Documentation of backend methods – see the “Database Query Functions” editor fold and the “SQL EXECUTION FUNCTIONS” editor fold.

Methods that are not in **bold** font should probably not be called directly from the front-end but are used internally by other methods.

**Listing and Selecting Methods**

**listAllRecipes()**

Returns a two-dimensional array of all recipes in the format

{ {RecipeId, RecipeName, PrepTime, CategoryName, SubCategoryName, Rating} }

**listIngredients(int recipeId)**

Takes a recipeId as a parameter and returns a two-dimensional array of the ingredients for the recipe with that ID. The array's format is

{ { IngredientName, Quantity, Measurement (e.g. cups/tsp), AltIngredient} }

**listInstructions(int recipeId)**

Takes a recipeId as a parameter and returns a two-dimensional array of the instruction steps for the recipe with that ID. The array's format is

{ { StepNumber, InstructionText} }

**listComments(int recipeId)**

Takes a recipeId as a parameter and returns a two-dimensional array of the comments for the recipe with that ID. The array's format is

{ { Comment, Comment datetime } }

**listCategories()**

Returns a two-dimensional array of category names and category IDs. The Recipe object must use the category ID rather than the category name. The array's format is

{ { CategoryID, CategoryName } }

**listSubCategories()**

Returns a two-dimensional array of subcategory names and subcategory IDs. The Recipe object must use the subcategory ID rather than the subcategory name. The array's format is

{ { SubCategoryId, SubCategoryName } }

**selectSingleRecipe(int recipeId)**

Accepts a recipeId as a parameter and returns to the caller a two-dimensional array with that recipe’s data. Assuming the recipeId was unique, this \*should\* only return one row. The format of the result array is { { RecipeName, PrepTime, CategoryName, SubCategoryName, Rating } }

**getCategoryId(String name)**

Accepts a category name and returns the integer ID for that category.

**getSubCategoryId(String name)**

Accepts a subcategory name and returns the integer ID for that category.

**Adding Methods**

**addCategory(String category)**

Inserts the user-inputted category name to the CodeCategory table in the database, which generates an ID for that category. Returns *true* on successful insertion

**addSubCategory(String subCategory)**

Inserts the user-inputted subcategory name to the CodeSubCategory table in the database, which generates an ID for that category. Returns *true* on successful insertion

**addComment(int recipeId, String comment)**

Accepts a recipeId and a string comment, grabs and formats the current date and time, and inserts these into the Comments table of the database.

**addRecipe(Recipe object)**

Accepts a Recipe object as a parameter. The recipe object must have these properties specified:

recipeName (String)

rating (integer)

category (integer, the category ID)

subCategory (integer, the subcategory ID)

preptime (string)

ingredients (String[][], format { { IngredientName, Quantity, Measurement (e.g. cups/tsp), AltIngredient} } )

instructions (String[], format {"Step 1 text", "Step 2 text", …"Step *n* text"} )

Checks to see if a recipe with identical name, rating, category, subcategory, and preptime exists. If an identical recipe exists, it gets the matching recipe's Id and replaces that recipe's Ingredients and Instructions with the new ingredients and instructions.

If an identical recipe did not exist, it inserts these values into the Recipe database, which generates a new unique recipeId. It then uses the new recipeId to execute insertInstructions() and insertIngredients().

insertInstructions(int recipeId, String[] recipeInstructions)

This should never be called directly. It is used by editInstructions() and addRecipe().

Inserts the new set of instructions to the database, numbering the steps from 1 to *n* in the order that they were given in the array. The format of the input array should be

{"Step 1 text", "Step 2 text", …"Step *n* text"}

This should only be executed if deleteInstructions() was executed first or if the recipe was just created.

insertIngredients(int recipeId, String[][] recipeIngredients)

This should never be called directly. It is used by editIngredients() and addRecipe().

Inserts a new set of ingredients to the database. The format of the input array should be

{ { IngredientName, Quantity, Measurement (e.g. cups/tsp), AlternateIngredient } }

This should only be executed if deleteIngredients() was executed first or if the recipe was just created.

**Deleting Methods**

deleteComments(int recipeId)

Takes a recipeId and deletes all comments associated with that recipe from the database. Should only be used when deleting a recipe.

deleteInstructions(int recipeId)

Takes a recipeId and deletes all instructions associated with that recipe from the database. Should only be used when updating the set of all instructions or when deleting a recipe.

deleteIngredients(int recipeId)

Takes a recipeId and deletes all ingredients associated with that recipe from the database. Should only be used when updating the set of all ingredients or when deleting a recipe.

**totallyDeleteRecipe(int recipeId)**

Takes a recipeId and executes deleteComments(), deleteInstructions(), and deleteIngredients() for the given recipeId. Also deletes the row from the Recipes table that has the given recipeId.

**Editing Methods**

**editInstructions(int recipeId, String[] recipeInstructions)**

Takes a recipeId and an array of instructions. The format of the input array should be

{"Step 1 text", "Step 2 text", …"Step *n* text"}

Executes deleteInstructions() for the given recipeId and insertInstructions() for the given Id and array

**editIngredients(int recipeId, String[][] recipeIngredients)**

Takes a recipeId and a two-dimensional array of ingredients. The format of the input array should be

{ { IngredientName, Quantity, Measurement (e.g. cups/tsp), AlternateIngredient } }

Executes deleteIngredients() for the given recipeId and insertIngredients() for the given ID and array.

**editRecipe(Recipe object)**

Accepts a Recipe object as a parameter, and updates the database to set all property values of this object equal to the corresponding properties of the recipe with the given object's recipeId in the database.

**Other Methods**

**search(String searchParameters[])**

Accepts an array of search parameters. The array format should be

{RecipeName, PrepTime, CategoryId, SubCategoryId, Rating, Ingredient, Instruction}

Any parameters the user did not elect to use should be passed in as empty string. The ingredient should be a single ingredient only. The Instruction can be used to find key words anywhere within the instructions.

This will return a two-dimensional array of all the recipes that match the parameters. The array will have the format

{ { RecipeId, RecipeName, PrepTime, CategoryName, SubCategoryName, Rating } }

**emailRecipe(int recipeId)**

Accepts a recipeId, gets that recipe data from the database, builds a mailto with the recipe data in the body of the email, opens the user’s default mail client with that email, and returns true to the caller.

**SQL Execution Methods**

execSQLSingleColumnSelect(String sqlStatement)

Should be executed with a parameter that is a sql query that selects a single column of data. Initially puts the result set into an ArrayList, but transfers that data over to a normal array which is returned to the caller.

execSQLMultiColumnSelect(String sqlStatement)

Should be executed with a parameter that is a sql query that may select any number of columns of data. Places the result set into a two-dimensional array which is returned to the caller.

execSQLUpdateOrDelete()

Should be executed with a parameter that is a sql query that is not expected to return any data to the caller. This includes inserts, updates, and deletes on the database. Returns an integer result of the number of rows affected by the query.