and filtering.
  getRecipesByUserId returns fetched array of Recipe objects created by given user.

- user.ts - retrieves information about the currently logged-in user by making a GET request to the /user endpoint

- geocode.ts - performs reverse geocoding using coordinates (latitude and longitude). Parameter 'coords' contains the latitude and longitude coordinates.
- review.ts - define functions related to managing reviews for recipes (createReview, getReviews, deleteReview,editReview,getStars). It uses various methods (PUT, DELETE, POST, GET) to /recipe/review retrieve information.

5.3.2 assets

Assets directory contains all the static files like images, etc. In this case the AFK logo in .png format, which is used and presented in application login page.

5.3.3 components

Components directory is organized in three subdirectories: recipe, routing and sidebar. Components use the Material-UI (mui) library.
In sidebar folder a menu directory and SidebarNavigation.ts file is located.

- SidebarNavigation.ts - component that encapsulates the rendering of a navigation list. The actual content and structure of the navigation items are defined within the SidebarMenu component.
- menu
  - SidebarMenu.ts - component responsible for showing user information and implementing buttons 'Create Recipe','My recipes', 'Search Recipes','Wall', 'Log out'. Every button redirects to specific subpage which is given in linkTo attribute in particular SidebarMenuItem.
  - SidebarMenuItem.ts - reusable component that defines a set of props that the SidebarMenuItem component can accept. It provides a clickable button with support for routing, custom text, titles, and associated icons.

In routing subdirectory two files can be found - PrivateRoute.tsx and RouterLink.tsx.

- PrivateRoute.tsx - component used for creating private routes in an application. It leverages the useAuth to determine if a user is authenticated. If the user is authenticated, it renders the nested content within the Outlet. If not, it redirects the user to the login page while preserving the original intended route.
- RouterLink.tsx - component that acts as a wrapper around the Link component. It provides a convenient way to create links for internal routing in a React application.

The recipe subfolder contains EditRecipe.tsx, RecipeCard.tsx, RecipeForm.tsx, RecipeFormPagination.tsx files and review subdirectory that contains rewiev related components.

- review
  - ReviewComment.tsx - component used to display and manage a review comment, allowing the author to edit or delete their own review.If the user is not authenticated, the component renders nothing.
    Editing or deleting triggers asynchronous API requests for handling review modifications.The onUpdate callback can be used to update the parent component's state upon successful modifications.
  - ReviewCommentListPagination.tsx - displays a paginated list of review comments within a recipe. The component is used internally to render each individual review.
    Pagination controls are only visible when there is more than one page of reviews. Page navigation triggers the handlePageChange callback to update the active page.
  - ReviewCreate.tsx - component used to provide a form for users to submit a new review for a specific recipe. It uses the ReviewForm component internally. CreateReview API function sends the review data to the server.

  - ReviewForm.tsx - responsible for rendering a form to submit or edit reviews. Component render a form with fields for stars rating and a comment. It utilizes the onSubmit callback for successful form submissions and supports the onCancel callback for canceling the form submission or editing.
  - ReviewSection.tsx - component responsible for displaying and managing recipe reviews. It displays the overall stars rating for the recipe, toggles the visibility of recipe reviews with a button, fetches and displays reviews dynamically with pagination and allows users to submit new reviews or edit their existing ones.
  - ReviewStars.tsx - component displaying stars rating for a recipe, including the average score, visual star representation, and the total count of reviews. It provides a visually appealing and informative representation of the stars rating.
  - ReviewSkeletonComment.tsx - component providing loading state representation for a review comment.

- EditRecipe.tsx - component allowing users to edit an existing recipe. It utilizes the RecipeForm component to present a form for modifying recipe details and communicates with the server to update the recipe data.
  The component renders a form for editing the recipe, populated with the existing recipe data. Upon submission, the updateRecipe function is called to update the recipe on the server. If successful, the user is redirected to the recipe details page with a success message; otherwise, error messages are displayed.

- RecipeCard.tsx - represents a card displaying details of a recipe, including the author, title, categories, ingredients, cooking time, and description. It is designed to use in various contexts, such as the wall view or within the recipe details page.
  It displays essential information about a recipe and supports dynamic rendering based on whether the user is viewing their own recipe or someone else's recipe. There is also a possibility for editing a recipe if the current is the author of the snippet.

- RecipeForm.tsx - component responsible for creating and editing recipes.Supports input for recipe title, cooking time, description, categories, ingredients, and location. Users are allowed to add and remove categories and ingredients dynamically. There is an option to use current user's geolocation or let manually enter user's location.

- RecipeListPagination.tsx - renders a paginated list of recipe cards with optional pagination controls.

5.3.4 config

index.ts located in config directory defines various configuration parameters, including environmental variables, API URLs, and specific constraints for different features. It retrieves the environment details using 'import.meta.env' and determines whether the application is in production. In the context of API it defines a function 'getApiUrl' to dynamically generate the API URL based on the environment and domain.
Additionaly, the component specifies constraints for recipe-related parameters, such as title length, description length, location length, latitude/longitude range, ingredient quantity, and category quantity, which ensures consistency and flexibility across different components and features.

5.3.5 context

Context directory contains three files: AuthContext.tsx, CookbookContext.tsx, SidebarContext.tsx. Context providers help manage and share stateful information across different components in an application.

- AuthContext.tsx - manages user authentication-related state and provides methods for handling user login and logout. This context is crucial for managing the authentication flow within the application. Implemented useAuth hook allows components to access the AuthContext.
  The AuthProvider component is a context provider responsible for managing the authentication

state and providing it to the entire application.
Two methods are defined: handleLogin and handleLogout.

  - handleLogin attempts to log in a user using the provided credential and returns a Promise resolving to a SuccessOrErrorMessage object.
  - handleLogout logs out the current user and returns a Promise resolving to a SuccessOrErrorMessage object.

- CookbookContext.tsx is a placeholder for future functionalities related to a cookbook. It provides a structure for potential future extensions.
  The useCookbook hook allows components to access the CookbookContext.The CookbookProvider component is a context provider responsible for managing the cookbook-related context.

- SidebarContext.tsx manages the state and functionality related to the sidebar. It provides a way to control the visibility of the sidebar within the application.
  The useSidebar hook allows components to access the SidebarContext. It provides access to the state and the function to set the visibility of the sidebar.
  The SidebarProvider component is a context provider responsible for managing the sidebar-related context. It initializes the default state for the sidebar's visibility.

5.3.6 hooks

The Hooks section provides opportunities for implementing custom hooks to manage various functionalities. Currently, the folder contains two created hooks: useForm.ts and useItemList.ts.

- useForm.ts is designed to manage form state and simplify handling input changes in React components. It is particularly useful for managing form data, such as text inputs, checkboxes, and text areas.
  The handleInputChange function is responsible for updating the form data based on user input. It automatically handles different input types.
  The resetForm function resets the form to its initial values.
  The setNewFormValues function allows setting new values for the entire form. This hook is used in ReviewCreate.tsx and CreateRecipePage.tsx components.

- useItemList.ts is a custom hook designed to manage a list of items in a React component. It provides functions to add, remove, and update items within the list.
  HandleItemChange function updates the item at the specified index with the new value. AddItem function Adds a new item to the list, subject to the optional maxItems limit if provided. RemoveItem is responsible for removing the item at the specified index from the list and resetList Resets the list to its initial values provided during the hook instantiation. The hook is used in CreateRecipePage.tsx component.

5.3.7 pages

The pages directory contains various components responsible for rendering different views for the user. Each page is designed to cater to specific functionalities and features of the application, with error handling implemented where necessary.

- errors subdirectory and NotFound.tsx located within are designed to handle 404 errors. The 'NotFound.tsx' component is automatically invoked when the application encounters a 404 error which ensure user-friendly experience for handling missing pages.

- CreateRecipePage.tsx allows users to create new recipes within the application. It provides an intuitive and responsive form with various input fields, including title, cooking time, description, categories, ingredients, and more.
  The component is integrated into the application's routing system, allowing users to access it through the appropriate navigation links. Upon submitting the form, the component triggers a request to create the recipe through the API. Success and error messages provide users with feedback on the outcome of their submission.

- DrawerView.tsx serves as a layout structure for pages within the AFK Cookbook application. It includes a responsive sidebar with navigation links and a main content area. The sidebar can be toggled open and closed, providing a user-friendly interface for navigating through the application.

- LoginPage.tsx handles user authentication through OAuth (Google login). It provides an interface for logging in, displaying the application logo and a Google login button. The component is designed to be used as the initial entry point for user authentication. It can be accessed by users who are not currently logged in.

- RecipePage.tsx retrieves and displays details of a specific recipe based on the provided ‘recipeId‘. It uses the ‘getRecipeById‘ API to fetch recipe data and renders the information using the ‘RecipeCard‘ component.If the recipe cannot be found (HTTP 404) it redirects to the home page.

- RecipesOfUserPage.tsx displays a paginated list of recipes associated with a specific user. It supports filtering recipes based on the user's selection of public and private recipes. The page also includes pagination controls to navigate through the recipe list.
  The component extracts the ‘userId‘ from the route parameters using React Router's ‘useParams‘ hook and verifies the current user's ID against the requested user's ID to ensure authorization. Also it fetches a paginated list of recipes associated with the specified user from the server and displays pagination controls to navigate through the list of recipes. Users are allowed to filter their recipes whether they are public or private.

- RecipeWallPage.tsx displays a paginated list of recipes based on different "walls," depending if it the main wall, category wall, or ingredient wall. It allows users to filter recipes based on whether they are authored by the current user(the user must be authenticated beforehand).
  The component utilizes pagination controls to navigate through the recipe list. It utilizes URL parameters for pagination and retains user-selected filters in the URL.

5.3.8 styles

In this directory pages.css file is configured. A logo animation in login page is configured here, as well as the custom styling in the application.

5.3.9 theme

The index.ts file in theme directory is responsible for configuring the Material-UI theme used. It utilizes the ‘createTheme‘ and ‘responsiveFontSizes‘ functions to define the visual appearance of the user interface. This theme configuration provides a consistent visual style throughout the application, ensuring a cohesive and appealing user experience.

5.3.10 types

This subdirectory contains TypeScript files defining various types used within components in the application. Each type provides a clear structure and definition for specific data models used in the corresponding components such as Action.ts, Category.ts, Coords.ts, ErrorFields.ts, RecipeToAddOrToEdit.ts, Ingredient.ts, Recipe.ts, User.ts, Review.ts, ReviewToAdd.ts, Stars.ts.

5.3.11 utils

Utils directory consists of one subfolder named date where 2 files are located: formatDate.ts and timeSince.ts.

- formatDate.ts is used for formatting date values into a human-readable string. It accepts either a string or a ‘Date‘ object as input and returns a formatted date string. The function uses the Intl.DateTimeFormat API for localization and formatting based on the user's locale. If the navigator.language is not available, the default locale is set to 'en-US'. This file is used in SidebarMenu.tsx component.

- timeSince.ts is a function that calculates and returns a string indicating the time elapsed since a given date. It accepts either a string or a 'Date' object as input. The result is formatted in terms of years ('y'), months ('mo'), days ('d'), hours ('h'), minutes ('min'), or seconds ('s'). This file is used in RecipeCard.tsx and ReviewComment.tsx.

5.3.12 App.tsx

The App.tsx file defines the main application structure, routing, and page components. It utilizes the react-router-dom library to define routes and render different components based on the URL path. The structure includes authentication handling for private routes, navigation to various pages, and error handling for routes that do not match any defined pattern. Some of the routes are protected and wrapped by PrivateRoute component - it checks the authentication status using the useAuth context and redirects unauthenticated users to the login page.

5.3.13 main.tsx

The main entry point for the AFK Cookbook web application is the 'main.tsx' file, responsible for initializing the React application. The primary dependencies and context providers are configured in this file to ensure a seamless integration of features and user authentication. The 'main.tsx' file establishes the application structure and theme.

5.4 index.html

The index.html file serves as the foundation for the AFK Cookbook web application.
The main entry point for the AFK Cookbook application is the main.tsx file, located at /src/main.tsx. This TypeScript file serves as the starting point for the application, initializing the React components and defining the application's structure.

6 Directory: backend

6.1 Used libraries

- prisma/client (5.2.0) - Database toolkit for TypeScript and Node.js.
- cookie-parser (1.4.6) - A middleware for parsing cookies in Express applications.
- cors (2.8.5) - Cross-Origin Resource Sharing middleware for Express, allowing secure communication between different origins.
- dotenv (16.3.1) - Loads environment variables from a .env file into process.env.
- express (4.18.2) - Web framework for Node.js, used for building the web server.
- google-auth-library (9.2.0) - Google's official library for authenticating with Google APIs.
- jsonwebtoken (9.0.2) - JSON Web Token (JWT) implementation for Node.js. Used for creating and verifying JWTs, commonly used for authentication.
- simple-oauth2 (5.0.0) - A simple OAuth2 client for Node.js, used for handling OAuth2 authentication.
- uuid (9.0.1) - A library for generating universally unique identifiers (UUIDs).
- winston (3.10.0) - A versatile logging library for Node.js, providing support for multiple transports.
- zod (3.22.4) - A TypeScript-first schema declaration and validation library. It is likely used for data validation in the application.

6.2 config

Config directory contains one file named index.ts, which is responsible for the application's configuration files and parameters that control the behavior of the application.
The checkConfigFields function provided in the index.ts file is crucial for validating the integrity of the configuration. It ensures that all required fields are defined and not set to null or undefined. This validation process helps catch potential configuration issues early in the application lifecycle.

6.3 prisma

In the schema.prisma file there is a database created for the project. The database connection is configured using the datasource block,which is responsible for specifying the details of the database connection.The database provider is set to mysql. The database consists of the following tables: User, RefreshToken, Settings, Recipe, RecipeReview, Ingredient, RecipeIngredient, Category and RecipeCategory. Fields and relations between tables are presented on Figure 2.

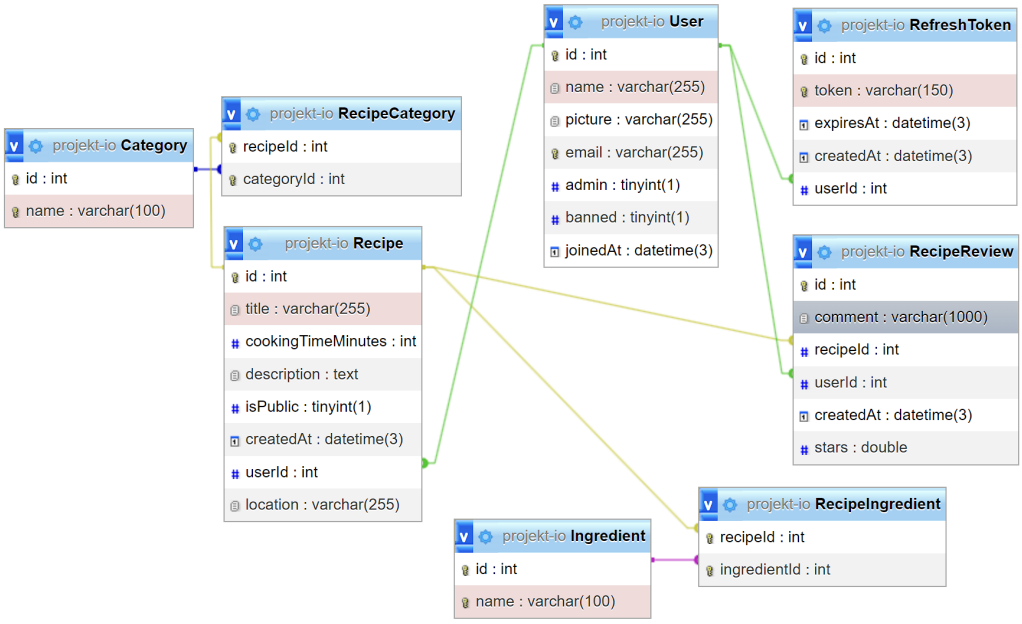

Figure 2: Database schema

6.4 src

6.4.1 server.ts

Server.ts file organizes and centralizes the setup of application which makes it modular and easy to maintain.
Cross-origin requests are permitted using the cors middleware, with allowed origins defined in the config.ALLOWED_ORIGINS array which is implemented in config/index.ts.
Application uses cookieParser to handle cookie parsing and express.json() to parse incoming JSON data.
There is use of custom middlewares - authenticate and requestLogger, which are implemented in 'middlewares' subdirectory.
The application integrates the main router (router), which contains various routes and endpoints. The 404 Not Found Handler is used, as it is implemented in messages.ts module.

6.4.2 initdb.ts

Function responsible for initializing a database.

6.4.3 index.ts

File responsible for serving as the entry point for initializing the application. The functionality is structured to ensure a smooth startup process and provide informative logging.
At first, the module checks the configuration fields using the checkConfigFields and initializes the database using the initDB function. After the steps are completed, there is a message what port server is listening on. In the event of any errors during the initialization process, the application catches and logs the error message using the logger.

6.4.4 middlewares

- authenticate.ts - component responsible for authenticating users based on the provided access token. It is designed to be integrated into Express.js routes to secure and control access to protected.
  The middleware handles scenarios where the access token is not provided, is invalid, or does not correspond to a valid user. In such cases, it proceeds with the next middleware or route handler without setting authentication status.
  Authenticate.ts etrieves the user details from the database based on the user ID extracted from the access token and sets res.locals.authenticated to true and res.locals.user to the authenticated user. This makes user information available in subsequent middleware or route handlers.

- requestLogger.ts - designed to log HTTP requests in an Express.js application. It captures key information such as the client's IP address, authentication status, HTTP method, and the requested URL.
  'RequestLogger' retrieves the client's IP address from headers such as 'CF-Connecting-IP' (header provided by Cloudflare) and 'x-forwarded-for' (header set by NGINX proxy). If unavailable, it falls back to the default IP provided by req.ip.
  Also the component checks if the incoming request is associated with an authenticated user. It displays 'A' in green if authenticated and 'U' in red if unauthenticated.

- requireAuth.ts - responsible for ensuring that an incoming request is only processed if the user is authenticated. If res.locals.authenticated is set to true the middleware allows the request to proceed to the next middleware or route handler.

6.4.5 routes

In routes directory files with particular extensions are located. The extensions are:

- .router.ts - where application routes are defined, specifying which functions handle these routes, and determining which middleware should be used during request processing.

- .controller.ts - responsible for defining functions that handle HTTP requests (request handlers).

- .service.ts - responsible for providing service functions that handle interactions with the database. In practice, these files contain methods for manipulating data in the database, providing business logic, and are called by controllers.

Subdirectiories inside routes folder are responsible for particular functions:

- user is responsible for user-related operations, such as registration, login, and retrieving user information.

- auth is dedicated to authentication and security. It handles requests related to authentication and token generation.

- recipe handles operations related to recipes. It contains functions for adding, removing, updating, and retrieving recipes.

- review deals with operations related to recipe reviews. Endpoint associated: /recipe/review/

The directory path inside the "routes" directory creates the API URL schema. For instance: router file for /recipe/review/ API endpoints are located under routes/recipe/review/ directory path.

6.4.6 utils

Folder containing subdirectories and modules related to HTTP code responses, pagination, object validation, logging, and console colors.

- checkObjectValuesNotNull.ts - function that recursively checks if there are any null or undefined values in an object. If there are null or undefined values, returns the path of the first null field as a string.
- colors.ts - module used to provide ANSI escape codes for various colors that can be used in the console.
- logger.ts - module that configures and exports a logger using Winston for logging messages. The logger can output messages to the console and, if configured, to a log file.
- logger.ts - module that configures and exports a logger using Winston for logging messages. The logger can output messages to the console and, if configured, to a log file.
- validateObject.ts - provides a function for validating objects. It can return either the validated data or an array of validation errors.
- pagination - Pagination subdirectory contains module paginationParams.ts which is responsible for calculating pagination parameters based on input values. It is used in review.controller.ts and recipe.controller.ts
- httpCodeResponses
  - messages.ts module serves as a centralized collection of functions that generate API responses for various HTTP statuses. These functions aim to provide consistent and standardized responses for different scenarios encountered in an API.
  - respond.ts is responsible for creating and sending standardized JSON responses in an Express application.
    Function generates a JSON response with the specified message, HTTP status code, and optional data. The response includes a success flag indicating whether the operation was successful, the provided message, and any additional data.

7 Directory: nginx

7.1 Dockerfile

Dockerfile is responsible for copying .conf files to the /etc/nginx directory in the NGINX image and then listing and displaying the contents of the nginx.conf file.

7.2 nginx.conf

An nginx.conf file specifies global configuration settings for the NGINX HTTP server. It is used as a reverse proxy to forward requests to the appropriate backend servers. The configuration file defines how NGINX handles different types of requests and where it forwards them. NGINX is configured to run with the 'nginx' user and dynamically sets the number of worker processes based on available CPU cores.

- worker_connections sets the maximum number of simultaneous connections each worker can handle
- worker_rlimit_nofile sets the maximum number of file descriptors

MIME (Multipurpose Internet Mail Extensions) types are also included. They are used to specify the type of content that is being sent over the network. The default type is set to application/octet-stream.
The nginx.conf file also includes configurations for requests redirection using proxy_pass, such as

forwarding requests to PHPMyAdmin, Frontend and Backend.
Requests to the /api endpoint are forwarded to the backend server (http://backend:9000). The Cache-Control header is set to no-store which means that responses should not be cached.
Requests to the /pma endpoint are forwarded to the PHPMyAdmin server (http://phpmyadmin/). The Cache-Control header is set to "no-store."
The location block for the root path (/) explicitly sets the Cache-Control header to cache responses for 10 minutes. Responses are forwarded to the frontend server (http://frontend:3000).
For all other paths, responses are cached for 1 day, and the immutable directive is used to indicate that the resource will not change. Requests are forwarded to the frontend server (http://frontend:3000).

8 HTTP REST API

Every API endpoint returns JSON response in shape: { success, message, data } where success is a boolean indication success, message is a string status message and data is an optional object with additional data returned from an endpoint.

8.1 POST /auth/login

8.1.1 Input parameters

Request body:

- credential - string containing OAuth token

8.1.2 Success output

- token - JWT access token string
- user - currently logged User object

8.1.3 Status codes

- 200 - user logged in
- 201 - user registered in (logged in for the first time)
- 400 - some request body fields are missing or error occurred when creating new row in the User's table in the database
- 401 - user not found in the database
- 403 - user is banned

8.1.4 Additional description

Logs in the user or creates new account if he have not already logged in.

8.2 POST /auth/refresh

8.2.1 Input parameters

HTTP-only cookies:

- refreshToken - string containing JWT refresh token

8.2.2 Authorization

JWT refresh token inside HTTP-only cookie named 'refreshToken'

8.2.3 Success output

- token - refreshed JWT access token

8.2.4 Status codes

- 200 - access token has been renewed
- 401 - refresh token has not been supplied or is invalid

8.2.5 Additional description

Renews JWT access token.

8.3 GET /auth/logout

8.3.1 Input parameters

There are no input parameters.

8.3.2 Authorization

JWT access token inside Bearer token.

8.3.3 Success output

There is no output however HTTP-only cookie named 'refreshToken' is being set to age equal 0 which means it is terminated immediately. Its value is also set to empty string to make sure it is no longer usable.

8.3.4 Status codes

- 200 - user has been logged out
- 401 - authorization failed

8.3.5 Additional description

Logs user out and destroys his JWT tokens.

8.4 GET /user

8.4.1 Input parameters

There are no input parameters.

8.4.2 Authorization

JWT access token inside Bearer token.

8.4.3 Success output

- user - User object of currently logged user

8.4.4 Status codes

- 200 - success
- 401 - authorization failed

8.4.5 Additional description

Returns User object of currently logged user.

8.5 GET /user/{id}

8.5.1 Input parameters

There are no input parameters.

8.5.2 Authorization

JWT access token inside Bearer token.

8.5.3 Success output

- user - User object of user with given ID

8.5.4 Status codes

- 200 - success
- 400 - user ID URL param is invalid or missing
- 401 - authorization failed
- 404 - user with given ID not found

8.5.5 Additional description

Returns User object of user with given ID.

8.6 GET /recipe/{id}

8.6.1 Input parameters

There are no input parameters.

8.6.2 Authorization

JWT access token inside Bearer token.

8.6.3 Success output

- recipe - Recipe object (with its categories, ingredients, stars and author) of recipe with given ID

8.6.4 Status codes

- 200 - success
- 400 - recipe ID URL param is invalid or missing
- 401 - authorization failed
- 404 - recipe with given ID not found

8.6.5 Additional description

Returns Recipe object (with its categories, ingredients, stars and author) of recipe with given ID.

8.7 GET /recipe

8.7.1 Input parameters

Query parameters:

- limit - amount of recipes per page (maximum is 25)
- page - page number
- userId - optional parameter telling to fetch only recipes created by user with this ID (if not supplied - public recipes of all users will be fetched)
- includePublic - optional boolean flag indicating whether to include user's public recipes (NOTE: this parameter matters only if userId parameter is defined)
- includePrivate - optional boolean flag indicating whether to include user's private recipes (NOTE: this parameter matters only if userId parameter is defined AND userId parameter is the same as currently logged user's ID)
- excludeMyRecipes - optional boolean flag indicating whether to exclude currently logged user's recipes (NOTE: this parameter matters only if userId parameter is NOT defined)

8.7.2 Authorization

JWT access token inside Bearer token.

8.7.3 Success output

- limit - amount of recipes per page
- page - page number
- totalRecipes - number of all recipes able to display
- totalPages - number of pages available
- recipes - List of Recipe objects (with its categories, ingredients, stars and authors) sorted by creation date (descending)

8.7.4 Status codes

- 200 - success
- 400 - some query params are missing or invalid OR userId param is invalid (if supplied) OR page parameter's value exceeds total number of pages
- 401 - authorization failed
- 404 - no recipes found

8.7.5 Additional description

If 'userId' param is defined and 'includePublic' and 'includePrivate' params are false or null then enpoint will return empty recipes list. This endpoint uses pagination. Returns List of Recipe objects (with its categories, ingredients, stars and authors) sorted by creation date (descending).

8.8 POST /recipe

8.8.1 Input parameters

Request body:

- title - string, title of the recipe, must be between 5 and 255 characters long
- cookingTimeMinutes - number, must be positive and less than 100 000
- description - string, must be longer than 10 characters
- isPublic - optional boolean indicating whether recipe is public
- location - optional string with location, must be shorter than 255 characters
- ingredients - array of strings, quantity of items inside array must be between 1 and 25
- categories - array of strings, quantity of items inside array must be between 1 and 5

8.8.2 Authorization

JWT access token inside Bearer token.

8.8.3 Success output

- recipe - created Recipe object

8.8.4 Status codes

- 201 - recipe has been created
- 400 - body params are missing or invalid
- 401 - authorization failed
- 500 - server error

8.8.5 Additional description

Saves recipe in the database.

8.9 PUT /recipe/{id}

8.9.1 Input parameters

Request body:

- title - string, title of the recipe, must be between 5 and 255 characters long
- cookingTimeMinutes - number, must be positive and less than 100 000
- description - string, must be longer than 10 characters
- isPublic - optional boolean indicating whether recipe is public
- location - optional string with location, must be shorter than 255 characters
- ingredients - array of strings, quantity of items inside array must be between 1 and 25
- categories - array of strings, quantity of items inside array must be between 1 and 5

8.9.2 Authorization

JWT access token inside Bearer token.

8.9.3 Success output

- recipe - updated Recipe object

8.9.4 Status codes

- 200 - recipe has been updated
- 400 - body parameters are missing or invalid OR recipe ID URL param is invalid or missing OR recipe with given ID does not exist
- 401 - authorization failed
- 500 - server error

8.9.5 Additional description

Edits recipe in the database.

8.10 DELETE /recipe/{id}

8.10.1 Input parameters

There are no input parameters.

8.10.2 Authorization

JWT access token inside Bearer token.

8.10.3 Success output

There is no output.

8.10.4 Status codes

- 200 - recipe has been deleted
- 400 - body parameters are missing or invalid OR recipe ID URL param is invalid or missing OR recipe with given ID does not exist
- 401 - authorization failed
- 500 - server error

8.10.5 Additional description

Removes recipe from the database.

8.11 POST /recipe/review/{recipeId}

8.11.1 Input parameters

Request body:

- stars - number of stars, must be between 1 and 5
- comment - optional string with comment

8.11.2 Authorization

JWT access token inside Bearer token.

8.11.3 Success output

- review - created Review object

8.11.4 Status codes

- 201 - review created
- 400 - body parameters are missing or invalid OR recipe ID URL param is invalid or missing
- 401 - authorization failed
- 404 - recipe with given ID does not exist
- 409 - currently logged user have already created review for recipe with this ID OR currently logged user trying to comment his own recipe
- 500 - server error

8.11.5 Additional description

Creates review for given recipe.

8.12 GET /recipe/review/{recipeId}

8.12.1 Input parameters

Query parameters:

- limit - amount of recipes per page (maximum is 25)
- page - page number

8.12.2 Authorization

JWT access token inside Bearer token.

8.12.3 Success output

- limit - amount of recipes per page
- page - page number
- totalRecipes - number of all review of given recipe
- totalPages - number of pages available
- currentUserReview - Review object of currently logged user or null if such a review does not exist
- reviews - List of Review objects (with its authors) sorted by creation date (descending)

8.12.4 Status codes

- 200 - success
- 400 - some query params are missing or invalid OR page parameter's value exceeds total number of pages
- 401 - authorization failed
- 404 - recipe with given ID does not exist

8.12.5 Additional description

This endpoint uses pagination. Returns List of Review objects (with its authors) sorted by creation date (descending).

8.13 PUT /recipe/review/{reviewId}

8.13.1 Input parameters

Request body:

- stars - number of stars, must be between 1 and 5
- comment - optional string with comment

8.13.2 Authorization

JWT access token inside Bearer token.

8.13.3 Success output

- review - edited Review object

8.13.4 Status codes

- 200 - success
- 400 - body parameters are missing or invalid OR review ID URL param is invalid or missing
- 401 - authorization failed
- 404 - review with given ID does not exist
- 500 - server error

8.13.5 Additional description

Edits review with given ID.

8.14 DELETE /recipe/review/{reviewId}

8.14.1 Input parameters

There are no input parameters.

8.14.2 Authorization

JWT access token inside Bearer token.

8.14.3 Success output

There is no output.

8.14.4 Status codes

- 200 - success
- 400 - review ID URL param is invalid or missing
- 401 - authorization failed
- 404 - review with given ID does not exist
- 500 - server error

8.14.5 Additional description

Removes review with given ID.

8.15 GET /recipe/review/stars/{recipeId}

8.15.1 Input parameters

There are no input parameters.

8.15.2 Authorization

JWT access token inside Bearer token.

8.15.3 Success output

- count - number of all reviews of given recipe
- average - average mean of all reviews of given recipe

8.15.4 Status codes

- 202 - success
- 400 - recipe ID URL param is invalid or missing
- 401 - authorization failed
- 404 - recipe with given ID does not exist

8.15.5 Additional description

Returns number of recipe's reviews and average mean of them.

8.16 GET /recipe/ingredient/{name}

8.16.1 Input parameters

Query parameters:

- limit - amount of recipes per page (maximum is 25)
- page - page number
- excludeMyRecipes - optional boolean flag indicating whether to exclude currently logged user's recipes

8.16.2 Authorization

JWT access token inside Bearer token.

8.16.3 Success output

- limit - amount of recipes per page
- page - page number
- totalRecipes - number of all recipes able to display containing given ingredient (name parameter inside URL)
- totalPages - number of pages available
- recipes - List of Recipe objects (with its categories, ingredients, stars and authors) containing given ingredient (name parameter inside URL) and sorted by creation date (descending)

8.16.4 Status codes

- 200 - success
- 400 - some query params are missing or invalid OR name param is invalid OR page parameter's value exceeds total number of pages
- 401 - authorization failed
- 404 - no recipes with given ingredient found

8.16.5 Additional description

This endpoint uses pagination. Returns List of Recipe objects (with its categories, ingredients, stars and authors) containing given ingredient (name parameter inside URL) and sorted by creation date (descending).

8.17 GET /recipe/category/{name}

8.17.1 Input parameters

Query parameters:

- limit - amount of recipes per page (maximum is 25)
- page - page number
- excludeMyRecipes - optional boolean flag indicating whether to exclude currently logged user's recipes

8.17.2 Authorization

JWT access token inside Bearer token.

8.17.3 Success output

- limit - amount of recipes per page
- page - page number
- totalRecipes - number of all recipes able to display containing given category (name parameter inside URL)
- totalPages - number of pages available
- recipes - List of Recipe objects (with its categories, ingredients, stars and authors) containing given category (name parameter inside URL) and sorted by creation date (descending)

8.17.4 Status codes

- 200 - success
- 400 - some query params are missing or invalid OR name param is invalid OR page parameter's value exceeds total number of pages
- 401 - authorization failed
- 404 - no recipes with given category found

8.17.5 Additional description

This endpoint uses pagination. Returns List of Recipe objects (with its categories, ingredients, stars and authors) containing given category (name parameter inside URL) and sorted by creation date (descending).

8.18 GET /recipe/ingredients/{commaSeparatedNames}

8.18.1 Input parameters

Query parameters:

- limit - amount of recipes per page (maximum is 25)
- page - page number
- excludeMyRecipes - optional boolean flag indicating whether to exclude currently logged user's recipes

8.18.2 Authorization

JWT access token inside Bearer token.

8.18.3 Success output

- limit - amount of recipes per page
- page - page number
- totalRecipes - number of all recipes able to display containing only given ingredients (name parameter inside URL)
- totalPages - number of pages available
- recipes - List of Recipe objects (with its categories, ingredients, stars and authors) containing only given ingredients (commaSeparatedNames parameter inside URL) and sorted by creation date (descending)

8.18.4 Status codes

- 200 - success
- 400 - some query params are missing or invalid OR commaSeparatedNames param is invalid OR page parameter's value exceeds total number of pages
- 401 - authorization failed
- 404 - no recipes with only given categories found

8.18.5 Additional description

'commaSeparatedNames' URL parameter is a list of category names separated with comma character. This endpoint uses pagination. Returns List of Recipe objects (with its categories, ingredients, stars and authors) containing only given ingredients (commaSeparatedNames parameter inside URL) and sorted by creation date (descending).

8.19 GET /recipe/categories/{commaSeparatedNames}

8.19.1 Input parameters

Query parameters:

- limit - amount of recipes per page (maximum is 25)
- page - page number
- excludeMyRecipes - optional boolean flag indicating whether to exclude currently logged user's recipes

8.19.2 Authorization

JWT access token inside Bearer token.

8.19.3 Success output

- limit - amount of recipes per page
- page - page number
- totalRecipes - number of all recipes able to display containing only given categories (name parameter inside URL)
- totalPages - number of pages available
- recipes - List of Recipe objects (with its categories, ingredients, stars and authors) containing only given categories (commaSeparatedNames parameter inside URL) and sorted by creation date (descending)

8.19.4 Status codes

- 200 - success
- 400 - some query params are missing or invalid OR commaSeparatedNames param is invalid OR page parameter's value exceeds total number of pages
- 401 - authorization failed
- 404 - no recipes with only given categories found

8.19.5 Additional description

'commaSeparatedNames' URL parameter is a list of category names separated with comma character. This endpoint uses pagination. Returns List of Recipe objects (with its categories, ingredients, stars and authors) containing only given categories (commaSeparatedNames parameter inside URL) and sorted by creation date (descending).

9 Tests

After implementation phase, several tests have been conducted in order to check for both functional and non-functional requirements set in the requirements document. Each functionality was tested and then recorded in a table 1 with additional comments. Same principle applies to the nonfunctional requirements, which are listed in the table 2. In addition, the backend REST API endpoints have been tested.

9.1 Functional requirements

9.1.1 Manual tests

To test how the application functions from a user's perspective, how the frontend behaves, how the page loads, and what functionalities it allows, manual tests were conducted. Requirements that did not work as a result of the tests were explained and corrected.

Table 1: Manual Test Results

Functionality	**Does it Work**	**Additional Comment**
Authenticated user being able to browse public recipes on the wall.	Yes	The recipes render correctly on the wall page. User can click on each of the recipes to check details. The results render quickly due to pagination.
Authenticated user is able to select whether the wall page contains his public recipes.	Yes	This functionality is provided by "Show my public recipes" checkbox.
The recipe contains such details as: title, author, cooking time in minutes, ingredients, categories, description, public attribute, creation time, location (optional) and reviews.	Yes	The details are available when the user clicks on one of the recipes.
Authenticated user can add a recipe containing previously mentioned details, excluding the author, creation time and reviews.	**No**	Using exceptionally long character sequences for the title, cooking time, description, location, category and ingredients, the recipe is not being submitted.
After trying multiple combinations, the issue was narrowed to the "cooking time" field. The bug occurred while trying to enter a value as large as "99999999999999999" or bigger. It was later observed that there are no error messages being rendered on the frontend site when submitting a recipe, as well as no data validation on the backend side. The issue was fixed by adding proper data validation, with regards to "cooking time" field, on both fronted and backend side. (See files recipe.controller.ts and RecipeForm.tsx)		
Authenticated user can edit his own recipes.	Yes	Feature works as intended.
During the testing of the editing recipe feature, it was revealed that any user can enter /recipe/{id}?action=edit page, which renders an editing view for any recipe user enters. Fortunately, the request was still rejected by the backend side. The bug was later fixed (See file RecipeCard.tsx).		
Authenticated user can review only public recipes of other users.	Yes	Users own recipes are not possible to review while other's users private recipes are not available nor possible to review. This feature needs to be further tested on the backend API side.
Authenticated user can enter user's profile to view their public recipes.	Yes	This feature works as intended.
Authenticated user can filter recipes with ingredients or categories.	**No**	Some of the categories or ingredients result in no recipes found, even though they are in the recipes.
URI encoding was missing when creating an URL in *getRecipes* function. The bug was fixed shortly after. (See file recipe.ts)		

9.1.2 Automated tests

In this section, Jest framework and Supertest library are employed for automated testing of the backend API. Automated tests allow a more thorough and efficient examination of the application compared to manual testing. The testing process involves providing query or body parameters to the endpoint, depending on the type of the endpoint, and verifying whether the output aligns with the expected behavior of the application. Typically, output includes a success or failure message, status code, and data such as recipes or reviews. The accumulated test suite ensures proper functioning as new features are introduced. Below, tests executed for each endpoint are outlined. All test files are located in the "backend/tests" folder and can be run using the *npm run test* command.

9.2 Automated Tests

- **GET /auth/logout**
  - Unauthorized user cant log out
  - Unauthorized users cookie gets maxage 0 and is expired
- **GET /user**
  - Unauthorized user cant see its profile
  - Authorized user can see its profile
- **GET /user/{id}**
  - Unauthorized user cant see others profiles
  - Authorized user can see others profile
  - Authorized user cant enter float as profile id
  - Authorized user cant enter string as profile id
  - Authorized user cant see nonexistent users profile
- **GET /recipe/{id}**
  - Unauthorized user cant see recipes by id
  - Authorized user can see public recipes by id, which contain author, id, title, cookingtime, description, isPublic flag, creation time, location, userId, categories, ingredients, and stars
  - Authorized user cant enter float as recipe id
  - Authorized user cant enter string as recipe id
  - Authorized user cant see nonexistent recipes
- **GET /recipe**
  - Unauthorized - should respond with a 401 status code
  - Authorized user able to browse public recipes on the wall
  - "Authorized user enters page param as string
  - Authorized user enters page param as float
  - Authorized user fetches recipes of nonexistent user
  - Authorized user fetches recipes with limit over 25
  - Authorized user fetches public recipes with other userid
  - Authorized user fetches non-public recipes with other userid
  - Authorized user fetches no public and no private recipes with other
  - Authorized user fetches private recipes
  - Authorized user fetches everything but his own recipes
- **POST /recipe**

  - Unauthorized - should respond with a 401 status code
  - Authorized can post a valid recipe
  - Authorized user cant post an empty recipe
  - Authorized user cant post a title as a number recipe
  - Authorized user cant post a cookingTime as a string recipe
  - Authorized user cant post ingredients as a number[] recipe
  - Authorized user cant post more than 25 ingredients in a recipe
  - Authorized user cant post the same ingredients in a recipe
  - Authorized user cant post the same categories in a recipe

- **PUT /recipe/{id}**
  - Unauthorized - should respond with a 401 status code
  - Authorized can put a valid own recipe
  - "Authorized user cant put an empty recipe
  - Authorized user cant put a title as a number recipe
  - Authorized user cant put a cookingTime as a string recipe
  - Authorized user cant put ingredients as a number[] recipe
  - Authorized user cant put more than 25 ingredients in a recipe
  - Authorized user cant put the same ingredients in a recipe
  - Authorized user cant put the same categories in a recipe
  - Authorized cant put someone's recipe

- **DELETE /recipe/{id}**
  - Unauthorized - should respond with a 401 status code
  - Authorized user can delete his own recipe
  - Authorized user cant delete not his recipe
  - Authorized user cant delete nonexistent recipe
  - Authorized user cant delete string id recipe
  - Authorized user cant delete float id recipe

- **POST /recipe/review/{recipeid}**
  - Unauthorized - should respond with a 401 status code
  - Authorized user can review someone's public recipe
  - Authorized user cant review someone's private recipe

- **GET /recipe/review/{recipeid}**
  - Unauthorized - should respond with a 401 status code
  - Authorized user can get other users public recipes reviews
  - Authorized user cant get other users private recipes reviews
  - Authorized user can get his public recipes reviews
  - Authorized user can fetch no reviews for his recipe
  - Authorized user cant fetch no more than 25 reviews
  - Authorized user cant get his private recipe reviews

- **PUT /recipe/review/{reviewid}**
  - Unauthorized - should respond with a 401 status code

  - Authorized user can edit his review on others users public recipes
- **DELETE /recipe/review/{reviewid}**
  - Unauthorized - should respond with a 401 status code
- **GET /recipe/review/stars/{recipeid}**
  - Unauthorized - should respond with a 401 status code
  - Authorized user can get stars of public recipe
- **GET /recipe/ingredient/{name}**
  - Unauthorized - should respond with a 401 status code
  - Authorized user able to get recipes by ingredient
  - Authorized user enters page param as string
  - Authorized user enters page param as float
  - Authorized user fetches recipes with nonexistent ingredient
- **GET /recipe/category/{name}**
  - Unauthorized - should respond with a 401 status code
  - Authorized user able to get recipes by category
  - Authorized user enters page param as string
  - Authorized user enters page param as float
  - Authorized user fetches recipes with nonexistent category
- **GET /recipe/ingredients/{commaSeparatedNames}**
  - Unauthorized - should respond with a 401 status code
  - Authorized user able to get recipes by ingredients
  - Authorized user enters page param as string
  - Authorized user enters page param as float
  - Authorized user fetches recipes with nonexistent ingredients
- **GET /recipe/categories/{commaSeparatedNames}**
  - Unauthorized - should respond with a 401 status code
  - Authorized user able to get recipes by categories
  - Authorized user enters page param as string
  - Authorized user enters page param as float
  - Authorized user fetches recipes with nonexistent categories

9.3 Non-Functional requirements

9.3.1 Manual tests

Table 2: Manual Test Results

Requirement	**Met?**	**Additional Comment**
User authenticates using a Google account.	Yes	-
Page loads in less than 5 seconds.	Yes	Using the website webpagetest.org, a page loading speed test was conducted, which showed that for the location Frankfurt, Germany, using a 3G mobile network and the Chrome browser, the first content loads in 3.598 seconds, which is below the target of 5 seconds.
Page should be accessible on both computers and mobile devices.	Yes	Page loads correctly on both type of devices.
Page should load equally fast with a very large number of recipes in the database.	Yes	The pagination mechanism allows avoiding long page loading by fetching a small portion of data from the database, which will actually be displayed on the page.
Page should be displayed correctly on all popular browsers.	Yes	The page is displayed correctly on popular browsers like Chrome, Firefox, Edge and Opera.
Page should be resistant to XSS attacks.	Yes	TSX automatically escapes dynamic values, treating them as plain text by default, preventing XSS vulnerabilities. When trying to enter script code as part of the recipe, no javascript code got executed. Of course, it is not possible to have one hundred percent certainty regarding the system's resistance to XSS attacks.