RECIPE

Attributes:

- int id
- String title
- String description
- HashSet<Ingredient> ingredients
- List<Feedback> feedbacks
- MealType mealType
- List<RecipeTag> recipeTags
- List<Direction> directions
- String sideNote

Methods:

- int getId()
- void setId(int id)
- String getTitle()
- void setTitle(String title)
- String getDescription()
- void setDescription(String description)
- MealType getMealType()
- void setMealType(MealType mealType)
- List<RecipeTag> getRecipeTags()
- void setRecipeTags(List<RecipeTag> recipeTags)
- void addRecipeTag(RecipeTag recipeTag)
- List<Feedback> getFeedbacks()
- void addFeedback()
- List<Direction> getDirections()
- void setDirections(List<Direction> directions)
- void addDirection(Direction direction)
- String getSideNote()
- void setSideNote(String sideNote)

MealType is an Enum type variable which describes whether the meal is breakfast, brunch, lunch, dinner or sweets and snacks.
RecipeTag is also an Enum type variable that names categories of meals. This variable can be used to filter vegan, vegetarian, meat meals, dairy free, gluten free or nut free options.
Other attributes and methods are self explanatory.

Direction

Attributes:

- int recipeId
- int stepNumber
- String text
- double stepDuration

Methods:

- int getRecipeId()
- void setRecipeId()
- int getStepNumber()
- void setStepNumber()
- String getText()
- void setText()
- double getStepDuration()
- void setStepDuration()

This class gives more information about the recipe. Text holds the description of the given preparation step in the recipe. The variable "stepDuration" represents the duration in minutes which will later be used in the implementation of a progress bar.

Feedback

Attributes:

- int userId
- int recipeId
- int rating
- String comment

Methods:

- int getUserId()
- void setUserId(int id)
- int getRating()
- void setRating(int rating)
- String getComment()
- void setComment(String comment)
- int getRecipeId()
- void setRecipeId(int id)

Feedback will be shown on the page of each recipe.

User

Attributes:

- int userId
- String name
- String lastName
- String mail
- String address
- String country
- UserRole userRole
- List<Recipe> savedRecipes

Methods:

- int getId()
- void setId(int id)
- String getName()
- void setName(String name)
- String getLastName()
- void setLastName(String lastName)
- String getMail()
- void setMail(String mail)
- String getAddress()
- void setAddress(String address)
- String getCountry()
- void setCountry(String country)
- UserRole getUserRole()
- void setUserRole(UserRole userRole)
- List<Recipe> getSavedRecipes()
- void setSavedRecipes(List<Recipe> savedRecipes)

UserRole is an Enum type variable which is used to give users certain permissions (such as Admin privileges, or VIP additional access).

Ingredient

Attributes:

- int id
- int amount
- String title

Methods:

- int getId()
- void setId()
- String getTitle()
- void setTitle(String title)
- int getAmount()
- void setAmount(int amount)

RecipeRepository

Methods:

- List<Recipe> getAllRecipes()
- Recipe getRecipe(int id)
- void insertRecipe(Recipe recipe)
- void deleteRecipe(int id)
- void updateRecipe(Recipe recipe)
- List<Recipe> getFilteredRecipes(MealType mealType, RecipeTag recipeTag)
- List<Recipe> getSavedRecipes(int userId)
- int saveRecipeForUser(int userId, int recipeId)

IngredientRepository

Methods:

- List<Ingredient> getAllIngredients()
- Ingredient getIngredient(int id)
- void insertIngredient(Ingredient ingredient)
- void insertIngredients(List<Ingredient> ingredients)
- void deleteIngredient(int id)
- void updateIngredient(Ingredient ingredient)

UserRepository

Methods:

- List<User> getAllUsers()
- User getUser(int id)
- void insertUser(User user)
- void deleteUser(int id)
- void updateUser(User user)

RecipeController

Methods:

- List<Recipe> getAllRecipes()
- Recipe getRecipe(int recipeId)
- int saveRecipe(Recipe recipe)
- int deleteRecipe(Recipe recipe)
- List<Recipe> getSavedRecipes(int userId)
- int saveRecipeForUser(int userId, int recipeId)
- int addIngredient(int recipeId, Ingredient ingredient)

IngredientController

Methods:

- List<Ingredient> getAllIngredients()
- Ingredient getIngredient(int ingredientId)
- int saveIngredient(Ingredient ingredient)
- int deleteIngredient(Ingredient ingredient)

UserController

Methods:

- List<User> getAllUser()
- int saveUser(User user)
- int deleteUser(User user)
- User getUser(int userId)

Controller classes contain instances of needed repository classes to fetch data and pass them to a user.
Repository classes communicate with the database and do CRUD operations on assigned tables. Additionally, RecipeRepository filters recipes accordingly to recipe tags and meal types.

SOLID principles

S states that every class should have only one responsibility, which is in our case true. For example, RecipeRepository class is responsible to only fetch data for recipes.

O states that every class should be open for extension and closed for modification. In our case, every class can be extended with new methods, but other methods should not be modified.

L states "you should be able to use any derived class instead of a parent class and have it behave in the same manner without modification". In our case, we do not have any derived classes because they are not needed and would only make things more complicated without any benefits.

I states "clients should not be forced to implement interfaces they don't use. Instead of one fat interface, many small interfaces are preferred based on groups of methods, each one serving one submodule.". IN our case, we do not have any interfaces, so we do not need to worry about this principle.

D states that high-level modules/classes should not depend on low-level modules/classes. Both should depend upon abstractions. Secondly, abstractions should not depend upon details. Details should depend upon abstractions. In our case, low-level classes are basically model and repository classes that are losely coupled because they are responsible for handling information about certain objects. Controller classes are losely coupled with repository classes because they handle communication with user and if an user asks for something, controller must be able to provide them with needed information. Recipe/feedback/user classes cannot implement or be derived from abstract classes because that does not make any sense. Repository classes also cannot implement an interface or be derived from an abstract class because different models ask for different SQL queries which cannot be generalized.