FoodBuddy - Food Recommendation System

Team Members:

Vikhyat Goyal

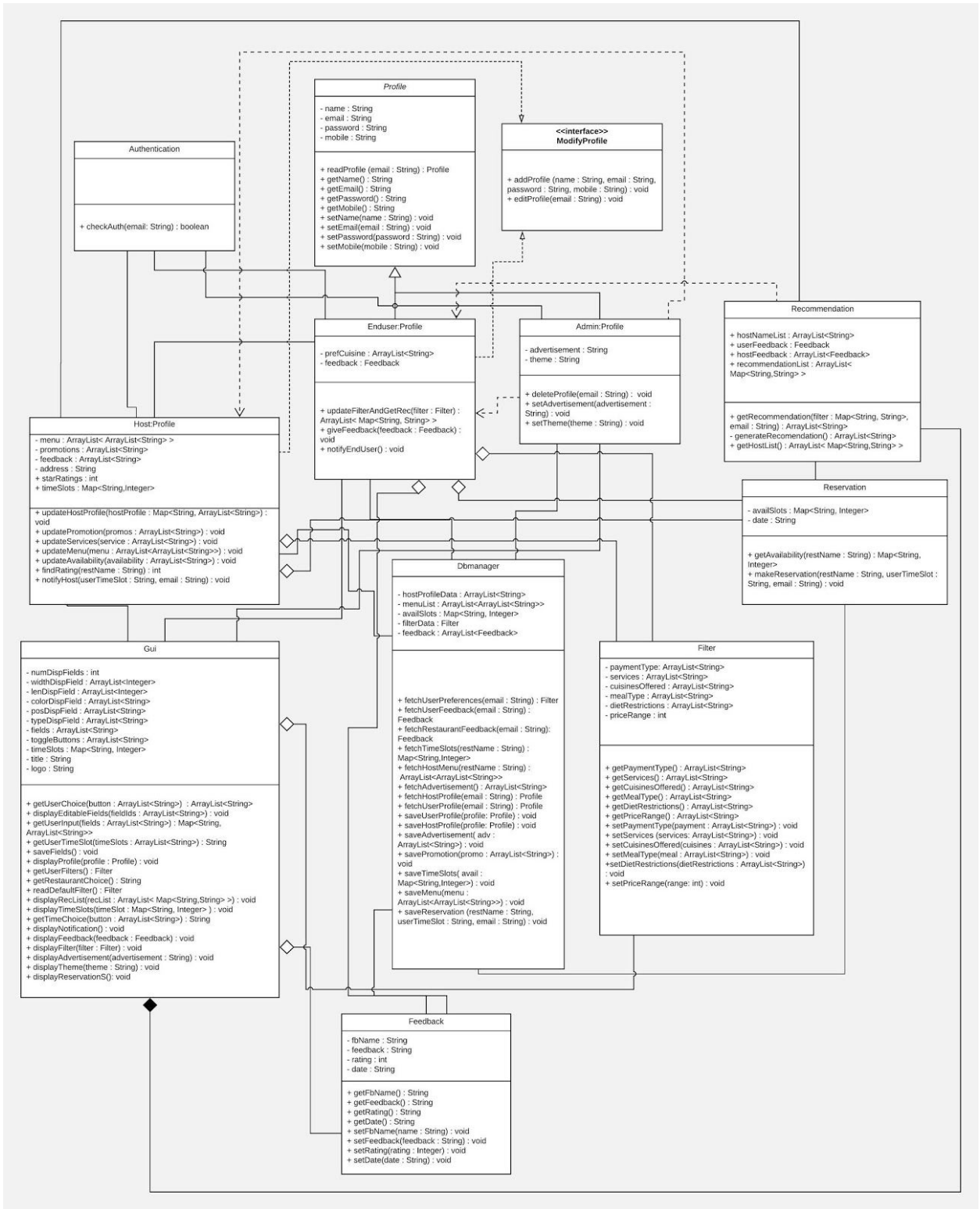
Pavan Dhareshwar

Rishi Soni

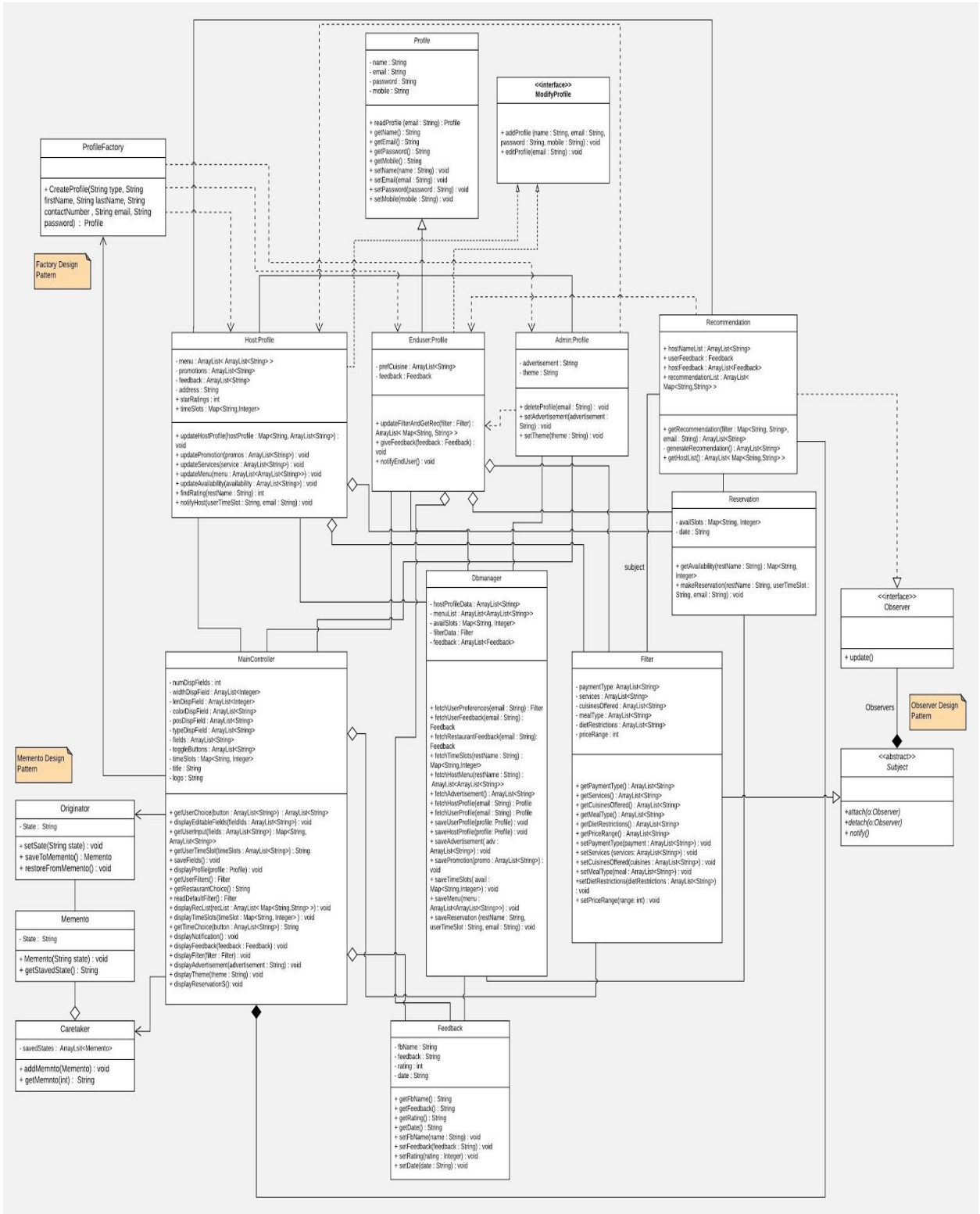
Summary:

Our system for the project will be a website that will enable end users to find new restaurants or food product to try based on their profile, past experiences and random projection. It will also allows new restaurants to get registered and promote their business, old restaurants and food product manufacturers to add new cuisines/dishes and get feedback.

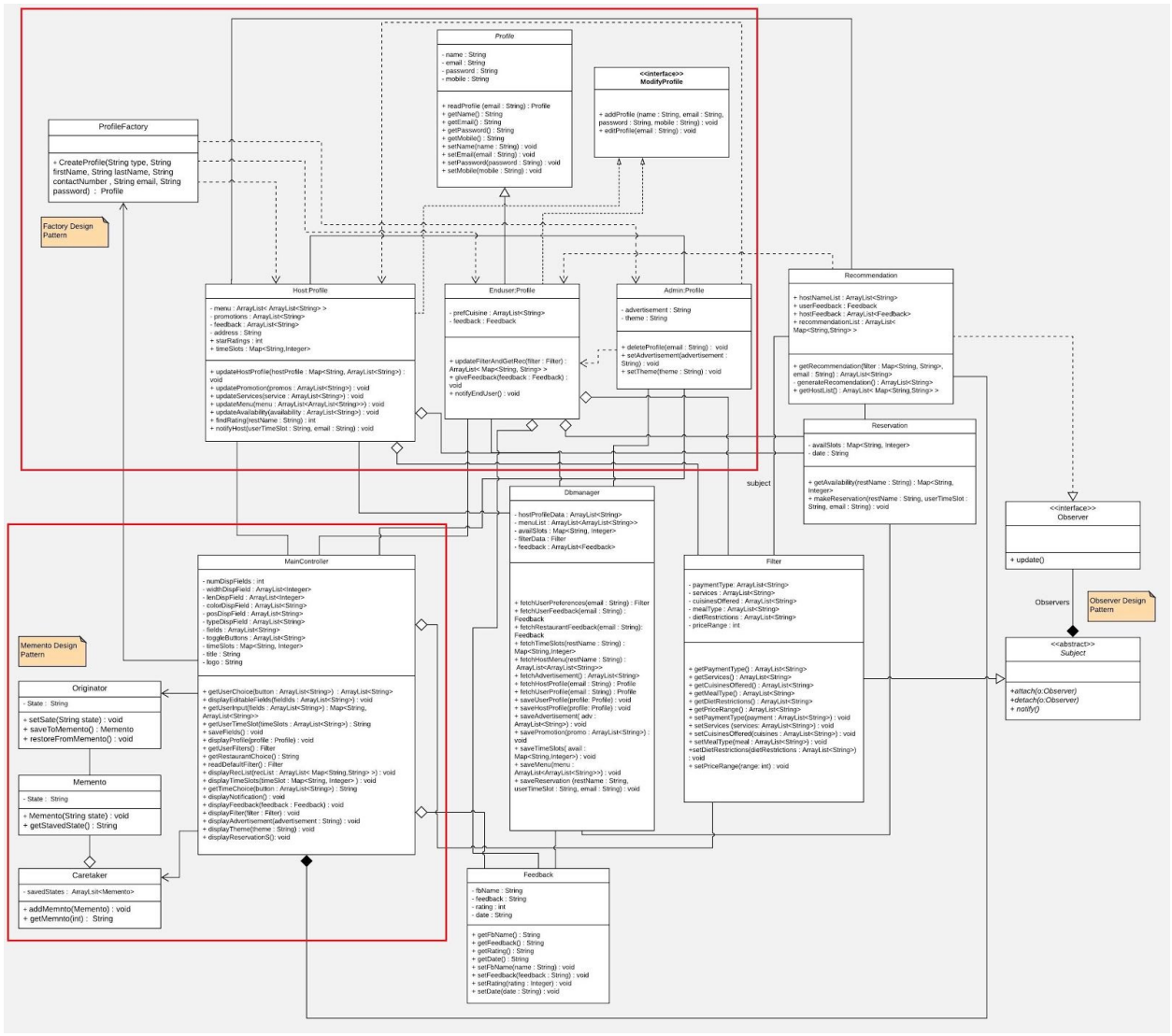
Previous Class Diagram:



Refactored Class Diagram:



Completed Class Diagram:



Breakdown:

The team worked on first understanding and correcting all the points which were a part of Part 3 feedback. Each team member participated in the understanding of the Spring MVC framework and setting it the project as Spring Boot application using Spring Boot. Furthermore, the team discussed the design patterns to be used in the project and then made the required changes in the class diagram individually.

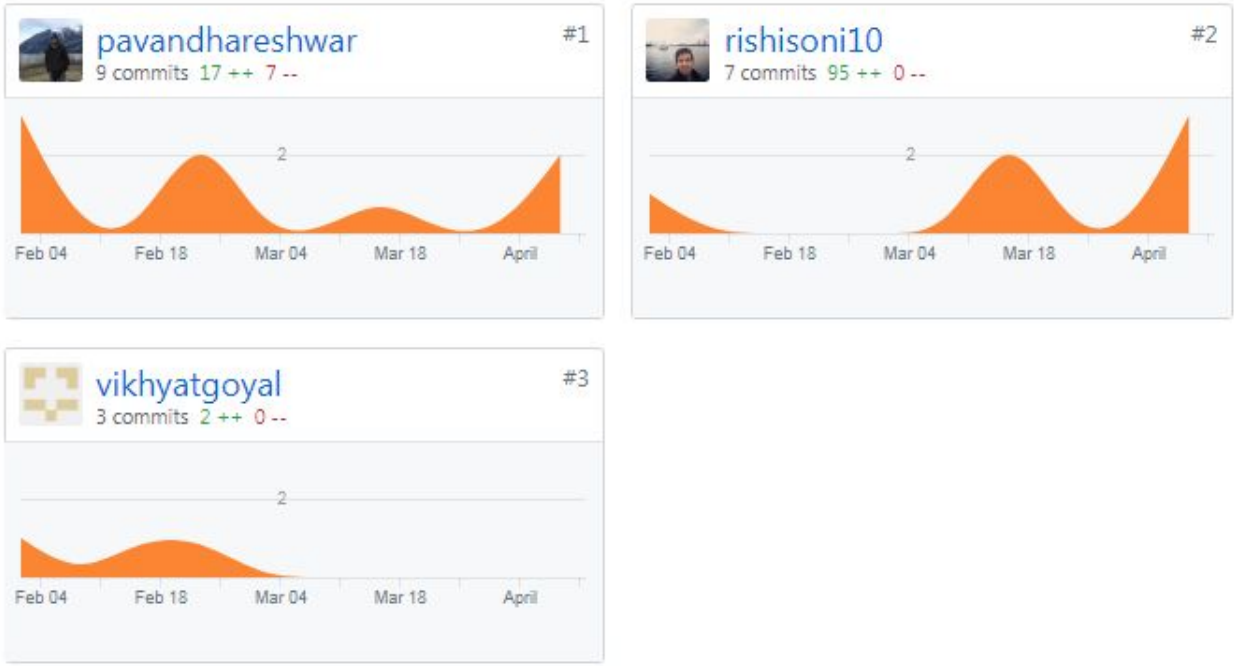
Once, we had a better understanding of our requirement and their mapping to the UML diagrams, each of us worked on refactoring the class diagram. In particular :

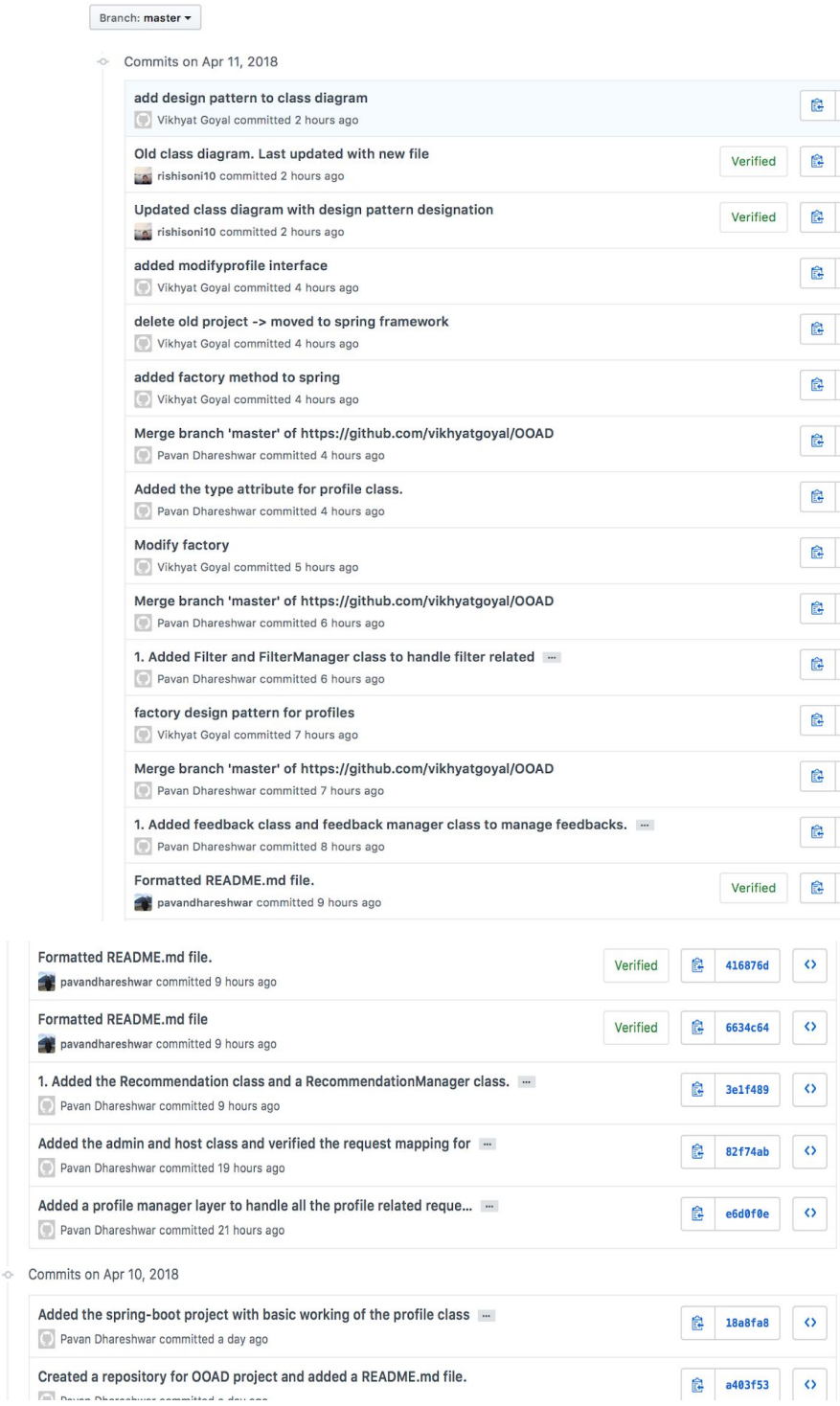
Pavan : Worked on setting up the Spring Boot and the SQL database. Made changes in the class diagram for implementing the **Memento** design pattern. Implemented and tested the functionality of profile related requests (add, delete, get) and feedback related requests using the controller, manager and model classes.

Vikhyat : Worked on the class diagram by modifying it to show the **Factory** design pattern implementation. Implemented the ProfileFactory class with some classes and methods related to profile creation and handling in the project..

Rishi : Worked on the class diagram by modifying it to show the **Observer** design pattern implementation. Implemented a few classes and methods in the project.

GitHub Graph:





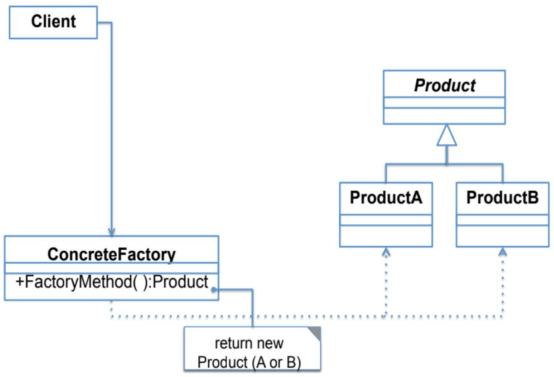
The contributors chart from Git isn't showing the commits made by two team members (Pavan and Vikhyat) from command-line because the git username in system's git configuration isn't matching the username in git repository submitted.

Design Pattern:

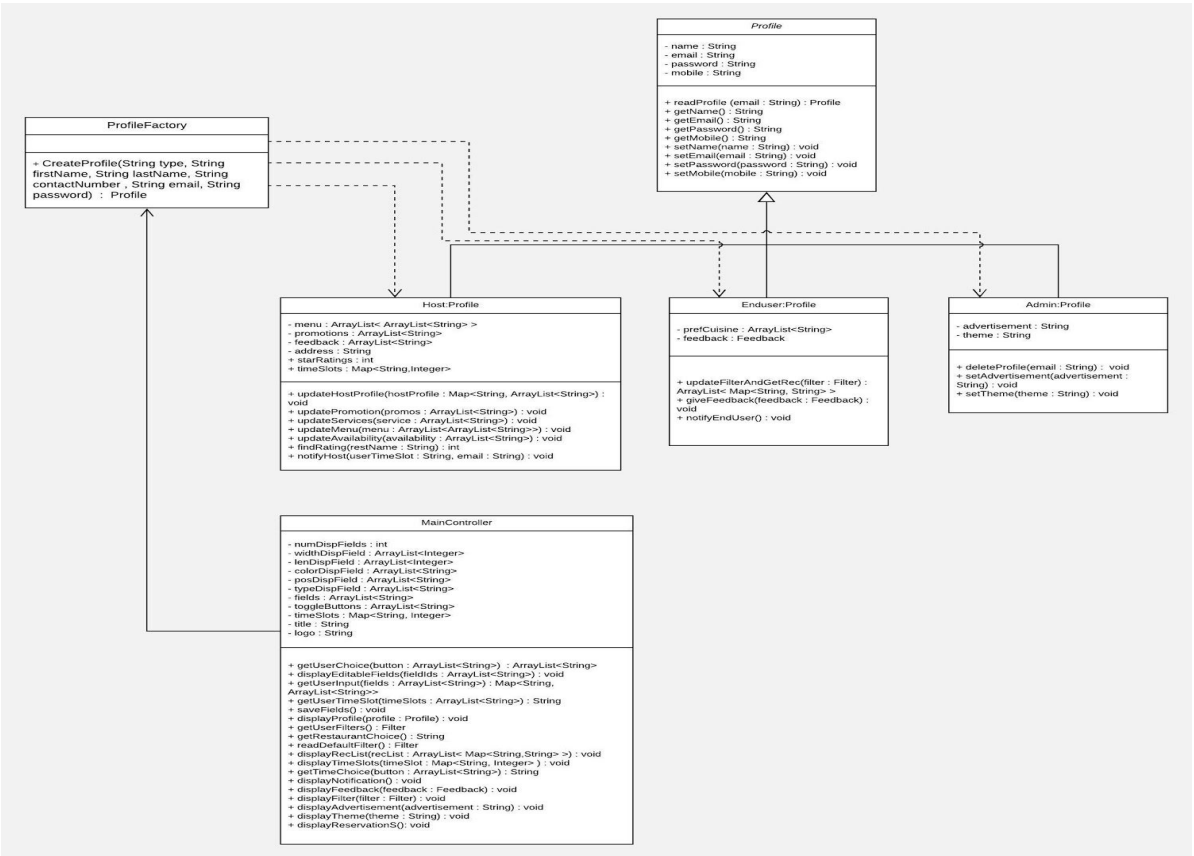
1) Factory Design Pattern:

The system requires creation of new profiles which can be of any one of the types : Admin, Host, EndUser. This is the exact common problem which factory design pattern solves. We have added “ProfileFactory” class which depending on the kind of profile required creates a new object of either “Host” or “Admin” or “EndUser” type.

Factory Design Pattern is identified by the diagram as below:



Our Class Diagram Portion which implements the Factory Design Pattern (participant classes: ProfileFactory, Host, Admin, Enduser, MainController) :



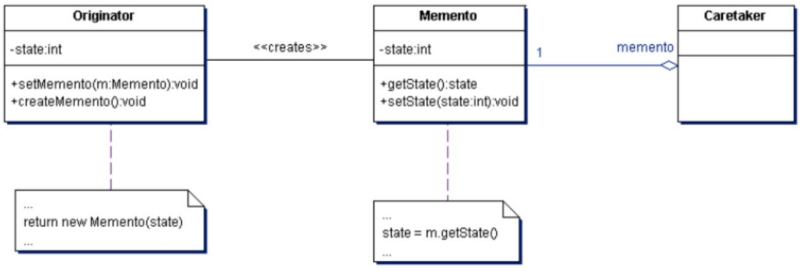
2) Memento Design Pattern

Memento Design Pattern allows us to reset the system to a prior saved point, if required. In our system, there are two scenarios in which this design pattern can help us:

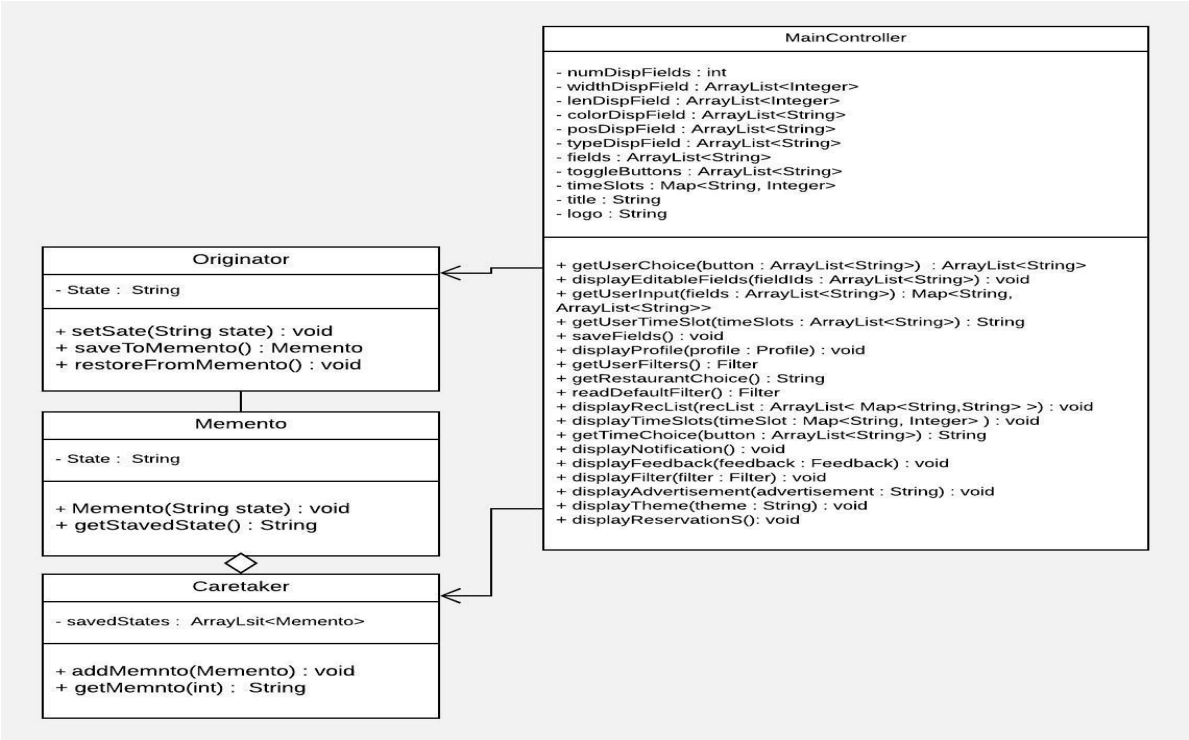
a) Reverting to a default recommendation list : If the EndUser wants to revert back to the default recommendation which is only based on his feedback and profile inputs, and not on the filters he had set in current session, then memento which was saved at the start of session can be used.

b) Reverting back and re-selecting a reservation timeslot: If the Enduser cannot find a appropriate timeslot while reserving for one of the recommended host and wants to revert back to the generated recommendation list and select a different host, memento which had saved that state can be used.

Memento Design Pattern is identified by the diagram as below:



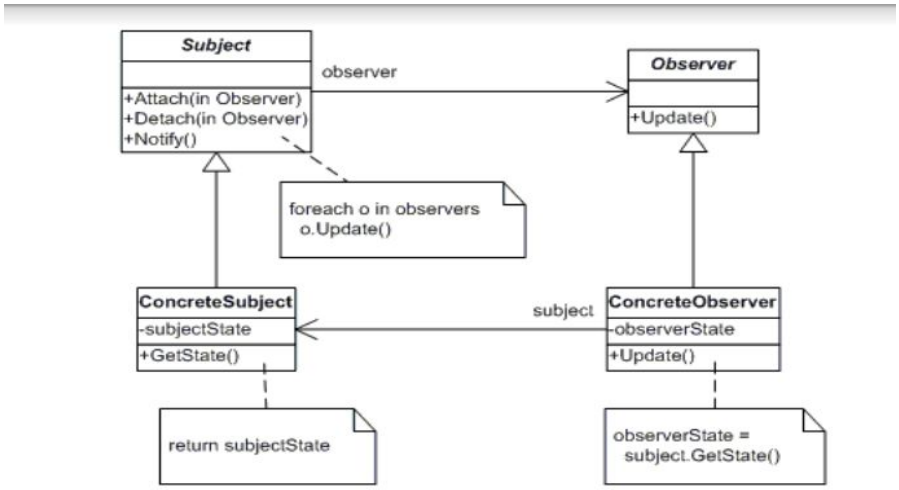
Our system implements memento with the “MainController” class (as also seen in the class diagram portion seen below) :



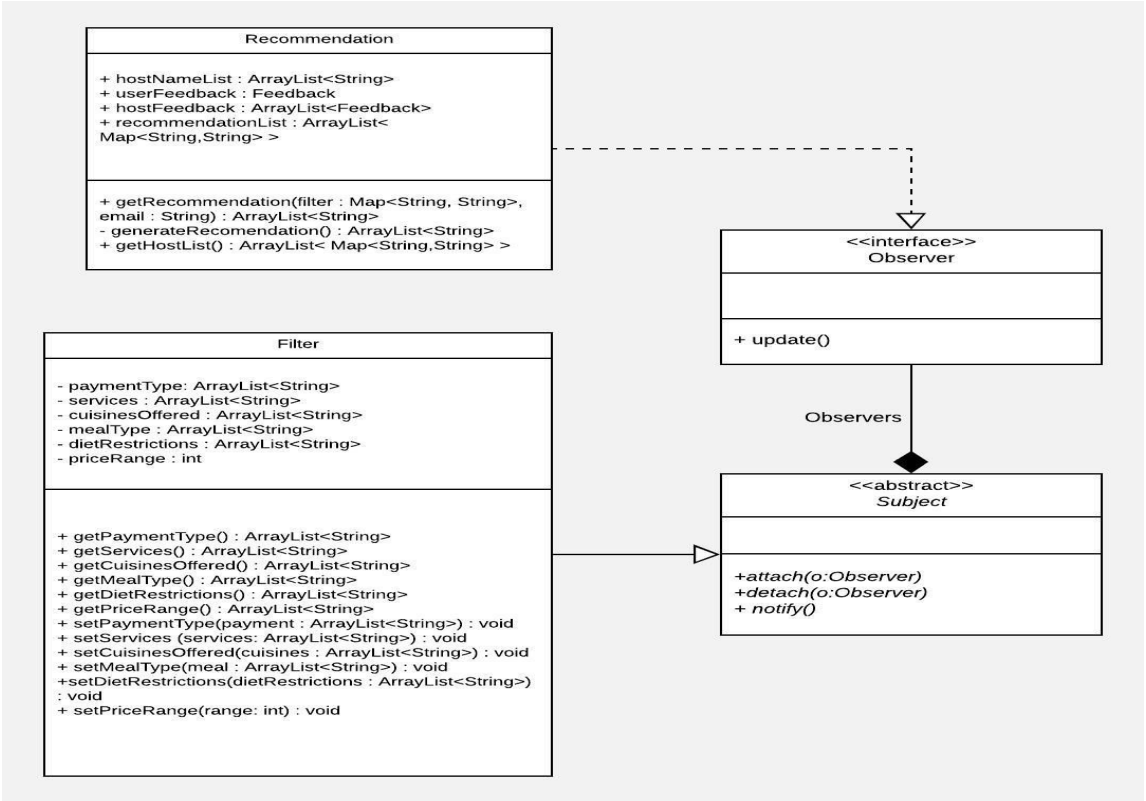
3) Observer Design Pattern

The system requires that whenever the Filter is updated, new Recommendations are generated. This problem is solved with the Observer design pattern. The Recommendation class acts as an observer, while the Filter class acts as a subject. The Filter class attaches itself to an observer object, which it uses to notify the Recommendation class that the filter has been updated. The Recommendation class itself waits on the Observer object for receiving the notification.

Observer Design Pattern is identified by a diagram given below:



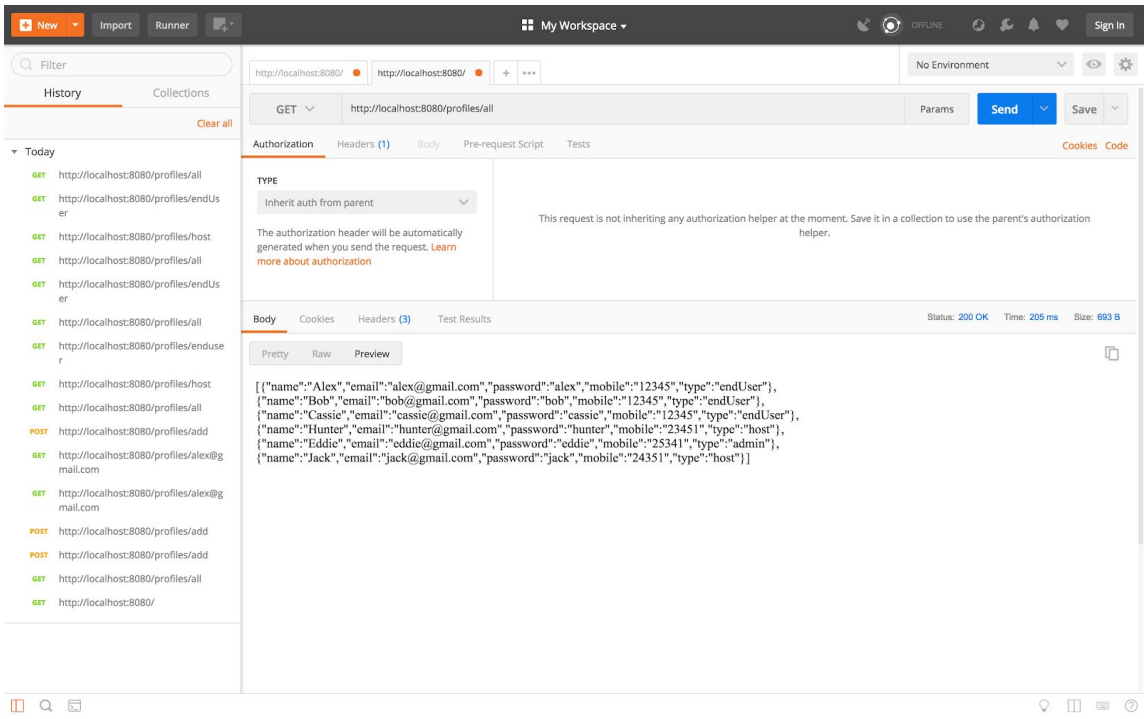
Our class diagram portion which implements the Observer Design Pattern (participant classes: Recommendation, Observer, Filter, Subject):



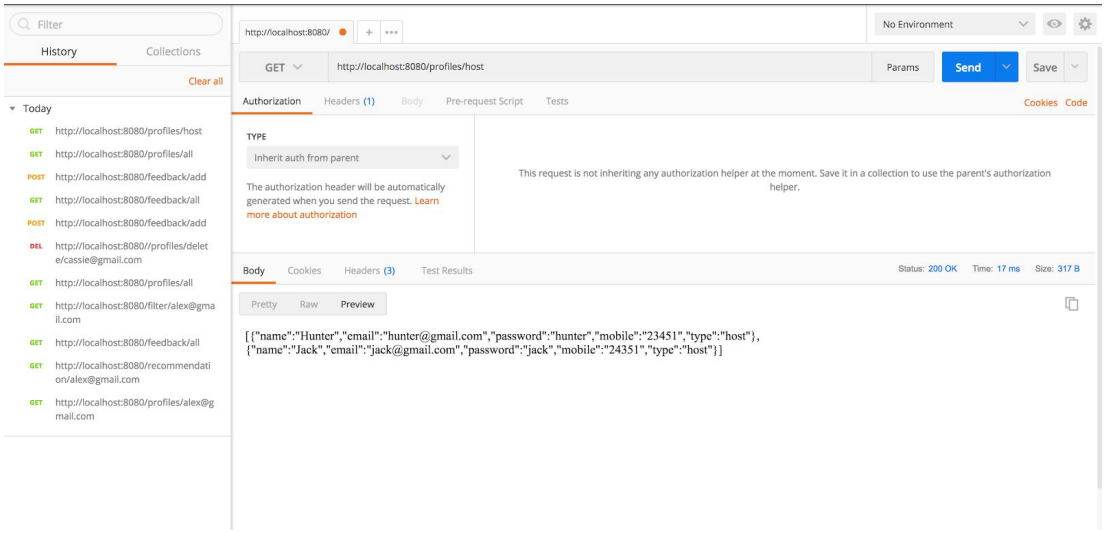
Estimated Remaining Effort:

Attached below are the screenshots of few of the functionalities that are part of the final system:

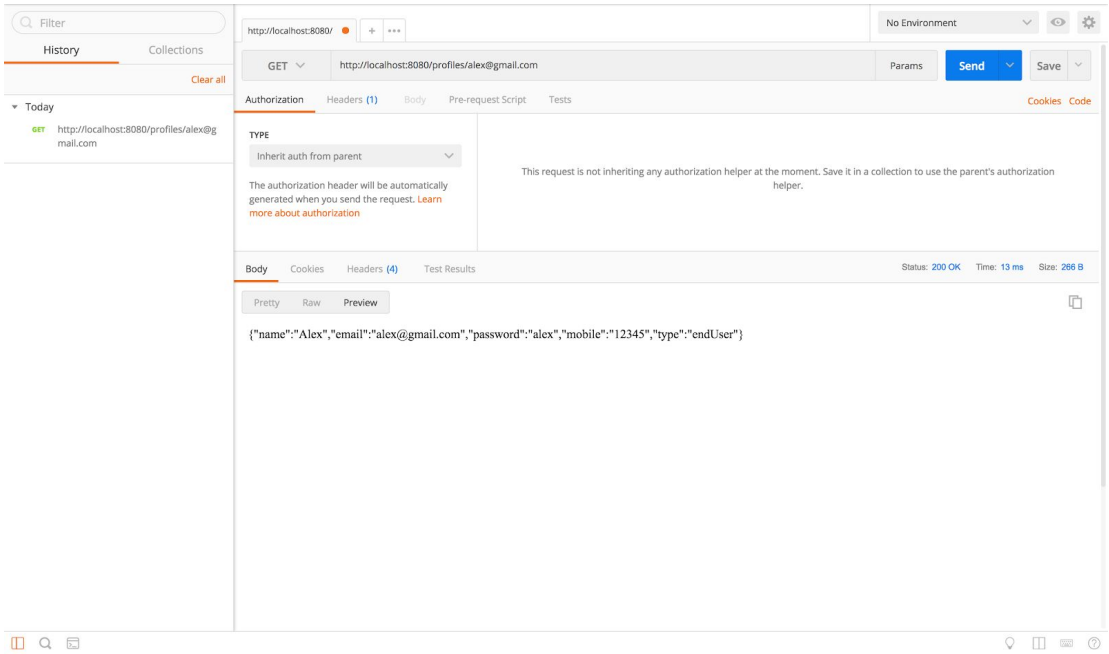
1. Fetching a list of profiles (enduser, host and admin) from the database (mysql) and displaying it



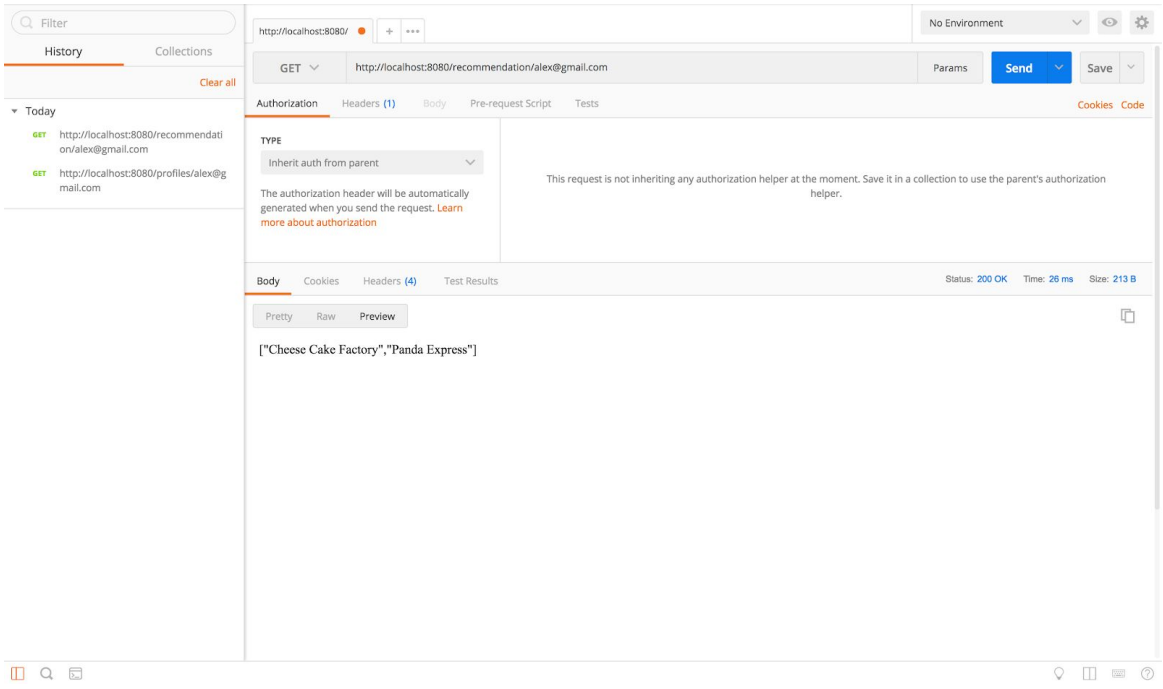
2. Fetch profiles by type (just host, enduser or admin)



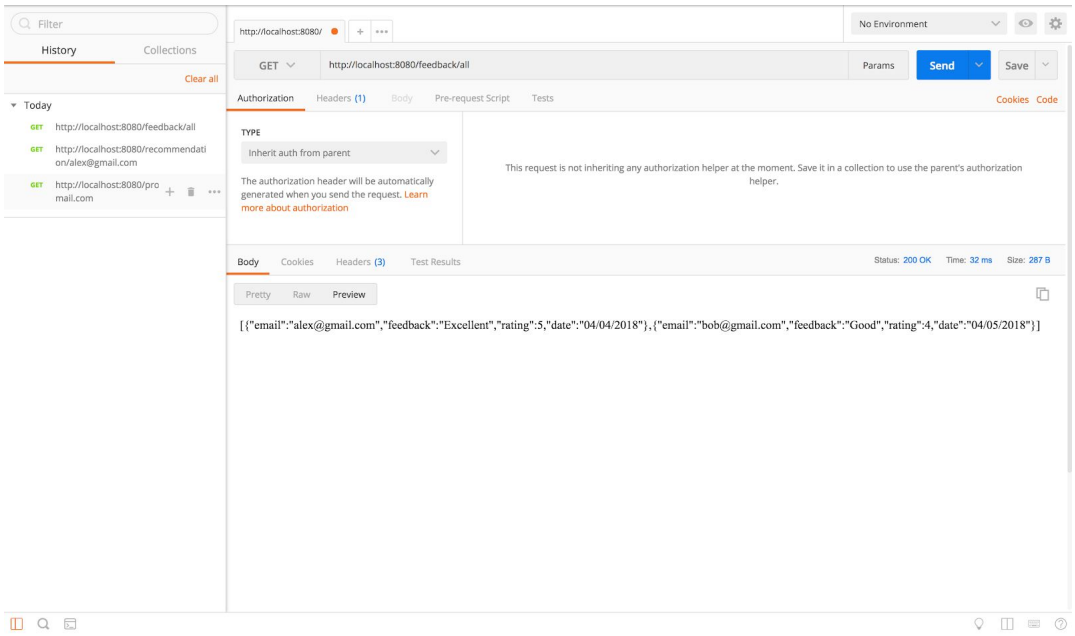
3. Fetching information of a specific profile (using email as the key) and displaying it



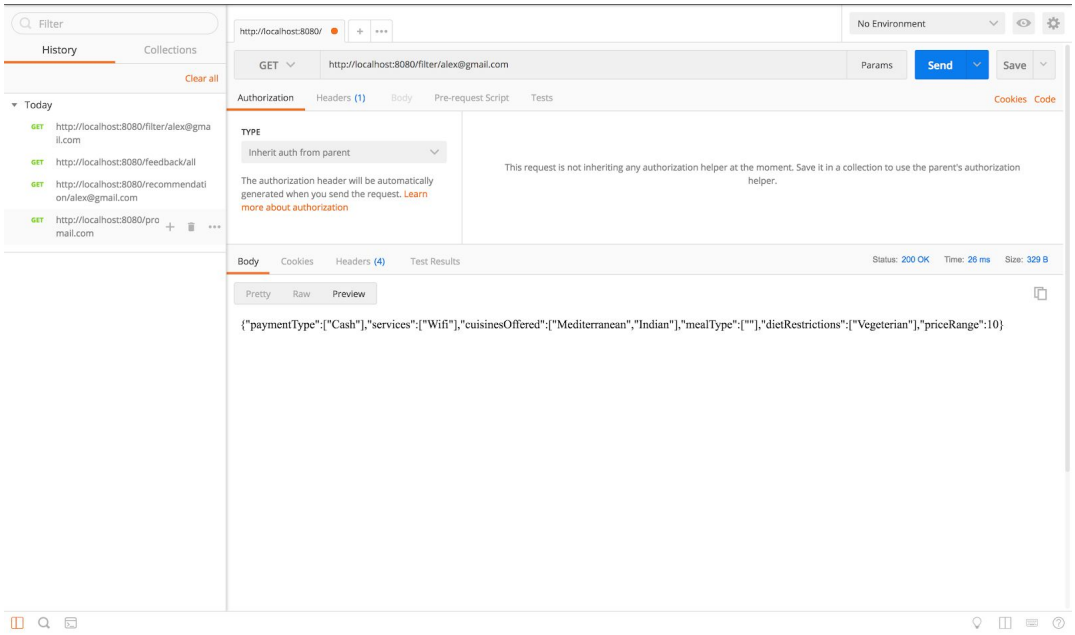
4. Showing recommendation for a specific user (this isn't the full implementation - we wanted to check the end-to-end request-response handling)



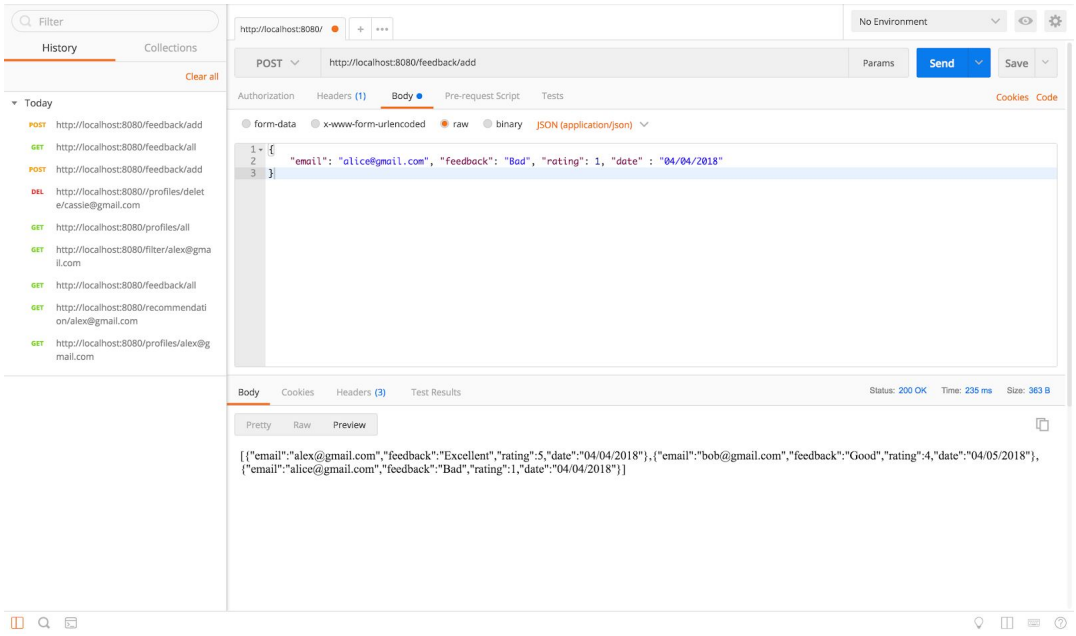
5. Fetching the feedback of all the users



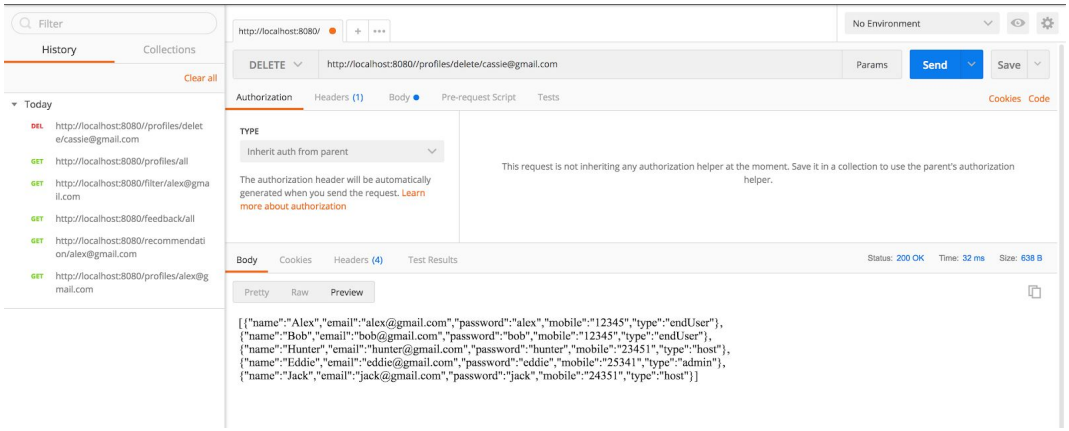
6. Fetching the filter for a specific user



7. Add a feedback



8. Delete a profile



Final Iteration:

1. Implementation of methodologies to save and retrieve the data from the database.
2. Implementation of the Recommendation class algorithm to generate the recommendation list based on user preferences, user feedback and restaurant reviews.
3. Develop test cases for testing all the modules separately (unit testing) and verifying the desired functionality of the implemented design patterns.