Project Testing Document

Team Red (Student Nutrition Mobile App)

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

Date: 2 December 2016

1. Unit Tests

1.1. AppState class

1.1.1. Required information

1.1.1.1. Set up an AppState object to be used for all tests

```
AppState testState = new AppState();
```

1.1.2. AppState class Methods to Test

1.1.2.1. getListMenuItems() : List<MenuItem>

Test name	ID
testGetListMenuItems()[i]	001

1.1.2.2. setListOfMenuItems(items : List<MenuItems>)

Test name	ID
testSetListMenuItems()[ii]	001

1.1.2.3. constructListOfMenuItems(venue : string)

Test name	ID
testConstructListOfMenuItems_PandaExpress()[iii]	001
testConstructListOfMenuItems_PizzaHut()	002
testConstructListOfMenuItems_Nonexisting()[iv]	003

1.1.2.4. getUserProfile(): UserProfile

Test name	ID
testGetUserProfile()[v]	001
?	

1.1.2.5. setUserProfile(profile : UserProfile)

Test name	ID
testSetUserProfile()[vi]	001

1.1.2.6. constructUserProfile()

Test name	ID
testConstructUserProfile()[vii]	001

1.1.2.7. getListOfVenues() : List<Venue>

Test name	ID
testGetListOfVenues()[viii]	001

1.1.2.8. setListVenues(list : List<Venues>)

Test name	ID
testSetListVenues ()[ix]	001

1.1.2.9. constructListVenues()

Test name	ID
testConstructListVenues()[x]	001

1.1.2.10. getRecommendationsList() : List<Recommendation>

Test name	ID
testGetRecommendationsList()[xi]	001

1.1.2.11. setRecommendationsList(list : List<Recommendations>)

Test name	ID
testSetRecommendationsList()[xii]	001

1.1.2.12. constructRecommendationsList()

Test name	ID
testConstructRecommendationsList()[xiii]	001

1.1.2.13. getSystemTime() : Date

Test name	ID
testGetSystemTime()[xiv]	001

Test will be performed later once we include this information in the database. Not in this version of the app.

1.1.2.14. sortMenuItems()

Test name	ID
testSortMenuItems()[xv]	001

1.1.2.15. sortMeals()

Test name	ID
testSortMeals()[xvi]	001

1.1.2.16.sortVenues()

Test name	ID
testSortVenues()[xvii]	001

1.1.2.17.exportUserProfileTo()

Test name	ID
testExportUserProfileTo()[xviii]	001

1.2. RecommendedDailyAllowance class

1.2.1. Required information

1.2.2. RecommendedDailyAllowance class methods

1.2.2.1. getAdjustedRDA(calories : int): List<NutrientTotal>

Test name	ID
testGetAdjustedRDA1000()[xix]	001
testGetAdjustedRDAZero()[xx]	002
testGetAdjustedRDANegative()[xxi]	003

1.3 **NutrientCalculator class**

1.3.1. Required information

1.3.1.1. Set up an NutrientCalculator object to be used for all tests

```
NutrientCalculator testCalculator = new NutrientCalculator();
```

1.3.2. NutrientCalculator class Methods to Test

1.3.2.1. getRecommendations(num : int) : List<MenuItem>

Test name	ID
testgetRecommendationsCorrectReccomendations() Will determine if the method returns the correct recommendations for the nutrient amount entered. This will likely require a test venue with specific test meals to know exactly what meal is expected to be returned. This may also require several separate tests to test each nutrient for recommendation.	001
testgetRecommendationsLessThanZero() Will test to make sure a correct error is thrown. It may be smart to just have the method round up to 0 in this case.	002

1.3.2.2. getListDates(numDaysPriorToDay : int) List<Date>

Test name	ID
testgetListDates() Will create a list of Date objects that is equal to the list that is to be returned by the getListDates method. These dates should be hardcoded in to ensure that the test is running correctly so will need to be checked to ensure that they correct for the day of the test.	001

1.3.2.3. getPDV(nutrientTotal : NutrientTotal) double

Test name	ID
testgetPDV() Will determine that the Percent Daily Value for a certain Nutrient is correct. This may require testing each nutrient individually. A certain double value would be hardcoded in to ensure that the getPDV value returned is correct.	001

1.3.2.4. getNutrientTotals(fromDate : date, toDate : Date) : List<NutrientTotal>

Test name	ID
testgetNutrientTotals() Will determine if the List of NutrientTotal objects is correct by hardcoding in an expected List of NutrientTotal objects and comparing it to the results of the method.	001
testgetNutrientTotalsDatesInWrongOrder() Will ensure that the method throws an error if the toDate comes before the fromDate	002

1.3.2.5. getRDANutrientTotals(numDays : int) List<NutrientTotal>

Test name	ID
testgetRDANutrientTotalsCorrectList() Will compare a hardcoded list to that which is returned by the getNutrientTotals method. This tests that the lists are identical in both the content and length of the lists.	001

1.3.2.6. lowestNutrients(numLowest : int) List<Nutrients>

Test name	ID
testlowestNutrientsCorrectLowest() Using a hardcoded list of Meal objects this method will test that the list of Nutrients returned is equal to what is expected.	001

1.3.2.7. getAllMealEntries(date : Date) List<MealEntry>

Test name	ID
testgetAllMealEntries() Will compare a hardcoded list of meal entries to the list of MealEntry objects returned by this method.	001

1.3.2.8. sortNutrientTotals() : List<NutrientTotal>

Test name	ID
testsortNutrientTotalsCorrectLlist() Will compare a hardcoded list of NutrientTotals to the list of NutrientTotal objects that is returned by this method.	001

1.3.3 Database Unit Testing

1.3.3.1 getVenues()

Test name	ID
testGetVenues() Sends mysql statement into the database which looks for certain columns, such as name. checks if the statement fails. Can be expanded for whatever columns should be checked for	001

1.3.3.2 getMeals

Test name	ID
testGetMeals() Sends mysql statement into the database which looks for certain columns, such as serving_size or calories. checks if the statement fails. Can be expanded for whatever columns should be checked for	001

1.3.3.3 no Specific function it tests

Test name	ID
testCheckDuplicate Looks for any duplicate data entries in DB. If finds duplicates, will let user know	001

2. Code Review

2.1. App Engine

11/20/16 by Ryan Nelson

General

- Need to implement Serializable interface on all classes within UserProfile, so the profile can be stored in file as an object.
- Bad equals() method implementation (see below for example of how it should be done)
- Mitch needs help finishing his classes. His are the most important, and none of it is there. This is of HIGHEST PRIORITY possible. Without these classes, we have nothing.

Class	Method	Comment	Priority
Donovan, Recommendation	getNutrient() setNutrient(Nutrient n)	Both are missing.	*****
Donovan, Comparator <NutrientTotal>	Missing	This class is missing. We need to be able to compare nutrient total objects.	***
Donovan, Date interface	serializable	Make serializable	*****
Donovan, MealDate	toLongString()	Needs to have the following format: DD/MM/YYYY	*****
Donovan, MealDate	toString()	Use the format from original toLongString() “Year: “ + year + …	***
Donovan, MealDate	Equals()	Write this method	***
Donovan, NutrientTotal	serializable	Make serializable	*****
Donovan, NutrientTotal	Equals()	Re-write. This is not the proper way to write an equals() method (see below)[i]	***
Donovan, NutrientTotal	hashCode()	Remove this method	*

Donovan, Standard Recommended DailyAllowance	getProteinLevel() getCalciumLevel() …	Let's add a getter for each of the fields in the class.	*****
Donovan, Standard Recommended DailyAllowance	PROTIEN_STANDARD	Should be spelled PROTEIN	*****
Donovan, Standard Recommended DailyAllowance	Class	Class must implement RecommendedDailyAllowance interface. Must include all methods described there.	*****
Donovan, Venue	serializable	Make serializable	*****
Donovan, Venue	toString() equals()	Write these methods	***
Donovan, Venue	memberName field	Remove memberName	*
Jordan, Comparator <MenuInterface>	Missing	We need to be able to sort menuItems by category. Category order (for reference): 1 Appetizer 2 Entree 3 Soup 4 Salad 5 Side 6 Drink 7 Dessert	***
Jordan, Meal	getMealEntries()	Should return mealEntries object.	*****
Jordan, Meal	setMealEntires(List<MealEntry> list)	Should set mealEntries object	*****
Jordan, Meal	toString() equals()	Would be good to have.	***
Jordan, MenuItem	Serializable	Should implement serializable interface	*****
Jordan, MenuItem	Overloaded Constructor	Should take List<NutrientTotal>,	*****

		not single NutrientTotal. I believe I fixed this already, but check your version.	
Jordan, MenuItem	Equals()	Not proper (see below)	***
Jordan, MenuItem	Equals() toString()	Should have these to be able to compare menu items	***
Jordan, MenuItem	Constructor	Remove this	*
Jordan, MenuItem	hashCode()	Remove this method	*
Jordan, MenuItem	getNutrientTotals()	Pulls list, not single nutrientTotal. You have it right.	*
Jordan, Nutrient	Class	Class shouldn't extend Menuitem.	*****
Jordan, Nutrient	serializable	Make serializable	*****
Jordan, Nutrient	Equals()	This is not how you write a proper equals method. (See below). Also, something is wrong with the method too. Need this method to work for comparison purposes.	***
Jordan, Nutrient	hashCode()	Remove this method	*
Jordan, StandardUserProfile	getMealsFor(Date date)	Should compare only day of the date object, not time. Time won't be the same for every meal that day.	*****
Jordan, StandardUserProfile	serializable	Make serializable	*****
Jordan, StandardUserProfile	StandardUserProfile(int calories, List<Meal> meals)	Overloaded constructor 1 method should include setting calorie and meals	****
Jordan,	StandardUserProfile(int calories,	Overloaded constructor 2 method	**

StandardUserProfile	List<Meal> meals, int weight, int height, int gender, int age, int activity level)	should include all fields	
Jordan, StandardUserProfile	All methods	Put methods in correct format (header one line, body below)	*
Jordan, UserProfile Interface	serializable	Make serializable	*****
Mitch, StandardAppState	All methods	None of the methods are supported yet. Finish the methods! (See note below)[ii]	*****
Mitch, AppState interface	constructListOfMenuItems(String venue)	Add method to class pulls data of menu items from specific venue on the server, and constructs a list of menu items from it	*****

Mitch, AppState interface	constructUserProfile()	Add method to class
		//pulls data from txt file where current userprofile object is stored
		//if no user profile data stored, creates new user profile object to be used in the app
			*****

Mitch, AppState interface	constructRecommendationsList()	Add method to class //pulls data of recommendations from server, and constructs list of recommendations from it	*****
Mitch, AppState interface	constructListVenues()	Add method to class //pulls data of venues from server and constructs list of venues from it	*****
Mitch, Calculator interface	getRecommendations(int num, AppState state)	AppState is needed so calculator can get recommendations from somewhere	*****
Mitch, Calculator interface	getPDV(NutrientTotal nutrientTotal,	Rda is needed so calculator can compare nutrient totals against a	*****

	RecommendedDailyAllowance rda)	standard.	
Mitch, Calculator interface			
Mitch, MealEntry	Class	Does not extend menu item	*****
Mitch, MealEntry	Serializable	Should implement serializable interface	*****
Mitch, MealEntry	Equals()	Not the proper way to write an equals method (see below)	***
Mitch, MealEntry	hashCode()	Remove this method	*
Mitch, MenuInterface	Serializable	Should implement serializable interface	*****
Mitch, MenuInterface	Equals() toString()	Should have these to be able to compare menu items	***
Mitch, NutrientCalculator	getRecommendations(int num, AppState state)	See note below[iii]	*****
Mitch, NutrientCalculator	getListDate(int numDaysPriorToToday)	See note below[iv]	*****
Mitch, NutrientCalculator	getPDV(NutrientTotal nutrientTotal, RecommendedDailyAllowance rda)	See note below[v]	*****
Mitch, NutrientCalculator	getNutrientTotals(Date from, Date to, UserProfile profile)	See note below[vi]	*****
Mitch, NutrientCalculator	Other methods	These are just suggestions of how I would go about creating helper methods to facilitate	*

		implementation of the public methods. These helper methods should be private!	
Mitch, Recommended DailyAllowance interface	getProteinLevel() getCalciumLevel() …	Let's add a getter for each of the fields in the class.	*****
Mitch, Recommended DailyAllowance interface	Okay	okay	-

---

[i]

```
@Override
public boolean equals(Object o)
{
	if(  !(o instanceof Auto) )
	{
		return false;
	}
	else
	{
		Auto objAuto = (Auto) o;		//typecast to gain access to private fields of Auto class

		boolean b1 = model.equals( objAuto.model);
		boolean b2 = milesDriven == objAuto.milesDriven;
		boolean b3 = Math.abs(gallonsOfGas - objAuto.gallonsOfGas) < 0.0001

		if(b1 && b2 && b3)
		{
			return true;
		}
		else
		{
			return false;
		}
	}
}
```

[ii] StandardAppState class methods
getter and setter for each field, easy to understand

constructListMenuItems(String venue)
/*
This method needs to talk to SQL server, and for a particular venue, pull all the menu item data, parse the data and construct menu item objects, and add these objects to the menuItems field.
*/

constructUserProfile()
/*
This method needs to determine the location of the txt file storing the user profile object, reading the file, parsing the object that is there (ObjectInputStream). If there is no txt file there, recognize that no user profile exists, and create a default user profile to store in the user profile field.

If there is a txt file, read the object, and let the method construct the user profile object. Store in user profile field.
*/

constructListVenues()
/*
This method needs to talk to the SQL server, and get venue data, then parse the data into venue objects, and store these objects in the field venues.
*/

constructRecommendationList()
/*
This method needs to talk to the SQL server, get recommendation data stored there, and parse the data as recommendation objects, and store these objects in the field recommendationsList. If there are no recommendations, shouldn't fail. List should simply remain empty.
*/

getSystemTime()
//should check the time of the system and construct and return a Date object

sortMenuItems()
/*sort menuItem field by category
Category order (for reference):
1 Appetizer
2 Entree
3 Soup
4 Salad
5 Side
6 Drink
7 Dessert
*/

sortVenues()
//alphabetical order

sortMeals()
//sort by date, then time

exportUserProfileTo(String filename)
/*
Should use ObjectOutputStream to save the user profile object in a txt file. It should overwrite the existing data in the file, if there is a user profile already stored.
*/

[iii]
AppState provides the list of possible recommendations to choose from.
Num is the number of recommendations needed for the app. (Could be 3, or 2, etc.)
This method needs to determine the lowest (and second lowest, and third lowest, for as many as are needed to reach the required number) percentage daily values out of the nutrients, then get the correct recommendation objects for those particular nutrients from the AppState class, and add them to a list to output.

If no recommendation exists in the recommendation list to match the nutrient (or list is empty), should return an "Error" recommendation (Title: "Processing Error", Message: "No available recommendation right now. Sorry!")

[iv] numDaysPriorToToday is the number of days to add to a list to output.
This method needs to see today's date, then determine the dates of each day previous to the current, for the number required. Once the date is determined, create a Date object and add to list. Output the list.

[v] PDV is percent daily value. Say you consumed 40 g of protein today. And you are required (recommended daily allowance) to consume 50 g for a single day. Your PDV for protein is 0.8 (or you have consumed 80% of your daily need for this nutrient).

nutrientTotal contains the nutrient you are choosing to calculate PDV for, and the amount is needed for comparison against the recommended daily allowance.

RecommendedDailyAllowance provides the nutrient value standards you are comparing against.

[vi] getNutrientTotals() goes through all meal data a given profile has recorded in the given date range, and calculates a sum total for each nutrient.

This method will have to take a user profile, go through all its recorded meals for a given date range. The method will have to create a running total for each nutrient, then go through each MealEntry in every meal in the given date range, and for each MealEntry, multiply the amount of each nutrient in the menu item stored in the meal entry (amount in NutrientTotal in MenuItem) by the recorded serving (amount in MealEntry), then add that amount to the running total for the given nutrient. Do this for every meal entry in every meal, and for every nutrient.

The running totals are constructed as a NutrientTotal object, as well as the appropriate nutrient. These objects are added to a list for output.

[i] testGetListMenuItems()
Checks whether getter works properly after setting list of menu items.

[ii] testSetListMenuItems()
The method should not fail.

[iii] testConstructListMenuItems_PandaExpress()
Will determine if the menuItems field has a list with the correct number of menuItems for a chosen venue listed in the server database. For Panda Express, there are 46 items in the database.

```
testState.setListMenuItems("Panda Express");          //pulls from server and sets value for field
int expectedCount = 46;
int actualCount = testState.getListMenuItems().Count();
```

[iv] testConstructListMenuItems_Invalid()
Checks whether method validates input before pulling from server.

```
testState.setListMenuItems("Test 1");  //invalid location
```

[v] testGetUserProfile()
Checks whether method correctly pulls user profile object from class.

```
testState.setUserProfile();
UserProfile actualProfile = testState.getUserProfile();
```

[vi] testSetUserProfile()
Method should not fail.

[vii] testConstructUserProfile()
Checks whether method correctly constructs user profile object from database or text file. Needs to correctly convert object into user profile. Method should not fail.

[viii] testGetListOfVenues()

```
//create venues
StandardVenue venue1 = ("Panda Express");
StandardVenue venue2 = ("Pizza Hut");

List<Venue> expected = {venue1, venue2};
List<Venue> actual = testState.getListOfVenues();
```

[ix] testSetListVenues()
Method should not fail.

[x] testConstructListVenues()
Method should not fail. Should parse data from server.

appState.setListVenues();

[xi] testGetRecommendationsList()

Recommendation r1 = ("Low in potassium", "Try eating almonds, banana, ground beef, or broccoli.", "WebMD.com");

Recommendation r2 = ("Low in calcium", "Try eating some cheese, yogurt, milk, or dark leafy greens like spinach and kale.", "WebMD.com");

List<Recommendation> expected = {r1, r2};

[xii] setRecommendationsList()
Method should not fail.

Recommendation r1 = ("Low in potassium", "Try eating almonds, banana, ground beef, or broccoli.", "WebMD.com");

Recommendation r2 = ("Low in calcium", "Try eating some cheese, yogurt, milk, or dark leafy greens like spinach and kale.", "WebMD.com");

[xiii] testConstructRecommendationsList
Method should not fail.

[xiv] testGetSystemTime()
Check whether the method works with the computer system.

[xv] testSortMenuItems()
Items should be sorted alphabetically by category, then by name

First item should be Chicken Egg Roll at Panda Express.
StandardMenuItem expected = new StandardMenuItem("Chicken Egg Roll", 2.75, "ounces", new List<Nutrient> (), "Appetizer", "Panda Express website", "Panda Express");

appState.setListOfMenuItems("Panda Express");
StandardMenuItem actual = appState.getListOfMenuItems().SortMenuItems().First();

[xvi] testSortMeals()
Meals should be sorted by date, then by time

```
//create meals to add to user profile
Venue v1 = new Venue("Panda Express");
MealDate d1 = new MealDate(2016, 11, 14);
List<MealEntry> l1 = new List<>();

Venue v2 = new Venue("Carino's");
MealDate d2 = new MealDate(2016, 10,08);

Meal meal1 = new Meal(v1, d1, l1);
Meal meal2 = new Meal(v2, d2, l1);

//add to user profile
StandardUserProfile testProfile = testState.getUserProfile();
testProfile.addMeal(meal1);
testProfile.addMeal(meal2);

//sort meals
testState.SortMeals();

//check if correctly ordered
//first meal should be from 10/08/2016
Meal expected = meal2;
Meal actual = testProfile.getMeals().First();
```

[xvii] testSortVenues()
Pizza Hut should come after Panda Express.

[xviii] testExportUserProfileTo()
Should not fail.

[xix] testGetAdjustedRDA1000()
Values should be half of what they are currently.

[xx] testGetAdjustedRDAZero()
Zero shouldn't be allowed to be chosen as a value, because of divided by zero issues.

[xxi] testGetAdjustedRDANegative()
Negative values should be allowed