Recipe App Documentation

# Contents

[I. Contents 1](#_Toc92746609)

[II. App overview 3](#_Toc92746610)

[III. Entities description 4](#_Toc92746611)

[1. Ingredient 4](#_Toc92746612)

[1.1 Get Ingredients 4](#_Toc92746613)

[1.2 Get Ingredient 5](#_Toc92746614)

[1.3 Add Ingredient 6](#_Toc92746615)

[1.4 Add Ingredients 7](#_Toc92746616)

[2. Recipe 8](#_Toc92746617)

[2.1 Get Recipes 9](#_Toc92746618)

[2.2 Get Recipe 9](#_Toc92746619)

[2.3 Add Recipe 10](#_Toc92746620)

[2.4 Update Recipe 12](#_Toc92746621)

[2.5 Delete recipe 14](#_Toc92746622)

[2.6 Add Ingredient To Recipe 14](#_Toc92746623)

[2.7 Remove Ingredient From Recipe 15](#_Toc92746624)

[3. RecipeIngredient 16](#_Toc92746625)

[4. User 16](#_Toc92746626)

[4.1. Get Users 16](#_Toc92746627)

[4.2. Get User 16](#_Toc92746628)

[4.3. Add User 17](#_Toc92746629)

[4.5. Update User 18](#_Toc92746630)

[5. Bookmark 19](#_Toc92746631)

[4.1. Get Bookmarks By User Id 19](#_Toc92746632)

[4.2. Get Bookmarks With Recipe Details For User 20](#_Toc92746633)

[4.3. Add Bookmark 21](#_Toc92746634)

[4.4. Get User Bookmarks By Date 22](#_Toc92746635)

[4.5. Delete Bookmarks By User Id 23](#_Toc92746636)

[4.6. Delete Bookmark 24](#_Toc92746637)

[6. User Settings 24](#_Toc92746638)

[6.1. Get User Setting 25](#_Toc92746639)

[6.2. Add User Setting 26](#_Toc92746640)

[IV. Business Requirements 27](#_Toc92746641)

[V. App Usage 28](#_Toc92746642)

# App overview

The app represents the server side of a web cooking platform where users can view, create, edit recipes, use bookmark and set user settings.

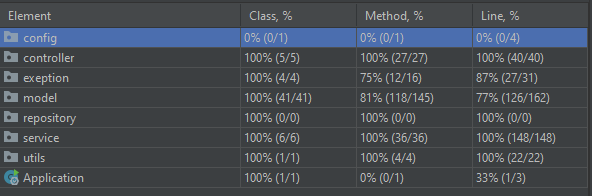
Spring, Spring Data, Hibernate, PostgreSql and Mokito were used to implement it.

The app has six entities and shows examples of these types of relationships:

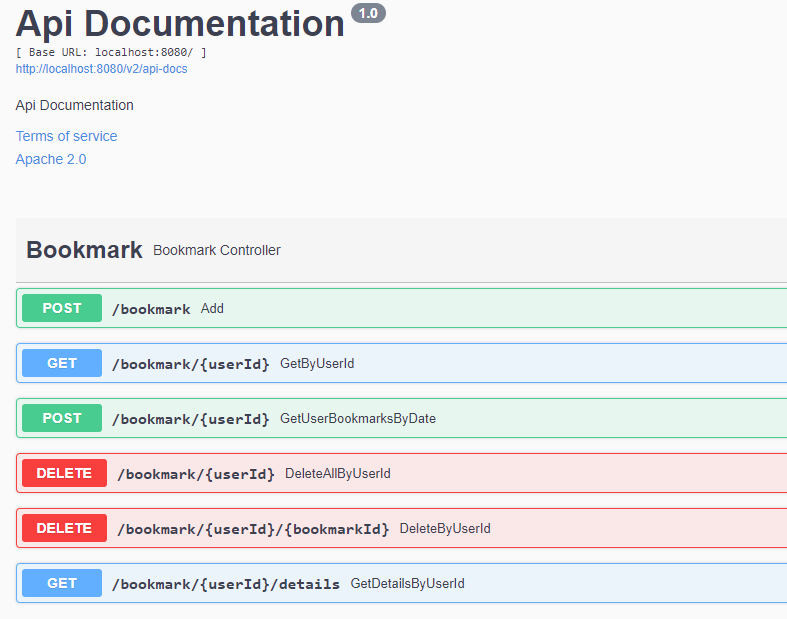
* 1:1
* 1:M
* M:M (resolved into two 1:M relationships with a connection table with an extra column).

The project implements custom exception handling, detailed in the entities description chapters and has a high coverage percentage.



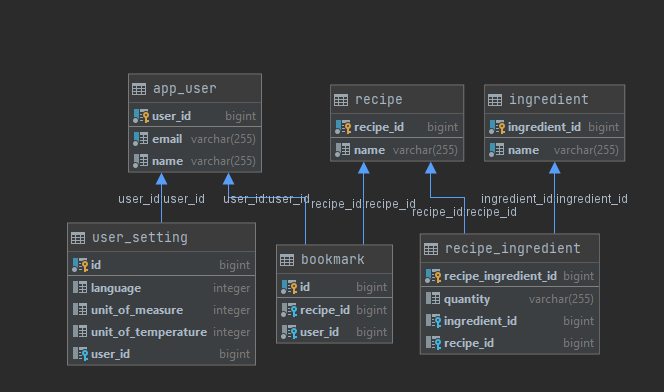


All the endpoints are called in the Postmen collection (found in the resources folder) and presented in the Swagger documentation accessible at: [*http://localhost:8080/swagger-ui/*](http://localhost:8080/swagger-ui/)



# Entities description

The entities in the app are connected in the following way. In the next sections, their implementations and usages will de described.

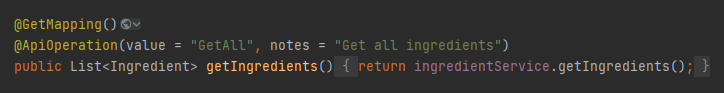


## Ingredient

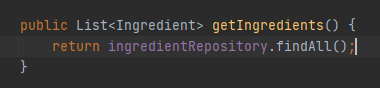
The *Ingredient* entity has an auto-generated id, a name with a min length of 3 characters and a 1-Many relationship with the *RecipeIngredients* entity (which is not added in the JSON and string representations).

### Get Ingredients

The *IngredientController* defines the **getIngredients** method which calls **getIngredients** from the service.

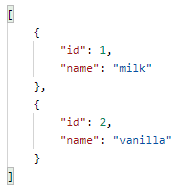


The service will call the **findAll** method of the repository.



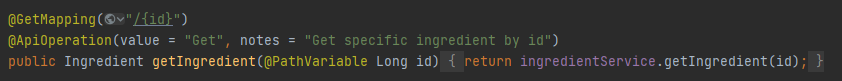
**Returns:**

* List of Ingredients

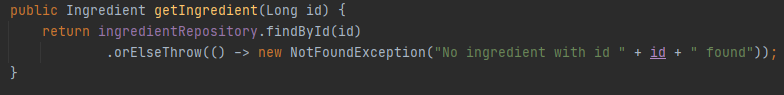


### Get Ingredient

The *IngredientController* defines the **getIngredient** method which calls **getIngredient** from the service with the path variable id.

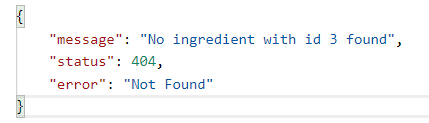
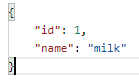


The service will call the **findById** method of the repository.

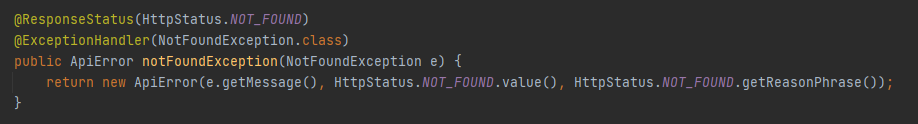


**Returns:**

* Ingredient
* Api Error – ingredient not found

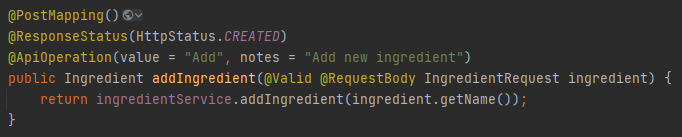


The *NotFoundException* is caught, among others, in a *GlobalExceptionHandler* file which returns a custom Api Error instance.

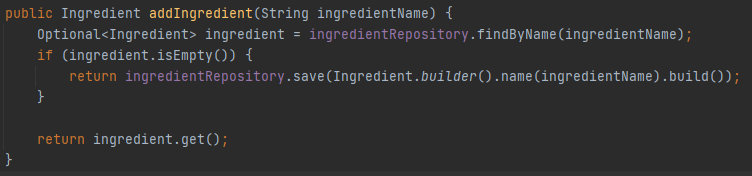


### Add Ingredient

The *IngredientController* defines the **addIngredient** method which calls **addIngredient** from the service with the *IngredientRequest* object.

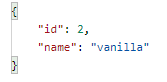


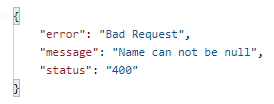
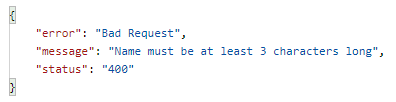
The service will check that an ingredient with the same name does not exist and then call the repository to create it. No error is thrown if a similar record if found because recipes also create ingredients and in that case we just want to avoid duplicates, not send out errors.



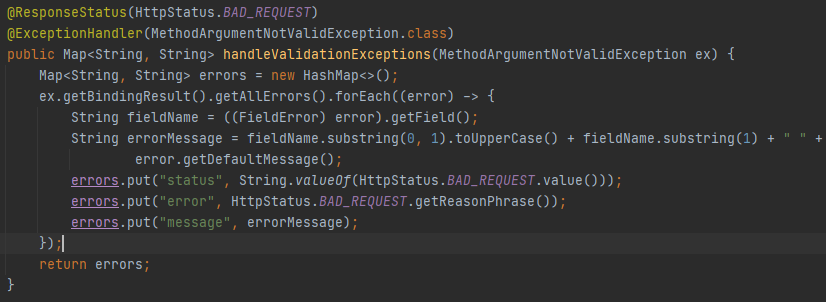
**Returns:**

* Ingredient Response
* Api Error – min length
* Api Error – not null

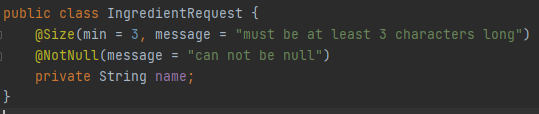




These kind of errors are also caught in the global handler by the *MethodArgumentNotValidException* handler.



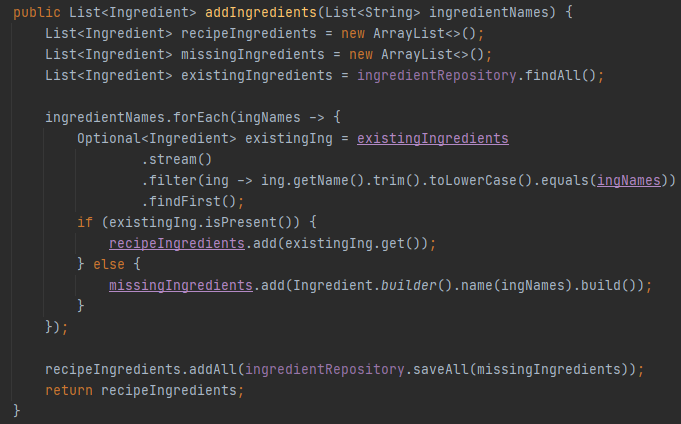
The custom messages are set in the request entities.



### Add Ingredients

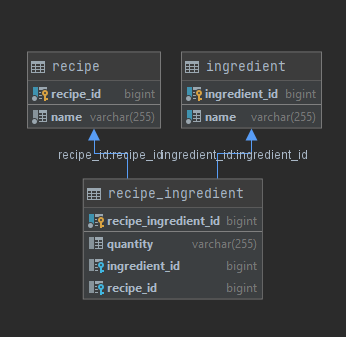
The service also implements an **AddIngredients** method used when creating new recipes. The values are passed as a list of names.

The name list is looped and if an equivalent if not found among the database records the element is added to a missingIngredients list. After checking all the values the missing items are saved, by calling the add method from the repository, and all ingredients, new and present, are returned.



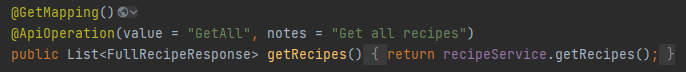
## Recipe

The recipe entity has an auto-generated id, a not null name and a 1-Many relationship with the *RecipeIngredients*. This is used to form a bidirectional link to the ingredients the recipe has and their quantity.

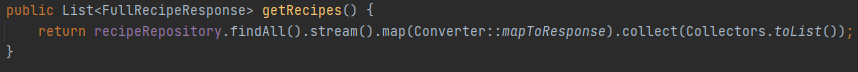


### Get Recipes

The *RecipeController* defines the **getRecipes** method, which calls **getRecipes** from the service.



The service will get all the recipes with the **findAll** repository method and then map them to a *FullRecipeResponse*, using a helper converter function (a recipe with a list of ingredient-quantity values).

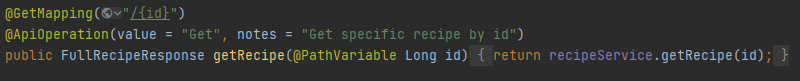


**Returns:**

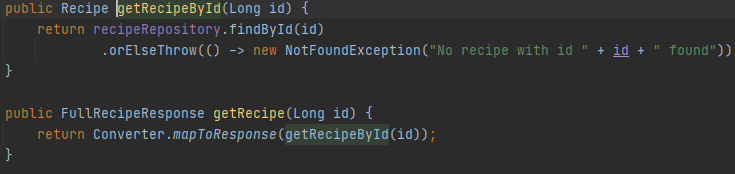
* List of Full Recipe Response

### Get Recipe

The *RecipeController* defines the **getRecipe** method which calls **getRecipe** from the service with the path variable id.

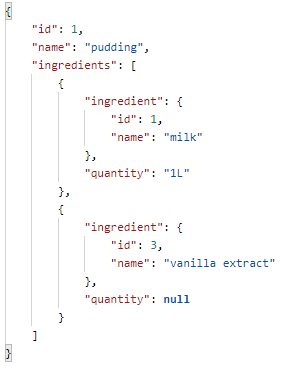


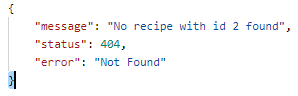
The service will call the **findById** method of the repository and convert the response.



**Returns:**

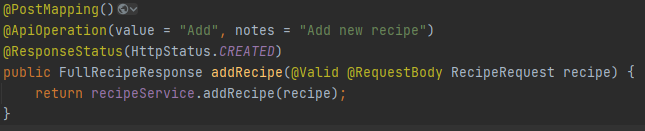
* Full Recipe Response
* Api Error – recipe not found



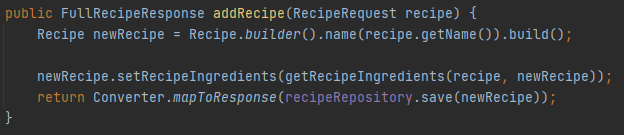


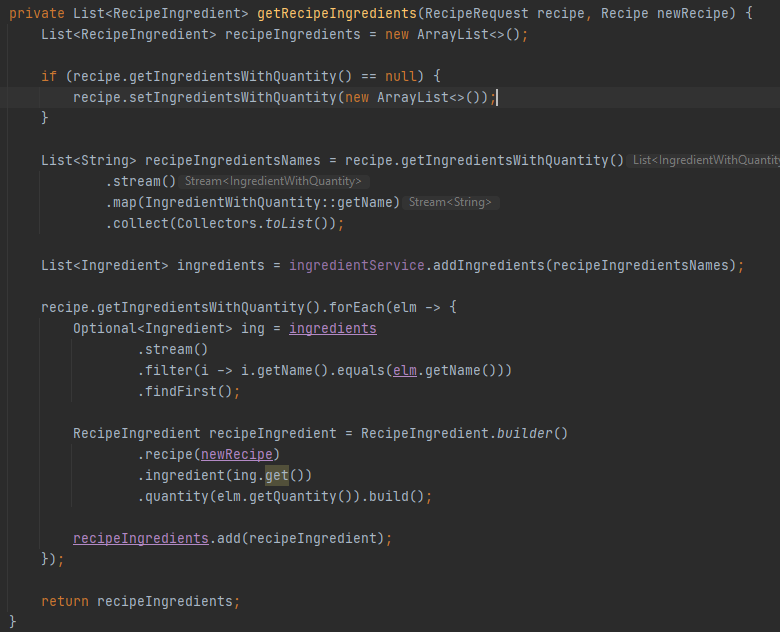
### Add Recipe

The *RecipeController* defines the **addRecipe** method which calls **addRecipe** from the service with the *RecipeRequest* object.



The service will call **addIngredients** from the ingredient service in order to create any missing ingredients in the database. Then it will create and save new ingredient-quantity records. It will then link those records to the recipe.

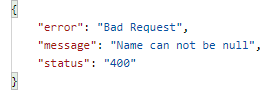




**Returns:**

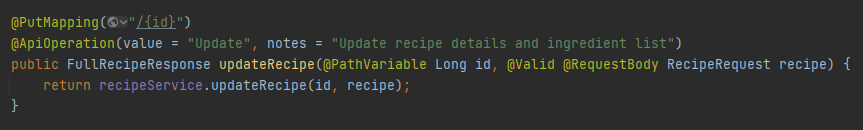
* Full Recipe Response
* Api Error – name not null



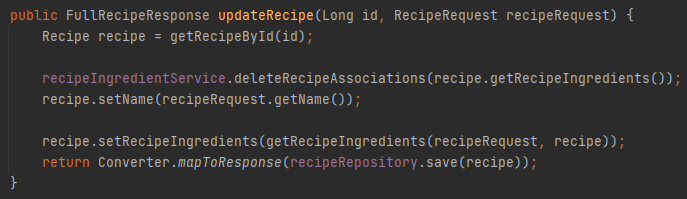


### Update Recipe

The *RecipeController* defines the **updateRecipe** method which calls **updateRecipe** from the service with the recipe id and the *RecipeRequest* object.



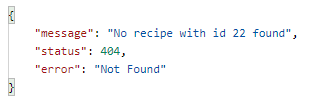
In the service the recipe is retrieved from the database using the given id. The recipe ingredient associations are deleted and new connections are made, in the same way as for create (new ingredients are added and ingredient-quantity entities are made).

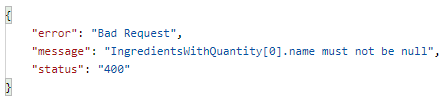
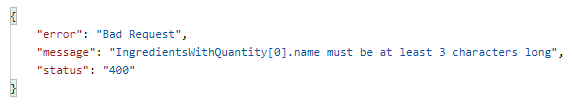


**Results:**

* Full Recipe Response
* Api Error – recipe not found
* Api Error – name not null
* Api Error – ingredient name not null
* Api Error – ingredient min length

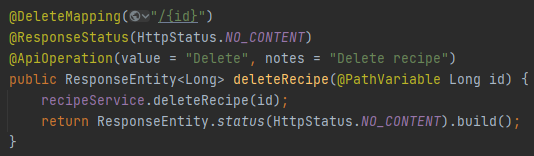




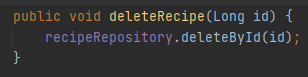
 

### Delete recipe

The *RecipeController* defines the **deleteRecipe** method which calls the service with the given id and returns a No Content response entity even if the record is found or not.

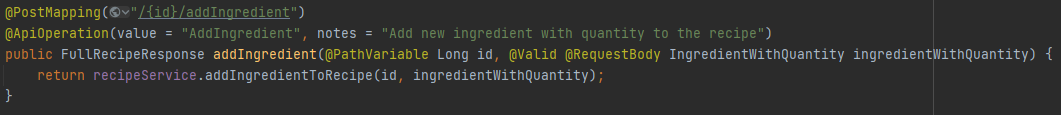


The service calls the repository delete by id method.

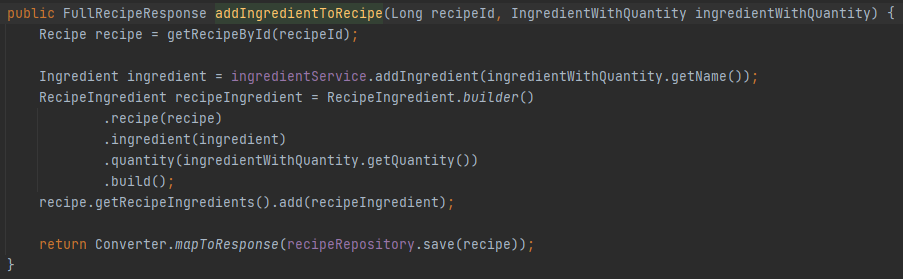


### Add Ingredient To Recipe

The *RecipeController* defines the **addIngredient** method which calls the service with the recipe id and *IngredientQuantity* object.



In the service the ingredient is saved by the *IngredientService*. A new *RecipeIngredient* entity is build and added to the recipe which is converted to a *FullRecipeResponse* and returned.

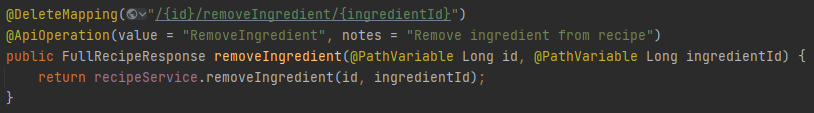


**Results:**

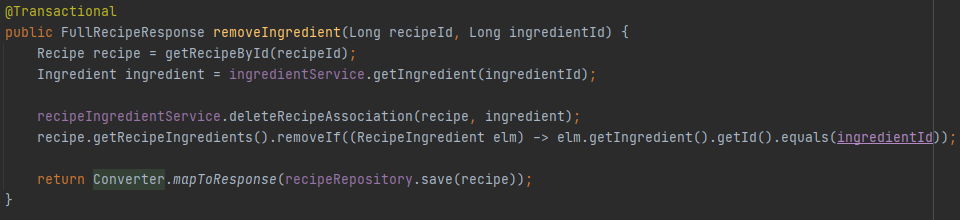
* Full Recipe Response
* Api Error – recipe not found
* Api Error – ingredient name not null
* Api Eror – ingredient min length

### Remove Ingredient From Recipe

The *RecipeController* defines the **removeIngredient** method which calls the service with the recipe id and the ingredient id.



The service deletes the record form the *RecipeIngredient* table, and then the same association is removed form the recipe entity. In order to persist context after the first deletion the @Transactional annotation is used.



**Results:**

* Full Recipe Response
* Api Error – recipe not found
* Api Error – ingredient not found

## RecipeIngredient

The RecipeIngredient entity resolves a many to many relationship between Recipe and Ingredient. It also stored as extra parameter for the relationship – quantity.

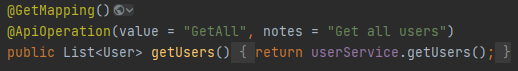
Since it is not used on her own it does not have a controller, just a repository and a service used by other entities services.

## User

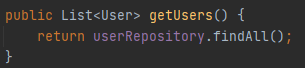
The user entity has an auto-generated id, a not null name and a unique, not null email.

### 4.1. Get Users

The *UserController* defines a **getUsers** method, which calls **getUsers** in the service.



The service returns all the users using the repository **findAll** method.

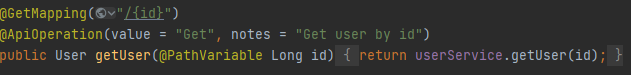


**Results:**

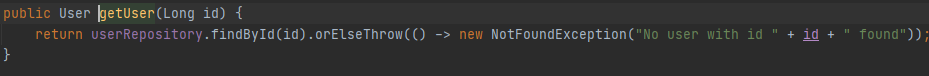
* User List
* Empty List

### 4.2. Get User

The *UserController* defines the **getUser** method which calls **getUser** from the service with the path variable id.

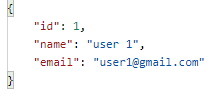
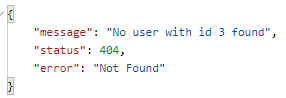


The service will call the **findById** method of the repository.



**Returns:**

* User
* Api Error – user not found

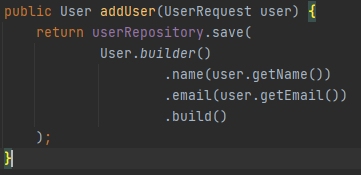
 

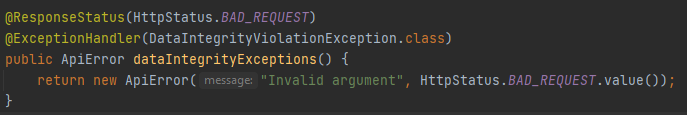
### 4.3. Add User

The *UserController* defines the **addUser** method which calls **addUser** from the service with the *UserRequest* object.



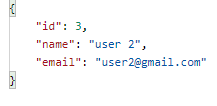
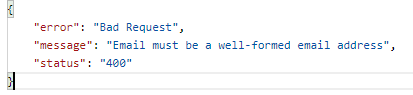
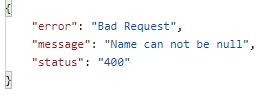
The service will create and return a new user. If the email in the user request already exists, a bad request error will be caught in the *GlobalExceptionHandler* class.





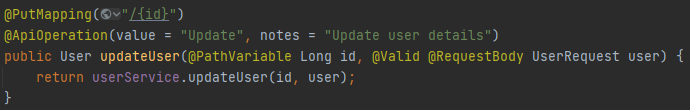
**Returns:**

* User
* Api Error – email not valid
* Api Error – name not null
* Api Error – invalid argument (email not unique)

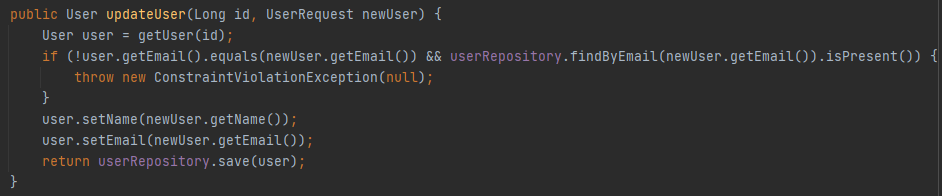
  

### 4.5. Update User

The *UserController* defines the **updateUser** method which calls the service with the user id and the *UserRequest* object.



The service will check if the email changed and if another record with the new email exists. In that case it will return an error caught by the constraintExceptions exception handler.

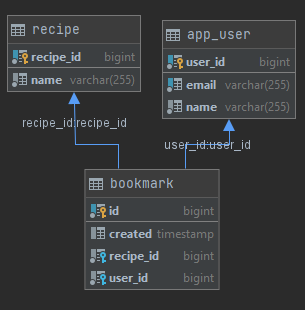


**Returns:**

* User
* Api Error – email not valid
* Api Error – name not null
* Api Error – invalid argument (email not unique)

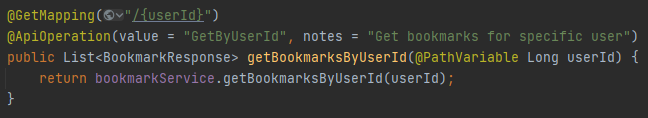
## Bookmark

The recipe entity has an auto-generated id, a timestamp property (LocalDateTime in ISO format) and a 1: Many relationship to User and Recipe entities.

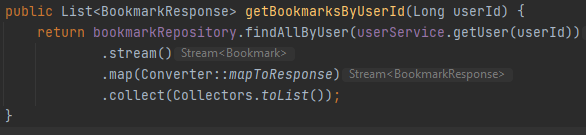


### 4.1. Get Bookmarks By User Id

The *BookmarkController* defines the **getBookmarksByUserId** method which calls the service with the user id.

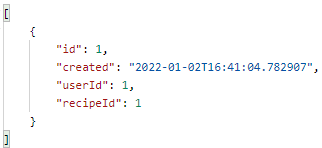
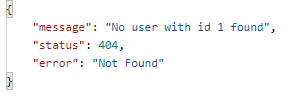


The service uses the findAllByUser method from the repository. And converts the *Bookmark* list to a *BookmarkResponse* one, using a helper function.



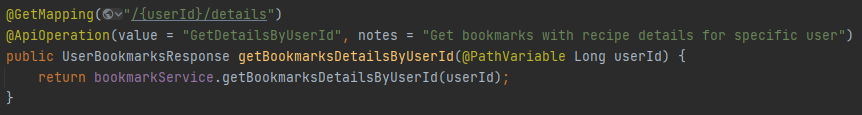
**Results:**

* Boomarks Response List
* Api Error – user not found

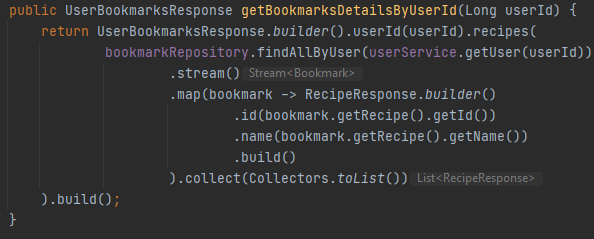
 

### 4.2. Get Bookmarks With Recipe Details For User

The *BookmarkController* defines the **getBookmarkDetailsByUserId** method and calles the service with the user id.

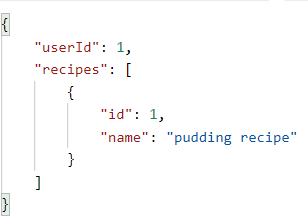


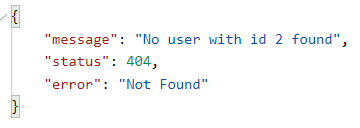
The service finds the bookmarks for the user using the **findAllByUser** method from the repository and creates an User*BookmarksResponse* which contains a list of *recipe* objects (with id and name).



**Results:**

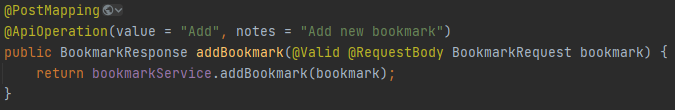
* UserBookmarksResponse
* Api Error – user not found



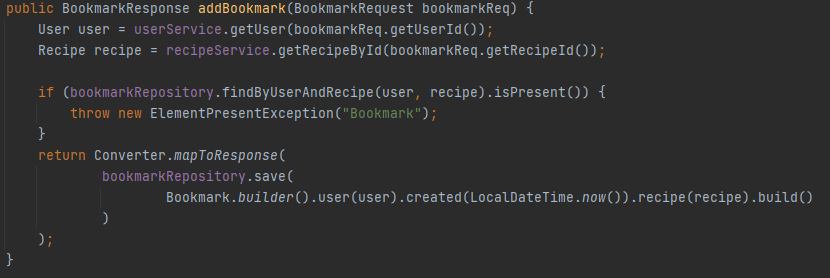


### 4.3. Add Bookmark

The *BookmarkController* defines the **addBookmark** method and calls the service with the *BookmarkRequest*.

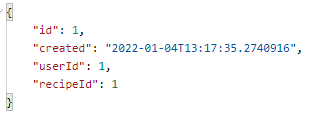


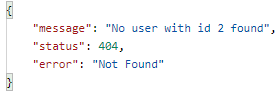
If a bookmark with the same recipe and user association exists an *ElementPresentException* is thrown which is caught in the *GlobalExceptionHandler* class.

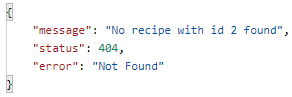


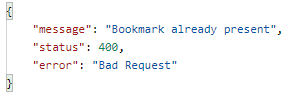
**Results:**

* Bookmark Reponse
* Api Error – user not found
* Api Error – recipe not found
* Api Error – bookmark already present



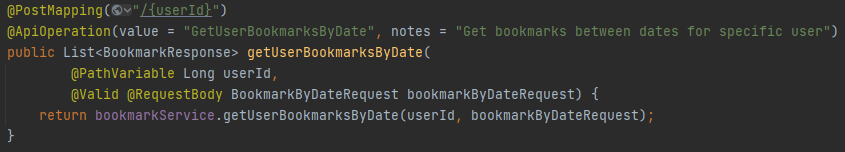




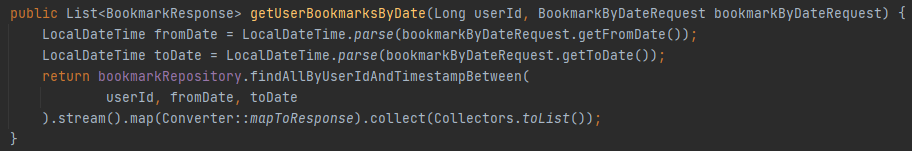


### 4.4. Get User Bookmarks By Date

The *BookmarkController* defines the **getUserBookmarksByDate** method which calls the service with the user id and a valid *BookmarkByDateRequest.*

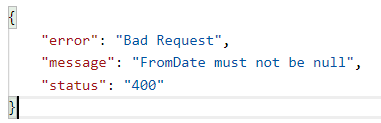
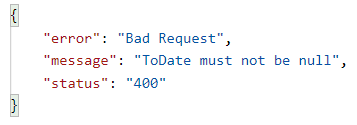


The service parses the dates and finds the bookmarks with the timestamp between those dates by calling **findAllByUserIdAndTimestampBetween** from the repository.



**Results:**

* Bookmark Response
* Api Error – fromDate not null
* Api Error – toDate not null

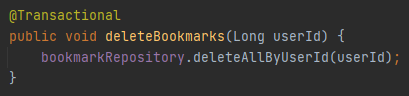
 

### 4.5. Delete Bookmarks By User Id

The *BookmarkController* defines the **deleteBookmarks** method which calls the service with the user id path variable and returns a no content response.



The service calls the **deleteAllByUserId** method from the repository.

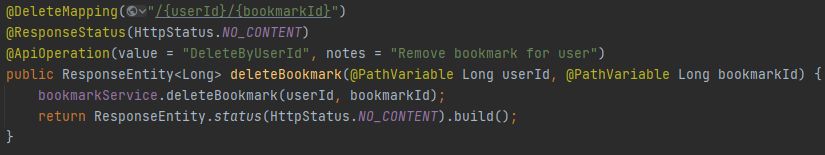


Results:

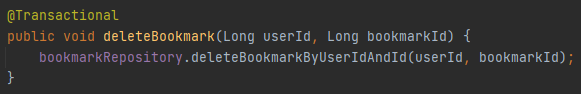
* No Content

### 4.6. Delete Bookmark

The *BookmarkController* defines the **deleteBookmark** method, which calls the service with the user and bookmark ids and returns a no content response.



The service calls the **deleteBookmarkByUserIdAndId** method from the repository.

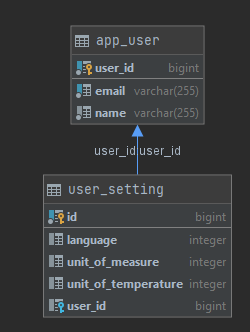


**Responses:**

* No Content

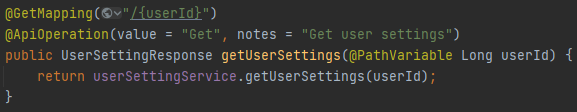
## User Settings

The *UserSettings* entity has an auto-generated id and three enum properties: language, unit of measure and unit of temperature. It also has a 1:1 relationship with the User entity.

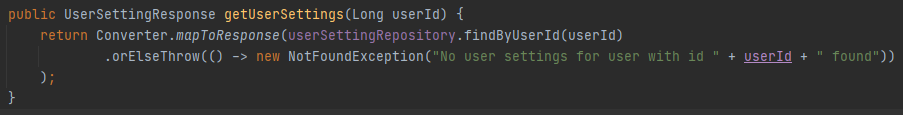


### 6.1. Get User Setting

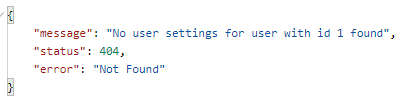
The *UserSettingController* defines a **getUserSetting** method witch calls the service with the user id.



The service finds the record by user id and maps it to an user setting response.



**Results:**

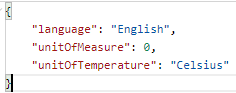
* User Setting
* Api Error – userSetting not found
*  

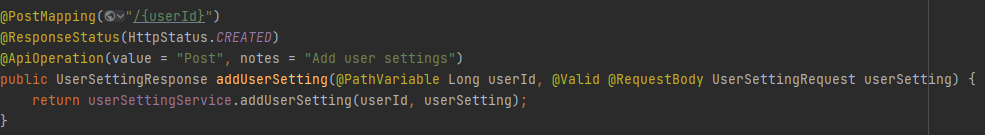
### 6.2. Add User Setting

The *UserSettingController* defines the **addUserSetting** method which calls the service with an *UserSettingRequest*. The enum properties in the request can be the sting or the numeric value. If null, or not provided default values will be applied:

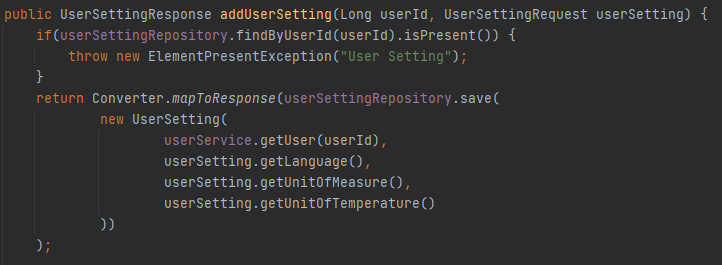
* language – English,
* unit of measure – Metric,
* unit of temperature - Celsius

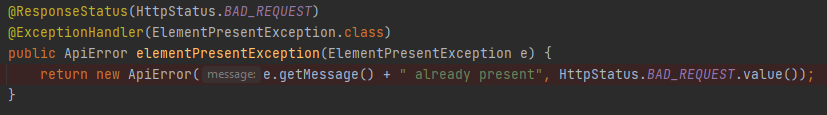
A request could look as fallows:





The service checks that a setting for the same user does not exist and then saves it. If it does, an *ElementPresentException* is thrown which is caught in the *GlobalExceptionHandler* class.

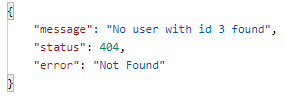
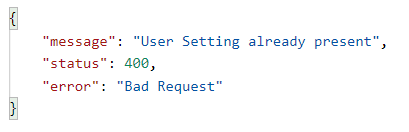




**Results:**

* User setting
* Api Error - user not found
* Api Error - user setting already present



# Business Requirements

1. As an app user I should be able to view the list of ingredients
2. As an app user I should be able to view an ingredient
3. As an app user I should be able to add an ingredient
4. As an app user I should be able to view the list of recipes
5. As an app user I should be able to view a recipe
6. As an app user I should be able to add a recipe with it’s ingredients and quantities
7. As an app user I should be able to update a recipe and it’s ingredients and quantities
8. As an app user I should be able to delete a recipe
9. As an app user I should be able to add an ingredient with quantity to a recipe
10. As an app user I should be able to remove an ingredient with quantity from a recipe
11. As an app user I should be able to bookmark a recipe
12. As an app user I should be able to view the list of my bookmarks
13. As an app user I should be able to view the list of my bookmarks with the recipe details
14. As an app user I should be able to view the list of my bookmarks between certain dates
15. As an app user I should be able to delete a bookmark
16. As an app user I should be able to delete all my bookmarks
17. As a administrator I should be able to view all the users
18. As a administrator I should be able to a user
19. As a administrator I should be able to add new users
20. As a administrator I should be able to update a user
21. As an app user I should be able to get my user settings
22. As an app user I should be able to add my user settings

# App Usage

* From recipeApp/docker run ***docker-compose up***
* Run the application
* Use the endpoints at *http://localhost:8080*