Project 6: Semester Project – Final Report

Project title: Travalign - plan your trips together

Team Members:

Andy Fan

Lincoln Schafer

Alex Orlowski

Final State of System:

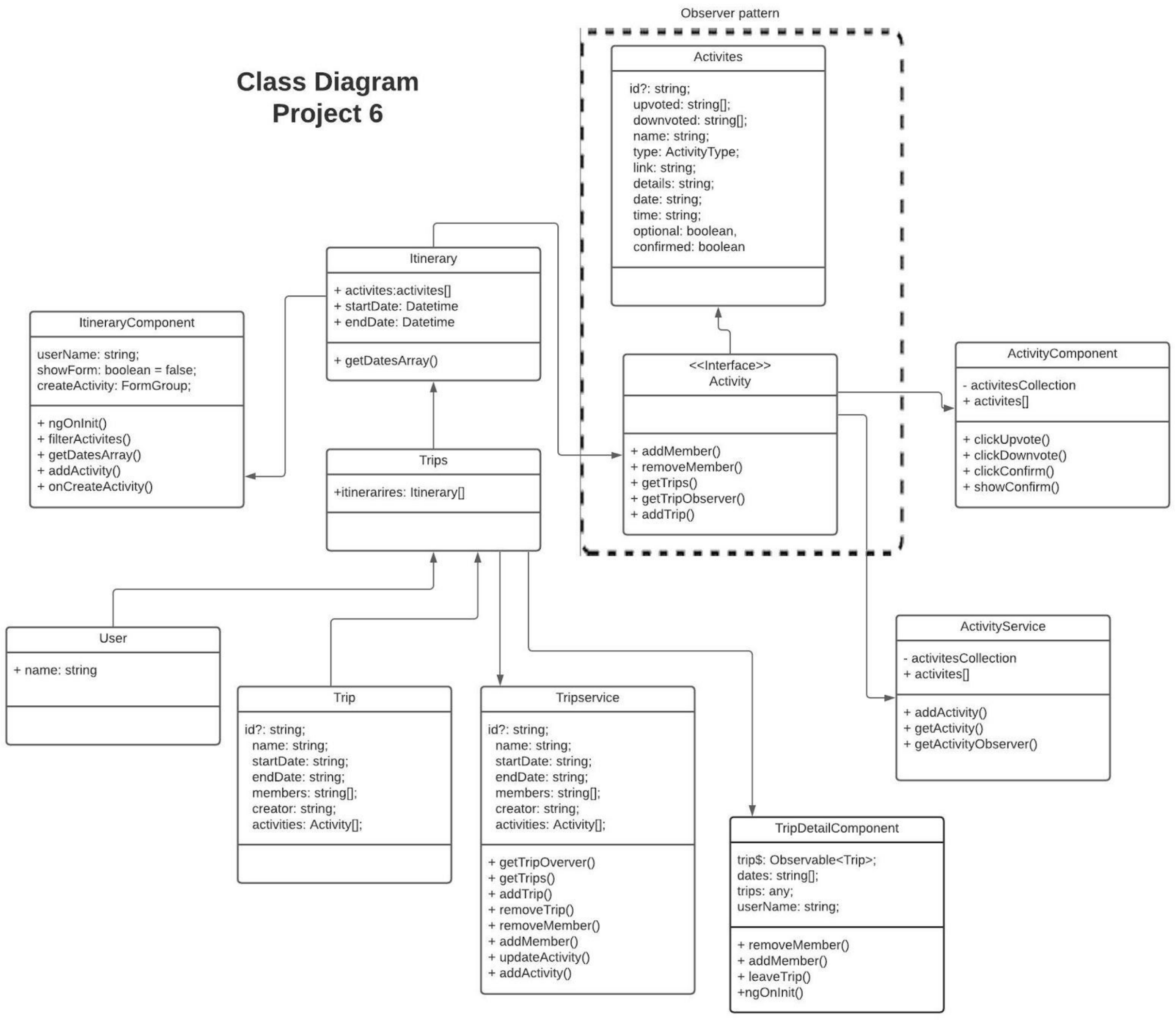
The final state of our system implemented the following features:

- Display trips that you are part of
- Create a trip
- Add travelers to trips
- Add ideas to a trip
- Upvote/downvote ideas of other travelers
- Trip leader can confirm an idea and make it part of the itinerary
- Google Authentication as primary login
- Non-mvp: real time collaboration - travelers see each others changes on the trip in real time using Firestore's real time capabilities

The only feature we set out to implement that we didn't was a budget for the itinerary, ideas and travelers. This was a non-mvp stretch goal of ours, so ultimately we decided to not implement it as we were crunched on time. From project 5 to now, we implemented adding activities to each day in the itinerary, finished adding and removing users once the trip had already been created, the upvote and downvote system as well as confirmation powers for the trip creator. Overall, not many things changed from our initial design for project 4 to project 6. The main difference would be that the UI we developed differed from our mocked UI, only in appearance and not in functionality. For example, instead of showing a number saying how many travelers were included in the trip, we opted to list all of the travelers by email.

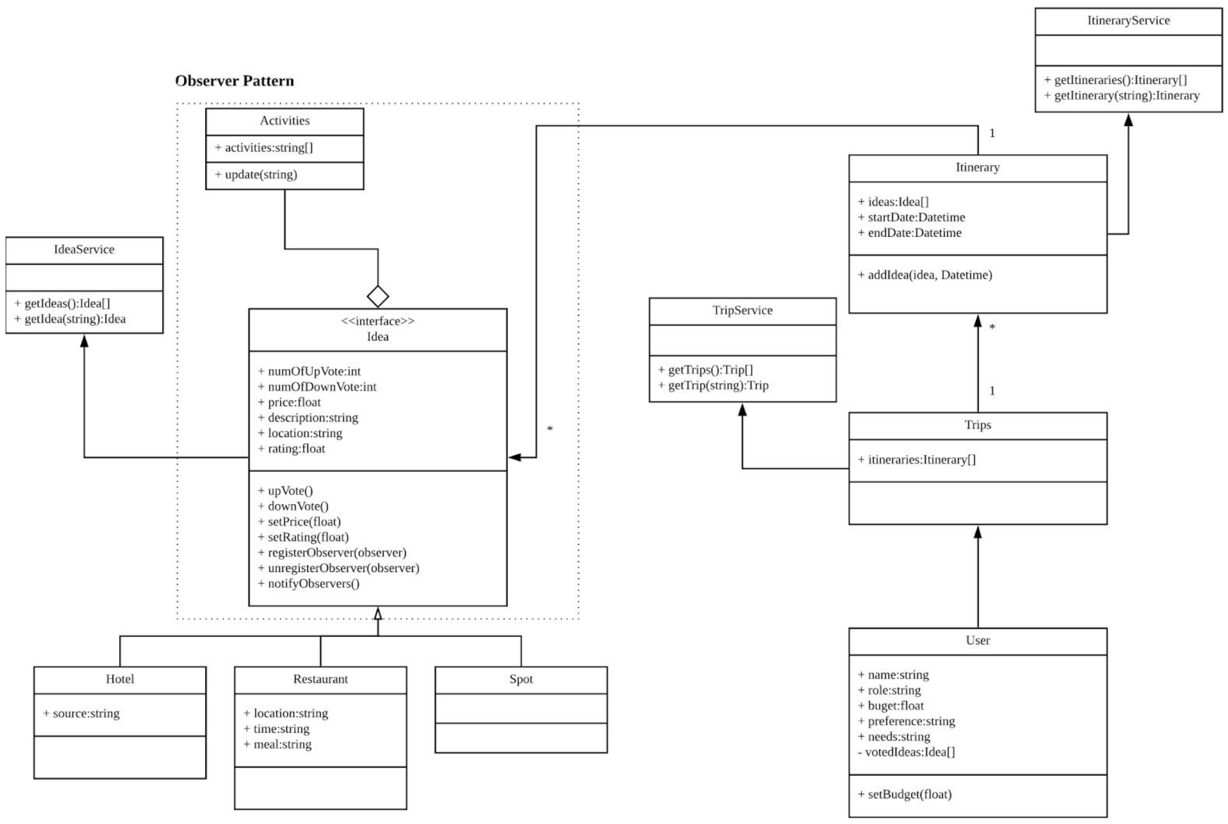
Final Class Diagram and Comparison Statement:

Final Class Diagram for project 6:



Class Diagram from Project 4:

Class Diagram



One of the biggest changes between these two is to not have different types of Ideas or activities via inheritance. We opted for an enum system that we felt wouldn't pigeonhole us as much as having a class dedicated only to Hotels or only to Restaurants. The types we chose are eat, sleep, logistics, recreational, and default. We felt that the inheritance here would actually stunt the program more than it would help it, as it would restrict the options and have to be more rigid.

Third Party Code vs. Original Code Statement:
The Angular CLI was used to generate the application including all boilerplate code for a basic application that can load a basic page. All code, especially in controllers, services, and html, was written by our group with some cases where we used online sources including Firestore documentation, Angular documentation, Angular Material documentation, and other code references. Whenever a code reference is used there is a comment with a link in the source code.

Additionally, we used libraries for lots of functionality including Angular itself, Firestore, Firebase, and Angular Material for UI Components. Code references to these libraries are clear via imports within files, so no references were added for these.

Statement on the OOAD process for your overall Semester project:

1. State management for a web application is a bit different than what we were thinking in our activity diagram. Everything is based on the web lifecycle and lives for the life of the web page, however the persistent database and multiple users creates extra states that we didn't expect in our activity diagram. This manifested in some UI bugs where one user could make a change that impacts another user, for example removing a user from a trip while they are looking at that trip.
2. The class relationships went well for us even though we ended up changing names of entities. It was interesting to see that the concept of the entities remains the same even though what we want to call it changes. The class diagram proved valuable in helping us identify trips and activities that happen in the trip.
3. It was a little difficult to map out how the Angular concepts of services and components work in typical UML since it's not a perfect conceptual overlap with typical OOP classes. Thinking through the analysis and design was valuable, we just needed to make adjustments to how we applied the concepts of UML. For example, we needed to make a decision about how to model trips. Do we make one entity for trips in the UML? Or are we better making an entity for trip service, component, and data type which is what we will have in the Angular app.