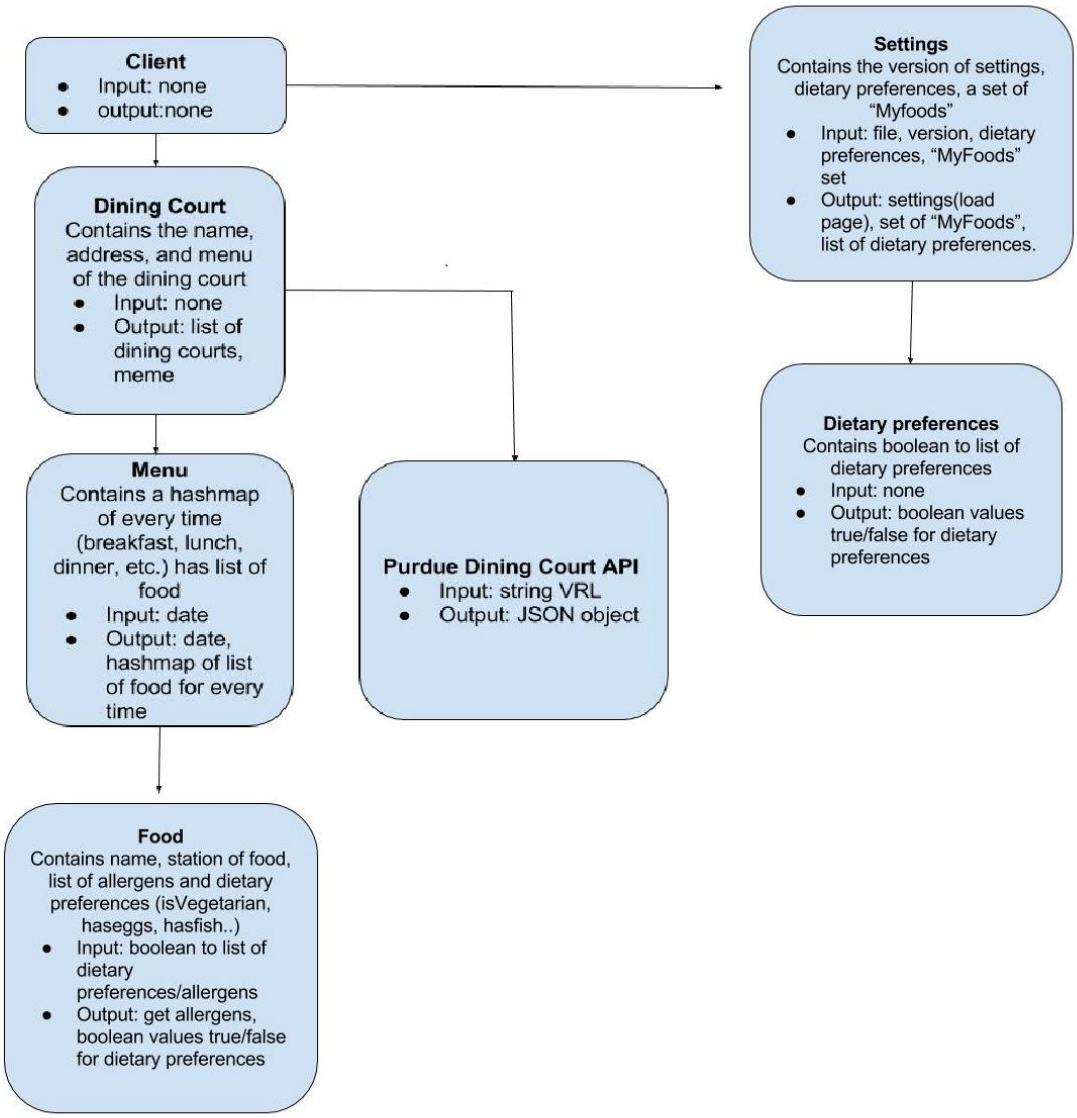
1.1) Components:



Testing

We decided to use the JUnit 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.
(Top-down)

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

Module DietaryPreferences:

- Incremental Testing:

DietaryPreferences001
Create a new DietaryPreferences module. Call setVegetarian(true), setNoFish(true), setNoMilk(true), setNoPeanuts(true), and setNoShellfish(true). Call the getter functions for the specified values changed. The values returned should all be true.
Test Result: Passed

DietaryPreferences002
Create a new DietaryPreferences module. Call the getter functions for all boolean components of the DietaryPreferences module. The values returned should all be false.
Test Result: Passed

- Regression Testing:

Module Settings:

- Incremental Testing:

Settings001
Description: Call Settings.load(new File(DATA_DIR + "settings-load.json")). Call getDietaryPreferences() on the resulting Settings module. The preference components in the returned DietaryPreferences module should match the true/false values present in the “settings-load.json” file.
Test Result: Passed

Settings002
Description: Call Settings.load(new File(DATA_DIR + "settings-save.json")). Call getDietaryPreferences() on the resulting Settings module. Call setNoFish(true), setNoShellfish(true), setNoPeanuts(true), setVegetarian(false), and setNoEggs(false) from the returned DietaryPreferences module. Call save() from the Settings module. The “settings-save.json” file should contain the true/false values for the keys changed.
Test Result: Passed

Settings003
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

Defect No.	Description	Severity	How Corrected
1	When adding a duplicate for “MyFoods” in Settings module, the duplicate shouldn’t be added to the json file	Critical	We changed using a List for MyFoods to using a Collection for MyFoods to store the information in the json file.

Defect No.	Description	Severity	How Corrected
1	Settings component needs to use a different library other than org.json to save preferences, otherwise coder needs to manually modify json file for each change to a user’s	Workaround	We changed the file so that it uses Google’s Gson library. The Gson library would provide automatic serialization and deserialization of POJOs (plain old java

	preferences. This is a very error prone coding style.		objects) through the use of reflection behind the scenes. Thus, we avoid the user error that could arise with manual parsing.

- Regression Testing:

Defect No.	Description	Severity	How Corrected
1	When Settings component uses a different library other than org.json to save preferences, Dietary Preferences component should be updated with new information after saving	Critical	Using the Gson library to automatically convert our objects to and from Json, and saves the dietary preferences list in a json file.

Defect No.	Description	Severity	How Corrected
1	Changing the type of MyFoods to a Collection type shouldn't affect the DietaryPreferences component.	Critical	After changing the type of MyFoods it doesn't change or affect the DietaryPreferences component.

Module Food:

- Incremental Testing:

Food001
Description: Create a new Food module. Call setEggs(true) and setGluten(true). Call the getter functions for all boolean components of the Food module. Only the return values for getGluten() and getEggs() should be true.
Test Result: Passed

Food002
Description: Create a new Food module. Call the getter functions for all boolean components of the DietaryPreferences module. The values returned should all be false.
Test Result: Passed

- Regression Testing:

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 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 list should include Food object with name “Breakfast Polenta”.
Test Result: Passed

Defect No.	Description	Severity	How Corrected

1	Module Menu holds number of meals types with number of foods, module Menu shouldn't contain a certain meal type (ex. Late lunch) if it isn't available for the chosen dining court.	Important	We return an empty list when no meal type is found for dining court.

- Regression Testing:

Defect No.	Description	Severity	How Corrected
1	Changing meal type for module Menu should not affect dining court information for DiningCourt module.	Critical	Created a function that would get meal hashmap of meal type and list of food without changing the given dining court.

Module DiningCourt:

- 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

Defect No.	Description	Severity	How Corrected
1	DiningCourt.java class contains dynamic arrays for only 3 different meal types (Breakfast, Lunch, Dinner), when there could be varying numbers of meals that a dining court can serve.	Important	Created module Menu.java to hold arbitrary number of meal times with an arbitrary number of foods using a dynamic list of dynamic lists.

- Regression Testing:

Defect No.	Description	Severity	How Corrected
1	Adding the Menu module shouldn't affect or change the dining court output.	Important	Dynamic List of Menus was added as a global variable in the DiningCourt module so that creating the Menu module doesn't change the contents of the rest of the DiningCourt module.

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

Defect No.	Description	Severity	How to Correct

1	Buffer overflow in the PurdueDiningCourtAPI.java file, should have a bound on number of characters it can process at once.	Workaround	We could theoretically use a BoundedInputStream from Apache Commons I/O library to put a maximum count on the number of bytes processed from an API call at once.

- Regression Testing:

Defect No.	Description	Severity	How to Correct
1	Buffer overflow in the PurdueDiningCourtAPI.java file, should not affect Dinning court functionality	Workaround	Use a BoundedInputStream from Apache Commons I/O library to put a maximum count on the number of bytes processed from an API call at once doesn't affect the functionality of dining court.