Bilkent University

Department of Computer Engineering

Senior Design Project

Foodster: Maintain your diet easier

Low Level Design Report

Khasmamad Shabanovi, Gledis Zeneli, Balaj Saleem, Ibrahim Elmas, Perman Atayev

Supervisor: Ozcan Ozturk

Contents

1. Introduction

Food is the first of all fundamental human needs, yet how many times today have you, personally, consciously thought of what you have and will be eating today, what its nutritional value is and how it fits into your greater nutritional, fitness, and lifestyle goals? With the ease of falling into a routine and the abundance of staple food, keeping track of all these variables, and making sure that you plan your meals such that your priorities are heeded, becomes increasingly mundane, monotonous, and just unnecessary extra work.

With the fast-paced lifestyle of the 21st century, monotony and unnecessary work are the last things an individual needs in his/her busy life. Figuring out what to eat, to order or to cook, how to cook it, where to get the ingredients from, and how beneficial this meal would be for your body are questions that would take precious time and energy that could be better employed elsewhere.

Furthermore, the complications of searching for the nutritional data, planning a healthy diet according to one's needs, planning the budget for such a diet, and finding recipes to support this time is truly a cumbersome endeavor.

This is where Foodster comes in.

1.1 Object design trade-offs

In this section performance, security, maintainability, supportability, reliability and scalability of the Foodster application is discussed. Also, which trade-offs are and will be made while developing the application are explained.

1.1.1 Performance vs Security

- Performance and Security are very important for Foodster, because both of those contribute a lot to the User Experience and the reliability of our services.
- Security is very important when it comes to users' passwords and payment information; however, other information like the diet types they have or the meals that a user eats are not as sensitive. Therefore, for the security related operations for hashing passwords and encrypting the card information the CPU and RAM will be provided as much as needed. However, for less sensitive information less security measures might be taken to reduce the load of our servers in favor of performance of the recommendation or scheduling meals algorithms.

1.1.2 Maintainability vs Supportability

- Maintainability of the Foodster is more important than supportability of it because we are dedicated to provide the best experience to our users for the platforms we support and we will make sure that their experience will not get worse as we expand to different Operating Systems and platforms. For now, we decided that our application will be focused on providing services for Android and iOS users. Therefore, until we build a platform that provides services to our users without any major frictions and / problems, we will not expand to Web or Desktop applications.

1.1.3 Reliability vs Scalability

- Both Reliability and Scalability are essential for a good business / application. For Foodster, reliability of our systems is more important than scalability, because we do not want our users to lose their data as we expand. Any data that is saved to our system should not get lost for any reason. Therefore, we will do frequent backups for our databases and also to make sure that a crash of our main server does not kill our ability to provide services. We will have a backup idle server that will be replaced by the main server if the main server crashes.

1.2 Interface documentation guidelines

All the class names are named with PascalCase, where the first letter of every word in the identifier is upper case. On the other hand, camelCasing is used in naming attributes and methods, where the first letter of every word in the identifier is upper case except the first word. The following table displays the template we adopted for interface documentation in this report.

ClassName
ClassName is a class responsible for...
Attributes

private int firstAttr	
public String secondAttr	

Methods	
public void setFirstAttr(int val)	Sets the firstAttr attribute
public int getFirstAttr()	Returns the first attribute

1.3 Engineering standards

In order to describe the class interfaces, the diagrams, scenarios, use cases, subsystem compositions and hardware depictions of the Foodster application, the UML guidelines are used. The UML is a very important part of developing object oriented software and the software development process.

1.4 Definitions, acronyms, and abbreviations

API - Application programming interface

DB - Database

NoSQL - Non relational database

HTTP - Hypertext transfer protocol

Cross Platform application - An application that is supported in different platforms such as Android and iOS

UI - User interface

UX - User experience

OS - Operating system

2. Packages

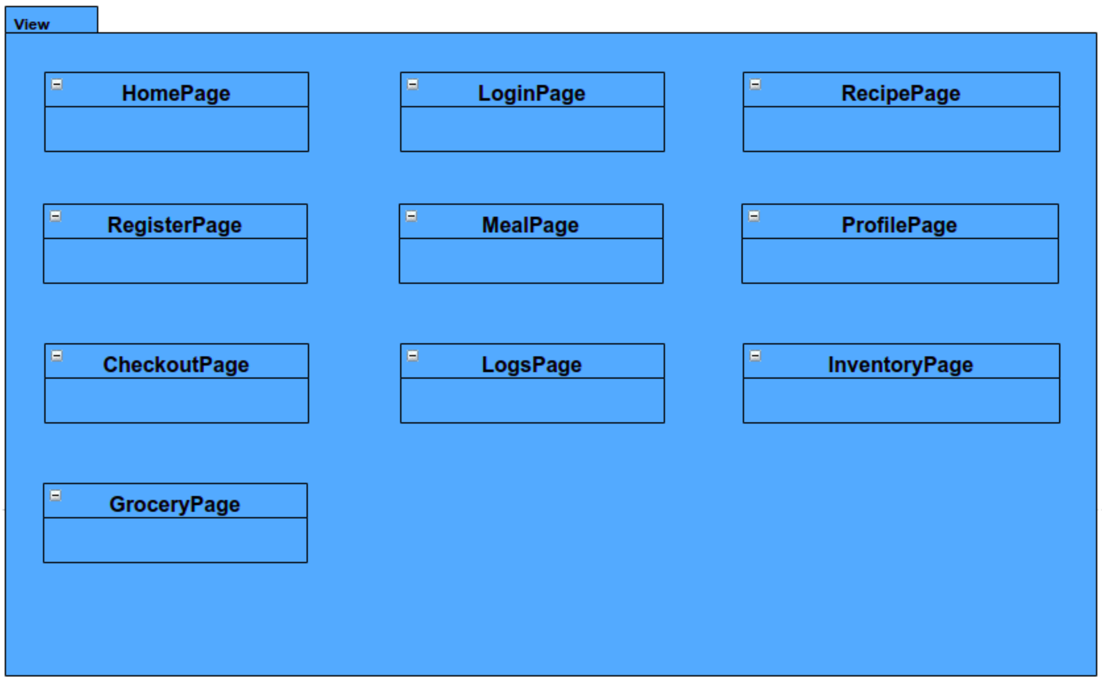
Foodster’s subsystems are grouped under Client and Server packages. Client package encapsulates three components, namely View, Controller, and Model. Server package, on the other hand, consists of Route, Logic, and Data tiers. The client side is responsible for fetching information from the server side, presenting it to the user, and updating the information based on the user input. While doing so, the client side sends HTTP requests to the server side. Server side waits for requests and handles them as they come. It does most of the heavy work and responds back with the results.

2.1 Client

Client side of our application has three components: view, controller, and model. This section presents detailed information on these components.

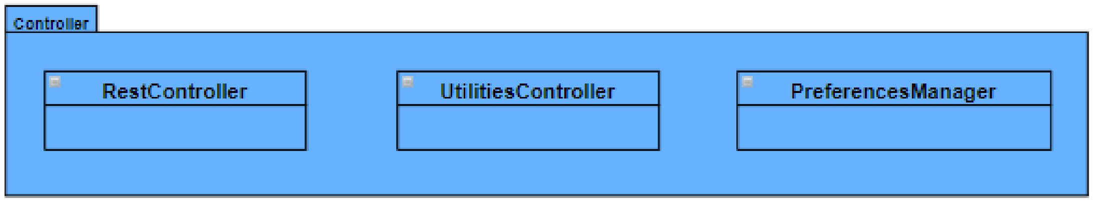
2.1.1 View

The View component of the client consists of all the classes which directly contribute to the UI.



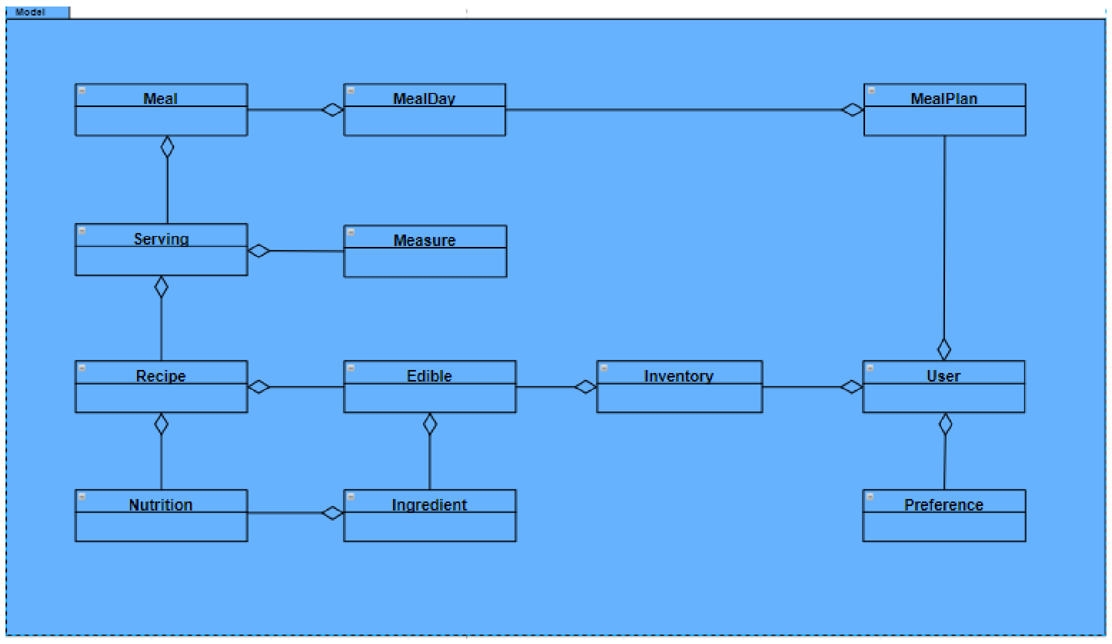
2.1.2 Controller

The Controller component of the Client is composed of classes which regulate communication between the View and Model of the Client, but also between the Client and the Server.



2.1.3 Model

The Model component of the client consists of all the classes that store information regarding the real life entities of the dieting process.

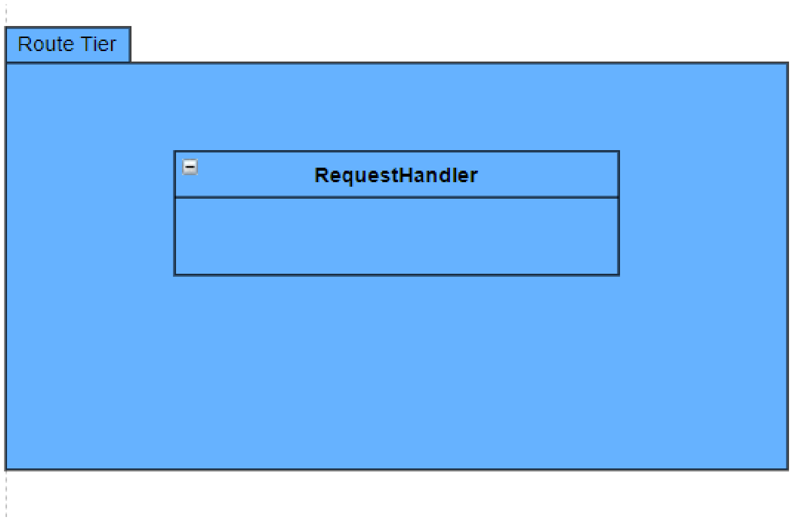


2.2 Server

Server side of our application has three components: route tier, logic tier, and data tier. This section presents detailed information on these components.

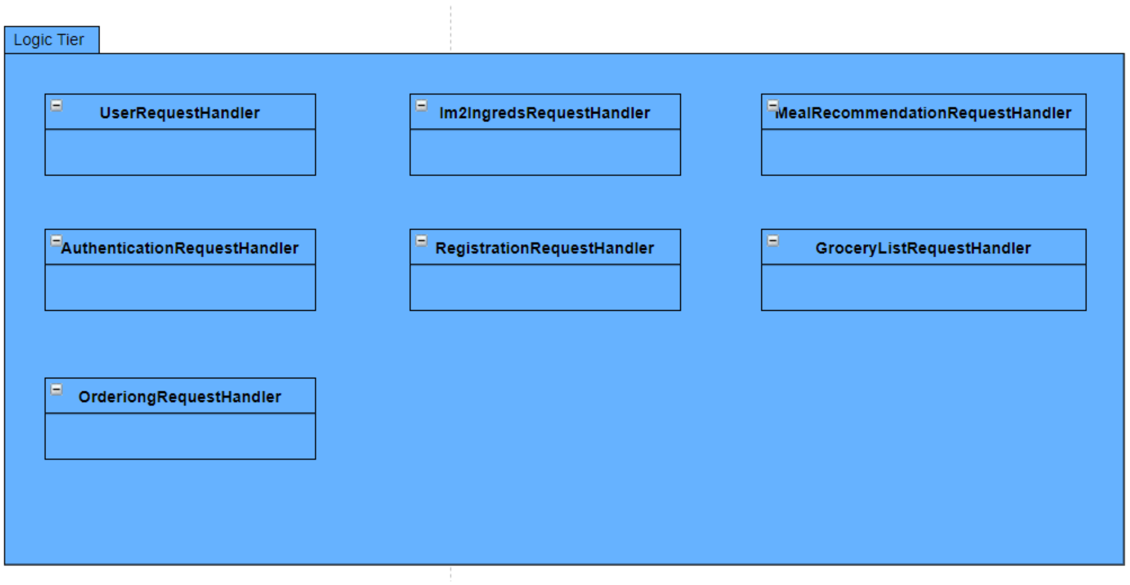
2.2.1 Route Tier

Route tier is the component that receives all the requests from clients and delegates them to the respective logic tier components.



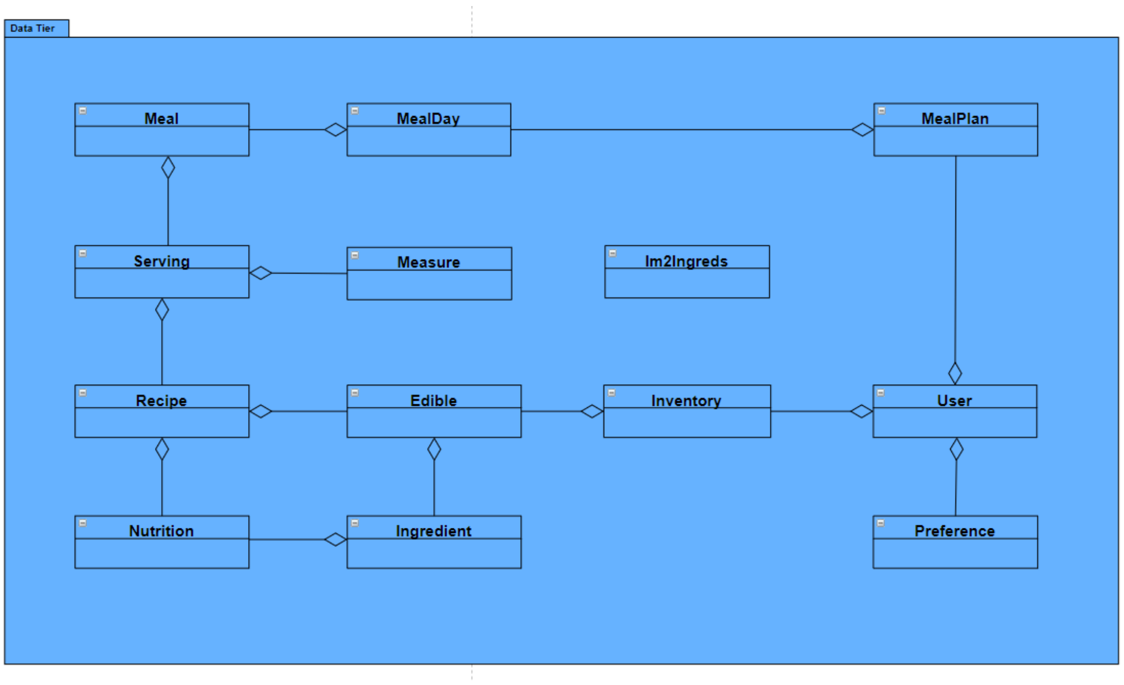
2.2.2 Logic Tier

Logic tier components are each responsible for a specific task. For example, OrderingRequestHandler, as the name suggests, deals with the requests regarding ordering food from third-party services.


Logic Tier
UserRequestHandler
Im2IngredsRequestHandler
MealRecommendationRequestHandler
AuthenticationRequestHandler
RegistrationRequestHandler
GroceryListRequestHandler
OrderiongRequestHandler

2.2.3 Data Tier

Lastly, data tier is where all the server side model classes are. This component is similar to the model component of the client side, except that it includes one additional class - Im2Ingreds.



3. Class Interfaces

In this section, class methods and parameters together with their explanations are provided. Subsections of this section are structured similar to the previous section.

3.1 Client

In this section, the classes grouped under the three components of the client package are presented.

3.1.1 View

HomePage
HomePage is class that contains the widget the displays the landing page for logged in user
Attributes
private int selectedNavIndex
Methods

public void initState()	Initializes the (state) variables of the class
public void build(BuildContext context)	Rebuilds the widgets / UI classes whenever the state is updated in the current context.
public void buildAppBar(BuildContext context))	Builds the app bar which contains the header
public void buildBottomNavigationBar(BuildContext context))	Builds the bottom navigation bar to move to different tabs
public void onNavItemTapped()	performs an action when the bottom navigation bar item is tapped

RegisterPage
RegisterPage is class / widget the displays the sign up page for a user
Attributes
private String email
private String password
private String gender
Methods

public void initState()	Initializes the (state) variables of the class
public void build(BuildContext context)	Rebuilds the widgets / UI classes whenever the state is updated in the current context.
public void handleRegister(BuildContext context))	Validates and handles registration functionality.

LoginPage

Login is class / widget the displays the sign in page for a user

Attributes

private String email
private String password
private boolean isLoading

Methods

public void initState()	Initializes the (state) variables of the class
public void build(BuildContext context)	Rebuilds the widgets / UI classes whenever the state is updated in the current context.

MealPage

MealPage is class that contains the widget the displays the meal plan page to a logged in user

Attributes

private MealPlan mealPlan

Methods

public void initState()	Initializes the (state) variables of the class

public void build(BuildContext context)	Rebuilds the widgets / UI classes whenever the state is updated in the current context.
public void buildDatePicker(BuildContext context))	Builds the datepicker to select a date for meals
private void handleMealGeneration()	Handles the fetching of mealPlan

RecipePage

RecipePage is class that contains the widget the displays the meal plan page to a logged in user

Attributes

private Recipe recipe

Methods

public void initState()	Initializes the (state) variables of the class
public void build(BuildContext context)	Rebuilds the widgets / UI classes whenever the state is updated in the current context.
public void buildHeader()	Builds the header for recipe page
private void buildIngredients()	Builds the ingredients list for the recipe

ProfilePage

ProfilePage is class that contains the widget the displays the user profile to a logged in user

Attributes

private User user
private Preferences preferences

Methods

public void initState()	Initializes the (state) variables of the class
public void build(BuildContext context)	Rebuilds the widgets / UI classes whenever the state is updated in the current context.
public void buildHeader()	Builds the header for the user details
public void buildUserDetails()	Builds the user details section of the page
public void buildPreferences()	Builds the user preferences section of the page

CheckoutPage

Checkout is class that contains the widget the displays the checkout form to a logged in user

Attributes

private String cardNo
private String name
private String cvv
private String paymentType

Methods

public void initState()	Initializes the (state) variables of the class
public void build(BuildContext context)	Rebuilds the widgets / UI classes whenever the state is updated in the current context.
public void buildForm()	Builds the card details form for the user to fill
private void buildSuccessPopup()	Builds the success popup if the transaction is accepted.

LogsPage

Logs is class that contains the widget the displays the meal logging and trends logged in user

Attributes

private Nutrition nutrition private Trends trends

Methods

public void initState()	Initializes the (state) variables of the class
public void build(BuildContext context)	Rebuilds the widgets / UI classes whenever the state is updated in the current context.
public void buildForm()	Builds the nutrition details form for the user to fill
private void buildTrends()	Builds the past trends data in form of graphs

InventoryPage

InventoryPage is class / widget the displays the inventory of ingredients for a logged in user

Attributes

private Inventory inventory

Methods

public void initState()	Initializes the (state) variables of the class
public void build(BuildContext context)	Rebuilds the widgets / UI classes whenever the state is updated in the current context.

GroceryPage

Grocery Page is class / widget the displays the grocery page for a user

Attributes

private List<String> vendorList

Methods	
public void initState()	Initializes the (state) variables of the class
public void build(BuildContext context)	Rebuilds the widgets / UI classes whenever the state is updated in the current context.

3.1.1 Controller

RestController
This controller is responsible for Restful API interactions of the mobile app
Attributes
private String baseUrl

Methods	
public static String token singup(User user)	Handles the restful api call to signup
public static String token login(String email, String password)	Handles the restful api call to log in
public static MealPlan getMeals(String token)	Handles the restful api call to get meals for a user
public static Trends getTrends(String token)	Handles the restful api call to get past trends
public static User getUserDetails(String token)	Handles the restful api call to get all user details
public static Preferences getPreferences(String token)	Handles the restful api call to get user preferences
public static User updateUserDetails(String token, User user)	Handles the restful api call to update user details
public static Recipe getRecipe(String token)	Handles the restful api call to get a unique recipe

public static Inventory getInventory(String token)	Handles the restful api call to get the Inventory details
public static List<String> getVendors(String token, String location)	Handles the restful api call to get Vendors List
public static Inventory postInventory(String token,, Inventory inventory)	Handles the restful api call to update the inventory
public static boolean processPayment(String token)	Handles the restful api call to process payment
public static boolean logMeals(String token, Nutrition nutrition)	Handles the restful api call to log meals

UtilitiesController

This class contains basic warning / message utilities for the app

Attributes

Methods

public void showToast(BuildContext context)	Shows a toast for a specific UI context
public void showSnackbar(BuildContext context)	Shows a snackbar for a specific UI context
public String uppercaseText(String s)	Converts a string to uppercase

PreferencesManager

This class handles shared preferences for a session

Attributes

private String token

Methods

public String initialize()	Initializes the manager to store shared data
public boolean storeToken(String token)	Stores a token for the user in session
public String getTokent()	Gets the token for the user in session
public boolean removeToken()	Removes the token for the user in session

3.1.1 Model

Recipe
This class is responsible for holding all the relevant information of a recipe.
Attributes

private String Name
private ENUM difficulty
private int prepTime
private int cookTime
private String imgUrl
private Array<String> instructions
private Nutrition nutrition
private double estimatedPrice
private Array<Edible> ingredients

Methods

public static Recipe fromJSON(String json)	Returns a Recipe object from a json string.
public static String toJSON(Recipe recipe)	Returns the json representation of a Recipe object
public Nutrition getScaledNutrition(Measure serving)	Returns a Nutrition object holding data for an amount of serving of that recipe.
public double getScaledPrice(Measure serving)	Returns the price for the amount of servings for the recipe object.
public Array<Edible> getScaledIngredients(Measure serving)	Returns the ingredients required to cook an amount of servings for the recipe.

getters and setters	

Ingredient
Ingredient is responsible for holding information relevant to a real life cooking ingredient.
Attributes

private String name
private String imgUrl
private Nutrition nutrition
private double estimatedPrice

Methods

public static Ingredient fromJSON(String json)	Returns an Ingredient object from a string json.
public static String toJSON(Ingredient ingr)	Returns the json representation of an Ingredient object.
public Nutrition getScaledNutrition(Measure measure)	Returns the nutritions of the Ingredient object for a measure as a Nutrition object.
public double getScaledPrice(Measure measure)	Returns the price of a measure of the ingredient.
getters and setters	

Measure
Measure is a class responsible for unit management. Different ingredients and food components are measured in different units and this class handles all the relevant conversion and information storage.
Attributes

private static Map<String, double> conversionTable
private double magnitude
private String unit

Methods

constructor(double mag, String unit)	Constructs a Measure object with given magnitude and unit.
public void convert(String newUnit)	Converts the objects unit to a new unit
getters and setters	

Serving

Serving is a class which holds information about a recipe and what amount of that recipe is cooked.

Attributes

private Recipe recipe
private Measure measure

Methods

public static Serving fromJSON(String json)	Returns a Serving object from its json representation.
public static String toJSON(Serving serving)	Returns the json representation as a string of a Serving object.

Edible

Edible is a class which holds information about an ingredient and what amount of that ingredient is used.

Attributes

private Ingredient ingredient
private Measure measure

Methods

public static Edible fromJSON(String json)	Returns an Edible object from its json representation.
public static String toJSON(Edible edible)	Returns the json representation as a string of an Edible object.

Meal
Meal is a collection of Serving objects which would represent a multi dish meal in real life.
Attributes

private String label
private Array<Serving> servings

Methods

public static Meal fromJSON(String)	Returns a Meal object from its json representation.
public static String toJSON(Meal meal)	Returns the json representation as a String of a Meal object.

MealDay
MealDay is a collection of all the meals planned for consumption on a particular day.
Attributes

private Date date
private Array<Meal> meals

Methods

public static MealDay fromJSON(String)	Returns a MealDay object from its json representation.
public static String toJSON(MealDay mealDay)	Returns the json representation as a String of a MealDay object.

MealPlan
MealPlan is a collection of all the daily meal plans, planned for consumption on a multi day time duration.
Attributes

private int duration
private Date startDate
private Date endDate
private Array<MealDay> plan

Methods

public static MealPlan fromJSON(String)	Returns a MealPlan object from its json representation.
public static String toJSON(MealPlan mealPlan)	Returns the json representation as a String of a MealPlan object.
public MealDay getMealDay(Date date)	Returns the planned MealDay object for a certain date.
public void setMealDay(MealDay mealDay, Date date)	Set the planned MealDay object for a certain date.
getters and setters	

Inventory
Inventory is responsible for storing an inventory of ingredients and their measures.
Attributes

private Array<Edible> inventory

Methods

public static Inventory fromJSON(String)	Returns an Inventory object from its json representation.

public static String toJSON(Inventory inventory)	Returns the json representation as a String of a Inventory object.
public static Inventory fromMealPlan(MealPlan plan)	Returns an Inventory with all the ingredients required to cook a MealPlan
public Inventory getSubInventory(Array<boolean> mask)	Returns a subset of the inventory. Specifically, it returns an inventory with all the ingredients whose indexes have a value 'true' in the mask parameter.

User

The user class holds relevant information needed to produce recommendations for a user, and some user account related information.

Attributes

private String username
private String email
private ENUM gender
private double height
private double weight
private String profileImage
private Array<Ingredient> allergies
private Array<Preference> preferences
private Array<Recipe> likedRecipes
private Array<Recipe> dislikedRecipes
private Array<Ingredient> likedIngredients
private Array<Ingredient> dislikedIngredient

Methods

public static User fromJSON(String)	Returns a User object from its json representation.
public static String toJSON(User user)	Returns the json representation as a String of a User object.
public void addPreference(Preference newPreference)	Adds a new Preference to the list of user preferences

public void rmPreference(Preference preference)	Removes a Preferences from the list of user preferences.
public void addAllergy(Ingredient ingr)	Adds an ingredient to the list of user allergies.
public void rmAllergy(Ingredient ingr)	Removes an ingredient from the list of user allergies.
public void addLikeRecipe(Recipe likedRecipe)	Adds a recipe to the list of liked recipes.
public void rmLikedRecipe(Recipe recipe)	Removes a recipe from the list of liked recipes.
public void addDislikedRecipe(Recipe dislikedRecipe)	Adds a recipe to the list of disliked recipes.
public void rmDislikedRecipe(Recipe recipe)	Removes a recipe for the list of disliked recipes.
public void addLikedIngredient(Ingredient ingredient)	Adds an ingredient to the list of liked ingredients
public void rmLikedIngredient(Ingredient ingredient)	Removes an ingredient from the list of liked ingredients.
public void addDislikedIngredient(Ingredient ingredient)	Adds an ingredient in the list of disliked ingredients.
public void rmDislikedIngredient(Ingredient ingredient)	Removes an ingredient from the list of disliked ingredients.
getters and setters	

Nutrition
Nutrition holds macro and micro nutrient information.
Attributes

private Measure base
private int calories
private int carbs
private int proteins
private int fats
private Map<String, Measure> micros

Methods

public static Nutrition fromJSON(String)	Returns a Nutrition object from its json representation.
public static String toJSON(Nutrition nutrition)	Returns the json representation as a String of a Nutrition object.
getters and setters	

Preference

Preference is responsible for grouping filters that the user would typically apply while generating meals or meal plans.

Attributes

private int mealsPerDay
private Pair<int, int> calRange
private Pair<int, int> fatRange
private Pair<int, int> carbRange
private Pair<int, int> protRange
private Pair<int, int> costRange
private Pair<int, int> cookingTimeRange
private String dietType
private Array<ENUM> acceptableDifficulties
private Array<Ingredient> restrictedIngredients

Methods

public static Preference fromJSON(String)	Returns a Preference object from its json representation.
public static String toJSON(Preference preference)	Returns the json representation as a String of a Preference object.

3.2 Server

In this section, the classes grouped under the three components of the client package are presented.

3.2.1 Route Tier

RequestHandler	
This class is responsible for handling requests and rerouting them to a relevant request handler if possible.	
Methods	
public boolean handleRequest(String url, Request req)	Sends the request to a relevant request handler. If unsuccessful, which means that the request is not relevant to any request handler, then it returns false and sends back the corresponding response.

3.2.2 Logic Tier

UserRequestHandler	
This class is responsible for handling user related requests. The handler methods of this class return true if successful, false otherwise. Request body that is provided to requests will contain all the relevant information that is needed by the method. Arguments to the methods (data) are passed inside the request body. For every request regarding a user a token is expected.	
Methods	
public boolean fetchUserInforHandler(String url, Request req	Fetches user related information.
public boolean updateUsernameHandler(String url, Request req)	Updates the username
public boolean updateHeightHandler(String url, Request req)	Updates the height of the user.
public boolean updateWeightHandler(String url, Request req)	Updates the weight of the user.
public boolean addAllergyHandler(String url, Request req)	Adds a new allergy type to the list of allergies of the user.
public boolean removeAllergyHandler(String url, Request req)	Removes an allergy type from the list of the allergies of the user.

public boolean updatePreferences(String url, Request req)	Updates the preferences of the user
public boolean likeRecipeHandler(String url, Request req)	Adds a recipe to the user’s list of liked recipes
public boolean unlikeRecipeHandler(String url, Request req)	Removes a recipe from the user’s list of liked recipes.
public boolean dislikeRecipeHandler(String url, Request req)	Adds a recipe to the user’s list of disliked recipes.
public boolean undislikeRecipeHandler(String url, Request req)	Adds a recipe to the user’s list of disliked recipes.
public boolean likeIngredientHandler(String url, Request req)	Adds an ingredient to the user’s list of liked ingredients
public boolean unlikeIngredientHandler(String url, Request req)	Removes an ingredient from the user’s list of liked ingredients.
public boolean dislikeIngredientHandler(String url, Request req)	Adds an ingredient to the user’s list of disliked ingredients.
public boolean undislikeIngredientHandler(String url, Request req)	Adds an ingredient to the user’s list of disliked ingredients.

Im2IngredsRequestHandler

This class is responsible for handling the Im2Ingreds class related requests. The handler methods of this class return true if successful, false otherwise. Arguments to the methods (data) are passed inside the request body

Methods

public boolean predictHandler(String url, Request req)	Fetches the prediction results of the Im2Ingreds model.

AuthenticationRequestHandler

This class is responsible for the authentication of the related request. The handler methods of this class return true if successful, false otherwise. Arguments to the methods (data) are passed inside the request body.
Methods

public boolean authenticationHandler(String url, Request req)	This method will check whether a user exists in the system and if so it will return a token for a user so that he can use that token as an identity in the subsequent requests.

RegistrationRequestHandler
This class is going to handle all the requests to the server related to registration. Arguments to the methods (data) are passed inside the request body.
Methods

public boolean registration(String url, Request req)	This method is going to register a user if all mandatory fields such as email and password are provided. If the email is not a real email or if the password is not strong enough the registration request will be rejected and the method will return false. Otherwise the user will be successfully registered in the Foodster.

MealRecommendationRequestHandler
This class is responsible for recommending meals to users taking into consideration the history of meals that users liked and other preferences that they provided to the Foodster such as their diet types, amount of calories they want to take in, their budget et cetera. Arguments to the methods (data) are passed inside the request body.
Methods

public boolean recommendMealFromHistoryHandler(String url, Request req)	This conservative meal recommender method recommends meals to Users looking at the meals that they liked in the past, and how the meals that are planning to be recommended are similar to the meals they liked. The similarity of meals will be calculated using mathematics such as Euclidean distance of meals' ingredients and scalar product of ingredients of the meal.
public boolean recommendMealRandomHandler(String url, Request req)	This non conservative meal recommender method recommends meals to Users randomly from the pool of meals that are allowed to the User considering User's allergies and diet types. However, the history of meals will not be considered for this method, because we want a User to try something new that he might potentially like and at the same time that he would not try himself.

GroceryListRequestHandler

The class that manages grocery list operations. The request body for these methods assumed to have all relevant information for the method.

Methods

public boolean groceryListGenerationFromScratchHandler(String url, Request req)	This method is going to generate a grocery list for the user that is required for a meal or for a meal plan assuming that User does not have any groceries. This method is going to return a list of missing ingredients if one or more of the ingredients for the meal or meal plan are not available in the grocery stores from which Foodster orders groceries.
public boolean groceryListGenerationHandler(String url, Request req)	This method is going to generate a grocery list for the items that User does not have. This method is going to return a list of missing ingredients if one or more of the ingredients for the meal or meal plan are not available in the grocery stores from which Foodster orders groceries.
public boolean getUserGroceryList(String url, Request req)	This method is going to return a grocery list that was generated already in the past for the user.

RecipeFetcher

The class that manages recipe operations. The request body for these methods assumed to have all relevant information for the method.

Methods:

public boolean getInstructions(String url, Request req)	Makes a database query to get the instruction details of the given recipe in the request.
public boolean getImagePath(String url, Request req)	Makes a database query to get the image path in the storage database of the recipe in the request.
public boolean getLikingUsers(String url, Request req)	Makes a database query to get the list of the users who liked the recipe in the request.
public boolean getRecipeWithNutritions(String url, Request req)	Finds a recipe with given nutrition constraints in the request.
public boolean getRecipe(String url, Request req)	Sets all attributes of the given recipe by making a database query with given recipe id in the request.

OrderingRequestHandler

The class that manages meal ordering operations. The request body for these methods assumed to have all relevant information for the method.

Methods:

private boolean login(String username, String password)	Logins the external ordering system with the given credentials in the body of the request.
public boolean searchForMeal(String url, Request req)	Makes a search for a given meal in the ordering system and lists relevant meals.
private void sortSearchResult(String url, Request req)	Sorts the result list of search according to the given sorting style.
private void prepareOrderAndShow()	Chooses the first meal in the sorted meal list, adds this meal to cart and prepares the order for payment.
public boolean makePayment(String url, Request req)	Makes payment with the given card details in the request such as card id, ec2 number etc.

3.2.3 Data Tier

Data tier classes are the same with the client side model classes, except Im2Ingreds, which is presented below.

Im2Ingreds	
This class is responsible for inferring ingredients of a meal given a picture of the meal.	
Attributes	
private <NNModuleType>[] layers	
Methods	
public loadWeights(String weightsPath)	Loads weights of the layers from a file given its path
public <JSONType> predict(ImgMtx img)	Predicts the ingredients of the meal based on the image matrix given as an input and returns the predictions as a json object