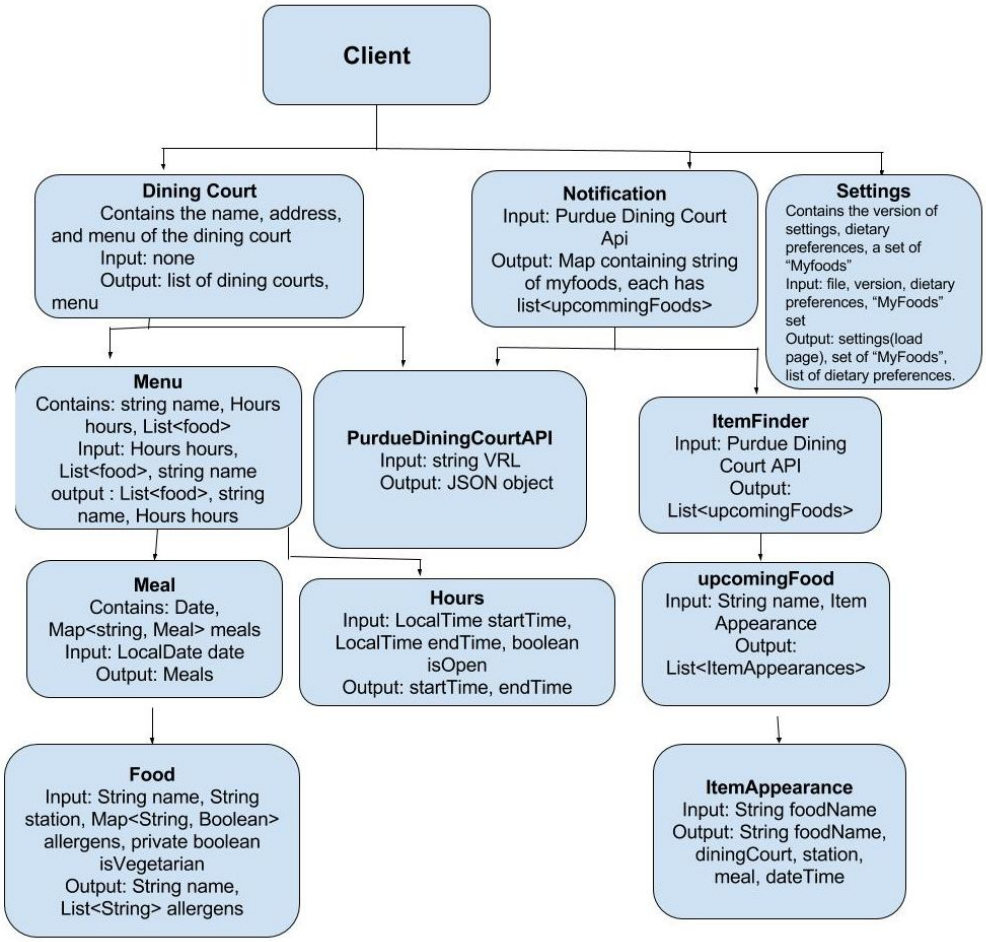
Sprint 2: Incremental and Regression Testing Log

Components:



Testing:

We decided to use the JUnit framework as well as the JWebUnit framework to automate our test cases. In order to test the functionality of our application, we had to make sure the API responses would remain consistent no matter how many times the test cases are run. To solve this problem, we implemented a MockDiningCourtAPI to redirect Purdue dining court API calls to return data from static files located in our test case folders. By creating a mock API, we were also able to broaden the scope of our test cases by modifying the mock API response files to test each and every feature that we parse. We strived to maximize the test case coverage to test out each unique feature after it was implemented. By using this automated test strategy, we would be able to not

only tell if a feature is broken but also tell whether a new feature breaks any of our previously implemented features. We used the JWebUnit framework in order to automate tests to the client interface. JWebUnit allows the tester to input http requests and verify that the returned response is what is expected. We also used a top-down testing strategy, as we felt this would be the best strategy to use in the case of our app, since we wanted to check the functionality of each individual component using stub modules.

Test Format:
Test Number
Description: Description of the test procedure
Test Result: (Passed|Failed)

Module Settings:

- Incremental Testing:

Settings001
Create a new Settings module. Call getDietaryExclusions().addAll(Arrays.asList({"fish", "shellfish", "peanuts", "vegetarian", "eggs"}). Call save(). Then call load(). Call getDietaryExclusions(). The strings returned should be "fish", "shellfish", "peanuts", "vegetarian", "eggs"
Test Result: Passed

Settings002
Create a new Settings module. Call getDietaryExclusions(). An empty collection should be returned.
Test Result: Passed

Settings003
Description: Call Settings.load(new File(DATA_DIR + "settings-load.json")). Call getDietaryExclusions() on the resulting Settings module. The preference components in the returned collection should match the values present in the “settings-load.json” file.

Test Result: Passed

Settings004
Description: Call Settings.load(new File(DATA_DIR + "settings-save.json")). Call getDietaryExclusions() on the resulting Settings module. Add string “fish” to the returned collection. Call save() from the Settings module. The “settings-save.json” file should contain the string value for “fish”..
Test Result: Passed

Settings005
Description: Call Settings.load(new File(DATA_DIR + "settings-save.json")). Call getMyFoods() on the resulting Settings module.From the resulting HashSet, call add("Blueberry"), add("Pizza"), add("Corn bread"), and add("Chili"). Call save() from the Settings module. The “settings-save.json” file should contain the true/false values for the keys changed.
Test Result: Passed

Module Food:

- Incremental Testing:

Food001
Description: Create a new Food module. Call addAllergen(“eggs”, true) and addAllergen(“gluten”, true). Call getAllergens(). A list should be returned containing “eggs” and “gluten”.
Test Result: Passed

Food002
Description: Create a new Food module. Call getAllergens(). An empty list should be returned.

Test Result: Passed

Defect No.	Description	Severity	How Corrected
1	As an allergen is added or removed from the Purdue dining court API, the app would no longer work because the Food module contains separate boolean variables for each of the allergens that a food might have.	Important	The Food module was changed to have a dynamic hashmap of string keys that represent allergens that map to true/false values. The app will now adapt to changes in types of allergens on the Purdue dining court API.

Module ItemAppearance:

- Incremental Testing

ItemAppearance001
Description: Create a new ItemAppearance module with foodname “pizza”. Call getStation(), getDiningCourt(), getMeal() and getDateTime(). All functions should return an empty list.
Test Result: Passed

ItemAppearance002
Description: Create a new ItemAppearance module with foodname “pizza”. Call setDiningCourt(“Earhart”) and setMeal(“Lunch”). Call getStation(), getDiningCourt(), getMeal() and getDateTime(). getDateTime() and getStation() should return empty, whereas the other functions should return the values input.
Test Result: Passed

- Regression Testing

Defect No.	Description	Severity	How Corrected

1	Information on matched food items from the Purdue Dining Court API would be parsed and then inserted into a list.This setup became too complicated and did not work properly with the ItemFinder and Notification modules.	Important	Module ItemAppearance was created which contains variables for important information on a given food, which made it easier to parse a list of food item appearances in the Purdue Dining Court API and extract necessary information.

Module Hours:

- Incremental Testing

Hours001
Description: Create a new Hours module with startTime = 5:00 and endTime = 7:00. Call getStartTime(). The returned value should be 5:00. Call getEndTime(). The returned value should be 7:00.
Test Result: Passed

Hours002
Description: Create a new Hours module with startTime = 25:00 and endTime = 10:00. An exception should be thrown.
Test Result:Passed

Module PurdueDiningCourtAPI:

- Incremental Testing:

PurdueDiningCourtAPI001
Description: Create a new PurdueDiningCourtAPI module. Call getJSON() from the PurdueDiningCourtAPI module. Call toString() from the returned JSONObject. Print the results of the resulting string. The output should match the “locations.json” file in the Data folder.

Test Result: Passed

Module Meal:

- Incremental Testing

Meal001
Description: Create a new Food module with name “Pizza”. Create a new list and add the food module. Create a new Meal module with the name “Late Lunch” and the Food module. Call getFoods(). The function should return a list containing Food module with name “Pizza”.
Test Result: Passed

Meal002
Description: Create a new Hours module with hours 3:00 to 5:00. Create a new Meal module with the name “Late Lunch” and the Hours module. Call getHours(). The function should return an Hours module with starttime component 3:00 and endtime component 5:00
Test Result: Passed

Meal003
Description: Create a new Hours module with hours 3:00 to 5:00. Create a new Meal module with the name “Late Lunch” and the Hours module. Call getFood(). The function should return an empty list.
Test Result Passed

Module Menu:

- Incremental Testing:

Menu001
Description: Call getMenu(LocalDate.parse("2017-02-07")) using existing DiningCourt object with name component “Earhart”. Function should return a Menu object containing Meal modules with names “Breakfast”, “Lunch”, and “Dinner”.

Test Result: Passed

Menu002
Description: Call getMenu(LocalDate.parse("2017-02-02")) using existing DiningCourt object with name component “Earhart”. Call getDate() using the returned Menu module. The date returned should match “2017-02-02”. Call getMenu(LocalDate.parse("2017-02-07")) using existing DiningCourt object with name component “Earhart”. Call getDate() using the returned Menu module. The date returned should match “2017-02-07”.
Test Result: Passed

Menu003
Call getMeal(“Breakfast”) on existing returned Menu object with date component "2017-02-02". Returned Meal should include a list with a Food object with the name “Breakfast Polenta”.
Test Result: Passed

Module DiningCcourt:

- Incremental Testing:

DiningCourt001
Description: Call getDiningCourt(“Earhart”)from DiningCourt module. “Earhart” should be included in the returned list.
Test Result: Passed

DiningCourt002
Description: Call getDiningCourt(“Earhart”) from DiningCourt module. “Earhart” should be included in the returned list. “Wiley” should not be in the returned list. Call

getDiningCourt(“Wiley”) from DiningCourt module. “Wiley” should be included in the returned list. “Earhart” should not be in the returned list.
Test Result: Passed

Module UpcomingFood:

- Incremental Testing

UpcomingFood001
Create new UpcomingFood module with name “sausage”. Call getAppearances(). An empty list should be returned.
Test Result: Passed

UpcomingFood001
Create new UpcomingFood module with name “sausage”. Create Create a new ItemAppearance module with foodname “pizza”. Call setDiningCourt(“Earhart”) and setMeal(“Lunch”) from the ItemAppearance module. Call addAppearance() from the UpcomingFood module. Call getAppearances(). A list containing an ItemAppearance module with dining court name “Earhart” and meal “Lunch” should be returned.
Test Result: Passed

- Regression Testing

Defect No.	Description	Severity	How Corrected
1	UpcomingFood module returns one ItemAppearance module. This did not work properly with the ItemFinder module which depended upon complete data about the given food.	Critical	ItemFinder module changed to create a list of multiple ItemAppearance modules and passes that list into a new UpcomingFoods module.

Module ItemFinder:

- Incremental Testing

ItemFinder001
Create new ItemFinder module. Call searchUpcoming(“Stirfry”). A list of UpcomingFoods should be returned containing an ItemAppearance that corresponds to every appearance of “Stirfry” in the Data folder.
Test Result: Passed

ItemFinder002
Create new ItemFinder module. Call searchUpcoming(“MEMES”). An empty list should be returned.
Test Result: Passed

Defect No.	Description	Severity	How Corrected
1	Parsing food item appearances directly from the Purdue Dining Court API was very error prone and did not work properly.	Critical	ItemFinder module created to parse items from the Purdue dining court API and place them into objects for easy access to information on food items.

Module Notifications:

- Incremental Testing

Notifications001
Create new Notifications module. Create new Settings module. Call getMyFoodAppearances using the Settings module. A map of lists of UpcomingFood modules corresponding to each entry under “myFood” in the “settings-save.json” file in Data should be returned.
Test Result: Passed

Defect No.	Description	Severity	How Corrected
1	Notifications returned a list of ItemAppearances, but each item did not correspond to a given food. Therefore, the data could not be used to map foods to appearances in the Purdue dining court API.	Critical	UpcomingFoods module was created to store a food name as well as a list of each of the ItemAppearances it corresponds to.

Module Client:

- Incremental Testing

Client001
Description: Create a new Client module. Access the Client module at "/". Verify the title of tab using the XPath assertion. The assertion should throw no errors.
Test Result: Passed

Client002
Description: Create a new Client module. Access the Client module at "/". Verify the text "BoilerHungry" is present at the title using an assertion. The assertion should throw no errors.
Test Result: Passed

Client003
Description: Create a new Client module. Access the Client module at "/". Verify the presence of Earhart Title using the XPath assertion. Verify the presence of Ford Title using the XPath assertion. The assertions should throw no errors.
Test Result; Passed

Client004

Description: Create a new Client module. Access the Client module at “/”. Verify that the Dining court on the webpage has “active” class and is being displayed using the XPath assertion. Verify that other Dining court does not have “active” class and hence not being displayed using the XPath assertion. The assertions should throw no errors.
Test Result: Passed

Client005
Description: Create a new Client module. Access the Client module at “/”. Verify that the Dining court visible on web page is Earhart using the XPath assertion. Click on the View Menu for Earhart using XPath. Verify that the “location” label is present on the web page using the XPath assertion. Verify that the location label is present on the web page using the XPath assertion. Verify that the location of Earhart is correct using Text assertion. The assertions should throw no errors.
Test Result: Passed

Client006
Description: Create a new Client module. Access the Client module at “/”. Verify the presence of “Earhart” using the XPath assertion. Click on “View Menu” for Earhart. Verify Carousel is not present. Verify that the “location” label is present on the web page using XPath. Verify that the location label is present on the web page using XPath. Verify that the location of Earhart is correct using Text. Verify “Late Lunch” Text is not present. Click on “Home”. Verify the presence of the Carousel. Verify presence of Earhart as the displaying dining court. Click on “next”. Click on “View Menu”. Verify location of Ford on web page. Verify “Late Lunch” Text on web page. The assertions should throw no errors.
Test Result: Passed

Client007
Description: Create a new Client module. Access the Client module at “/”. Verify the presence of Earhart on the web page using XPath. Verify the presence of notification ticker on the web page using XPath. The assertions should throw no errors.
Test Result: Passed

Client008
Description: Create a new Client module. Access the Client module at “/”. Verify that Earhart is present using XPath. Verify that the pictures on the background are present using XPath. The assertions should throw no errors.
Test Result: Passed

Client009
Description: Create a new Client module. Access the Client module at “/”. Verify the presence of Settings tab on the web page using XPath. Verify that My Foods is not present on the web page. Verify that Dietary Preferences is not present on the web page. Click on the Settings to open the tab. Verify the presence of “My Foods”. Verify the presence of “Dietary Preferences”. The assertions should throw no errors
Test Result: Passed

Client010
Description: Create a new Client module. Access the Client module at “/”. Verify the presence of “Earhart” on the page. Verify the presence of the image on the background. Click on “next”. Verify the absence of Earhart on the page. Verify the presence of a new image on the page, confirming that new Dining court is visible on the page. The assertions should throw no errors.
Test Result: Passed