Eating Smart

By: Roman Pelikh

PROG1090J – Intermediate Java Programming

December 10, 2019

Submitted by:	Accepted by:
_____	_____
Author, Roman Pelikh	Instructor, Chris Cusack

Abstract

This report includes documentation about the project detailing project architecture, resources used, possible extensions, summaries and conclusions, user manual and references. The application I developed is called “Eating Smart”. The project was inspired by the willingness to track the amount of nutrients intake per meal. There are many similar applications in web and for mobile platforms, however it is not common to find such an application for desktop platforms; therefore, I have chosen to create my own application to cover this gap.

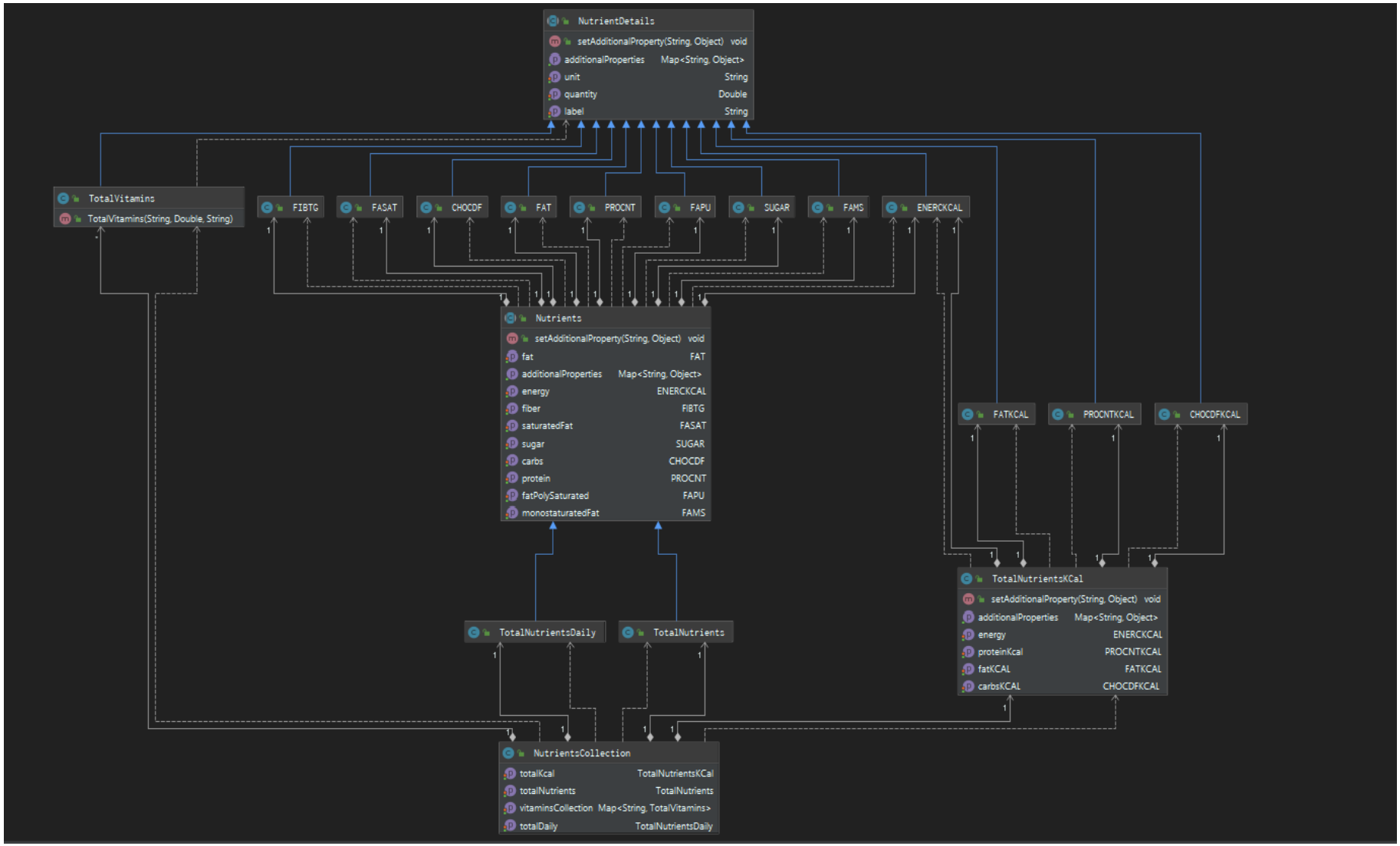
Table of Contents

Introduction

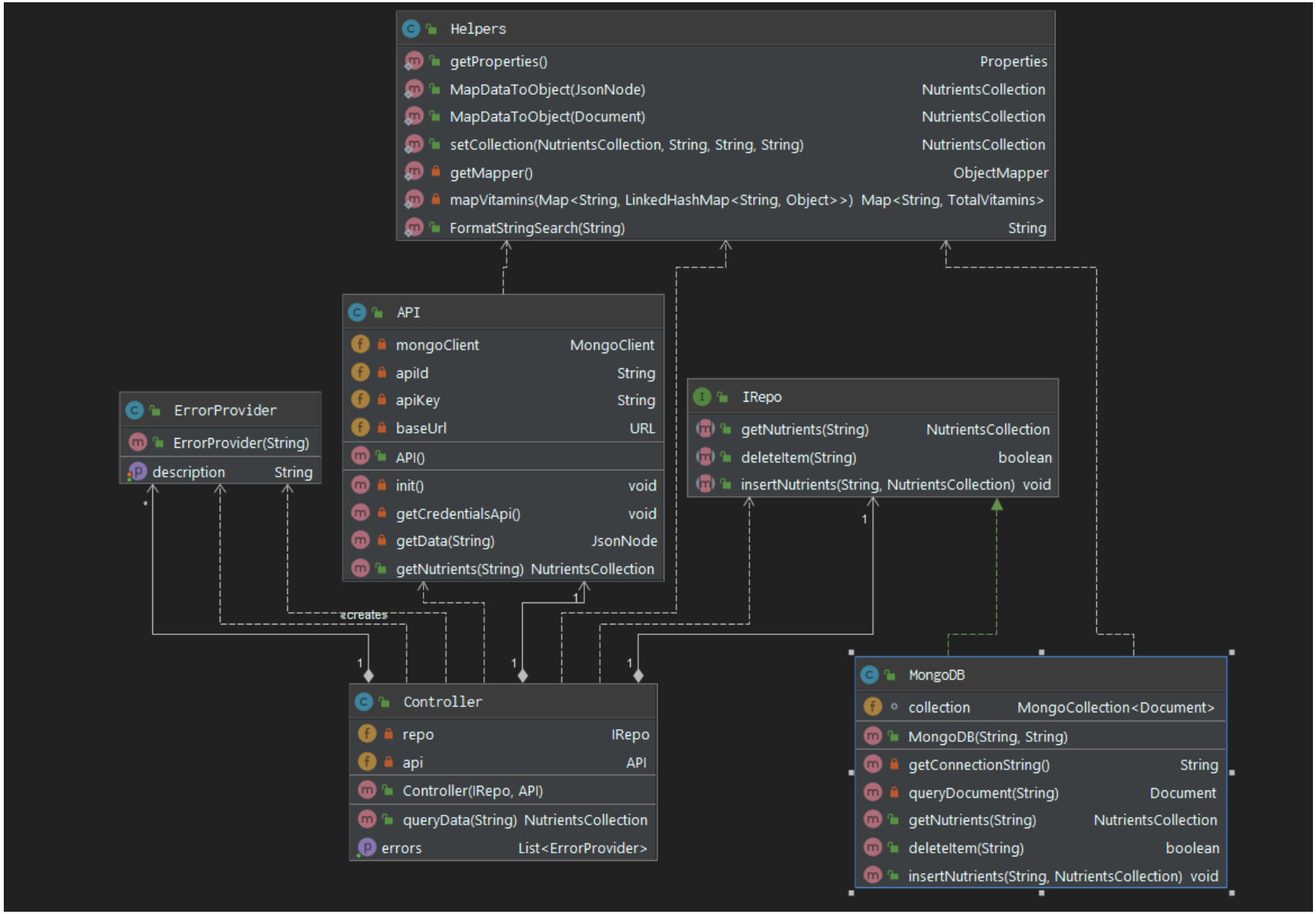
Eating Smart is an application that allows to get quick access to the macronutrient values in food. It allows to search and to display information about main nutrients (Fat (including variety of bonds of the carbon atoms in the aliphatic chain), Protein, Carbohydrates, Sugar), Calories, Vitamins similar to food labels that are very common in food industry, as well as ratio of calories intake from three main macronutrients. This report can be comprised as a documentation of the application and basic guideline.

Class Diagram

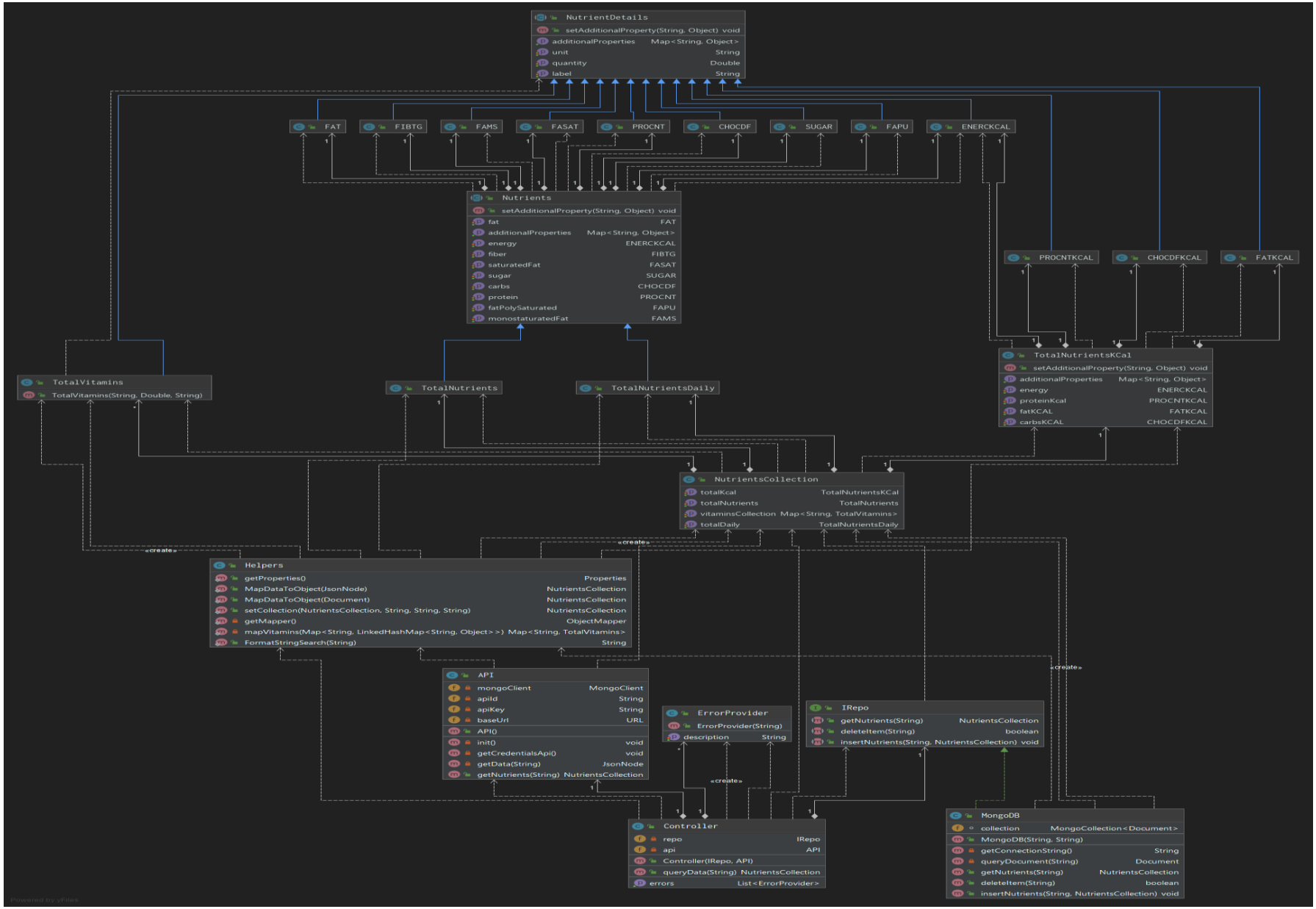
MODELS



APPLICATION



MODEL AND APPLICATION



Literature Review

There is no doubt that such a project requires out of class learning since some topics are beyond the scope of Intermediate Java Programming Course. In this section will be presented references being used in the developing of the application.

Java, JSON and API calls.

First step in my research was to get familiar with the way how to establish connection, make an HTTP request to the API end-point and get data back. Once I got results by making sure that data is buffered and consoling it, my next step was to parse buffered data to JSON format. To accomplish this task, I used JSON Jackson library https://github.com/FasterXML/jackson of 2.10 version released in September 2019. It is an awesome library that I would recommend anyone willing to work with JSON in Java. Once I got my data in JSON format I needed to find the way how to map it to my models. Fortunately, in the internet I found an great resource detailing the use of Jackson library, since on their official GitHub page they provide very few details about the library usage. This resource name is http://tutorials.jenkov.com/java-json/index.html . This resource covers many common cases of library usage. I would definitely get lost if I could not find this reference.

MongoDB

My next step was to learn how to use MongoDB driver with Java. Though I already had some exposure to this NoSQL database, I still had to learn some peculiarities and features when it comes to Java implementation. The official documentation of MongoDB https://mongodb.github.io/mongo-java-driver/3.11/driver/ helped me to succeed in it.

JFreeChart

When I connected my back-end to front-end, I decided to slightly enhance the latter and implement interactive charts on main JFrame. For such attempt I used and open-source framework JFreeChart. I must say, that it was a quite challenging task, due to the lack of structured documentation in free access. However, the entity standing behind JFreeChart is ready to offer a developer guide for the price at about 50 £; therefore, I had to google and find any resources that could be of use for me. Though these resources do not provide simple, clear and well-structured information, they at some point helped me to get started. To list some of them:

http://www.jfree.org/jfreechart/api/javadoc/index.html

https://www.caveofprogramming.com/java/charts-in-java-swing-with-jfreechart.html

http://zetcode.com/java/jfreechart/

https://www.javatpoint.com/jfreechart-tutorial

Java Map Interface (HashMap)

Finally, I had to research how to work with data-structures such as Map and HashMap to be able to efficiently work with models and perform basic operations such as get value, set value, iterate. To do so, I used resource that I mentioned above http://tutorials.jenkov.com/java-collections/map.html. It is an excellent reference about usage of Java Map Interface and many others.

Implementation of Eating Smart

The application has 4 modules: Search area allowing user to search for a product, Nutrition Facts in text representation displaying nutrients amount in grams and daily value, Calories Proportion representing the ratio of Protein, Carbohydrates and Fat that makes up the total calories, Nutrition facts in chart visually highlights the nutrients amount in the product, and Vitamin table displaying the percentage of vitamins in the daily norm. All modules are embedded in the JFrame called AppForm class.

The panels on the main window app were created using JPanel, labels using JLabel, charts using JFreeChart and table using JTable.

The configuration for database connection is stored in **db.properties** file. The configuration for API credentials is stored in remote database MongoDB.

Application has 2 sources of data.

First source of data is an API end point:

https://api.edamam.com/api/nutrition-details?app_id=${YOUR_APP_ID}&app_key=${YOUR_APP_KEY}

Second source of data:

mongodb+srv://roman:<password>@nutrients-lmd94.mongodb.net/admin?retryWrites=true&w=majority

The connection to MongoDB and CRUD operations were possible by using these packages:

- com.mongodb.client;
- com.mongodb.client.MongoClient;
- com.mongodb.client.MongoClients;
- com.mongodb.client.MongoCollection;
- com.mongodb.client.MongoDatabase;
- com.mongodb.client.model.Filters.and;
- com.mongodb.client.model.Filters.eq;
- com.mongodb.client.model.Filters.gt;
- com.mongodb.client.result.DeleteResult;
- org.bson.Document;
- com.mongodb.MongoException;

Parsing and manipulations with JSON was possible by using following packages

- com.fasterxml.jackson.core.JsonProcessingException;
- com.fasterxml.jackson.databind.ObjectMapper;
- com.fasterxml.jackson.databind.SerializationFeature;
- com.fasterxml.jackson.databind.DeserializationFeature;
- com.fasterxml.jackson.databind.JsonNode;
- com.fasterxml.jackson.databind.ObjectMapper;
- com.fasterxml.jackson.annotation.JsonAnyGetter;
- com.fasterxml.jackson.annotation.JsonAnySetter;
- com.fasterxml.jackson.annotation.JsonIgnore;
- com.fasterxml.jackson.annotation.JsonInclude;
- com.fasterxml.jackson.annotation.JsonProperty;
- com.fasterxml.jackson.annotation.JsonPropertyOrder;

When application starts up the first time, an instance of Controller class gets instantiated, where database source and API source instances are passed as parameters to constructor. Since application uses Interface **IRepo**, having defined methods for basic crud operations, constructor in controller can accept as its first parameter any class with database (SQL/NoSQL) that implements **IRepo** interface.

When the **API** class is getting initialized it calls **init()** method, which gets credentials for API endpoint from remote database. Once credential is received, user is presented with search area to analyse ingredient.

When user clicks the button **Search**, the method **btnSearchActionPerformed()** is called and application makes first request to remote database using repo method **getNutrients().** The **getNutrients()** method is getting called on MongoDB class and queries the MongoDB database and if both condition are true (data is available and data is fresh(no more than 24h passed since the data was stored) data is returned as **bson.Document** and then is mapped to models through the method **MapDataToObject()** in Helpers class. This method has 2 overloads and can accept data in form of **bson.Document** or as **Json Node**.

If data is not available in database, the controller moves to API source and makes a call to api-endpoint. If data is not available, the error is created using class **ErrorProvider** and is added to error list in the **Controller** class. Then this error is displayed to user as alert message in GUI. If data is available, it is returned as Json Node using Jackson library. Then it is mapped to models through the method **MapDataToObject()** in **Helpers** class. After that the retrieved data from API either gets added or replaced in database with a timestamp for the insert operation. For creating timestamp **java.time.LocalDateTime** class is used.

Once all models are populated with data, the process of mapping data to GUI is started. Methods in the **AppForm** class **getCaloriesProportion(), getLabel(), getVitamins(), getNutrientsChart()** are called to display data on GUI.

The Nutrition Facts

The Nutrition Facts Label consists of labels of type JLabel. Labels are set from the models **TotalNutrientsDaily** class and **TotalNutrients** class. Labels hold strings for names and double for quantities. Since API returns quantity with high precision more than 5 points after decimal, I use **NumberFormat** class to set 1 decimal place after coma. Units such as **%, g** and **kcal** are hardcoded.

Calories Proportion

Calories Proportion Pie Chart module consist of JPanel and JFreeChart. Chart is integrated in the panel for better layout. Chart has configurations in the AppForm.java. Pie chart has feature for scrolling, so user can change the layout of additional labels on Pie Chart using scroll wheel on the mouse. All labels and captions in chart are provided by JFreeChart library. Chart uses **TotalNutrientsKCal** class as its source for data.

Nutrients

Nutrients Bar Chart module consist of JPanel and JFreeChart. Chart is integrated in the panel for better layout. Chart has configurations in the AppForm.java. Bar chart has hoverable bars, so user can hover on each bar to get additional information about quantity of a nutrient. All labels and captions in chart are provided by JFreeChart library. Chart uses **TotalNutrients** class as its source for data.

Vitamins Table

The Vitamins Table consists of JTable element. Table has 3 columns for label, quantity and unit. Each column can be sorted by clicking on the header of the column. Table rows are created using are set of data from the model **TotalVitamins** class.

Testing

Testing process of the Eating Smart application was done iteratively. First, I used 3-rd party tool "**Postman**" to have a visual representation of my JSON data-structure, so I can build classes accordingly. Once my application was able to make an HTTP request, bring data back and parse JSON, console testing took a small part of all my testing methods. I printed to console the parsed values stored in object and compared it against API schema. Also, I used console testing when I connected my application to remote database (MongoDB), so I can make sure that I pass the correct values to store. Additionally, I used MongoDB tool called "**Compass**" to make sure that data I store and retrieve back is structured correctly and I do not loose anything along the way.

Nevertheless, the major part of testing tools was accounted for in-build debugger in NetBeans. This was by far the most convenient tool to test my application including checking values in variables and call stack.

Finally, when the GUI was built, I used it to test the general workability of application.

Possible Extensions

The Eating Smart is my attempt to create a basic application that allows to track micro-nutrients and micro-elements in food. The application does what it was intended to and provides the user with simple, visually structured information.

Critique

Event though the Eating Smart application provides the functional it was intended to do, there are still a room for improvements. The GUI is very basic and looks obsolete like from 90's. Application is able to retrieve information only about one product at a time. Application does not make use of Threads, so when user clicks the button "Search" the application "freezes", because fetching of data does not happen asynchronously. The application does not have a user account implementation and does not hold history of searches.

Improvements

First improvement to the application is to make more appealing GUI which would have modern and clean look. Though it is not critical, it would increase user experience significantly. Also, adding Threads to application would contribute to UX and to better workability of application. Moreover, the application should have the ability to accept many entries from user at a time, process them and display calculated values. The ideal use case scenario is the ability to pass a recipe so application can calculate the total amount of microelements and calories in the recipe. Finally, implementation of accounts for users would enable the tracking of search requests with a further reports and analysis of data. So, users would be able to set their goals in macroelements consumption and track their progress. This option would find a great positive feedback amongst athletes and amongst those who maintain a diet.

Summary and Conclusions

The Eating Smart application met all requirements and implements all features declared in proposal. Moreover, it has additional, not declared feature like charts. However, refined GUI, threads, extended search and implementation of user accounts would make it even better. Adding threads and multiline search will be the first step in the next iteration of development process.

By working on this application, I have learned a great deal of useful features that can be applicable not only for desktop applications but also for web apps. Working with API, though it was the most challenging part in developing of application, brought me the most joy. Also, implementation of Dependency Inversion to make my application loosely coupled when it comes to choice of database, made me extremely happy and I really enjoyed working with charts.

References and Bibliography

Jackson Project Home
https://github.com/FasterXML/jackson

Java JSON Tutorial
http://tutorials.jenkov.com/java-json/index.html

MongoDB Driver 3.11 Documentation
https://mongodb.github.io/mongo-java-driver/3.11/driver/

JFreeChart 1.5.0 API
http://www.jfree.org/jfreechart/api/javadoc/index.html

Charts in Java Swing With JFreeChart
https://www.caveofprogramming.com/java/charts-in-java-swing-with-jfreechart.html

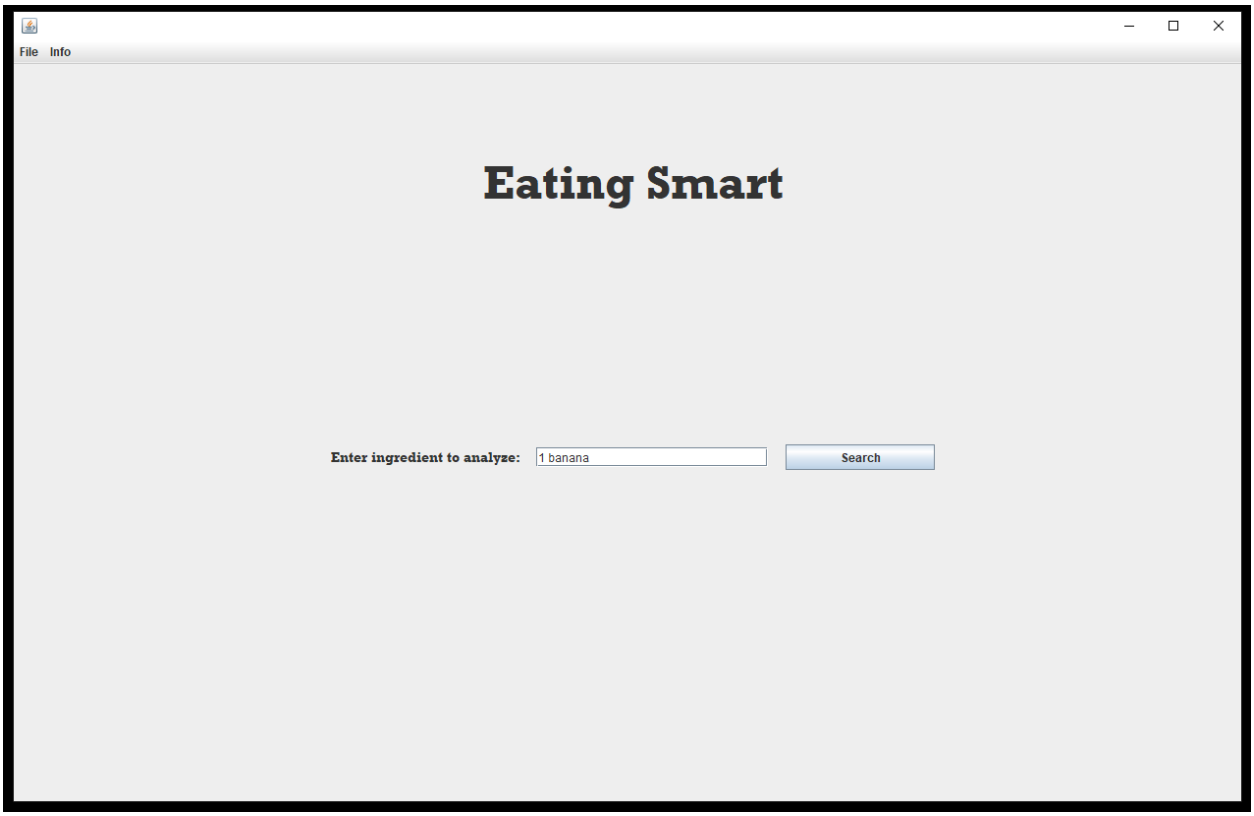
JFreeChart tutorial
http://zetcode.com/java/jfreechart/

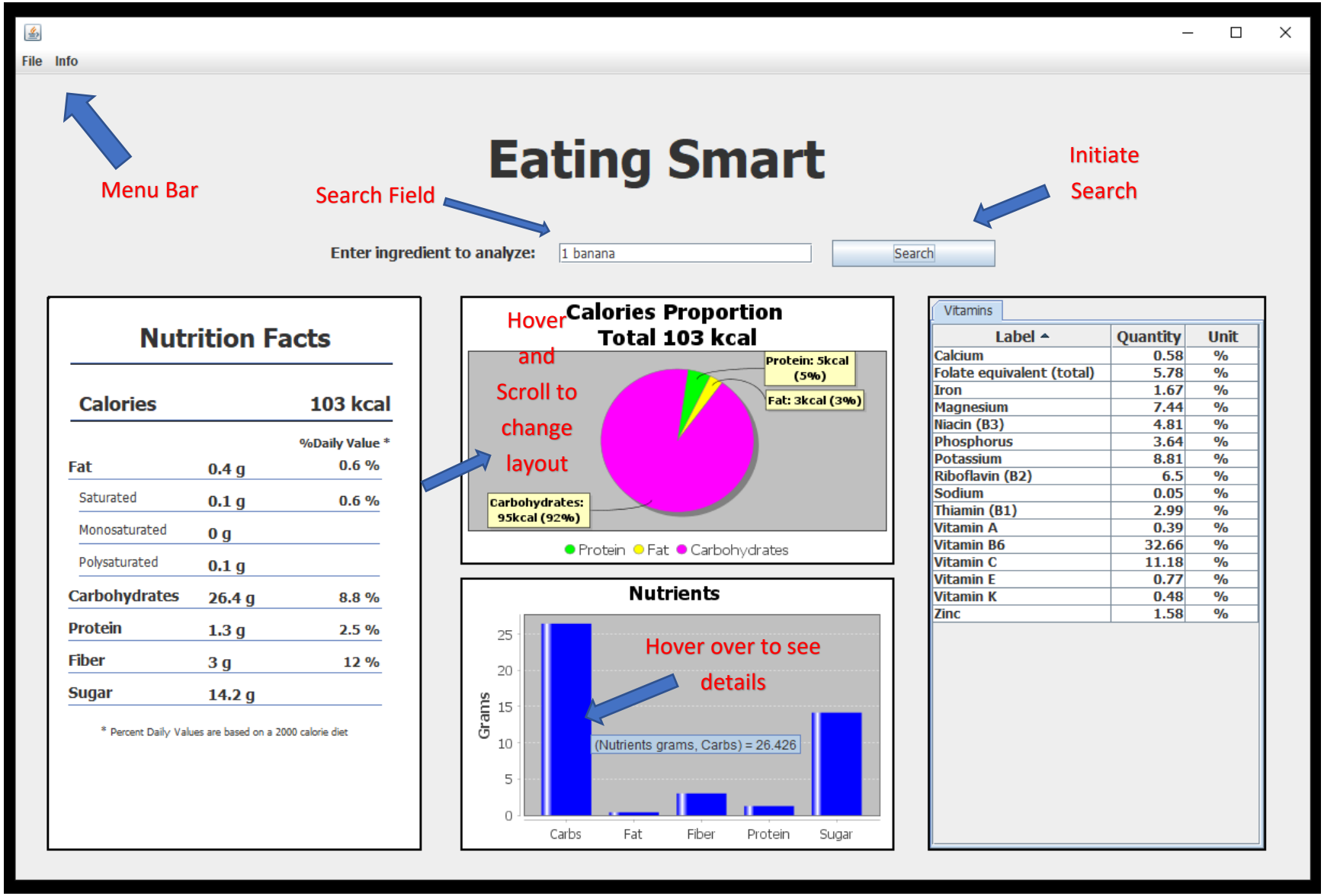
JFreeChart Tutorial
https://www.javatpoint.com/jfreechart-tutorial

Java Map
http://tutorials.jenkov.com/java-collections/map.html

Appendix A – User Manual

Eating Smart




File Info
Menu Bar
Eating Smart
Search Field
Initiate Search
Enter ingredient to analyze:
1 banana
Search
Nutrition Facts
Calories 103 kcal
%Daily Value *
Fat 0.4 g 0.6 %
Saturated 0.1 g 0.6 %
Monosaturated 0 g
Polysaturated 0.1 g
Carbohydrates 26.4 g 8.8 %
Protein 1.3 g 2.5 %
Fiber 3 g 12 %
Sugar 14.2 g
* Percent Daily Values are based on a 2000 calorie diet
Hover and Scroll to change layout
Calories Proportion
Total 103 kcal
Protein: 5kcal (5%)
Fat: 3kcal (3%)
Carbohydrates: 95kcal (92%)
Protein Fat Carbohydrates
Nutrients
Hover over to see details
Grams
25
20
15
10
5
0
(Nutrients grams, Carbs) = 26.426
Carbs Fat Fiber Protein Sugar
Vitamins
Label Quantity Unit
Calcium 0.58 %
Folate equivalent (total) 5.78 %
Iron 1.67 %
Magnesium 7.44 %
Niacin (B3) 4.81 %
Phosphorus 3.64 %
Potassium 8.81 %
Riboflavin (B2) 6.5 %
Sodium 0.05 %
Thiamin (B1) 2.99 %
Vitamin A 0.39 %
Vitamin B6 32.66 %
Vitamin C 11.18 %
Vitamin E 0.77 %
Vitamin K 0.48 %
Zinc 1.58 %


File Info
Menu Bar
Eating Smart
Info Window
Eating Smart
Eating Smart application allows to enter an ingredient for what you are cooking, like "1 cup rice" and learn its nutrition details in under couple seconds.
Nutrition Facts
Calories 10
%Da
Fat 0.4 g
Saturated 0.1 g
Monosaturated 0 g
Polysaturated 0.1 g
Carbohydrates 26.4 g
Protein 1.3 g
Fiber 3 g
Sugar 14.2 g
* Percent Daily Values are based on a 2000 calorie
10
5
0
Carbs
Fat
Fiber
Protein
Sugar
abel
Quantity
Unit
0.58 %
valent (total) 5.78 %
1.67 %
7.44 %
4.81 %
s 3.64 %
8.81 %
(B2) 6.5 %
0.05 %
1) 2.99 %
0.39 %
32.66 %
11.18 %
0.77 %
0.48 %
1.58 %

To close Eating Smart, click the **X** in the upper right corner of the main window or select **Exit** in **File > Exit** in in Menu Bar

To find the nutrients in product, type amount and unit (number, oz, g, kg, L, ml) and name of the product in the Search Field

To initiate search and get results, click Search button

To change layout of Calories proportion chart, hover mouse on the chart and scroll mouse up or down

To get more precise amount of nutrients, hover over a bar in Nutrients (G) chart to see popup-label

To get information about application and bring Info Window, select **About** in **Info** > **About** in Menu Bar