Course Professor: Dr. Johnson
Team Members: Bree Betts, Maya Dahlke, Adriana Donkers, Henno Kublin, Lexi Weingardt
CMSI 401
18 November 2020

Software Design Description Document (Detailed Section)

6.3 Detailed CSC and CSU Descriptions Section

6.3.1 Detailed Class Descriptions Section

The following sections provide the details of all classes used in the Daily Bites application:

6.3.1.1 Account Class - represents a user of the application
6.3.1.1.1 Id Field - unique key to distinguish users
6.3.1.1.2 Date Created Field - date the user signs up for the application
6.3.1.1.3 Name Field - name of user
6.3.1.1.4 Email Field - email of user
6.3.1.1.5 Diets Field - string representation of diets the user follows (used for recommendations)
6.3.1.1.6 Dietary Restrictions Field - specific allergies or restrictions of the user
6.3.1.1.7 Cuisine Preferences Field - preferred cuisines of the user
6.3.1.1.8 Facebook Id Field - if a user signs up with facebook, their facebook id is recorded
6.3.1.1.9 Set Account Method - creates a user
6.3.1.1.10 Get Account Method - gets the information for a user of the application
6.3.1.1.11 Update Account Method - makes updates to a user's information, whether it be a diet change, email update or cuisine preference addition

6.3.1.2 Meal Class - represents a meal that is logged by a user
6.3.1.2.1 Id Field - unique key to distinguish meals
6.3.1.2.2 Account Id Field - references the Account class to represent which user logged the meal
6.3.1.2.3 Date Logged Field - date the user logged the meal
6.3.1.2.4 Category Field - the category a meal falls into (Breakfast, Lunch, Dinner, Snack)
6.3.1.2.5 Set Meal Method - creates a meal for a user
6.3.1.2.6 Get Meals Method - returns the meals for a specific user/time period specified
6.3.1.2.7 Delete Meal Method - deletes a meal that a user has logged

6.3.1.3 Nutrition Class - specific nutrients for 100 grams of a food item
6.3.1.3.1 Food Id Field - unique key to distinguish the nutrients for specific food items
6.3.1.3.2 Calories Field - the calories found in this food item

6.3.1.3.3 Protein Field - the amount of protein in this food item
6.3.1.3.4 Fat Field - the amount of fat in this food item
6.3.1.3.5 Carbohydrate Field - the amount of carbohydrates in this food item
6.3.1.3.6 Get Nutrition Method - this is called internally to perform the math needed to find a user's nutrition for a food item and meal

6.3.1.4 Food Class - food that is consumed in a meal
6.3.1.4.1 Meal Id Field - references the Meal Class, the meal the food is a part of
6.3.1.4.2 Food Id Field - references the Nutrition Class, the nutrition information for the food
6.3.1.4.3 Log Id Field - the specific log a user consumes the meal in
6.3.1.4.4 Food Unit Field - unit representation for quantity of food consumed (ex: tablespoons)
6.3.1.4.5 Food Quantity Field - numeric quantity for the food unit consumed (ex: 2)
6.3.1.4.6 Set Food Method - creates a food item for a meal that a user is logging
6.3.1.4.7 Get Food Method - returns the data for a food item being requested

6.3.1.5 Food Unit Class - table to help with converting quantities from Food Class to grams in Nutrition Class
6.3.1.5.1 Food Id Field - references Nutrition Class, a specific food item
6.3.1.5.2 Food Unit Field - the string representation of a food item, can be matched with Food Unit Field of the Food Class
6.3.1.5.3 Grams Per Unit Field - grams conversion of the food for the specific food unit
6.3.1.5.4 Get Units Method - returns the units and grams equivalent for food item and unit requested

6.3.1.6 Food Detail Class - details about a food item to display to user
6.3.1.6.1 Food Id Field - unique key to distinguish between food items
6.3.1.6.2 Food Description Field - plain english representation of the food item (example 'Banana')
6.3.1.6.3 Barcode Field - barcode of the food item
6.3.1.6.4 Brand Field - brand of the food item
6.3.1.6.5 Food Group Field - food group that the food item belongs to
6.3.1.6.6 Ingredient List Field - string representation of the list of ingredients found in the food item
6.3.1.6.7 Processed Description Field - used for comparing with Chatbot tagger, takes the food description and processes it to remove unnecessary words and sorts in alphabetical order
6.3.1.6.8 Find Food Item Method - used to find foods that match the text the Chatbot has tagged

6.3.1.7 Recipe Class - recipes from Spoonacular that are recommended to users
6.3.1.7.1 Recipe Id Field - unique identifier for a recipe
6.3.1.7.2 Title Field - title of the recipe

6.3.1.7.3 Recipe Description Field - description of the recipe provided
6.3.1.7.4 Recipe Ingredients Field - ingredients required to make recipe
6.3.1.7.5 Recipe Instructions Field - instructions to follow to make recipe
6.3.1.7.6 Set Recipe Method - adds a recipe to the database
6.3.1.7.7 Get Recipe Method - returns the information for a recipe

6.3.1.8 Rasa Action Classes - used for custom actions in the chatbot, which extend from Rasa's existing Action classes, thus having a long list of default fields. These are only used in the Rasa application for the chatbot
6.3.1.8.1 No new fields are defined specifically for Daily Bites
6.3.1.8.2 Name Method - returns the name of the custom action which corresponds to the name in the domain.yml file
6.3.1.8.3 Run Method - when this custom action is used, this defines what the action should do (i.e. if it's food logging: send the food log data to the Flask backend app)

6.3.2 Detailed Interface Descriptions Section

The following sections provide the details of all interfaces used in the Daily Bites application:

6.3.2.1 The Dashboard CSU receives data from the Flask App CSU and displays the users' trends in a series of charts/graphs
6.3.2.2 The Settings CSU receives data from user input and sends that data to the Flask App CSU
6.3.2.3 The Chatbot CSU receives user input and responds accordingly with data pulled from the Flask App CSU
6.3.3.4 The Chatbot CSU collects data about a user from the Flask App CSU and makes recommendations to users
6.3.3.5 The Chatbot CSU allows users to navigate to a specific recipe in the Client Recipe Recommendation Information CSU
6.3.2.6 The Client Recipe Recommendation CSU receives data from the Flask App CSU. When the user clicks a button, the user id is passed up to the Flask App, which uses the user_id to gather all the data needed from the Database CSU, and looks for the */recipes/<user_id>* REST API route. Then, it returns a list of JSON information back to the front-end
6.3.2.7 The Client Recipe Recommendation Information CSU receives data from the Flask App CSU. When the user clicks on a recipe, the user id and recipe id is passed up to the Flask App, which uses the ids to gather all the data needed from the Database CSU, and looks for the */nutrients/<int:recipe_id>/<user_id>*, */ingredients/<int:recipe_id>/<user_id>*,

and */instructions/<int:recipe_id>/<user_id>* REST API routes. Then, it returns a list of JSON information back to the front-end

6.3.2.8 The Client Saved Recipes CSU receives data from the Flask App CSU. When the user clicks a button, the user id is passed up to the Flask App, which gathers all the data needed from the Database CSU, and looks for the */savedrecipes/<user_id>* REST API route. Then, it returns a list of JSON information back to the front-end

6.3.2.9 The RDS CSU sends data that is needed on the front end to the Flask App CSU, which then sends that data to the front end

6.3.3 Detailed Data Structure Descriptions Section

The following sections provide the details of all data structures used in the Daily Bites application:

6.3.3.1 Query results from Flask App CSU on Dashboard CSU

6.3.3.2 Data Structures in the Chatbot CSU

6.3.3.2.1 All of the Training Data is structured in YAML files with keys indicating the type of training data.

6.3.3.2.1.1 NLU data of the Training Data is used for training the NLU models so that the NLU models can learn the types of structured information (such as intents) to extract from user messages. It is unordered in nature.

6.3.3.2.1.2 Stories are a type of training data that is organized in a data structure and is used to train the assistant's dialogue management model. It is an ordered structure because it indicates how a conversation is expected to play out between the user and bot.

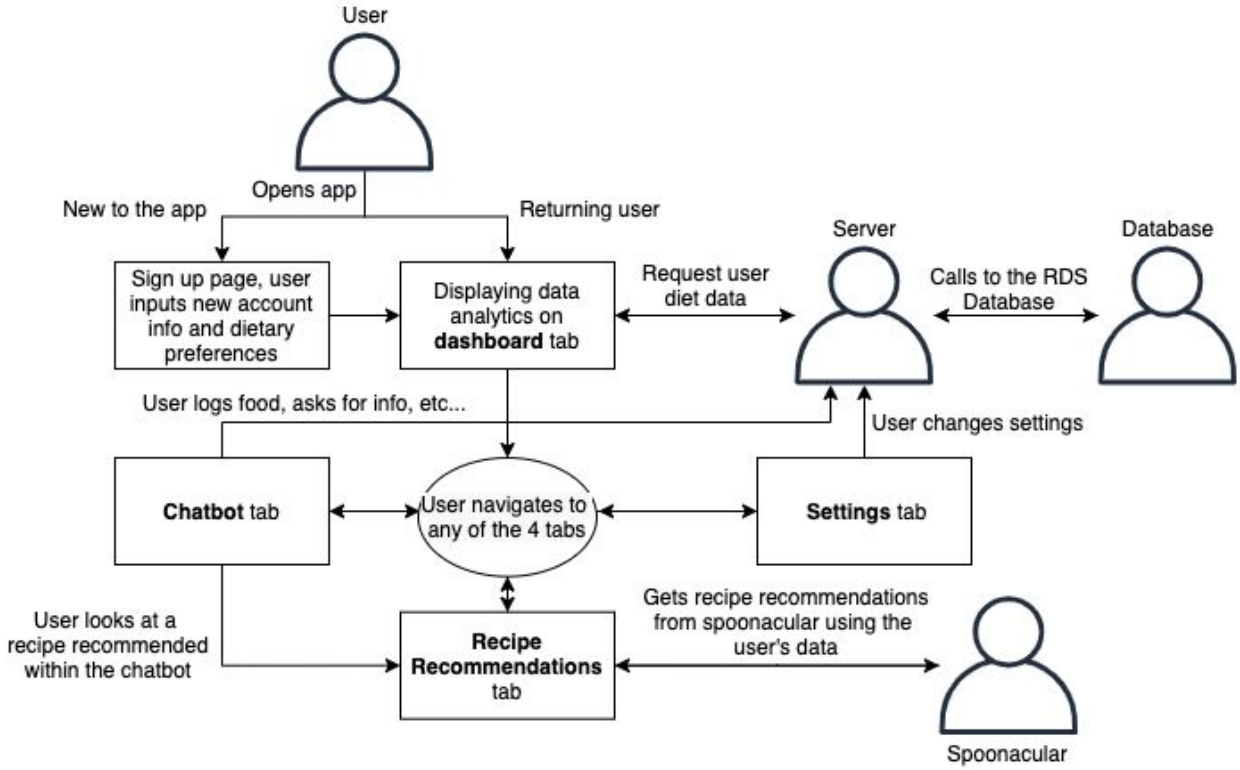
6.3.3.2.1.3 Rules are a type of training data that is also used to train the assistant's dialogue management model. Like stories, rules are ordered because they describe short pieces of conversations that should always follow the same path.

6.3.3.3 Query results from Flask App CSU on Client Recipe Recommendation CSU.

6.3.3.4 Query results from Flask App CSU on Client Recipe Recommendation Information CSU.

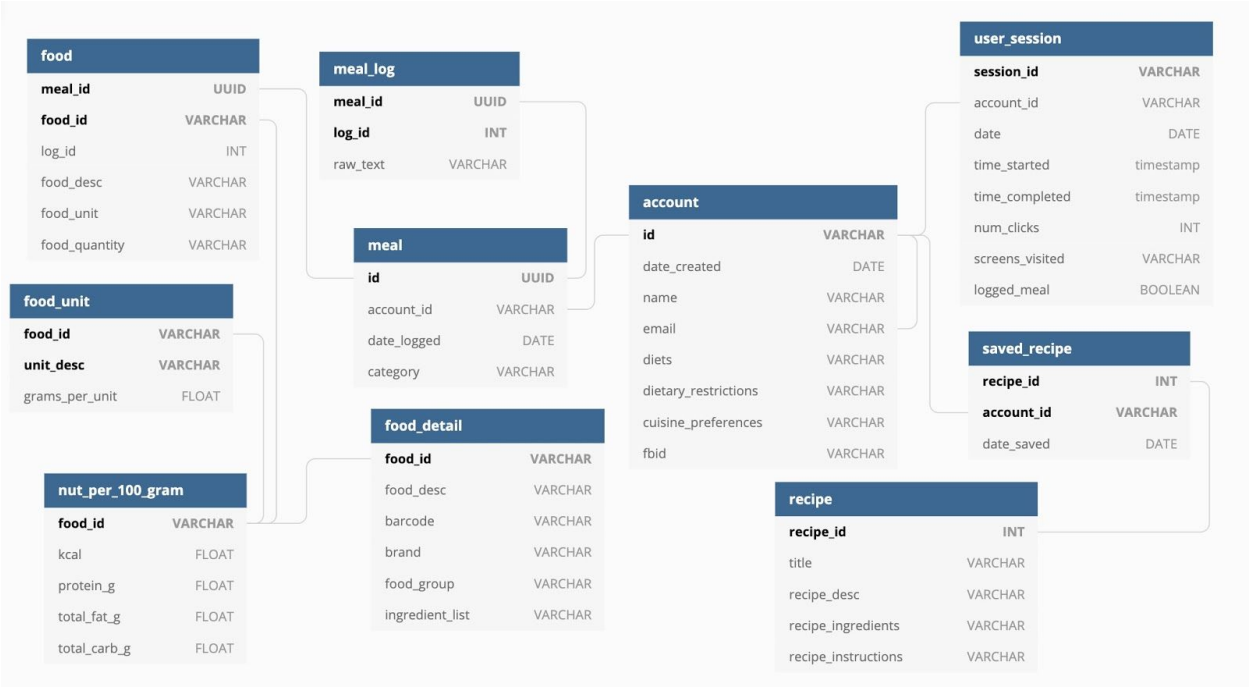
6.3.3.5 Query results from Flask App CSU on Client Saved Recipes CSU

6.3.4 Detailed Design Diagrams Section



6.4 Database Design and Description Section

6.4.1 Database Design ER Diagram Section



6.4.2 Database Access Section

The database is entirely accessed through a flask application that is run on an EC2 instance in a public subnet within our own personal “Daily Bites” VPC. This allows for higher security that is highlighted in section 6.4.3 below. When any data is required due to an interaction made with the front-end iOS by the user, a computation is made within the flask app that handles requests to the database and then either forwards that data to be displayed in the front end or runs a calculation with that data that will then be displayed.

Conversely, when a user adds data through the sign in process, chatbot, or any other means, that data is sent to the flask app which determines where to store that data in the database. The database is held in a private subnet without access to the internet and can only be accessed by that flask app to allow for a closed ecosystem with only access from the iOS front end to the internet gateway into the flask app. This stack structure is typical industry best practice since it is highly secure and efficient.

6.4.3 Database Security Section

The RDS database is held in a private subnet within a personal VPC to the project. This allows for all entities within the VPC to be virtually independent from other services being used by outside users on AWS. No outside user has direct access to the database since the database itself is not directly attached to the internet. Even if a hostile entity was able to get the username and password for the database, they would not be able to access it since there is no way to connect without it being internet accessible.

The only way we would have access to the database is through the AWS console, which even then doesn’t allow for display of what is held in the database but only the ability to shut down the database. As a final touch, even if somehow an outside force was able to get into the AWS console and deleted the database, AWS would automatically hold a snapshot of that database that could not be deleted for 14 days. All in all, this is an incredibly secure architecture.