**Recipes Application – documentation**

**About**

Recipes application is a culinary platform designed for enthusiasts to explore, share, and engage with a diverse range of recipes. Users can seamlessly add, delete, rate, and exchange recipes, creating a vibrant community of food lovers. The application offers the following key features:

1. Recipe management:

* Users can effortlessly add their favorite recipes, contributing to a dynamic collection of culinary delights.

1. Flexibility in recipe updates:
   * The platform allows users to easily delete and update recipes, ensuring that their culinary creations are always up-to-date.
2. Recipe rates and comments:
   * Users have the ability to rate and comments a recipe, providing valuable feedback and helping others discover the most popular and well-loved dishes.
3. Interactive recipe exchange:
   * The application facilitates the exchange of recipes, fostering a collaborative and engaging community where users can discover new culinary inspirations.
4. Messaging system:
   * In addition to recipe-related functionalities, users can communicate through a messaging system, enabling direct interaction and the sharing of culinary tips and experiences.
5. Shopping list managment:
   * Users can create and manage their shopping lists, making it convenient to plan and organize ingredients needed for their chosen recipes.

**Backend**

Technologies used:

* Using ASP .NET CORE - .NET 6 version
* Entity Framework
* SQL Server for database storage

Project Structure

* Data: Repositories and DBContext classes for SQL Server db integrations
* Service: Business logic and service classes
* Recipies: Web project with controllers for handling HTTP requests
* Dtos: Data transfer object used for communication
* Tests: Implemented unit tests for services and controllers

Error Handling:

* Exception handling: Custom exception handling on each level
* IExceptionFilter: Utilized for handling unexpected exceptions

Logging:

* Delegates and events: Implemented for logging significant events

API Documentation:

* Swagger: Integrated for intercative API documentation

Authorization and Authentication

* JWT Token: Used for both authorization and authentication
* Bearer token authentication: Ensures secure API access

Asynchronous operations:

* Async/Await Pattern: Throughout the backend, asynchronous programming is employed using the async/await pattern
* Enhance responsiveness and scalability, ensuring efficient handling of concurrent requests

Real-time communication:

* Implemented a SignalR Hub on the server-side to facilitate real-time communication
* The Chat Hub enables bidirectional communication, allowing the server to push messages to connected clients in real-time

**API Documentation:**

**Recipe Controller**

* Get All Recipes: GET /api/recipes (Authorization: Bearer token)
* Get Latest Recipes: GET /api/recipes/latest?count={count} (Authorization: Bearer token)
* Get Top Rated Recipes: GET /api/recipes/toprated?count={count} (Authorization: Bearer token)
* Get All User Recipes: GET /api/recipes/userrecepies?userId={userId} (Authorization: Bearer token)
* Get All User Favourite Recipes: GET /api/recipes/userfavouriterecepies?userId={userId} (Authorization: Bearer token)
* Get Recipe by ID: GET /api/recipes/getrecipe?id={recipeId} (Authorization: Bearer token)
* Create Recipe: POST /api/recipes/add (Authorization: Bearer token)
* Update Recipe: PUT /api/recipes/{id} (Authorization: Bearer token)
* Add Recipe to User Favourites: GET /api/recipes/addtofavourites?id={recipeId} (Authorization: Bearer token)
* Remove Recipe from User Favourites: GET /api/recipes/removefromfavourites?id={recipeId} (Authorization: Bearer token)
* Delete Recipe: DELETE /api/recipes/{id} (Authorization: Bearer token)
* Add New Rating: POST /api/recipes/ratings (Authorization: Bearer token)
* Add New Comment: POST /api/recipes/comments (Authorization: Bearer token)
* Get All Ratings: GET /api/recipes/ratings?recipeId={recipeId} (Authorization: Bearer token)
* Get All Comments: GET /api/recipes/comments?recipeId={recipeId} (Authorization: Bearer token)
* Delete Rating: DELETE /api/recipes/ratings/{ratingId} (Authorization: Bearer token)

**User Controller**

* Register User: POST /api/user/register - No Authorization
* Update User: POST /api/user/updateuser (Authorization: Bearer token)
* Change Password: POST /api/user/changepassword (Authorization: Bearer token)
* Login User: POST /api/user/login (Authorization: Bearer token)
* Get Current User: GET /api/user/current (Authorization: Bearer token)
* Get All Users: GET /api/user/getall (Authorization: Bearer token)
* Get User by ID: GET /api/user/getuser?id={userId}(Authorization: Bearer token)
* Get All Notifications: GET /api/user/notifications (Authorization: Bearer token)
* Make Notification as Read: GET /api/user/makeasread?notificationId={notificationId} (Authorization: Bearer token)
* Delete Notification: GET /api/user/deletenotification?notificationId={notificationId} (Authorization: Bearer token)
* Make All Notifications as Read: GET /api/user/makeallasread?userId={userId} (Authorization: Bearer token)

**Shopping List Controller**

* Get user shopping list: GET /api/shopping/shoppinglist (Authorization: Bearer token)
* Add ingredient to shopping list: POST /api/shopping/addIngredient (Authorization: Bearer token)
* Remove ingredient from shopping list: DELETE /api/shopping/removeIngredient (Authorization: Bearer token)

**Chat Controller**

* Send Message: POST /api/chat/send (Authorization: Bearer token)
* Receive Messages: GET /api/chat/receive/{userId} (Authorization: Bearer token)
* Get Messages Between Users: GET /api/chat/messages-between/{userId1}/{userId2} (Authorization: Bearer token)
* Get Sent Messages: GET /api/chat/send/{userId (Authorization: Bearer token)

**Frontend**

Frontend app, developed with Angular 12 and Node.js 14.15.5, employs a modular and efficient architecture to enhance the recipe management experience. Key features and technologies include:

* **Angular 12:** Angular is a widely-used open-source front-end web application framework developed and maintained by Google. It is written in TypeScript and enables developers to build dynamic, single-page web applications with a focus on a modular and maintainable code structure.
* **Node.js 14.15.5:** Powering the backend, Node.js delivers a scalable and efficient runtime environment for server-side operations.
* **Angular Material Forms:** The application incorporates Angular Material's form components to create intuitive and visually consistent forms. These forms facilitate seamless user interactions when adding, editing, or managing recipes.
* **Grouped Services with HttpClient:** Backend communication is streamlined through grouped services using Angular's **HttpClient**. This approach enhances maintainability, readability, and scalability by organizing API calls into logically grouped services.
* **Dependency Management with NPM:** Node Package Manager (NPM) is employed for efficient dependency management, simplifying the installation, updating, and maintenance of frontend libraries.
* **Angular Material Theming:** Customizable theming options provided by Angular Material are implemented to maintain brand consistency and enable easy adaptation to different visual styles.
* **Accessibility:** Our frontend prioritizes accessibility by integrating features provided by Angular Material, ensuring usability for individuals with diverse abilities.
* **Angular Material Components:** Beyond forms, various Angular Material components contribute to the overall aesthetic and functionality of the application, enhancing the user experience.

**Conclusion:**

By utilizing Angular's **HttpClient** for grouped services, our frontend establishes a robust and organized approach to communicate with the backend. This, combined with Angular 12, Node.js 14.15.5, and Angular Material Forms, ensures an efficient, responsive, and accessible frontend for our recipe management application.