OOAD Project 6: Final Report
Rohan Baishya, Omar Kaheel, Akhil Kunam

Project: Virtual BarTab Application

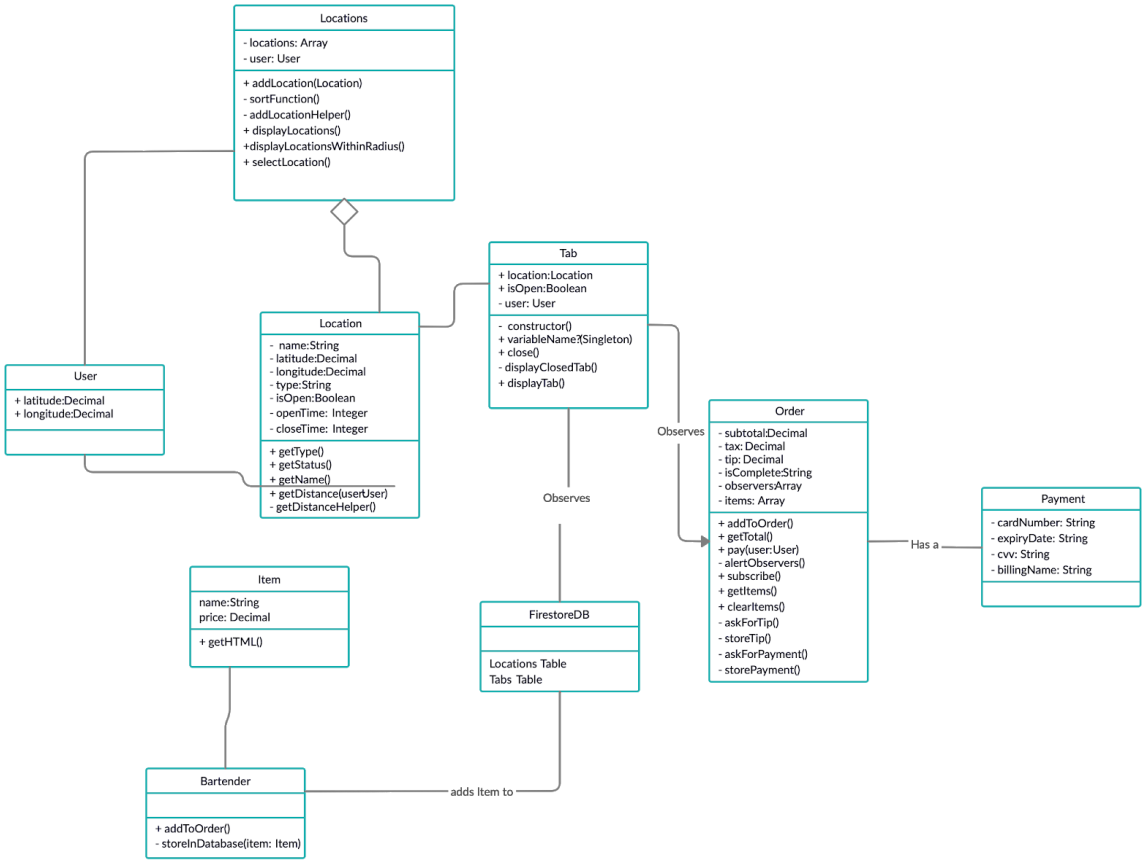
Final State of System Statement

Our application is a contactless, virtual, bar tab. When a user/bar patron opens the app, they see a list of restaurants/bars near their location, and if they click on a location, they effectively open a tab there. The user can then go about ordering food and drinks with a bartender/server. After the patron is done with their food and beverages, they can close their tab with the click of a button on their device, add their gratuity, and payments are processed automatically through pre-loaded payment information. This eliminates the time and effort of chasing down a check, swiping a card, or other payment interactions with others, as well as the inconvenience of entering card details through a unique online ordering platform at each restaurant/bar a user visits.

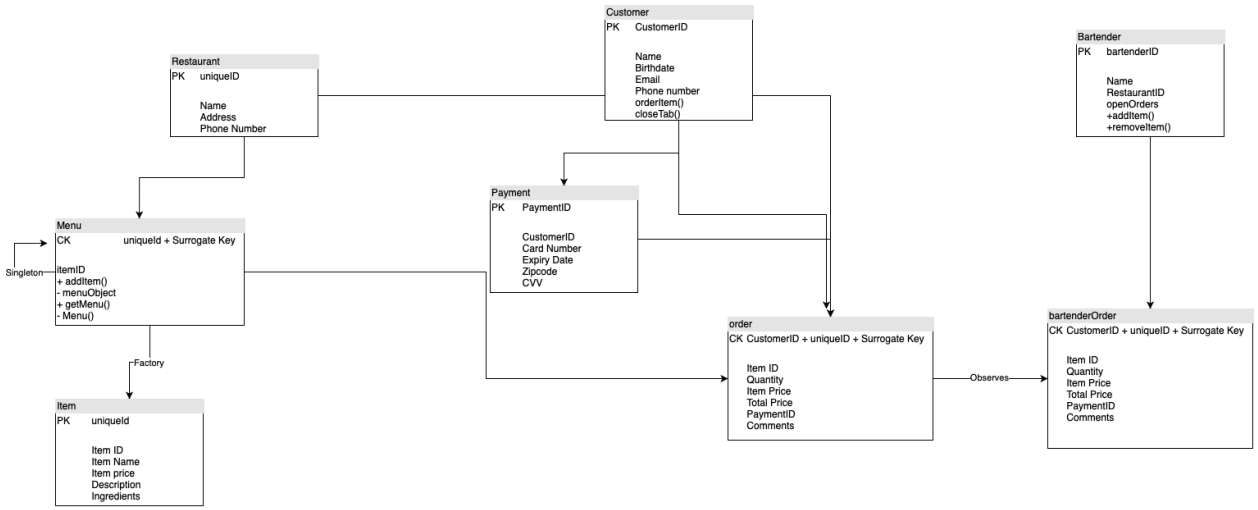
Since Project 5, our concept has stayed relatively similar, featuring a User class and Tab class, both of which have a HAS-A relationship to the Location class, an Order class, which stores users' food and drink order(s), and a Payment class featuring users' payment details. For the Tab class, we implement a singleton design pattern. We also implemented an observer design pattern for the Tab and Order classes . Since then, we have also added a Bartender class and an Item class, which are in place as a Bartender object adds Item objects to an Order object.

Final Class Diagram and Comparison Statement

Final:



Project 4:



For our final project we implemented a few design patterns. We tried to implement the model view controller. We combined the controller and the model into 1 as in JavaScript we can have functions inside objects. The views are the

HTML files such as Tab.html, Locations.html which are changed at run time. We also implemented the observer pattern in 2 ways. The first was when we manually implemented it ourselves by maintaining a list of observers and calling the subscribed function on any change which is what the Tab and Order relation does. This makes it possible for the Tab to automatically close whenever the Order has been paid. The next observer pattern was the one with the FirestoreDB. The tab subscribes to the Tab table so that whenever any item is added it adds it to the order and can update the view. The tab itself is a singleton as there will only ever be 1 tab active at a time. In JavaScript it is very unique that there isn't a way to make a traditional singleton and instead we had to create an object to use. We also have a locations class which aggregates all the location objects so that the user can choose between them.

Our project changed every time between 4, 5 and 6. During 4 we had just done the planning so our ideal system was what the UML depicted. Then in project 5 we had a huge slow down where the entire team had to learn javascript and we just made the backend system. We reduced the scope here significantly by removing some classes like the menu and bartender classes. In project 6 we again had a major setback as when we tried to integrate a front end and also split up the backend into multiple files we would run into issues. We finally settled on a system where the core functionality remained the same as project 4, however we did have to take out the detailed Item functionality as well as the menu. The bartender order and user order merged into 1 table in the firebase DB as we thought that would be a much more efficient method.

Third-Party code vs. Original code Statement

Majority of the code in this project was ours. We spent a lot of time watching youtube tutorials, reading articles and wading through stackoverflow to figure out JavaScript and HTML so we were heavily influenced by that. Where we felt there was a direct connection to the code and and outside resource we tried to cite it in the comments, but other instances where we pulled from many tutorials to get something working we were unsure on how to properly cite them.

We did use some outside libraries to supplement our code. To integrate with the FirestoreDB we used the FirebaseSDK which provides functions that can be called to interact with the database. This can be found here: https://firebase.google.com

Another tool we used was bootstrap for some styling. We read that this makes it easier to style the pages so we attempted it with the menu bar, but found it to be way too complicated for us at this level so it wasn't heavily relied on. That can be found here: https://getbootstrap.com
We used a lot of help from the following sources to learn about the tools we used: W3Schools, Youtube, StackOverflow, Reddit, Firebase Documentation.

Statement on the OOAD Process

We decided to do the bulk of our backend in Javascript for easy integration with our front end UI. Since Project 5, we also built our front end in Javascript and HTML and built our database using FirestoreDB. Object oriented programming with Javascript has been difficult, as none of us are very familiar with Javascript (especially compared to in Java or Python). Javascript is a prototype based object oriented language, which means it doesn't have classes, but rather defines behaviors using constructor function and then reuses it using the prototype. Although it supports encapsulation, inheritance (through prototypes), and polymorphism, implementation of the same design principles and patterns we learned in class has been a challenge with a new programming language. Furthermore, building a full-stack object-oriented application was a very new experience to us, and forming our architecture and data pipelines, learning all the new programming languages we used, and applying object-oriented principles to the application with new coding language(s) was a challenging, but rewarding experience. We also tried to initially build the frontend with a framework called Vue.js as some tutorials mentioned that it is easier and quicker to use that just raw HTML and javascript, but there was so much to learn that we couldn't accomplish it in the short 2 weeks we had so we went back to just HTML and javascript.

As far as design patterns go, we implemented a singleton pattern for the Tab class, as the user will only ever have one tab at a time and it may be referenced in multiple places. Therefore we wanted to have a singleton so we can easily refer to the same instance. It took awhile to figure out how, but in JavaScript it is essentially just an object you create with functions inside. We also implemented an observer pattern for the Tab and Order classes. This makes it possible for the bar tab to close as soon as the order details show that the payment is complete. Furthermore, the Order also observes the Firestore Database which contains a list

of all items that a user has ordered. These observer patterns were again very different to how it is done in Java, as instead of having a library for it and using an interface like in Java, we had to implement the pattern on our own with Javascript.