McMaster University

Computing and Software

Final Project

Nutrition Freak

Hahd Khan, Jeremy Joseph Klotz, Cesar Santana Penner, Rebecca Tran

L04

April 12, 2016

Group 23

SFWR ENG/ COMP SCI 2XB3

Revision

Search

Originally, we planned to search through the datasets with merge sort, as they are large datasets and we intended on implementing a search function for the BMI dataset (which is quite large). Although, towards the end of the project, we decided to implement the search function for the food products instead and change the sorting algorithm. Doing so can tell the user much more information about certain food items, than the BMI dataset. Changing the algorithm was more efficient to do so because of the size and dataset itself. We sorted the food groups dataset with binary insertion sort because the datasets were already partially sorted and were smaller in size. It took less time and was much easier with smaller sets of data compared to the BMI dataset. Running binary insertion sort on the large dataset was not efficient, so the work with the BMI dataset, we decided to sort it with shellsort using the numbers at the last index of the string arrays per line(split by commas). Shell sort was a better option because some of the lines in the dataset was unnecessary and were ignored. The size of the dataset decreased after, so merge sort was no longer needed and shell sort was used instead.

Graph

The graph we implemented also had to undergo some major changes. The original idea we had was to store the graph as a tree that stored the important information sequentially in order to find the person's desired number of servings. Instead of this, we chose to create the graph from the dataset that holds all of the information about the food in Canada's Food Guide. Every single food item is connected to each other food item that belongs in the same food group. Each edge is given a double value to represent the weight of the edge. This value is a random number from 1 - 10,000 (this is done in order to randomize the meal plan). From here, we gather the user's gender and age to find how many servings of each food group they need in a day. Then using a greedy algorithm, we travel the edge with the lowest weight.

Note:We used the Android Studio IDE to make the app. You can download it here:
http://developer.android.com/sdk/index.html

To download and run the NutritionFreak.apk:

1. You have an android phone:

a) ***Easiest:*** If you have an android phone you can download the .apk from your phone. Go to https://drive.google.com/drive/u/0/folders/0B4zxRDgFLe-CblRTdktBbDdvdlE and simply download the .apk on your phone and install.

b) If you have an android phone you can download the NutritionFreak.apk file onto your PC/Mac and transfer it onto the android phone. First, connect your android device into a PC or Mac via USB. In the internal storage of your phone and navigate to the Download folder. Next, copy and paste the .apk file into the Download folder in the android. Disconnect your phone from the computer. Note: Make sure you have Unknown source enabled on your phone (Settings -> Security -> Unknown sources).Using a file manager on your phone (we used File Manager by ZenUI but any file manager will do) navigate to the Download folder on your android device and tap on the .apk file. At this point you should be promoted to install the app. Install the app, and launch it.

c) If the above did not work try:

- https://www.youtube.com/watch?v=nY5EY1JWOaw
- http://science.opposingviews.com/move-apk-files-pc-android-phone-16071.html

2. You do not have an android phone:

I used a Genymotion emulator. It's free and its fast: https://www.genymotion.com/

a) https://www.youtube.com/watch?v=PoS2Vzt395I

b) http://stackoverflow.com/questions/3480201/how-do-you-install-an-apk-file-in-the-android-emulator

c) Google search

3. Last Case: Download Android studio and import the Android Studio project:

a) http://developer.android.com/sdk/index.html

b) https://www.jetbrains.com/help/idea/2016.1/importing-an-existing-android-project.html?origin=old_help

Members

Name:	Student Number:	Mac ID:
Hahd Khan	1209974	khanh27
Jeremy Klotz	1426853	klotzjj
Cesar Santana Penner	1411598	santanca

Rebecca Tran	1425611	tranr5

By virtue of submitting this document we electronically sign and date that the work being submitted by all the individuals in the group is their exclusive work as a group and we consent to make available the application developed through [CS] or [SE]-2XB3 project, the reports, presentations, and assignments (not including my name and student number) for future teaching purposes

Contribution

Name	Role(s)	Contributions	Comments
Hahd Khan	Programmer	CO-Designer of Graph Parsed the servings per day data file and the food guide menu data file. Design Document (MIS/MID for idServings)	Implemented the edges of graph
Jeremy Klotz	Programmer/ Algorithm specialist	Created BMI calculator, parsed BMI dataset and created the function that relays all BMI related information CO-Designer of Graph Design Document (Graph, Graphing, MIS/MID for ServingsPerDay)	Created the nodes of the graph, came up with the idea of how to weight the edges
Cesar Santana Penner	Designer/Programmer	Designed, created, and implemented the mobile app, layouts (in xml), activities (java). Modular Decomposition, MIS, MID. Parsed food	-did 95% of the front end (all the screen layout in xml code, All the java classes except BMI class, foodNode, foodEdge, and graphing classes), searching and

		products csv data. Implemented Binary Insertion Sort and Binary Search. Created launch icon for the app	sorting. MIS and MID for all of my code and BMI class - Parsed the vegetables data
Rebecca Tran	Designer/Programmer/ Logger	Designed buttons, logo Parsed food products data Logged meetings (meeting minutes, log) Design Document MIS/MID for foodNode, foodEdges, Recorded (except Revision Graph, Module Decomp, MIS & MID for GUI)	Parsed grains, meat and dairy data

Executive Summary

Roughly 40 percent of Canadians are self-reported to be obese or overweight. This application is to be targeted at solving this problem by providing a monthly meal plan and comparing the user's BMI to a healthy one. The datasets used to do this will include information from Canada's food guide, listing tallies of body mass index per sex, age and region and some nutritional facts about food. Since the user's BMI will be tracked monthly, this can be used as a validation that the product is working properly. The application will prompt the user to enter their personal details, and calculate everything for the user based on their inputs. A graph was created to hold all of the foods in the food guide, and we operate an algorithm on this graph to create the user's meal plan. The goal of doing this is to educate the user in how to make healthy food choices. This should theoretically help the user avoid poor diet related illnesses and maintain a healthy weight.

Contents

Environment

This project was heavily implemented with the use of Java and Android Studio. Android Studio helps create the mobile app and the Graphical User Interface, where it utilizes the layouts in xml. The activities and other components are created in Java.

Module Decomposition

LoginActivity

When building an android application, each activity (screen in which the user can interact with) is its own class (and their respective xml files). The LoginActivity is used as the login page of application. The user can login in with their email and password or if they have not yet created an account go to the CreateAccountActivity.

CreateAccountActivity

In the CreateAccountActivity the user the user can signup and create an account with their email and password. Once the user has made an account, the user will be forwarded to the ProfileSetUp activity where they will be able to enter some of their basic information such as name, age, weight, height, etc. From there, the dataset traversal is done, and shell sort is used to sort the BMI data, in order to calculate the BMI.

MainActivity

Next, the user will be in the MainActivity module where the user can choose to: view the generated meal plan on a calendar, go to the settings to change their personal statistics, search up food items for more information or view more about us and the idea behind the application. If the user goes to view the calendar, they will go to the calendarActivty which will display a calendar for that month. The user will than be able to click on any of the given days of that month and view their generated meal plan of the day. This generated meal plan is in the mealPlanOfTheDayActivity. If the user goes to the SettingsActivity (from the MainActivity) they will be able to view their personal stats and updated them (through a not yet created activity). If the user goes to the search page, they will be directed to searchActivity where they can enter the name of a food item and select the appropriate food group. This is to tell the user any nutritional information on the item. If the user wants to view more information about us or the inspiration for the project they can go to the AboutActivty. In addition to these activities (and their respective xml and class files), we have a BMI class which is used to calculate the bmi of a person given their weight and height. Refer to Figure 1 for the decomposition of all the modules.

Searching

To implement the search function, Binary Insertion Sort and Binary Search are used. Since binary sort only works with sorted arrays, binary insertion sort is necessary to sort through all of the data sets. Binary Insertion sort works by checking the middle element of the array to see if there is a match. If not, a comparison is done to check if the value we are searching for is greater than or less than the middle element. From there, the algorithm will search the upper half if the element is greater than, or the lower half if the element is less than. This process is done recursively until the element is found. Using this algorithm, we can find the item users are searching for by going through each sorted dataset and return the corresponding nutritional information. Some tested cases used: "Imitation cheese", "Carrots, raw", "Summer sausage, beef", "Celery, raw". Note: searching for a food must match exactly as in the food product data sets (e.g. DP.txt, GP.txt, MP.txt, or VP.txt). So for example, if the user wants to search for "Summer sausage, beef" and simply searches for "beef" the app will say that no food was found. The user must search for "Summer sausage, beef" to find the food. This was done because in the meat product data set there are dozens of foods with beef and it would have been much more difficult to implement a feature that returns all the foods with beef in the title name. Also, the proper food group filter must be applied to find a food. For example, for "Imitation cheese" the dairy products filter must be applied. If any other filter is applied the food will not be found.

Graphing

The graph we decided to implement was directed and connected. The graph was created from the foods-en_ONPP_rev csv file Each line in the file represents a node which holds strings that declare specific values that pertain to that node. The name of the food, and the amount of the food to make one serving and the food group they belong too. Four graphs were created, one for each food group. Lines from the file with the string "vf" belong to the fruits and vegetables graph, "gr" belong to the grains graph, "mi" belongs to the dairy graph, and "me" belongs to the proteins graph. Inside those graphs each food item is connected to each other. The edges are directed because you can go from point A to B or B to A. The way the edge is travelled could change the weight of the path. The weight of an edge is determined by a random double value given to the node travelled to.

The algorithm we decided to implement on this graph was a modification of Dijkstra's algorithm. The algorithm is to travel to x number of nodes in a given food group graph where x is the number of servings that person requires based on their age, and gender for that food group. First travel to the node with the smallest weighted incoming edges. Then select that food , with it's portion size and add it to the meal plan. Continue to do this x number of times. Repeat this process four times (one for each food group).

Extra Work

Along with our submission is a java eclipse project which includes the parsing of the BMI dataset and the shell sort used along with it. We simply did not have enough time to figure out how the buttons from android studio could be used as inputs for java files. It would be greatly appreciated if this java project was run separately from the application. The java project was handed in as an extra file to the submission. It is named "fullBMIproject.zip".

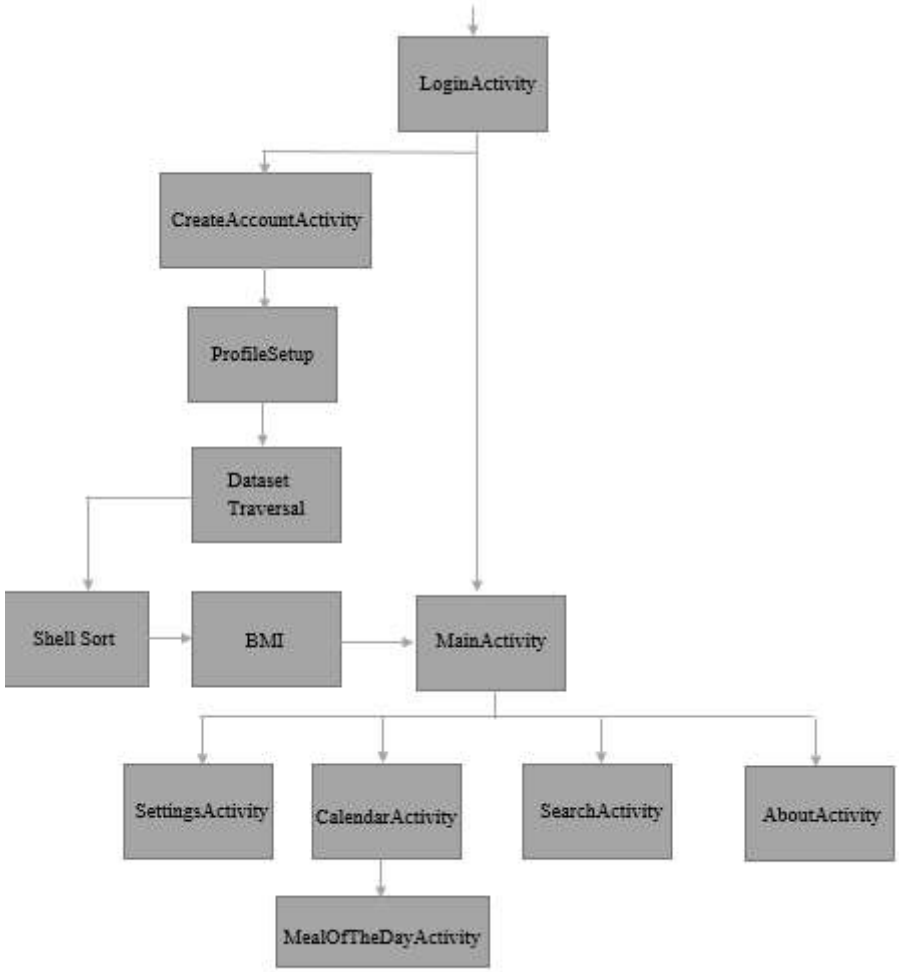

Figure 1: The Module Decomposition

Module Guide

Interface

The app starts out with a login page, where the user can login with their existing account with an email and password. If not, the user can click the text at the bottom that directs them to a signup page, where they can create an account. Once the user signs in, the main page will pop up with four buttons; the calendar, search, settings and about page. The calendar provides the user with an overview of their meal plan for the month. They can view the meal plan for a specific day by clicking on the day on the calendar. The search page allows users to search up a certain food item, which then provides the nutrients and vitamins of that specified search. Users will input the name of the item, select the food group it belongs to and can then hit search. From there, the nutritional information will appear underneath. This will work as long as the item is in the data set of the food groups. The settings page is where the user can input their age, weight, height and gender. Once the user submits this information, their BMI is automatically calculated. This page is where the user can also update their information, if any of their information changes. The about page is simply a page to tell the users more about the app.

MIS
CLASS: LoginActivity
Defines the login page/screen of the application.

INTERFACE
USES
The user can login with their email and password.
VARIABLE
None

ACCESS PROGRAMS
OnCreate(Bundle savedInstanceState): void
Called when the activity is first created. This is where all the setup happens such as creating views, binding the xml and java, etc.

MID
IMPLEMENTATION: LoginActivity

USES
The user can login with their email and password.
VARIABLES
emailEditText: EditText
Gets the user's email

passwordEditText: EditText
Gets the user's password

noAccountYet: TextView
Text displaying "No account yet? Create One", allowing the user to go to the

logInButton: Button
Navigates the user to the MainActivity, if the email and password are correct

prevAccounts: String
Used to read the accounts from a local text file

accounts: String[]
Array of strings representing all the current accounts

ACCESS PROGRAMS
setUp(): void

This is where all the setup happens such as creating views, binding the xml and java, etc

readFile(): void
Reads internal file

writeFileAddLine(String data): void
Write to the accounts text file

readFromFile(): String
Read the file containing all the accounts, return the accounts as a string

getUserNameID(String[] a, Stirng email, String password): int
Gets the user's ID given their email and password from an array of accounts.

checkEmail(String): boolean
Checks if there exists a user with the given email and password.

MIS
CLASS: CreateAccountActivity
Defines the create account page/screen of the application.

INTERFACE
USES
The user signs up using an email and password.
VARIABLE
None

ACCESS PROGRAMS
OnCreate(Bundle savedInstanceState): void
Called when the activity is first created. This is where all the setup happens such as creating views, binding the xml and java, etc. Also this method provides a Bundle containing the activity's previously frozen state.

MID
IMPLEMENTATION: createAccountActivity

USES
The user can create an account using an email and password.
VARIABLES

emailEditText: EditText
Get the user's email.

passwordEditText: EditText
Get the user's desired password.

confirmPasswordEditText: EditText
Get the user's password, to confirm that the password is the same as the first password edit text

signUpButton: Button
Navigates the user to the next page, if the information was filled in properly.

Accounts: String
All the accounts in the text file, in string format.

ACCESS PROGRAMS

setUp(): void
This is where all the setup happens such as creating views, binding the xml and java, etc

isCompleted(): boolean
Checks if all the information has been filled in.

checkPasswordEqual(): boolean
Checks if the password in the passwordEditText and the confirmPasswordEditText have been filled in.

writeFileAddLine(String data): void
Write to the accounts text file

MIS

CLASS: ProfileSetUp

Defines the profile setup (basic information) page/screen of the application.

INTERFACE

USES

This is where the user enters their basic information such as age, weight, height, etc.

VARIABLE

None

ACCESS PROGRAMS

OnCreate(Bundle savedInstanceState): void

Called when the activity is first created. This is where all the setup happens such as creating views, binding the xml and java, etc. Also this method provides a Bundle containing the activity's previously frozen state.

MID

IMPLEMENTATION: ProfileSetUp

USES

Used to get the user's personal stats and information.

VARIABLES

ageEditText: EditText

Gets the user's age.

weightEdittext: EditText

Gets the user's weight.

heightInchesEditText: EditText

Gets the user's height (the inches part).

heightFeetEditText: EditText

Gets the user's height (the feet part).

submitButton: Button

Navigates the user the main activity if all the information has been filled in.

userNameEditText: EditText

Get's the user's name.

provinceRadioGroup: RadioGroup

Get's which province the user is from.

bmi: BMI

Calculates the user's BMI.

email: String

Stores the user's email.

password: String

Stores the user's password.

accounts: String
String representation of all the accounts.

ACCESS PROGRAMS

setUp(): void
This is where all the setup happens such as creating views, binding the xml and java, etc

checkComplete(): boolean
Checks if all the information has been filled in.

writeFileAddLine(String data): void
Write to the accounts text file

MIS
CLASS: MainActivity
Defines the main page/screen of the application.

INTERFACE
USES

This is where the user can view their generated meal plan, view more about the application, and change some settings.

VARIABLE

None

ACCESS PROGRAMS

OnCreate(Bundle savedInstanceState): void
Called when the activity is first created. This is where all the setup happens such as creating views, binding the xml and java, etc. Also this method provides a Bundle containing the activity's previously frozen state.

MID
IMPLEMENTATION: MainActivity
USES

Used to get the user's personal stats and information.

VARIABLES

calendarImageView: ImageView
If this image is clicked, it will navigate to the calendar.

settingsImageView: ImageView
When this iamge is click on, it will navigate the user to the settings page.

aboutImageeView: ImageView
When this image is clicked, it will navigate the user to the about page.

account: Account
Stores the current account info.

name: String
Stores the name of the user.
age: int
Stores the age of the user

email: String
Stores the email of the user.

weight: double
Stores the weight of the user.

bmi: double
Stores the BMI score of the user.

inches:double
Stores the height of the user (the inches part of the height)

feet: double
Stores the height of the user (the feet part of the height)

province: String
Stores the province of where the user lives.

ACCESS PROGRAMS

setUp(): void
This is where all the setup happens such as creating views, binding the xml and java, etc

checkComplete(): boolean
Checks if all the information has been filled in.

onBackPressed(): void
Disables the back button

MIS
CLASS: AboutActivity
Defines the profile about page/screen of the application.

INTERFACE
USES
The user can view more about the application and its inspiration.

VARIABLE
None

ACCESS PROGRAMS
OnCreate(Bundle savedInstanceState): void
Called when the activity is first created. This is where all the setup happens such as creating views, binding the xml and java, etc. Also this method provides a Bundle containing the activity's previously frozen state

MID
IMPLEMENTATION: AboutActivity

USES
Shows the user more information about the application such as the inspiration.
VARIABLES
None

ACCESS PROGRAMS
None

MIS
CLASS: searchingActivity
Defines the search page/screen of the application.

INTERFACE
USES
Allows the user to search up foods and view their nutritional information
VARIABLE
None

ACCESS PROGRAMS

OnCreate(Bundle savedInstanceState): void

Called when the activity is first created. This is where all the setup happens such as creating views, binding the xml and java, etc. Also this method provides a Bundle containing the activity's previously frozen state

MID

CLASS: searchingActivity

Defines the profile about page/screen of the application.

INTERFACE

USES

Allows the user to search up foods and view their nutritional information

VARIABLE

searchButton: Button

When this button is clicked, it will search through the food product data set

showInfo: TextView

The found food's information will be shown here

searchEditText: EditText

This is where the search string is entered

veggiefoods: String[][]

Stores all the vegetable products and all their information

veggiefoodNames: String[]

Stores all the vegetables products names.

meatfoods: String[][]

Stores all the meats products and all their information

meatFoodNames: String[]

Stores all the meat product names

dairyfoods: String[][]

Stores all the dairy products and all their nutritional information

dairyFoodNames: String[]
Stores all the dairy product names

grainfoods: String[][]
Stores all the grain product and all their nutritional information

grainfoodNames: String[]
Stores all the grain product names

filterOptions: RadioGroup
Radio Group for all the filter options

veggiesRadioButton: RadioButton
Used as the vegetables filter option

meatRadioButton: RadioButton
Used as the meat filter option

dairyRadioButton: RadioButton
Used as the dairy filter option

grainsRadioButton: RadioButton
Used as the grains filter option

nutritionInfo: String[]
Nutrition information categories for dairy, vegetables and grain groups

meatsNutritionInfo: String[]
Nutrition information categories for the meat group

nutritionMeasurements: String[]
Measurements of each nutrition information category for dairy, vegetables and grain groups

meatsNutritionMeasurements: String[]
Measurements of the meat group's nutrition information

ACCESS PROGRAMS

setUp(): void
This is where all the setup happens such as creating views, binding the xml and java, etc

printFood(String[] food): String

Formats a given food (grains, dairy, vegetables) with each corresponding measurement and nutrient type

printMeats(String[] food): String
Formats a given meat food with each corresponding measurement and nutrient type

readFile(String filenName): ArrayList<String>
Reads a given csv data set and returns as a list of lines

parseInfo(ArrayList<String> data): ArrayList<String[]>
Parses a given data set (in ArrayList<String> format) and returns a ArrayList of array strings representing foods (and their information)

parseInfo2(ArrayList<String> data): String[][]
Parses a given data set (in ArrayList<String> format) and returns a 2D array of strings representing foods (and their information)

copyFoodNames(String[][] data): String[]
Used to transfer the vegetables list of foods, into an array of vegetables list of foods with only their food name

copyDairyFoodNames(String[][] data): String[]
Used to transfer the dairy list of foods, into an array of dairy list of foods with only their food name

copyGrainFoodNames(String[][] data): String[]
Used to transfer the grain list of foods, into an array of grain list of foods with only their food name

copyMeatFoodNames(String[][] data): String[]
Used to transfer the meat list of foods, into an array of meat list of foods with only their food name

copyArray(String[] c): String[]
Copies an array

sortBinary(Comparable[] x, Comparable[][] y, int n): void
Sorts both the lists of a given food using Binary Insertion Sort

exch(String[] x, int i, int j): void
Exchanges to elements in a string array

exch(Comparable[] a, int i, int j): void
Exchanges to elements in a comparable array

less(Comparable v, Comparable w): boolean
Checks either v is less than w

binarySearch(Comparable[] x, Comparable current, int low, int high): int
Binary Search on a list of foods, returning the location of where it was found.

MIS
CLASS: Settings
Defines the profile setup (basic information) page/screen of the application.

INTERFACE
USES
The user can change/update their personal stats.
VARIABLE
None

ACCESS PROGRAMS
OnCreate(Bundle savedInstanceState): void
Called when the activity is first created. This is where all the setup happens such as creating views, binding the xml and java, etc. Also this method provides a Bundle containing the activity's previously frozen state.

MID
IMPLEMENTATION: SettingsActivity

USES
The user can update their personal stats.
VARIABLES
ageTextView: TextView
Displays the user's current age.

weightTextView: TextView
Displays the user's current weight.

heightTextView: TextView
Displays the user's current height

bmiTextView: TextView
Displays the user's current BMI score

name: String
Stores the user's name

email: String
Stores the user's email

age: int
Stores the user's current age

weight: double
Stores the user's current weight

feet: double
Stores the user's current feet portion of their height

inches: double
Stores the user's current inches portion of their height

bmi: double
Stores the user's current BMI score

updateStates: Button
Allows the user to update their stats

ACCESS PROGRAMS

setUp(): void
This is where all the setup happens such as creating views, binding the xml and java, etc

MIS
CLASS: calendarActivity
Defines the calendar view page/screen of the application.

INTERFACE
USES

The user can view a calendar and click on any given day to view their generated meal plan of the day.

VARIABLE

None

ACCESS PROGRAMS

OnCreate(Bundle savedInstanceState): void

Called when the activity is first created. This is where all the setup happens such as creating views, binding the xml and java, etc. Also this method provides a Bundle containing the activity's previously frozen state.

MID

IMPLEMENTATION: calendarActivity

USES

Allows the user to view their generated meal plan for a month.

VARIABLES

Calendar: CalendarView

The generated meal plan for a month

ACCESS PROGRAMS

readFile(String fileName): void

Reads a text file and parses it

calculatePortionSizes(): void

Calculates a persons portion sizes

generateMealPlan(): void

Generates a meal plan for 365 days of the year

makeMealPlan(): String[]

Makes the meal plan for a day

randomizeFoodsOfDay(): String[]

Randomizes the order of the meal plan of the day

generateMealPllanOfTheDay(): Stirng[]

Used to generate the random meal plan of the day

copyArray(ArrayList<String> array): String[]

Copies an ArrayList into an array

MIS
CLASS: mealPlanOfTheDayActivity
Defines the generated meal plan of the day page/screen of the application.

INTERFACE
USES
The user can view their generated meal plan of the day.
VARIABLE
None

ACCESS PROGRAMS
OnCreate(Bundle savedInstanceState): void
Called when the activity is first created. This is where all the setup happens such as creating views, binding the xml and java, etc. Also this method provides a Bundle containing the activity's previously frozen state

MID
IMPLEMENTATION: mealPlanOfTheDayActivity
USES
The use can view a given day's meal plan of the day.
VARIABLES
mealPlan: ListView
Views the meal plan as a list view.

ACCESS PROGRAMS
setUp(): void
This is where all the setup happens such as creating views, binding the xml and java, etc

MIS
CLASS: BMI
Defines the BMI calculation, the result of the of the BMI calculation (based on the user's BMI are they underweight, Normal weight, Overweight, Obese), and a disclaimer calculator.

INTERFACE

USES

Calculate BMI, view results of BMI, and view BMI disclaimers.

VARIABLE

None

ACCESS PROGRAMS

OnCreate(Bundle savedInstanceState): void

Called when the activity is first created. This is where all the setup happens such as creating views, binding the xml and java, etc. Also this method provides a Bundle containing the activity's previously frozen state.

calculate(int feet, int inches, double weight): double

Calculates BMI based on height and weight.

result(double BodMassInd): String

Evaluates a proper result of their BMI (e.g. Underweight)

disclaimer(double BodMassInd): String

Returns a proper disclaimer of the BMI calculation

.

MID

IMPLEMENTATION: BMI

USES

Calculates BMI score and returns proper responses based on BMI score

VARIABLES

None

ACCESS PROGRAMS

None

MIS

CLASS: Account

Defines user's accounts in the application.

INTERFACE

USES

Easier organization of accounts.

VARIABLE

None

ACCESS PROGRAMS

Account(String name, String email, String password, int age, String province, double weight, double feet, double inches, double bmi)
Constructs an accounts object given information such as name, email, weight, etc.

getUsername(): String
Returns the user's name

getEmail(): String
Returns the user's email

getWeight(): double
Returns the user's weight

getInches(): double
Returns the user's height (the inches part)

getFeet():double
Returns the user's feet(the feet part of their height)

getPassword(): String
Returns the user's password

toString(): String
Returns the string representation of an account (email)

MID

IMPLEMENTATION: Account

USES

Makes it easier to view and organize accounts

VARIABLES

username: String
Stores the user's name

email: String
Stores the user's email.

password: String
Stores the user's password.

weight: double

Stores the user's weight

inches: double
Stores the user's height (the inches portion)

feet: double
Stores the user's height (the feet portion)

age: int
Stores the user's age

province: String
Stores the user's place of residence.

bmi: double
Stores the user's BMI score

ACCESS PROGRAMS
None

MIS
CLASS: foodEdge
Creates the edges of the graph

INTERFACE
USES
foodNode
VARIABLE
a: foodNode
First node of the food group

b: foodNode
Second node of the food group

weight: double
Cost of a particular food

descrip: String
Name of edge

order: Integer
To keep track of all of the edges

ACCESS PROGRAMS

foodEdge(a: foodNode, b: foodNode)
Holds the information of the edge

MID
CLASS: foodEdge
Creates the edges of the graph

INTERFACE
USES
foodNode
VARIABLE

a: foodNode
First node of the food group

b: foodNode
Second node of the food group

weight: double
Cost of a particular food

descrip: String
Name of edge

order: Integer
To keep track of all of the edges

ACCESS PROGRAMS

foodEdge(a: foodNode, b: foodNode)
Initializing a and b, from the foodNode class, initializes the description based on the food item name of both nodes and initializes the weight

MIS
CLASS: foodNode
Creates the nodes of the graph

INTERFACE
USES

None

VARIABLE

id: String
Determines the category of the food from which food group

idcat: String
The category the food belongs in

servSize: String
Amount of servings needed

foodName: String
The name of the food item

eWeight: Double
The weight of edge

ACCESS VARIABLES

foodNode(info: String[])
Initializes all variables to the proper information

foodNode(groupNode: ArrayList<foodNode>)
Another constructor, initializing a given group node

MID
CLASS: foodNode
Creates the nodes of the graph

INTERFACE
USES

None

VARIABLE

id: String
Determines the category of the food from which food group

idcat: String
The category the food belongs in

servSize: String
Amount of servings needed

foodName: String
The name of the food item

eWeight: Double
The weight of edge

ACCESS VARIABLES

foodNode(info: String[])
Initializes all variables to the proper information by going through the string array info and declaring each index of the array to the correct variable.

foodNode(groupNode: ArrayList<foodNode>)
Another constructor, initializing a given group node using a string array.

MIS
CLASS: Recorded
Keep track of the meal plans that have already been generated

INTERFACE
USES
idServings
foodEdge
foodNode
VARIABLE
existing: ArrayList<String[]>
ArrayList to hold all of the existing meal plans that have already been created

ACCESS VARIABLES

add(x: idServings)
Determines if a meal plan has already been created

MID
CLASS: Recorded
Keep track of the meal plans that have already been generated

INTERFACE

USES

idServings
foodEdge
foodNode

VARIABLE

existing: ArrayList<String[]>
ArrayList to hold all of the existing meal plans that have already been created

ACCESS VARIABLES

add(x: idServings)
Determines if a meal plan has already been created by getting the minimum of each food group edges, store the meal plan and checks to see if it exists already with a boolean statement. If the meal plan does not already exists in the arrayList, it will be added to the array.

MIS

CLASS: idServings

Generates graph based on number of servings per age group , gender and choice of meal.

INTERFACE

USES

To generate a minimized graph
foodEdge
foodNode

VARIABLE

id:String[ ]
holds name of id

idcat:String[ ]
number associated with id

servingsize:String[ ]
stores serving size

food:String[ ]
stores food in an array of string

createNode:String[ ]
stores the name of string of arrays

allNodes:foodNode[ ]
stores the foodnode group of arrays

allFruitNVeg:ArrayList<foodNode>
stores the foodNode specifically for vegetables group of array of strings

allDairy:ArrayList<foodNode>
stores the foodNode specifically for dairy group of array of strings

allMeat:ArrayList<foodNode>
stores the foodNode specifically for dairy group of array of strings

allGrain:ArrayList<foodNode>
stores the foodNode specifically for dairy obv

edgeWeights:String[]
The cost of using a food type

meatEdges:ArrayList<foodEdge>
stores the all the meat graph

vegeEdges:ArrayList<foodEdge>
stores the all the vege graph

dairyEdges:ArrayList<foodEdge>
stores the all the dairy graph

grainEdges:ArrayList<foodEdge>
stores all the grainEdges

ACCESS PROGRAMS

idServings(mActivity: AppCompatActivity):
initializes Servings data

randomServeSize()
get Serve size randomized in data structure

createEdges
creates edges between food nodes

getIndexOfMin(data: ArrayList<foodEdge>)

get the minimum edge of the arraylist for cheapest path

MID
CLASS: idServings
To generate a minimized graph

INTERFACE
USES
foodEdge
foodNode
VARIABLE
None

ACCESS PROGRAMS
readFile(fileName: String)
reads file in and puts them in appropriate data structures

MIS

CLASS: ServingPerDay

INTERFACE

USES: MyActivity, idServings, Recorded

VARIABLES:

Servings: ArrayList <String[]>

Holds all lines of information from the servings_per_day_day-en_OPP.csv file.

ServSizes: ArrayList <String[]>

Holds all serving sizes for each food from the servings_per_day_day-en_OPP.csv file.

foodgroup: String[]

Holds all food groups inside as strings from the servings_per_day_day-en_OPP.csv file for later access.

gender: String[]

Holds both genders in a string array from the servings_per_day_day-en_OPP.csv file for later access.

agegroup: String[]

Holds all age groups in a string array from the servings_per_day_day-en_OPP.csv file for later access.

AgeRange: ArrayList <Integer[]>

Stores all age ranges from the servings_per_day_day-en_OPP.csv file for later access in an integer array AgeRange[0] = lower bound, AgeRange[1] = higher bound.

numberofservings: String[]

Stores all number of servings needed in a string array from the servings_per_day_day-en_OPP.csv file

AppCompatActivity myActivity

Allows the text file be read from android studio.

ACCESS PROGRAMS

ServingPerDay(AppCompat(Activity mActivity))

Create the ServingPerDay ArrayList of String arrays from all lines in the text file. Used as input for the ArrayList<String[]> constructor.

ServingPerDay(ArrayList<String[]> Servings)

Initializes the stored information from the text file.

Recommended (String sex, int age) throws IOException: String

Returns the meal plan for the user as strings from the file with serving sizes.

MID

IMPLEMENTATION: ServingPerDay

USES: MyActivity, idServings, Recorded

VARIABLES

None.

ACCESS PROGRAMS

readFile(String filename): ArrayList<String[]>

Reads the file and returns the lines as an ArrayList of String arrays for later access. Also creates a random value to add to the end of each line that indicates the weight of all incoming edges.

Uses Relationship

LoginActivity

The loginActivity mainly works with two other components, depending on if the user already has an existing account. If the user already has an account, they can login and will be directed to the main page (mainActivity). If not, the user will go from the login page (where they do nothing) to the signup page, and sign up. With the loginActivity, the user's information and personal meal plan can be retrieved from the databases. Refer to Figure 2 for the LoginActivity uses relationship.

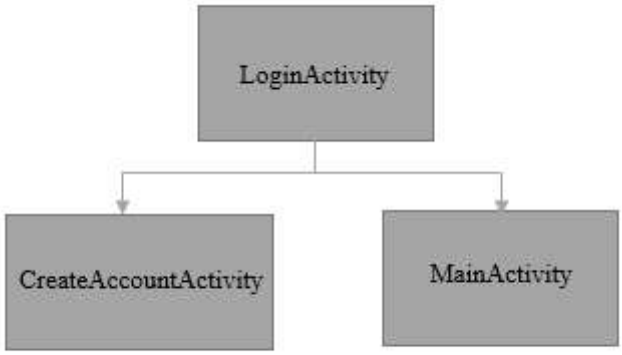

Figure 2: The LoginActivity Uses Relationship

CreateAccountActivity

When a user decides to create an account, they are directed to the ProfileSetup page. This will ask for the basic information of the user, their name, age, weight, height and what province they live in. All of this is necessary for the BMI that will be calculated later on. After the profile is complete, datasetTraversal is used to take in the data input from the user. It returns the data as strings and then then uses a class with shellsort, to sort the BMI dataset. The BMI is then calculated and the user is then directed to the main page (MainActivity). This deals with the account sign up, for users that do not have an existing account. Refer to Figure 2.1 for the CreateAccountActivity uses relationship.

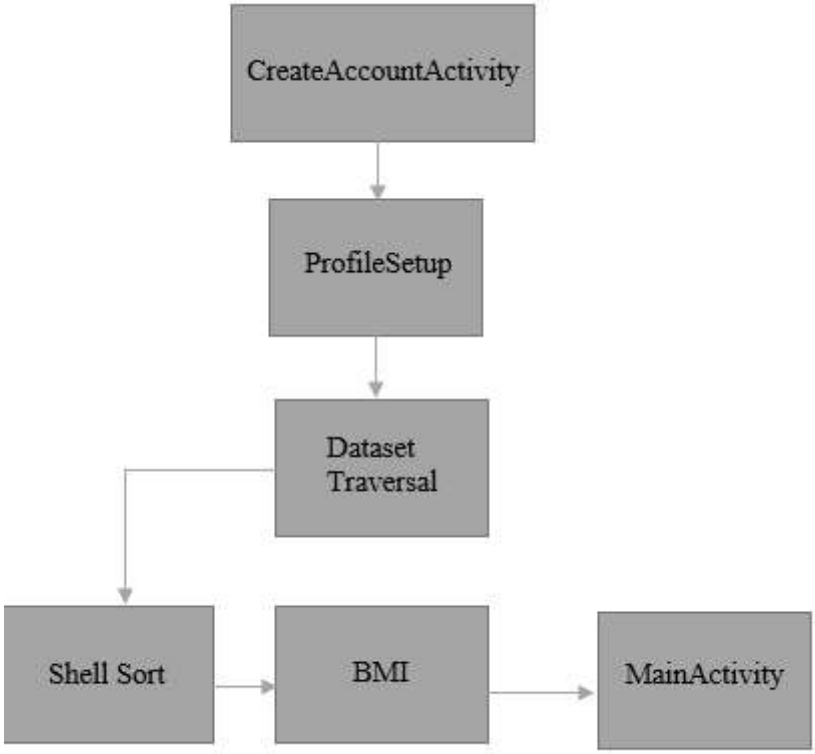

Figure 2.1: The CreateAccountActivity Uses Relationship

MainActivity

The main page is essentially the main component of the app. It uses four other components; the about page, the calendar, the search function and the settings page. From the main page, users can go to the about page to learn more about the app. They could check the calendar for the daily meal plan, or even search up a specific food item to learn more about the nutritional information with the search function. Users can also update any information about their body measurements in the settings. The MainActivity page is where all of the main functions can be found. Refer to Figure 2.2 for the MainActivity uses relationship.

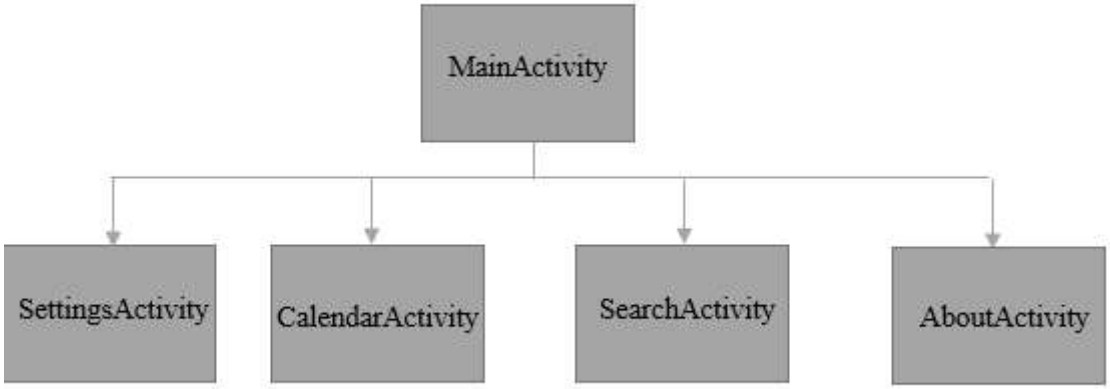

Figure 2.2: The MainActivity Uses Relationship

CalendarActivity

The CalendarActivity uses the class idServings, which is the main class of the graph. The graph class is made up of three other classes, FoodNode, FoodEdge and ServingsPerDay. FoodNode holds all of the nodes of the graph, where each node is a food item. FoodEdge creates the directed edges that connect vertices to each other. The class ServingsPerDay deals with the number of servings an individual should intake, based on their age and gender. After the graph has been made, the MealOfTheDayActivity is used to display the generated meal plan on each day of the calendar. Refer to Figure 2.3 for the CalendarActivity uses relationship.

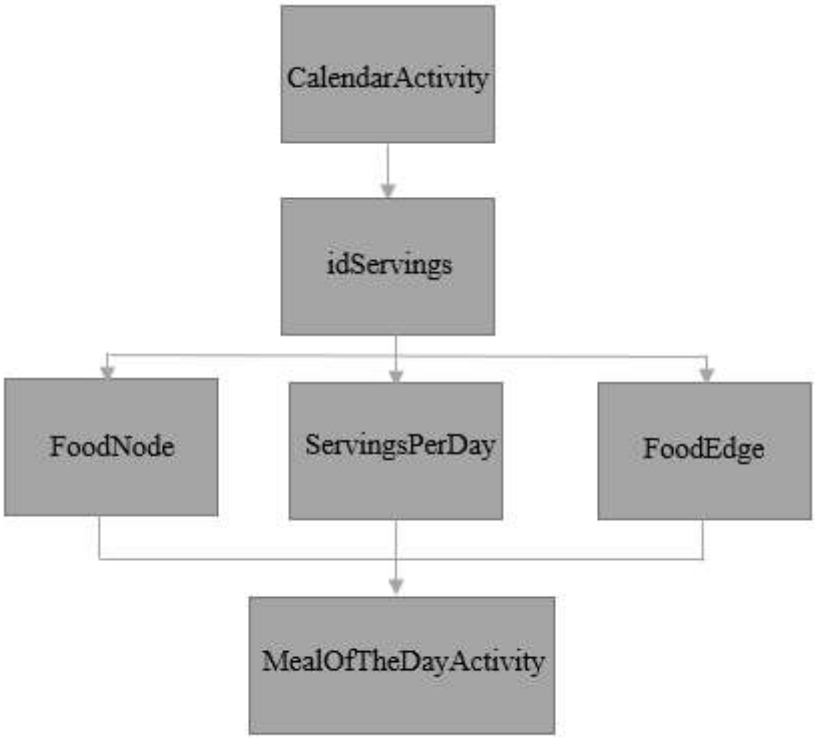

Figure 2.3: The CalendarActivity Uses Relationship

Traceability

Requirement	**Module**
Login/Signup	LoginActivity (onCreate) + CreateActivity (onCreate) + ProfileSetUp (onCreate)
Generate meal plan	CalendarActivity (idServings)
Generate calendar	CalendarActivity (onCreate)
Update BMI	BMI (calculate)
Returns information of a searched item	searchActivity(onCreate)

Evaluation of The Design

Overall Design

Overall, our design works best for what we had in mind. It achieves everything we wanted it to do, in a simple manner. We obviously could have done this as a webpage, but we decided to do an app

because individuals tend to lean more towards their smartphones and apps. In the end, we chose to use Android Studio for our platform, which also uses Java. We found that using Android Studio to make the GUI was a lot simpler than JavaFX, but all of the algorithms are done with Java and incorporated into the GUI.

Pros of the Design

We tried to make it as simple as possible, while still being functional -- it is very easy to use for anyone. Users can easily sign up or sign in to their existing account. There is only three buttons on the main page, one of which leads to more information about the app, another to a calendar and the last one prompts for user input. All of which are very simple and easy to navigate through.

Cons of the Design

One issue that we ran into was with the data sets. It is not being updated yearly so it will not be accurate as time passes. There is also an issue when a user is searching for an item with the search function. Users must input the name of the item exactly as stated in the data or else the item will "not be found", even if the item is actually in the dataset. For example, to search up summer sausage, users must input "summer sausage, beef" as that is how it is in the dataset. We did this because there are some food items that have multiple entries, but with different types. We also made the function to not be case sensitive.

This app is also tailored to Canadian citizens only, as the data found only provides BMI data for the provinces of Canada. It would not be functional for those outside of the country. We also realized that it wasn't practical to use a text file to store the user's account (which is stored locally to the phone). We only decided to use a text file for this project because it was easier to work with for the group. We originally had the idea to allow uses to omit certain foods if they so choose to, but in the end we were not able to implement this function because time constraints.

Test Report

We tested out our application by having family members play around with the app. We asked them to sign up, fill in the necessary information and play around with the main page. They were instructed to try to log in with an existing account as well and perform the same tasks.

Case	**Expected**	**Actual**	**Pass/Fail**
1.0 User can login	User will be directed to main page	Nothing happened	Fail

1.1 User can login	User will be directed to main page	User was directed to main page -- text file created to store information	Pass
2.0 User can sign up	User will go from login page to sign up page, shows stats page where users can input their information	Sign up page comes up, stats page shows up User can input required information to start	Pass
3.0 Calculates user's BMI with personal input	The user's BMI will be calculated upon new sign up	"BMI is: ____" shows up after	Pass
4.0 Generates meal plan	Meal plan is created	Wrong information of serving size, errors	Fail
4.1 Generates meal plan	Meal plan is created	Meal plan generated, gives necessary serving sizes and suggests food items	Pass
5.0 Buttons work on the main page	Users will be directed to corresponding pages	Clicked about page, shows more information about us and the app Did this for every button -- all passed	Pass
6.0 Search function returns information about item	Nutritional information about food item is returned	Searched item was said to be not found, even though it is in the dataset "Sorry! Food was not found" upon searching "skim milk"	Fail
6.1 Search function returns information about item	Nutritional information about food item is returned	Searched up "skim milk powder" as stated in the dataset: Skim milk powder Measure: 5ml Weight: 1g etc...	Pass