Title of Project:

Wheel and Reel

Members of Team:

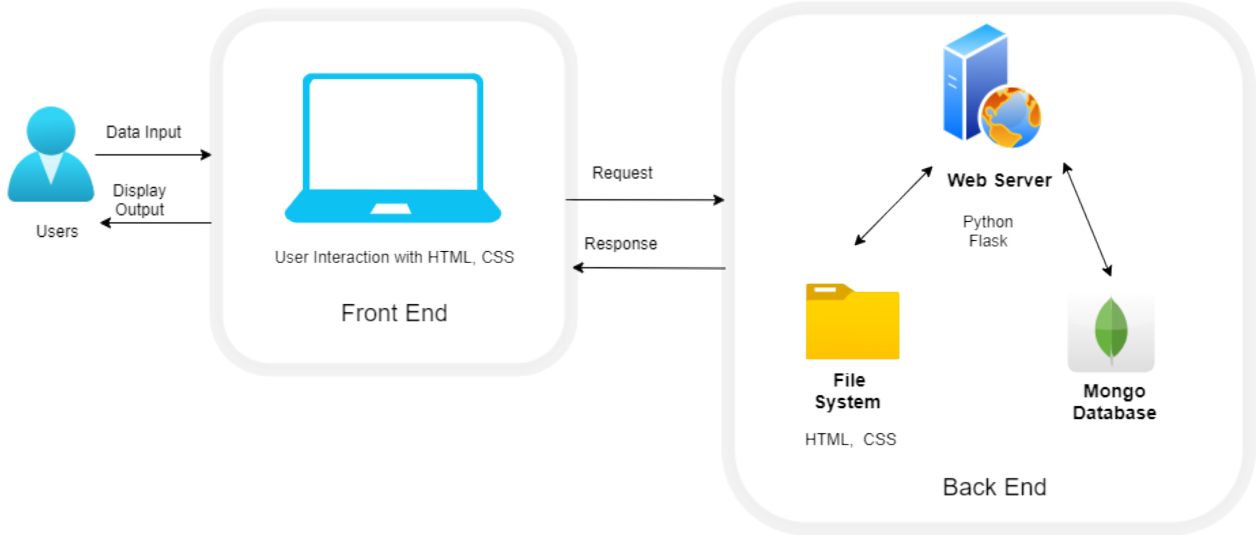
Xuefei Sun
Srihaasa Pidikiti
Alexander Louie

1. Summary:

Due to COVID-19, there has been a sharp increase in preference for drive-in theatres as opposed to traditional theatres. But there is no one particular website to search for the shows currently being played in all the nearby drive ins. To meet this new onset of demand a one place website for ticket booking at drive ins would be needed. We plan to design such a drive in specific websites catering to the drive in movie goers. This is a user specific website intended to provide a list of shows based on preset preferences upon signup. Our end system would focus on the following user preferences majorly-location (how far the user prefers to drive), genre of movies, ticket cost, food and drinks availability at the theater. These preferences can be updated anytime by the user. Once signed in, the user is provided with a list of shows playing based on their preferences. Users can choose from this list and go ahead for checkout. When a user purchases a ticket they will also be provided with an option to include food and drinks add-ons. Payment can be done using either debit card or credit card. The user can cancel their ticket purchase or view the purchased ticket in their profile dashboard’s transaction history.

2. Final State of the system:

2.1 System Architecture:



The above figure shows the system architecture of the web application. Users can interact with the web application using a front end which is designed with HTML and CSS. All the API requests received from front end are then sent to a back end web server which is run using Python Flask. This web server queries Mongo Database Atlas and uses a file system to process the requests and generate a response. Front end displays results accordingly from the generated response. Pymongo is used for connecting to the mongodb atlas database. Flask server running on a port has been used to connect api endpoints to the respective use cases and process user requests.

This was the initial proposed architecture and we were able to successfully implement the same.

2.2 System Features:

The final system has the following features implemented successfully:

1. Signup feature for new user which includes filling movie and theater preferences questionnaire
2. Sign-in, sign-out user features.
3. Feature to update existing user preferences in their profile dashboard.
4. Display currently playing shows to the user matching their preferences.
5. Process user request to purchase a ticket.
6. Enable and process user requests to order food add-ons
7. Process Payment done either by debit or credit card
8. Feature to cancel previous purchases displayed in the profile dashboard.

These were the initial proposed features and we were able to successfully implement all of them.

3. FINAL CLASS DIAGRAM AND COMPARISON STATEMENT:

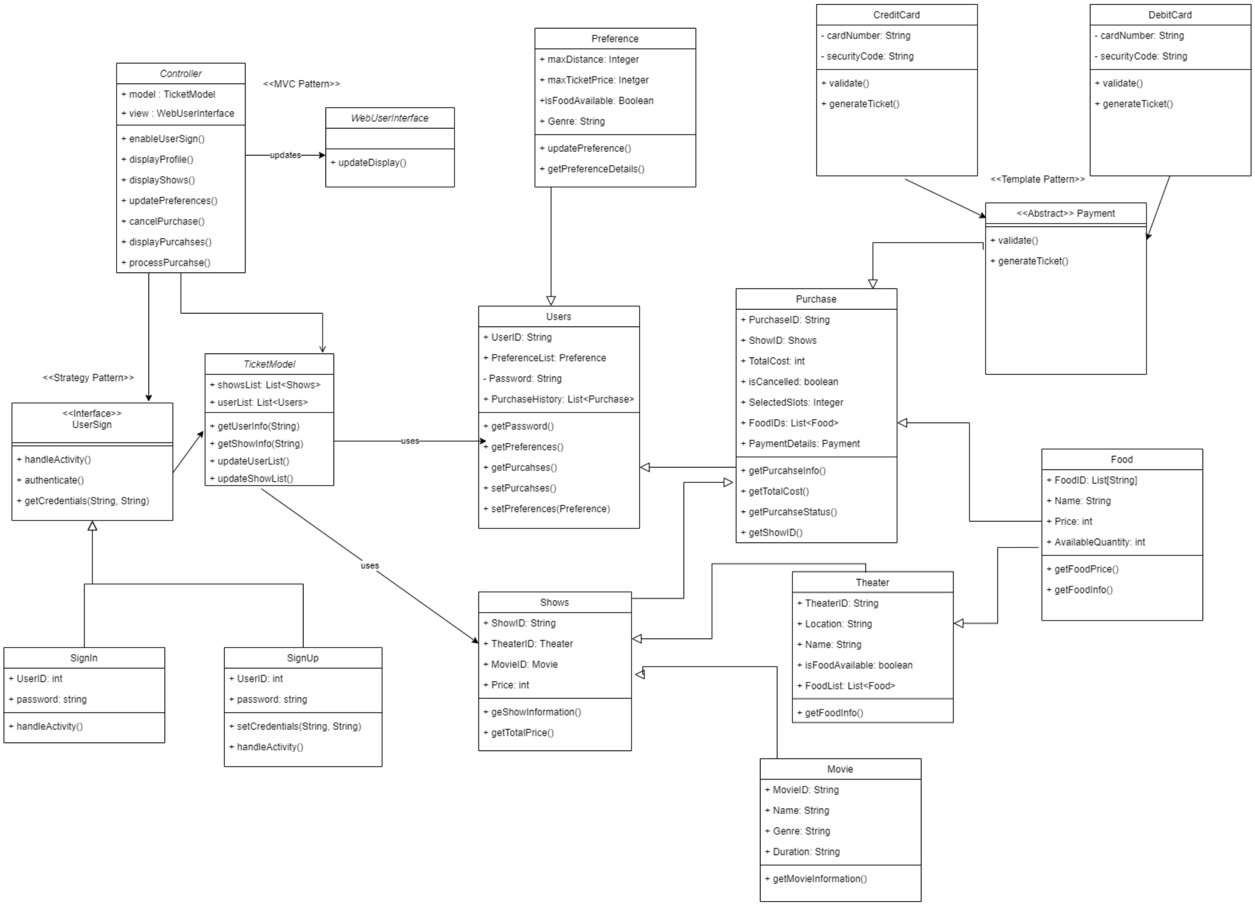


Fig 3a: Class UML Diagram in Project 4

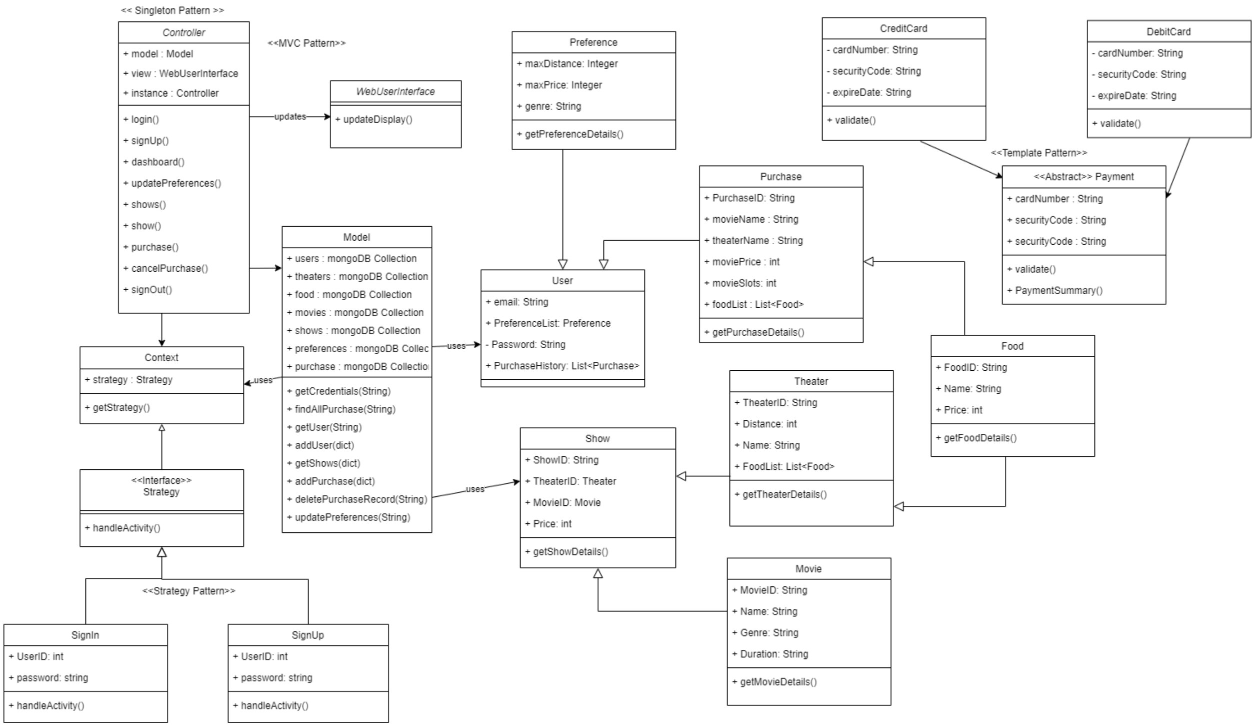


Fig 3b: Final Class UML Diagram

As shown in Fig 3b, we have included four design Patterns in total- MVC, Template, Strategy and Singleton Patterns
We used the following design patterns in our application:

1. MVC pattern - Our Model is the Class that helps us to connect with Mongodb Atlas, View is the Front end and Controller is the web server. This pattern helps us in separating the concerns of the back end, front end and flask server.
2. Template pattern - We have used it for the payment processing task. We assumed that both credit and debit cards share the same steps in processing a payment transaction except for the validate step which is done based on the initial digit of the card number type. Thus, to avoid rewriting the similar steps in payment algorithm implementation, template pattern has been leveraged.
3. Strategy Pattern- Since, the initial user activity behavior of starting a session with the web application can be categorised into two types - signin and signup we employ Strategy pattern to execute these respective algorithm implementations based on the behavior at runtime.
4. Singleton pattern- We need at all times only one instance of the controller class. This ensures that there are no port conflicts and the flask server always runs on the same port. Singleton pattern helps in this context.

From figures 3a and 3b, we can observe that there has been no major changes that occurred with respect to the class relationships and the only changes that happened are with respect to the attributes and methods in each class. These changes have been done to reduce the computational cost of joins in noSQL Mongodb.

4. OOAD PROCESS FOR OVERALL PROJECT

The following are the learnings and challenges encountered that our team experienced in analysis and design of the OO semester project:

1. **Design Process:** The initial phase of the project where we got to work on various UML diagrams like Activity, Sequence and Class Diagrams along with Database ER diagram, Architecture diagram helped us in doing a solid background analysis of the proposed solution and estimate its feasibility. Translating our vision of the web interface into WAVE test satisfying use cases posed a challenge.
2. **Design Patterns Incorporation:** Initially, we faced challenges to identify the patterns to be incorporated which can add value to the project. Identifying distinction between the application scenarios of almost similar design patterns like strategy and template was a roadblock. But eventually understanding the patterns in depth helped us in navigating these challenges and leverage the design patterns successfully.
3. **Design Patterns in Python:** This was our first attempt at implementing design patterns using Python. Unlike Java, we could not find abundant documentation on how to implement design patterns in Python. After some debugging sessions, we were able to figure out the correct implementations.
4. **Practical Application of Design Patterns and OO Principles:** On a whole, this project helped us apply all that we learned throughout the semester and understand the importance of design patterns in having an efficiently running software that can be tested and extended easily.

5. THIRD PARTY VS ORIGINAL CODE

We used the below mentioned sources for reference purposes only. Nevertheless, all the project work is our own original effort and is not copied from elsewhere.

1. https://refactoring.guru/design-patterns/strategy/python/example
2. https://refactoring.guru/design-patterns/template-method/python/example
3. https://www.w3schools.com/bootstrap/
4. https://pymongo.readthedocs.io/en/stable/

5. https://programminghistorian.org/en/lessons/creating-apis-with-python-and-flask
6. https://reinvently.com/blog/fundamentals-web-application-architecture/
7. https://www.tutorialspoint.com/python_design_patterns/python_design_patterns_singleton.html