Program Design Document

Team Red (Student Nutrition Mobile App)

Paul Abts
Gage Askegard
Donovan Beckmann
Brett Chastain
Jordan Falcon
Connor Fradenburgh
Zach Grosz
Tyler Johnson
Victoria Kyereme
Grant Moe
Sethu Monick
Ryan Nelson
Mitchell Olson
Jaron Pollman
James Raboin

Date: 29 November 2016

Preface

This document describes the program design for the Student Nutrition Mobile app project. It decomposes the system into a series of classes and interfaces, and describes the function and relationship of each of these parts.

The Student Nutrition Mobile App system is composed of the following modules:

- App GUI
- App Engine
- Phone Database (Data)
- Phone Database (Operations)
- MySQL Server (Data)
- MySQL Server (Operations)

App GUI

The App GUI is composed of into a number of Java classes as well as XML. This app is composed of three main groups of Java classes - those extending the Fragment class, those extending the AppCompatActivity class, and those extending the FragmentPagerAdapter class.

The XML portion is organized by each fragment in the app. For each fragment, the .xml file includes a 'Design' view and a 'Text' view. The 'Design' view shows the app screen for the chosen fragment, and allows the user to drag and drop included Android widgets, including TextViews, Buttons, etc.

Users are able to change properties of these widgets in the design view, and users will create the screen they want in the app. The 'Text' view goes more in depth of the properties. It starts with the fragment properties, including orientation, width, and height, and each widget that is added also appears with abilities to change properties in the 'Text' view.

Before introducing the breakdown of the App GUI, it is important to first explain a few of the essential terms of Android app development:

- Activity: a single, focused thing that a user can do.
- Fragment: a piece of an application's user interface or behavior that can be placed in an Activity
- Page: a single screen in an app
- Layout: invisible containers that hold other Views (or other ViewGroups)
- Widget: used to create interactive UI components (buttons, text fields, etc.)

Source: Developer.Android.com

App GUI

The App GUI contains a number of classes that extend the Fragment class. Before cxplaining the implementation of user-defind subclasses, let's explain a few of the standard classes that may be unfamiliar to new Android developers:

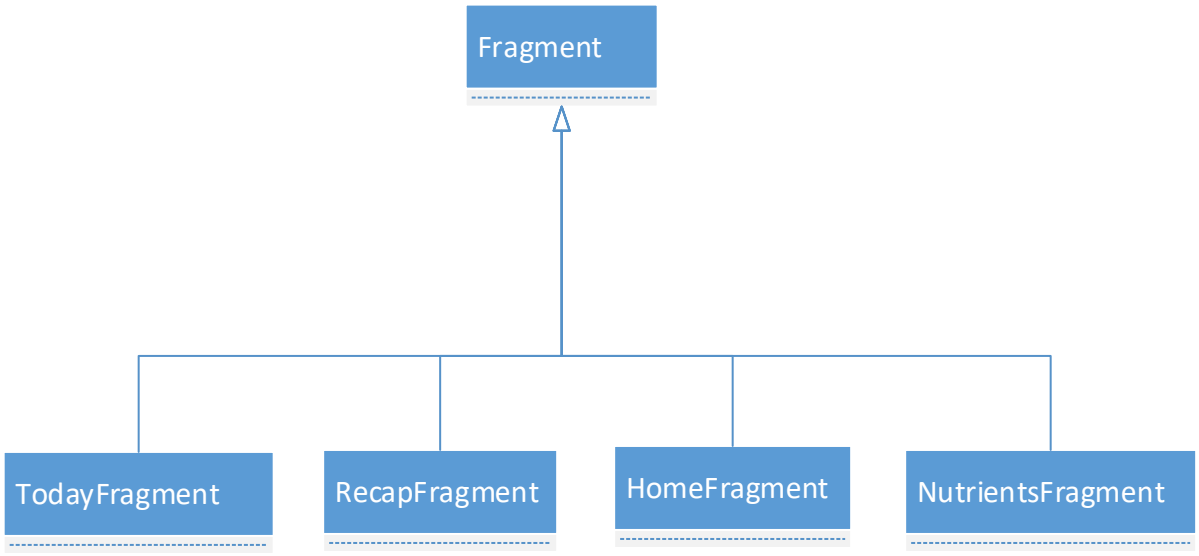
Activity class:

- An activity is a single, focused thing that a user can do.
- Almost all activities interact with the user, so the Activity class takes care of creating a window in which developer can place the UI

Fragment class:

- a piece of an application's user interface or behavior that can be placed in an Activity
- essentially it represents a particular operation or interface that is running within a larger Activity

Source: Developer.Android.com



Each fragment class represents an individual page of the Android app.

- The TodayFragment represents the Today's Meals screen displaying the meals a user has recorded for the current date.
- The RecapFragment represents the Recap screen displaying the list of past dates a user can view recorded meals from.
- The HomeFragment represents the Dashboard screen of the app, the main screen users navigate to all other screens from.
- The NutrientsFragment represents the User Nutrients screen, which displays the nutrient amounts and percentages from recorded meals.

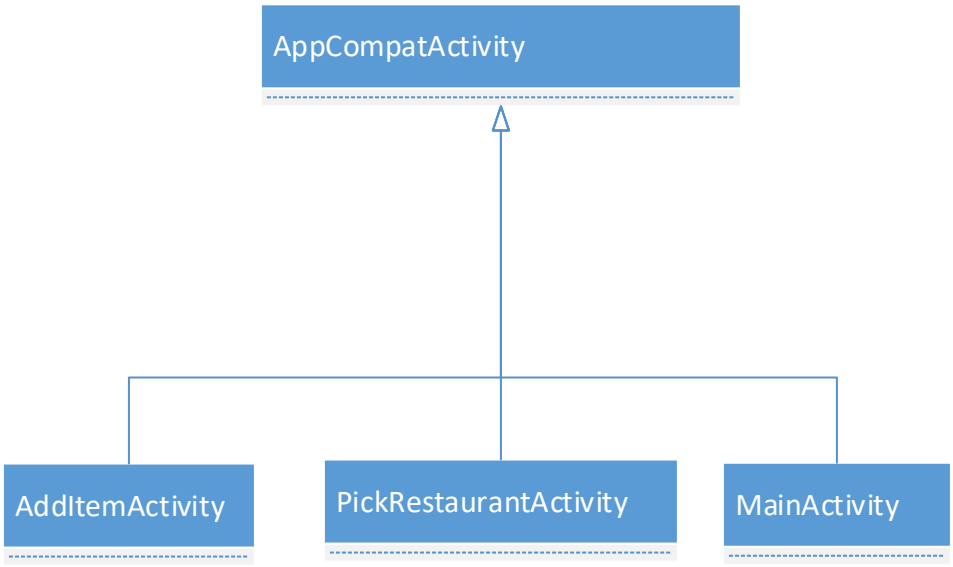
App GUI

The App GUI contains a number of Activity subclasses, which extend the AppCompatActivity class. Before cxplaining the implementation of user-defind subclasses, let's explain a few of the standard classes that may be unfamiliar to new Android developers:

FragmentActivity class:
- Extends Activity class
- acts as base class for activities that want to use the support-based Fragment and Loader APIs

AppCompatActivity class:
- Extends FragmentActivity class
- acts as base class for activities that use the support library action bar features

Source: Developer.Android.com



The app has been designed so that each Fragment sits on an Activity. In other words, each Activity class uses a particular set of Fragments to produce functionality. Designed in this way, it is easy to use buttons to switch between activities.

MainActivity is a class automatically created by the Android Studio IDE when building an Android app.
AddItemActivity represents the process of recording a meal.
PickRestaurantActivity represents the process of choosing the particular location for that meal.

App GUI

The App GUI contains a SectionsPagerAdapter class. This class extends the FragmentPagerAdapter class, which is an implementation of the PagerAdapter class. Before cxplaining the implementation of user-defind subclasses, let's explain a few of the standard classes that may be unfamiliar to new Android developers:

View class:
- represents the basic building block for user interface components
- A View occupies a rectangular area on the screen and is responsible for drawing and event handling
- base class for widgets

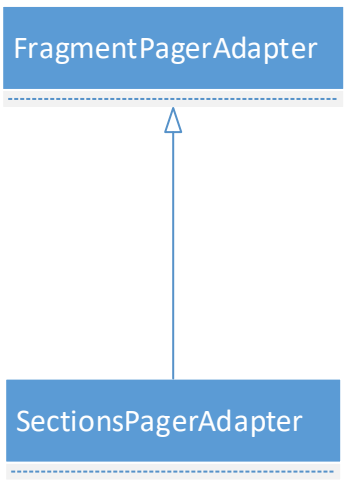
ViewGroup class:
- Extends View class
- special view that can contain other views (called children)
- base class for layouts and views containers

ViewPager class:
- Extends ViewGroup class
- Layout manager that allows the user to flip left and right through pages of data
- supply an implementation of a PagerAdapter to generate the pages that the view shows

PagerAdapter class:
- provides the adapter to populate pages inside of a ViewPager

FragmentPagerAdapter class:
- represents each page as a Fragment object which is persistently kept in the fragment manager as long as the user can return to the page

Source: Developer.Android.com



The SectionsPagerAdapter class is automatically created by the Android Studio IDE when building an Android app. It allows developers to link a set of Fragments to an Activity.

App GUI

Fragment subclasses have very similar code. Before cxplaining the implementation of user-defind subclasses, let's explain a few of the standard classes that may be unfamiliar to new Android developers:

IBinder
- interface for a remotable object
- describes the abstract protocol for interacting with a remotable object
- not implemented directly

Parcel:
- Container for a message (data and object references) that can be sent through an IBinder.

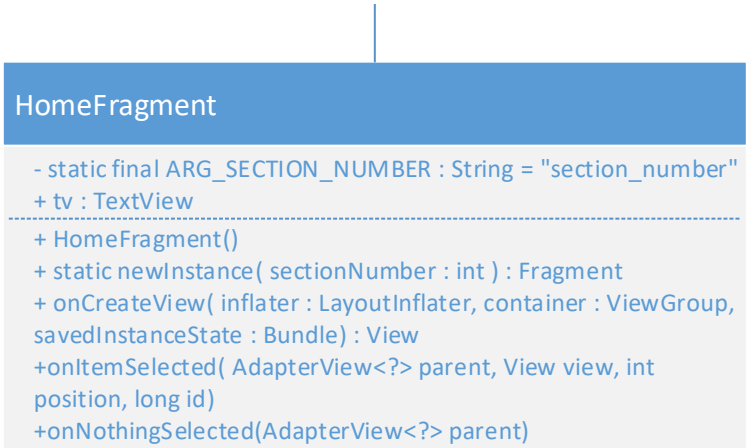
Parcelable interface:
- Interface for classes whose instances can be written to and restored from a Parcel.

Bundle class:
- A mapping from String keys to various Parcelable values

LayoutInflater class:
- Instantiates a layout XML file into its corresponding View objects.

Source: Developer.Android.com

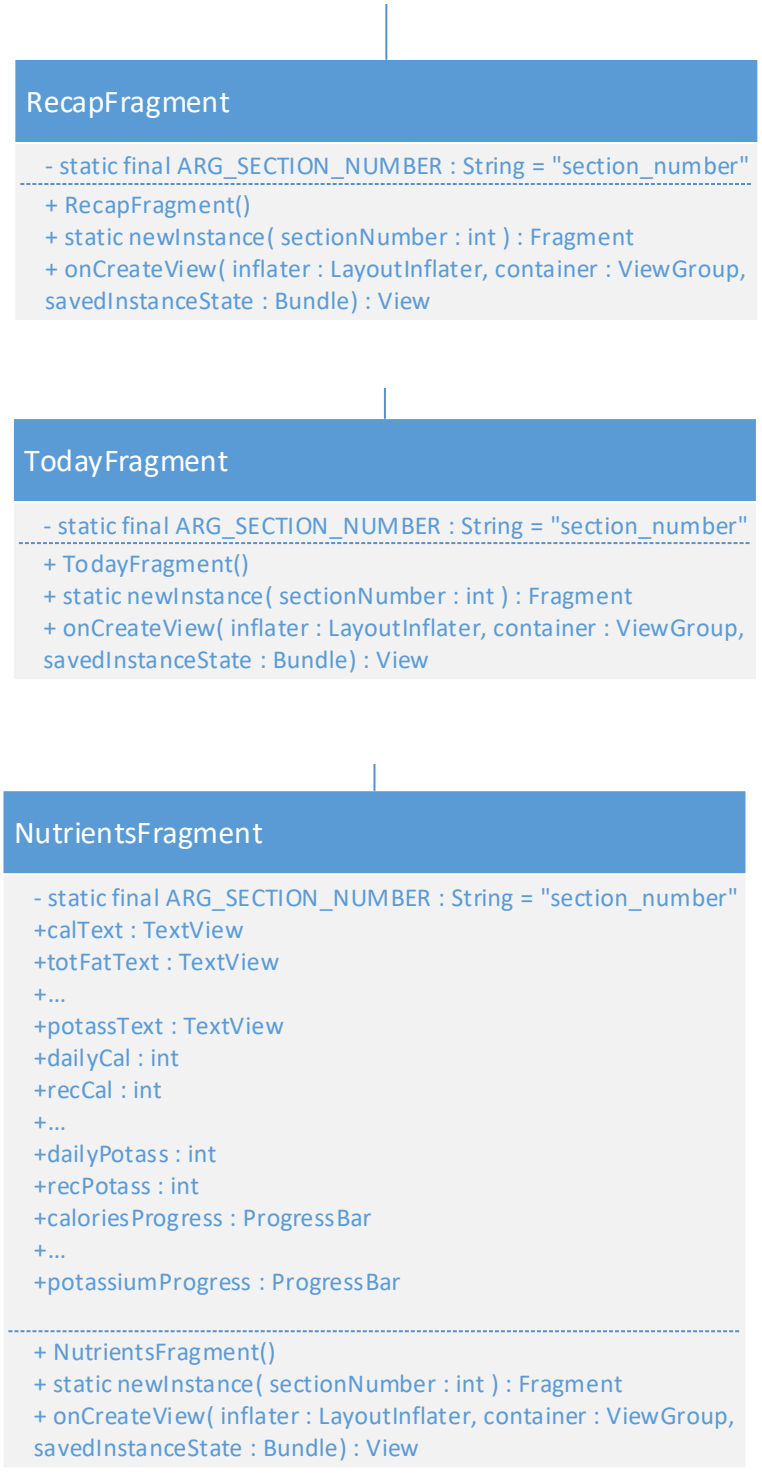
HomeFragment class:
The HomeFragment class was created for the Home screen fragment. The class contains all of the actions that need to be done every time the Home fragment is opened. Currently, the HomeFragment class does the following: Sets the spinner progress for the requested nutrient and displays a list of recommendations.



App GUI

The RecapFragment class was created for the Recap screen in the App GUI. The TodayFragment class was created for the Today's Meals screen. The NutrientsFragment class was created for the Percent Daily Values screen.

Their functionality is similar to the HomeFragment class.



App GUI

AppCompatActivity subclasses are each slightly different. Before cxplaining the implementation of user-defind subclasses, let's explain a few of the standard classes that may be unfamiliar to new Android developers:

TextView class:
- Derived from View class
- Displays text to the user and optionally allows user to edit it

Button class:
- Derived from TextView class
- Represents a push-button widget. Push-buttons can be pressed, or clicked, by the user to perform an action.

Menu interface:
- Interface for managing the items in a menu

MenuItem interface:
- Interface for direct access to a previously created menu item.

Spinner class:
- A view that displays one child at a time and lets the user pick among them.

Adapter class:
- An Adapter object acts as a bridge between an AdapterView and the underlying data for that view.
- The Adapter provides access to the data items.
- The Adapter is also responsible for making a View for each item in the data set

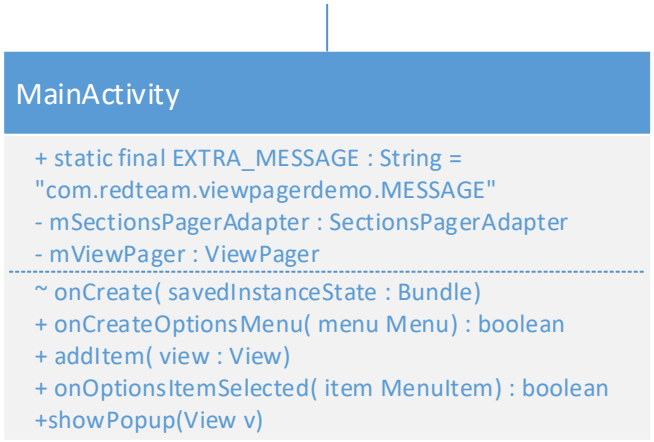
BaseAdapter abstract class:
- base class of common implementation for an Adapter that can be used in both ListView and Spinner classes

ArrayAdapter<E> class:
- A concrete BaseAdapter that is backed by an array of arbitrary objects
- This widget allows for the user to easily adapt an object using an Array.
- For example, it is used in the spinner object when a user selects the requested nutrient, from an array of nutrients, that they want to be displayed. The spinner object then changes to show the selected nutrient.

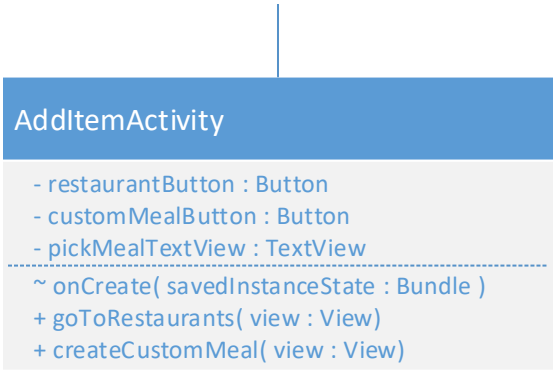
Source: Developer.Android.com

App GUI

MainActivity class:
The MainActivity class was created automatically for the app. It holds the main fragments of the app.

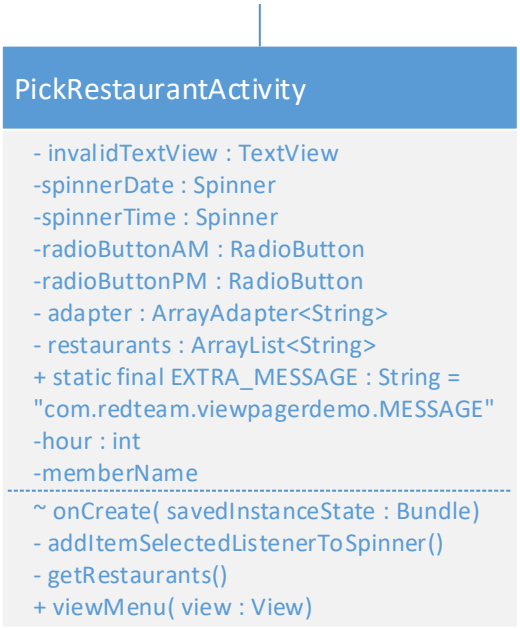


AddItemActivity class:
The AddItemActivity class was created for the Add a Meal process in the app. It works in conjunction with the activity_add_item.xml file. The only thing it does is prompt the user to choose whether the meal was a restaurant meal or a custom meal. Depending on the button pressed, the user will be taken to the next activity. Currently there is no custom meal option in the app so the user is taken to the pick restaurant activity.



App GUI

PickRestaurantActivity class:
The PickRestaurantActivity was created for the Add a Meal process in the app. It works in conjunction with the activity_pick_restaurant.xml file. The user is able to select a date, time, and a restaurant from the list of restaurants in the database. This information is then stored in an object that will affect the meal items that are shown in the next screen.



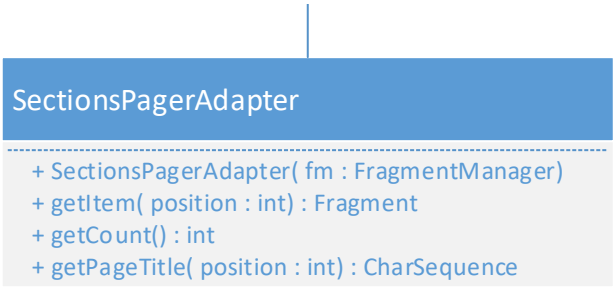
App GUI

The FragmentPagerAdapter subclass is called the SectionsPagerAdapter class. Before cxplaining the implementation of user-defind subclasses, let's explain a few of the standard classes that may be unfamiliar to new Android developers:

FragmentManager interface:
- Interface for interacting with Fragment objects inside of an Activity

CharSequence interface:
- Interface for a readable sequence of char values

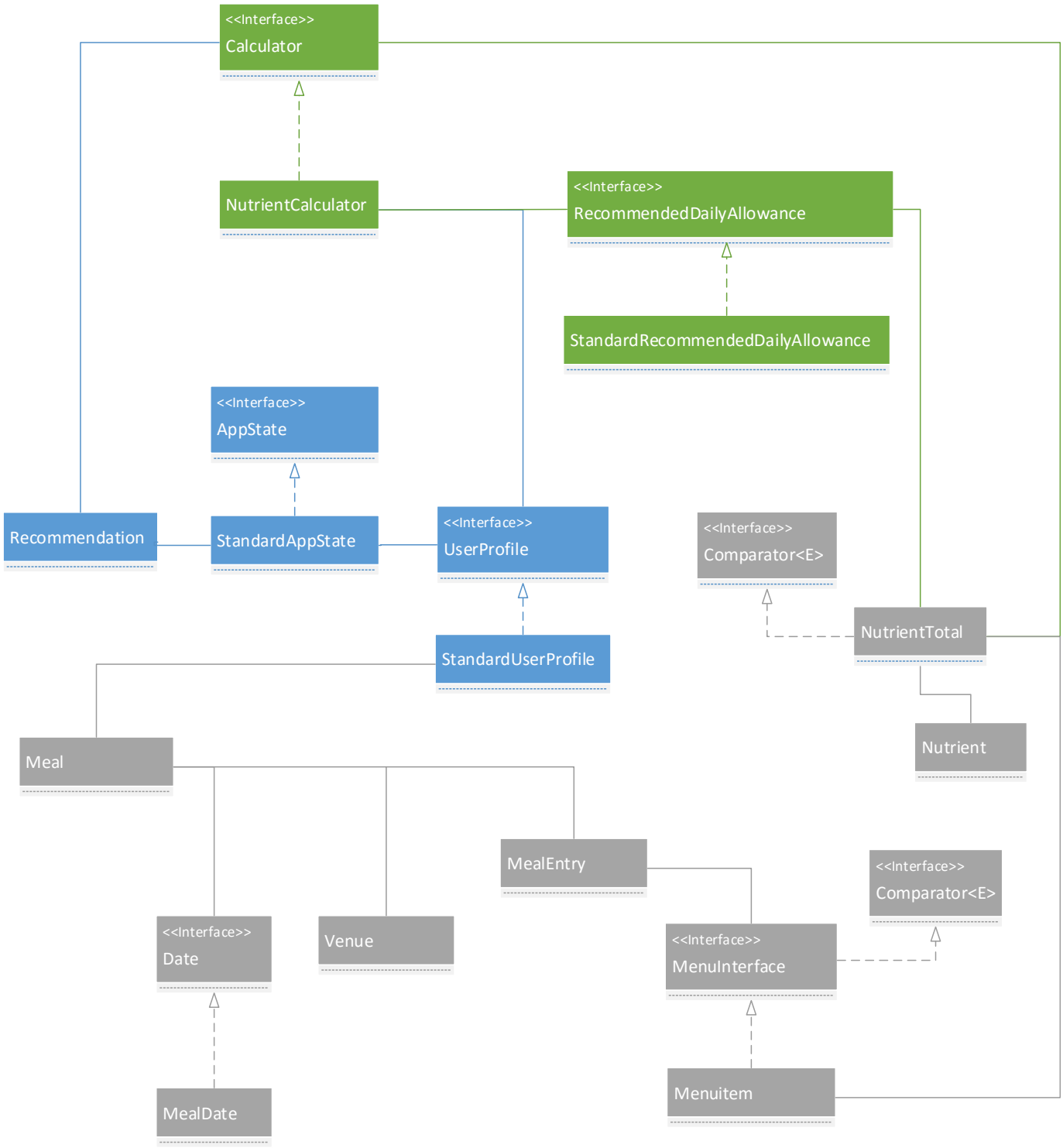
Source: Developer.Android.com



SectionsPagerAdapter class:
The SectionsPageAdapter class extends the FragmentPageAdapter class (an android class available to be imported) and purely keeps track of what page the user is on.

App Engine

The App Engine functions as the heart of the app, and communicates between the server, phone database, and GUI. Shown below is the overall UML diagram of the App Engine, composed of three main groups: calculations, state of the app, and meals. Note that classes that contains objects to be stored in the txt database implement the Serializable interface.



App Engine

Calculator Interface

The Calculator interface provides functionality for all calculations that needed to be performed in order to provided required features for the App GUI.

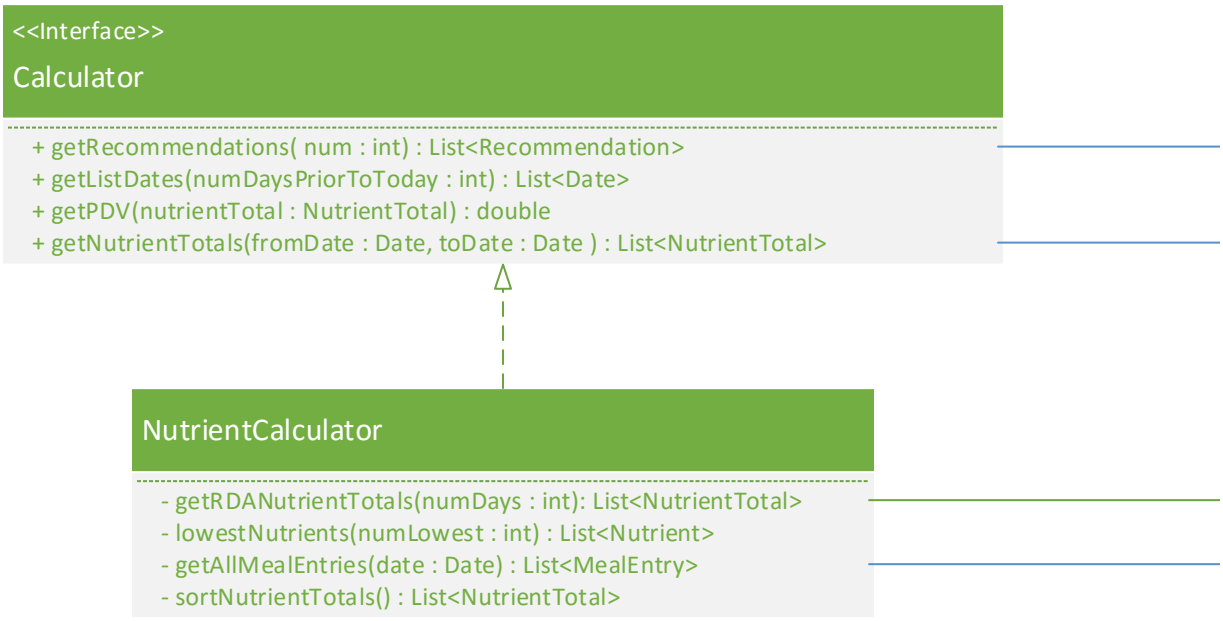
- getRecommendations() method takes an argument that represents how many recommendations the system is required to produce to populate the Recommendations section of the App GUI (Users will have the option to change the default number of recommendations shown from 3 to another value). It also takes an argument of the appState object containing the possible recommendations. The method determines the lowest nutrient percentage (or lowest two, three, etc.) of the user's available nutrient levels, and outputs a list of Recommendations pertaining to those lowest nutrients. For example, if calcium is lowest, the system will pull the Recommendation for Low Calcium and add it to the list.

- getListDates() method takes an argument that represents how many days prior to today to determine and format as a list. This is used in the App GUI for getting a list of dates to display for Recap screen items, and for determining how many days to include in Nutrient Total calculations and Percent Daily Values.

- getPDV() method takes a single NutrientTotal object as an argument, and compares that object against the given Recommended Daily Values tailored to the user. For example, Recommended Daily Values might say the user needs 65 g of protein for the day, and the NutrientTotal for Protein has 55 g. The method outputs the ratio of the two (the percentage of the nutrient the user has eaten for the time period).

- getNutrientTotals() method takes a start date and end date, and a user profile, and adds together all nutrients from all food items the profile has recorded in that period of time. The method outputs the amounts as NutrientTotal objects, in a list.

The NutrientCalculator class implements the Calculator interface.



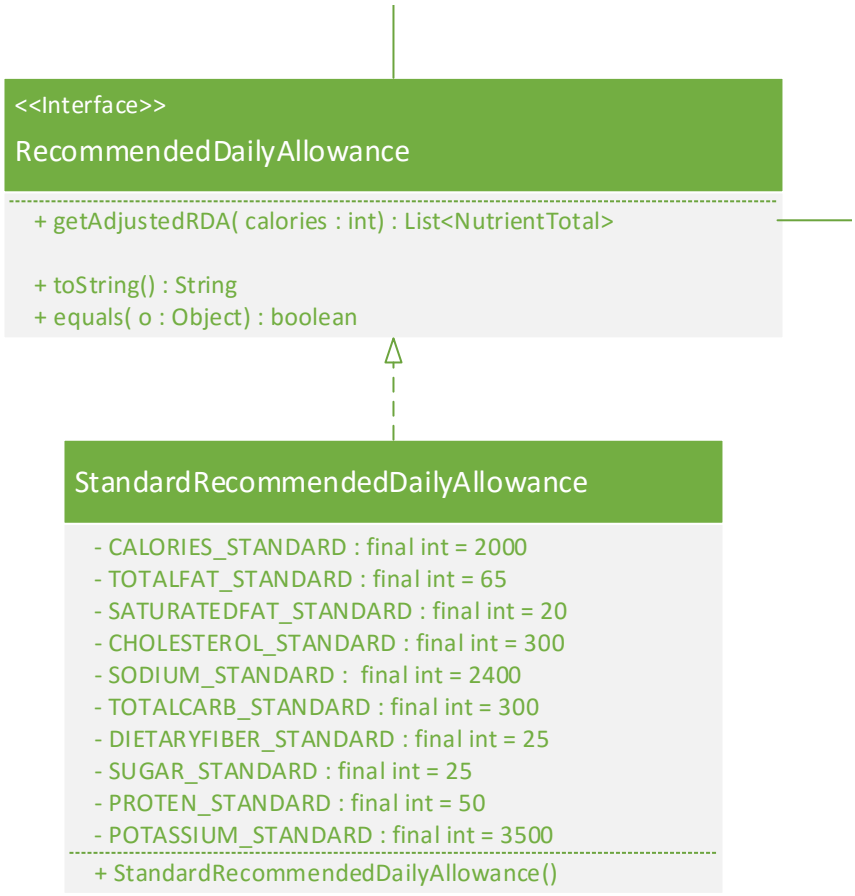
App Engine

RecommendedDailyAllowance Interface
The US Food and Drug Administration (FDA) defines recommended amounts of each nutrient a single individual should consume per day. These Recommended Daily Allowance (RDA) values are based on a 2000-calorie diet.

A main feature of the Student Nutrition Mobile app is the ability to view the Percentage Daily Values (PDVs) for each nutrient. PDVs tell a user what percentage of a single nutrient they have already consumed for the day.In the app, another requirement is to be able to customize the calorie level for an individual. This customization allows an individual to take into account the particular needs of their lifestyle, which may require more nutrients or less, and results in a more accurate calculation of PDVs.

The RecommendedDailyAllowance interface allows the system to adjust the RDA values, given a user's calorie level. It takes an input calorie level and creates a ratio against the 2000 calorie standard to determine how to adjust individual nutrient levels. It also allows a client to get the value of any single nutrient.

StandardRecommendedDaily class
The StandardRecommendedDailyAllowance class implements the RecommendedDailyAllowance interface, and contains the standard FDA values for each nutrient.



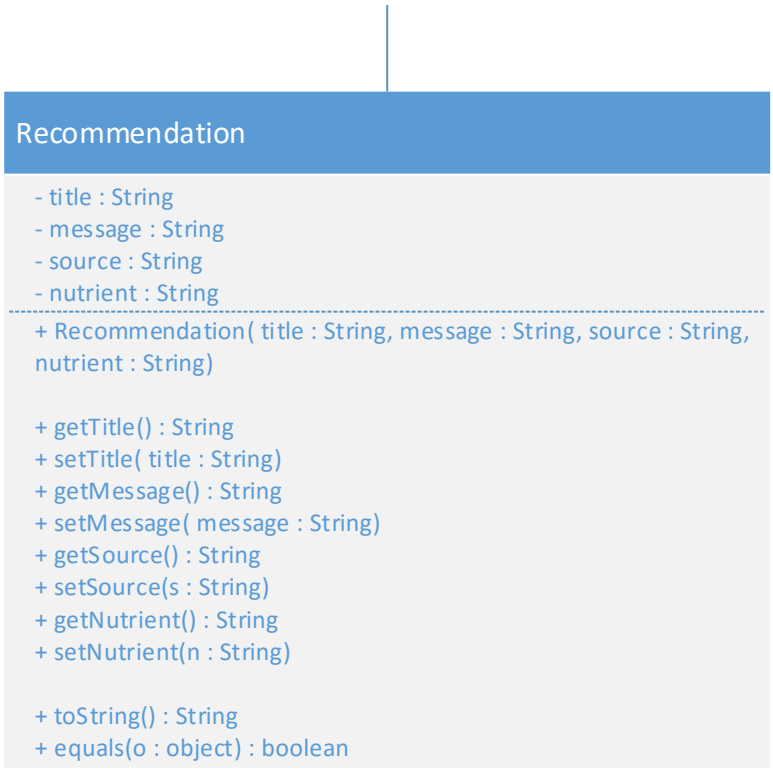
App Engine

Recommendation Class

One feature of the Student Nutrition App is Recommendations. After calculating nutrient amounts and PDVs for a period of time, the App determines whether the user has any nutrient deficiencies. If deficiencies are found, the App has the ability to suggest dietary choices to improve the user's nutrient levels in the future.

The Recommendation class represents the data needed for creating a Recommendation. A single Recommendation object contains the following:

- Title: The subject of the recommendation (e.g. "Low in calcium")
- Message: the body of the message (e.g. "Trying drinking milk, eating cheese, or having a kale salad.")
- Source: references the sources of the nutritional recommendation (e.g. "UDSA")
- Nutrient: defines which nutrient deficiency the recommendation is for



App Engine

AppState interface

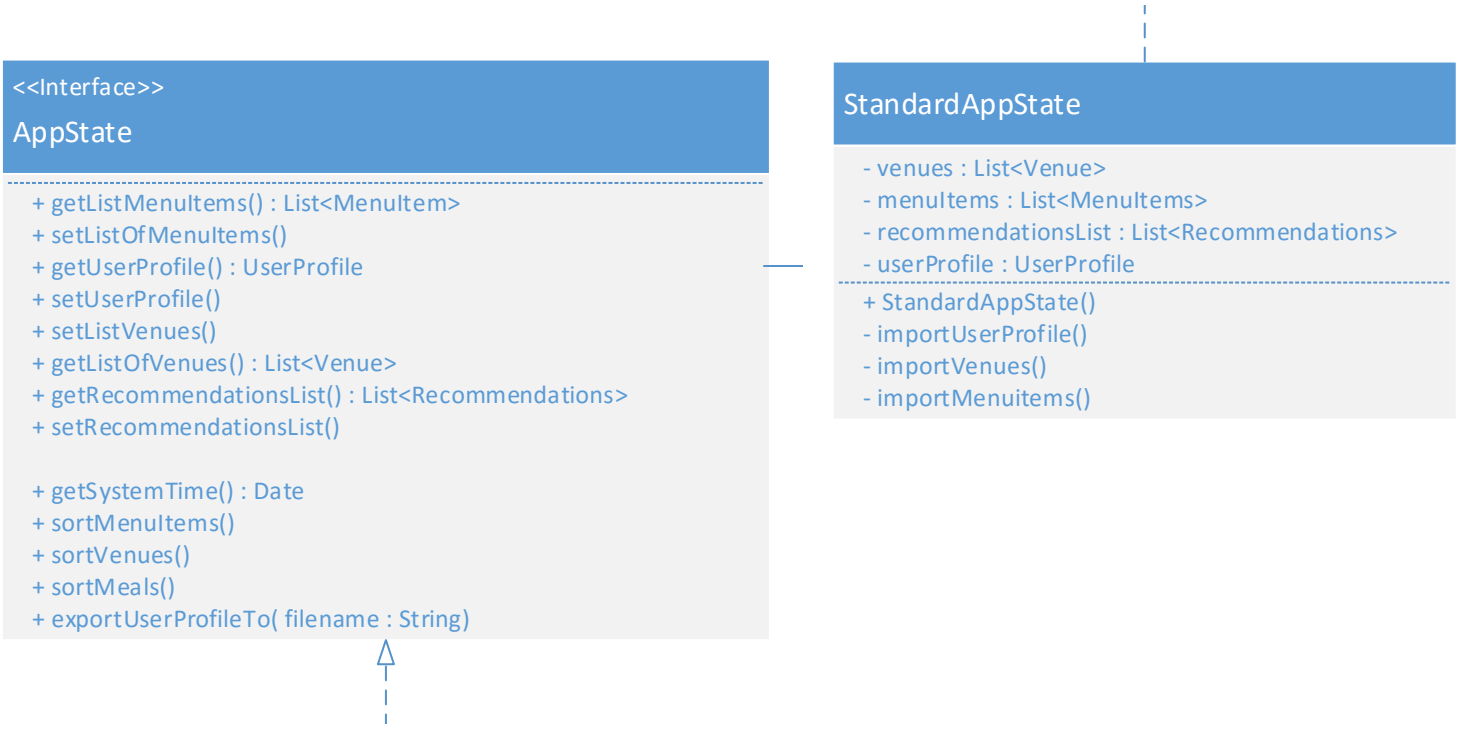
The AppState interface represents a snapshot of all information available in the app during runtime. This information is accessible by the GUI for populating screens. An AppState object includes the following behavior:

- getter and setter for each field
- constructListOfMenuitems : will pull data from the SQL server, based on a particular venue, and parse that data to form a list of menu items
- constructUserProfile() : will pull data from txt file stored on the phone and construct UserProfile object
- constructListOfVenues() : will pull all venue data from the SQL server, and parse that data to form a list of venues
- constructRecommendationList() : will pull all recommendation data from SQL server, and parse that data to form a list of recommendations
- ability to sort menu items: menu items are sorted by category first, then by name alphabetically
- ability to sort venues: venues are sorted alphabetically
- ability to sort meals: meals are sorted by date, then by time
- ability to export user profile information: this is a nice to have feature, so it will be added in the future

StandardAppState class

The StandardAppState class includes the following information:

- Venues: a list of all venues available on the server
- MenuItems: a list of all menu items available from the currently selected venue on the server
- Recomendations: a list of recommendations based on current nutrient deficiencies
- User Profile: an object containing all information a user records in the app



App Engine

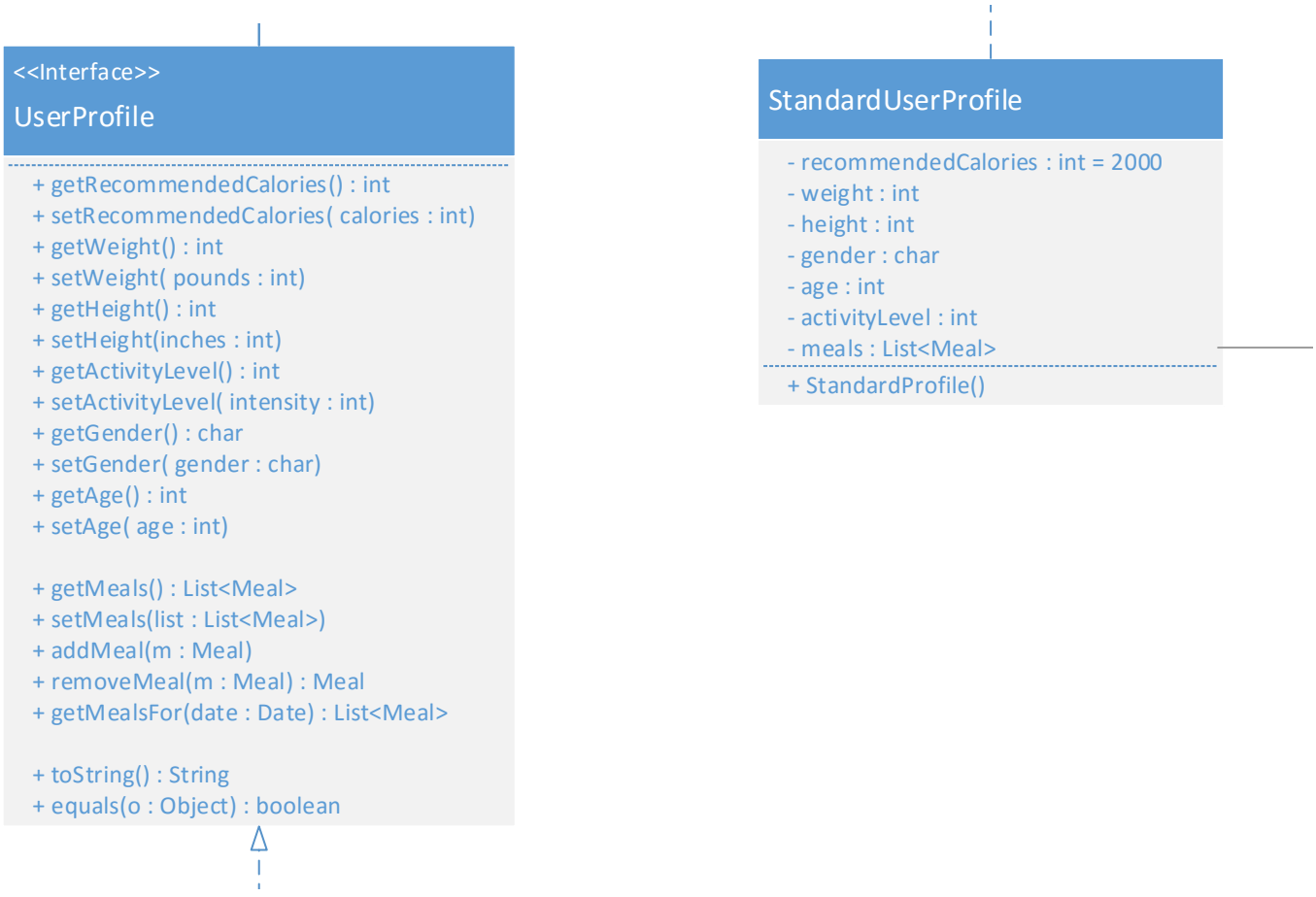
UserProfile interface

The UserProfile interface represents the mechanism for recording and retrieiving information about a user, includings meals and customized daily calorie need. The interface also includes access and modification of information needed to set the calorie level, including gender, age, height, weight, and activity level.

StandardUserProfile class

The StandardUserProfile class represents a single implementation of the UserProfile interface. Its main attributes include the following:

- recommended calories: the customized daily calorie needs for an individual, defaulted to 2000
- weight: user's weight
- height: user's height
- gender: user's gender
- age: user's age
- activity level: a numeric value representing the amount of activity a user undergoes in a typical day. values range from 0, which is low, to 3, which is high.
- meals: this is a list of all meals a user has recorded

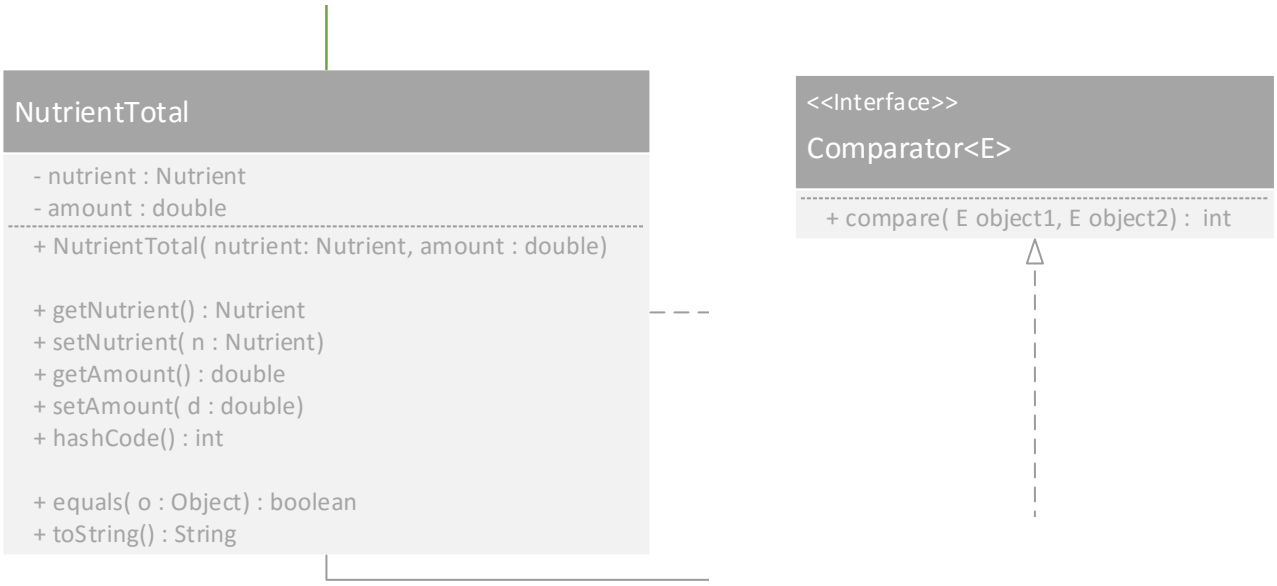


App Engine

NutrientTotal class

The NutrientTotal class represents a pairing of a nutrient and an amount. It allows the program to specify a particular amount of a nutrient.

The class implements the Comparator interface, so that NutrientTotal objects can be compared and sorted.



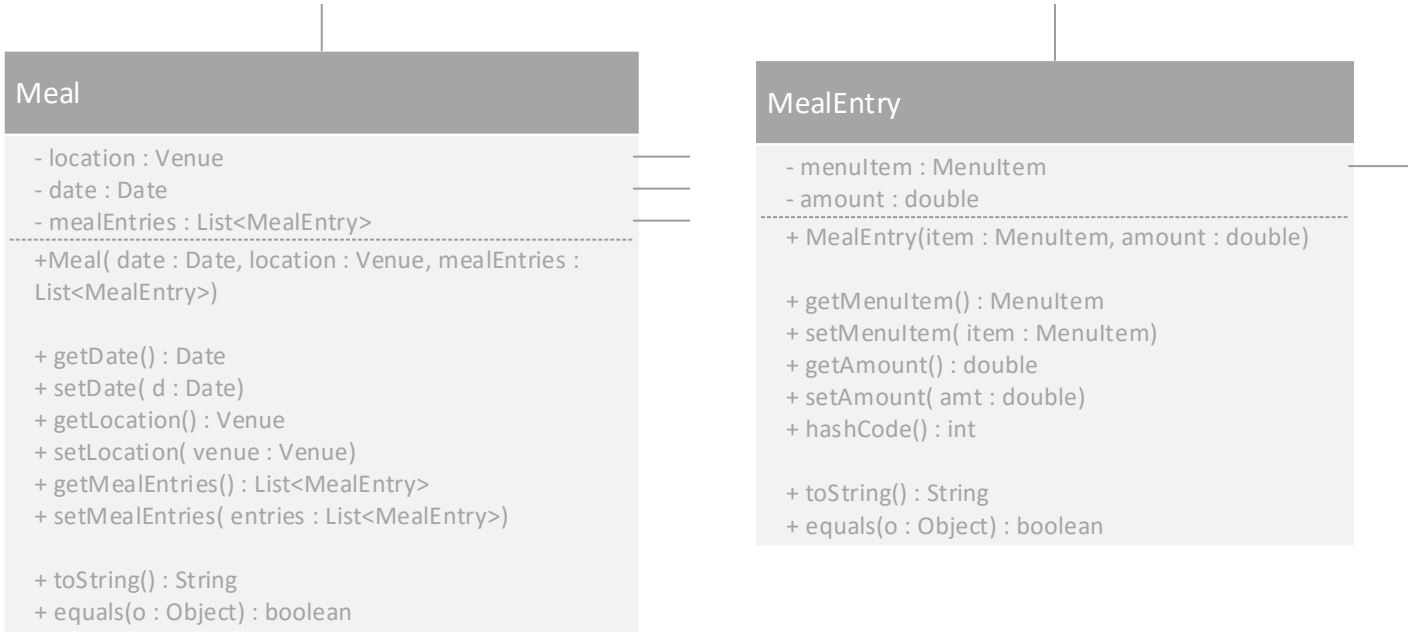
App Engine

Meal class
The Meal class represents a single recorded meal from a user. It includes the following attributes:

- Location:
- Date: a date object representing the time and date of the meal
- list of meal entries: list of meal entry objects

MealEntry class
The MealEntry class represents a pairing of a menu item and and amount of that item. It includes the following attributes:

- menu item: the particular menu item object selected by the user
- amount: a numeric representation of the amount of that menu item the user has recorded.

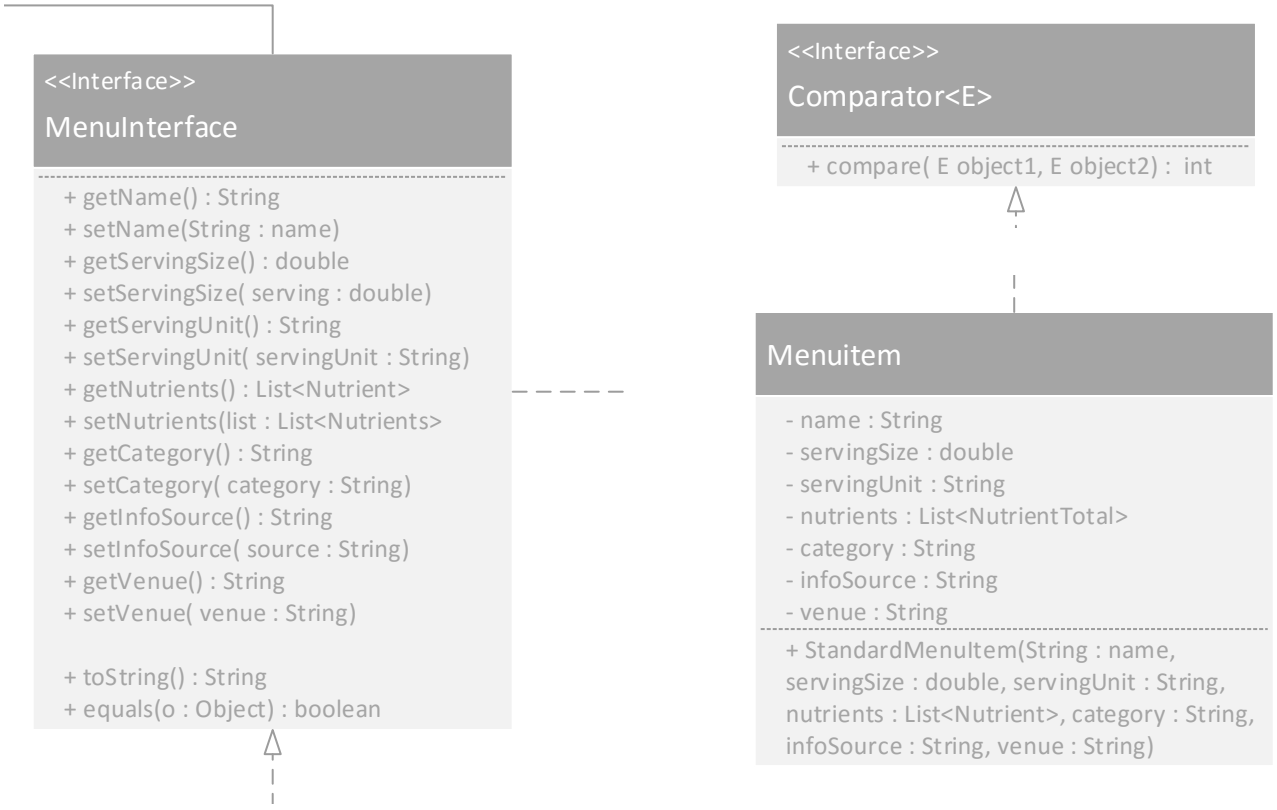


App Engine

MenuInterface interface
The MenuItem interface represents a single food item a user can select to add to a meal. The MenuItem interface defines the mechanisms for modifying and retrieving values for a MenuItem. It includes a getter and setter for each attribute of the StandardMenuItem class.

MenuItem class
MenuItem class represents a single implementation of the MenuItem interface. It also implements the Comparator interface so that menu items can be compared and sorted. This class includes the following attributes:

- Name: represents the name of the menu item
- Serving Size: numerical value representing a typicaly portion of the menu item
- Serving Size Unit: defines the typical unit the menu item is portioned out in (e.g. cups, ounces, pieces)
- Nutrients: contains a list of NutrientTotal objects, which define the amount of each Nutrient object the menu item contains
- Category: represents the menu category the item would typically be found under (e.g. Entree)
- InfoSource: represents the source of nutritional information for the menu item (e.g. USDA Database or particular restaurant website)
- Venue: the location this menu item is available at



App Engine

Nutrient class

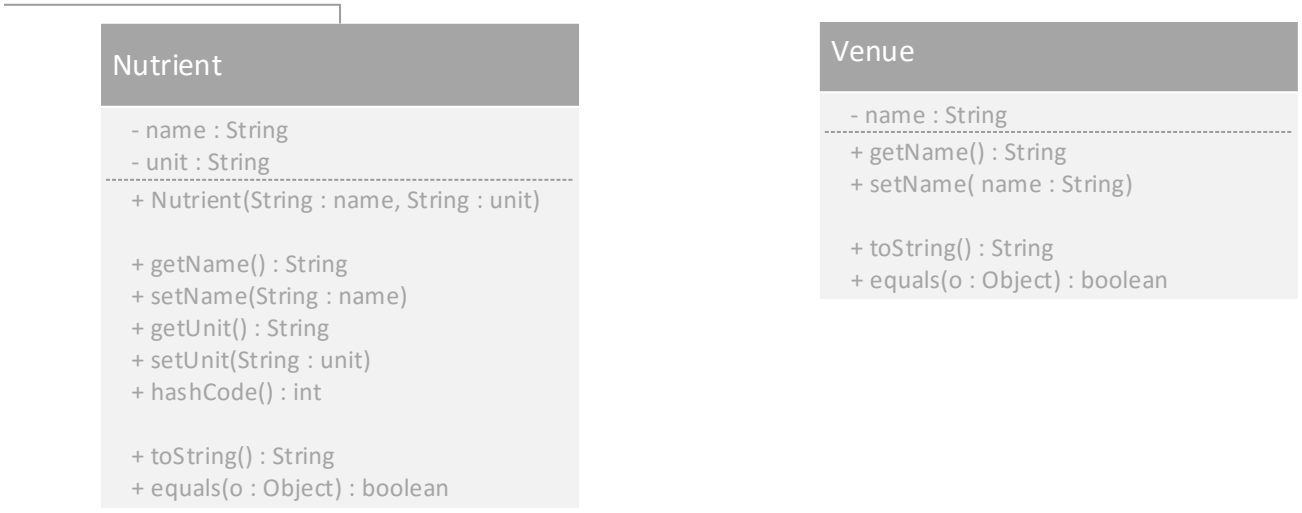
The Nutrient class represents a single food nutrient a menu item can contain. This is meant to standardize each nutrient so that it can be easily incorporated into each menu item object. Each nutrient consists of the following attributes:

- Name: representation of the name of a nutrient (e.g. “Protein”)
- Unit: represents the standard unit of measure the nutrient is measured in (e.g. “grams)

Venue class

The Venue class represents a venue, or location for recording a meal. It has a single field:

- Name: represents the name of the venue (e.g. “Panda Express”)

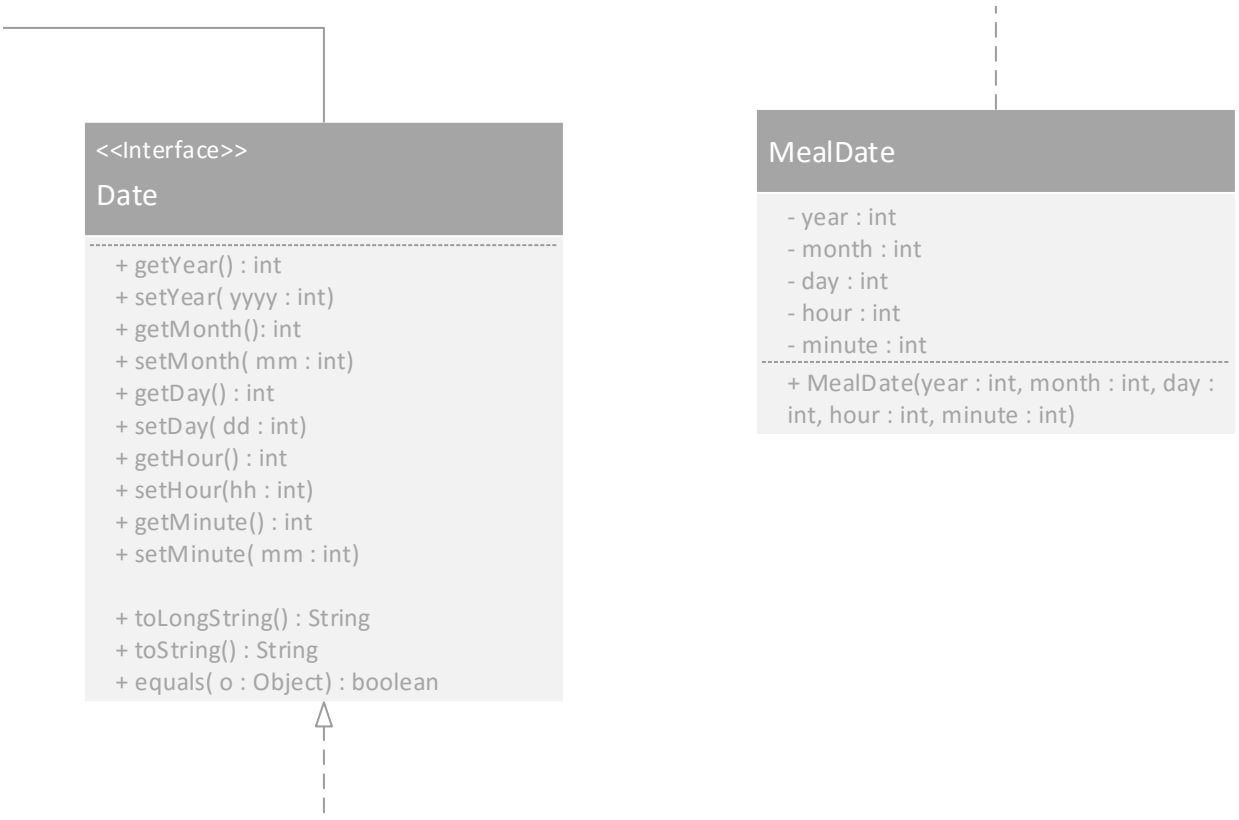


App Engine

Date interface
The Date interface defines the mechanism for setting and retrieving the values of Data objects. It includes a getter and setter for each particular field a Date object contains.

MealDate class
MealDate class is a single implementation of the Date interface. It represents a particular instance in time, as defined by the following attributes:

- Year: numerical value representing a calendar year
- Month: numerical value representing a standard month in a 12 month calendar
- Day: numerical value representing the particular day of the month
- Hour: numerical value representing an hour of the day
- Minute: numerical value representing a minute of the day



MySQL Server

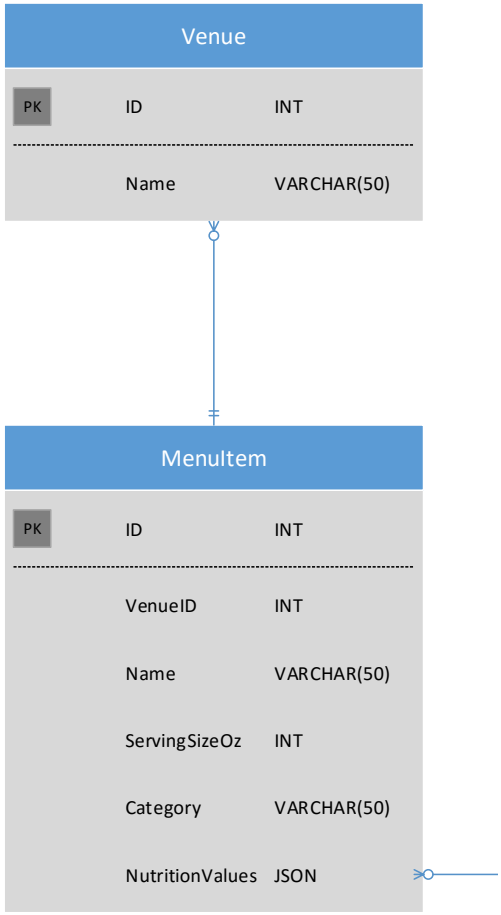
The MySQL server module of the project is composed of Data and Operations components. The Data component is structured in multiple tables, and the schema of these tables is defined below.

Venue table:
This table contains the attributes for each venue stored in the database. Attributes include an ID and a Name. The ID is the primary key of the table.

MenuItem table:
This table contains the attributes for each food item stored in the database. Attributes include the following: ID, VenueID, Name, ServingSizeOz, Category, and Nutrition Values. A few of the attributes are clarified below:

- ID is the primary key of each MenuItem.
- VenueID relates each MenuItem to a particular venue.
- ServingSizeOz allows for the storage of a numerical value describing the serving size of a particular menu item. In this case, the serving size is measured in ounces.
- Category describes the type of food item. Categories include Entree, Side, Dessert, Drinks, and more.
- NutrientValues is in JSON format.

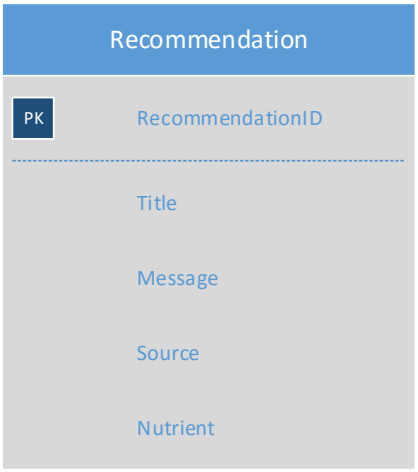
Each MenuItem is associated with a single Venue.



MySQL Server

Recommendations table:
This table contains the attributes for each recommendation stored in the database. Attributes include the follinwg: ID, Title, Message, Source, Nutrient.

- ID is the primary key of each Recommendation
- Title is the text string displayed as the title of the recommendation (e.g. “Low in Iron”)
- Message is the text string displaying instructions of how to increase the nutrient through eating (e.g. “Try eating beans, whole grains, dark leafy greens, or red meat.”)
- Source is the text string referencing on what the recommendation information is based (e.g. “USDA”).
- Nutrient is the text representation of the particular nutrient this recommendation is valid for (e.g. “Protein”)



MySQL Server

Server Operations
The server contains all venue and food item information. Each venue is stored with a name and ID,. Each food item is related to a particular venue, and contains all nutrient information, as well as a name of the item. This data is accessible through the server API the team has created.

The Server API is not a class, but a diagram that displays the methods the server contains. The API has its own documentation (see API Documentation), but it is important to explain the format of the information retrieved.

The following are requests the server can respond to:

/venues
Returns a collection of venues with ID and name. The server responses will be in JSON format. The format as shown over HTTP is the following:

Example:
{“count”:2,”venues”:[{“id”:1,”name”:”Panda Express”},{“id”:2,”name”:”Pizza Hut”}]}

/venues/<ID Number>/items
Returns a collection of meals belonging to the venue with the specified ID. The server responses will be in JSON format. Count is the number of meal items in the colletion. The following is a sample of the HTTP output:

Example:
{“count”:46,”meals”:[{“id”:1,”name”:”Chow Mein”,”servingsizeoz”:9.4,
”nutritionvalues”:{“fiber”:{“unit”:”g”,”count”:4},”sugar”:{“unit”:”g”,”count”:9},”sodium”:{“unit”:”mg”,
”count”:980},”protein”:{“unit”:”g”,”count”:13},”Calories”:510,”totalFat”:{“unit”:”g”,”count”:22},
”transFat”:{“unit”:”g”,”count”:0},”cholestrol”:{“unit”:”mg”,”count”:0},”saturatedFat”:{“unit”:”g”,
”count”:4},”carbohydrates”:{“unit”:”g”,”count”:65},”CaloriesFromFat”:200}},<...>]}

As stated in the API Documentation, calls to the API are directed to the base API address:
http://rin.cs.ndsu.nodak.edu:4567/

MySQL Server

Server Operations
The easiest way to access information from the server in Java is to use gson. gson is a Java serialization/ deserialization library that can convert Java Objects into JSON and back. gson is google's own goto json library.

Here is a sample of what the code will look like:

```
//just a string (grab all meals from pex)
String sURL = "http://rin.cs.ndsu.nodak.edu:4567/venues/1/meals";

// Connect to the URL using java's native library
URL url = new URL(sURL);
HttpURLConnection request = (HttpURLConnection) url.openConnection();
request.connect();

// Convert to a JSON object to print data
JsonParser jp = new JsonParser(); //from gson

//Convert the input stream to a json element
JsonElement root = jp.parse(new InputStreamReader((InputStream) request.getContent()));
JsonObject rootobj = root.getAsJsonObject();          //May be an array, may be an object.
zipcode = rootobj.get("count").getAsString();         //just grab the count of meals from pex
```

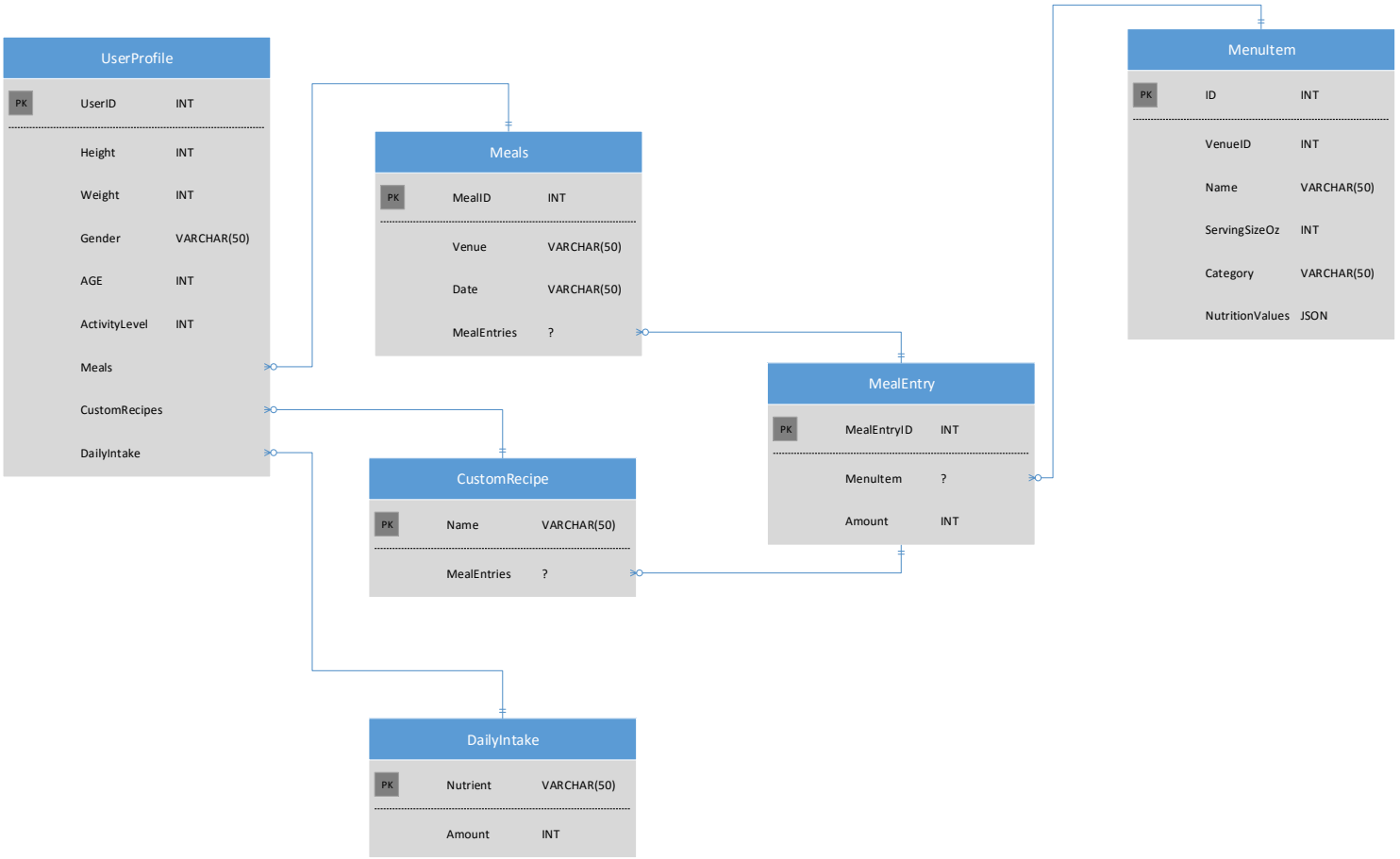
The link to more information: https://code.google.com/p/google-gson/

Phone Database

The Phone Database module is composed of Data and Operations components. The Data component is structured in multiple tables, and the schema of these tables is defined on the following pages.

The Data component contains the following tables:

- UserProfile
- Meals
- CustomRecipe (to be added later)
- DailyIntake (to be added later)
- MealEntry
- MenuItem



Phone Database

The Phone Database module is composed of Data and Operations components. The first draft of the database operations are listed below. (The SQLlite database component will be added in the future.)

- onCreate() : creates the database SQLlite txt file on the mobile device
- onUpgrade() : updates the database to a new version

- deleteVenue() : deletes a venue
- addVenue() : adds a venue
- getVenue() : returns a venue
- getAllVenues() : returns all venues

- deleteUserProfile()
- addUserProfile()
- getUserProfile()

- getMeal()
- addMeal()
- deleteMeal()
- getAllMeals()

- getCustomRecipe()
- addCustomRecipe()
- deleteCustomRecipe()
- getAllCustomRecipes()

- getMealEntry()
- addMealEntry()
- deleteMealEntry()
- getAllMealEntries()

- getMenuItem()
- addMenuItem()
- deleteMenuItem()
- getAllMenuItems()

Note:
The code that has been created so far is contained within the App repository in the project itself.