Milestone 3: Pseudo Code and UI

CJ Hillbrand

Overview

There are several parts to this deliverable:

- A database schema: the schema is to demonstrate the simplicity offered in some of the data retrieval mechanisms in the pseudo code.
- Class diagram: The class diagram is to aid in the visual explanation of the code, and the responsibilities of components.
- Pseudo Code: Logical steps of critical components.
- User interface: The user interface intended to be a part of the final deliverable.

Pseudo Code and Supporting Documents

Database Schema

As part of the reason in including the database schema is to offer the opportunity to explain that the data retrieval, and preprocessing is taken care of before the development of the application. The data engineering to satisfy the schema of the below database is assumed to be completed by the time of development of the R shiny application.

The database schema can be seen in figure 1. The schema offers all information necessary for the statistical analysis. This allows for no data preparation necessary in the application layer.

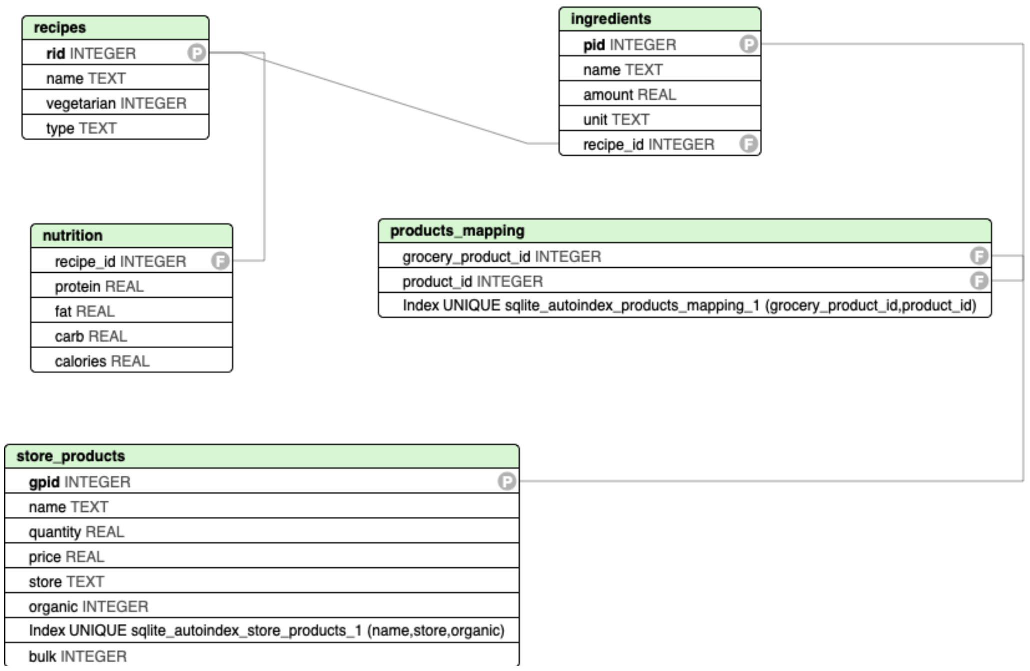

Figure 1: Database Schema

Class Diagram

The class diagram displays the relationship between components in the application. The class diagram shows where the later mentioned pseudo code will live in relation with other components.

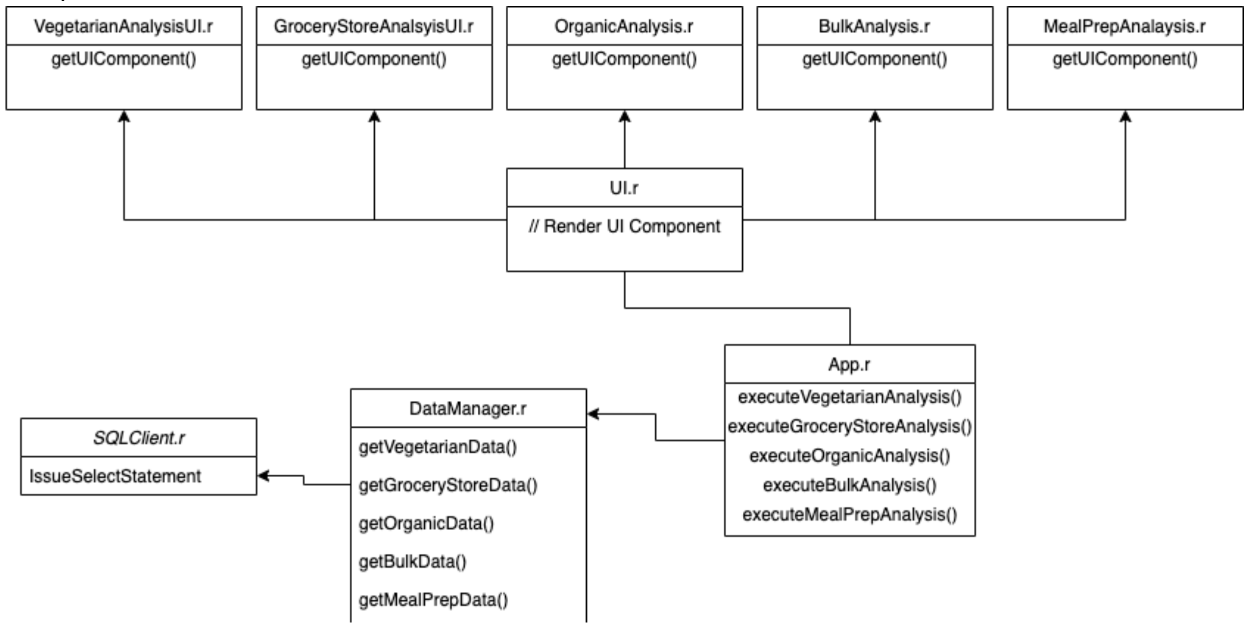

Figure 2: Class Diagram of the Application

The components are split into data management components and user interface components. The UI components oversee offering the UI components, while the data management components oversee the analysis:

- SQLClient.r: Responsible for integrating with native R libraries to issue sql queries to the database.
- DataManager.r: Responsible for issuing pre-defined queries to the SQLClient and returning data frames.
- App.r: Responsible for performing the analysis to answer the five questions.

Pseudo Code

Pseudo code for components performing analysis is offered below.

App.r executeVegetarianAnalysis

```
executeVegetarianAnalysis:
  df <- dataManager.getVegetarianData()
  dist_veg <- getDistributionForVegetarianRecipe()
  dist_non_veg <- getDistributionForNonVegetarianRecipes()
  // save distributions in server components
  t_stat <- generate_t_stat(dist_veg, dist_non_veg)

executeGroceryStoreAnalysis(store_one, store_two):
  df_1 <- dataManager.getGroceryStoreData(store_one)
  df_2 <- dataManager.getGroceryStoreData(store_two)
  dist_store_one <- getDistribution(df_1)
  dist_store_two <- getDistribution(df_2)
  // save distribution in server components
  t_stat <- generate_t_stat(dist_store_one, dist_store_two)

executeOrganicAnalysis:
  df <- datamanager.getOrganicData()
  dist <- getDistribution(df)
  // save dist in server components
  hyp_test <- evaluate_null_hypothesis(h_0, h_a, dist)

executeBulkAnalysis:
  df <- dataManager.getBulkData()
  dist <- getDistribution(df)
  // save dist in server components
  Hyp_test <- evaluate_null_hypothesis(h_0, h_a, dist)

executeMealPrepAnalysis:
  df <- datamanager.getMealPrepData()
  samples <- generateWeeksWorthOfMeals()
  simulate_meal_prep <- replaceMeals(samples)
  dist <- plotPricePoints(simulate_meal_prep.meals_prepped,
simulate_meal_prep.price)
  regression_model <- getRegressionModel(dist)
  // save distribution and regression in server components
```

User Interface

See attached PDF for user interface mocks. Each analysis has their own dedicated page. Each analysis has some graphical component, some summary component and if applicable a section for users to adjust the visual and summary statistics.