Sam Mauck
Jon Meade
Alex Oliver
CSCI 5/4448

Project 6 - MonoClass Final Report

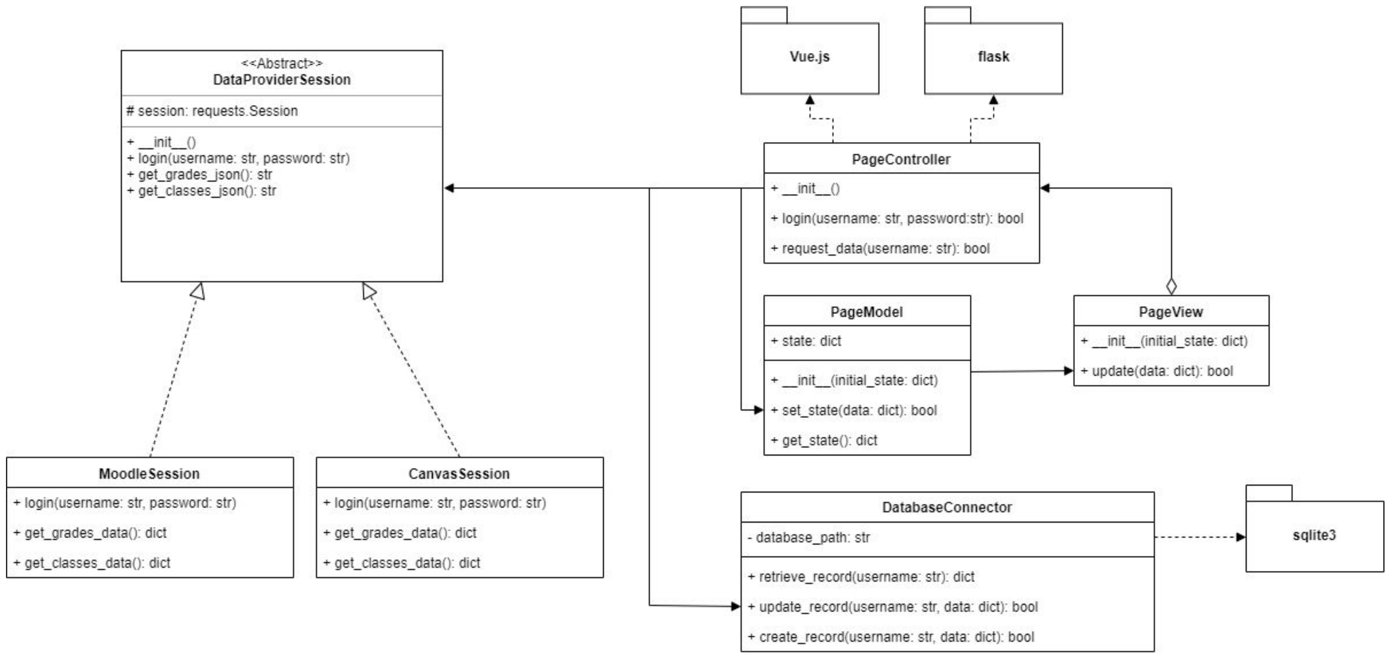
Final State of System Statement

The final state of the system incorporated most all of our initial goals and use cases. Several initial use cases, including “Login”, “Access course grades”, “Access assignment grades”, and “Reload page”, were each implemented in full, while others like “Automatic reload” and “Access course information” were reworked in order to accommodate the systems we were using. Specifically, “Automatic reload” was dropped as a use case, as the backend caches API calls, and since grade information isn’t necessarily time sensitive data, we did not see this as a necessary use case. Similarly, “Access course information” was merged into “Access course grades”, but also reduced in the scope. The main dashboard provides information about the course name, the course code, and the course grade. Finally, we were not able to pull the same amount of data from Moodle as we were from Canvas. Moodle provides an AJAX API, which we were able to leverage in order to get a list of each user’s classes, but the AJAX API does not allow access to the user’s gradebook. As such, retrieving grade information had to be done by manually scraping the HTML, and formatted on a course-by-course basis, and will obviously not work for all users at this time.

When users initialize the application, they enter the home page splash screen, where they can click the “Login” button at the top right side of the home page, which sends them to a login form, prompting them to enter their IdentiKey and associated password. Once submitted, the backend uses these credentials to login them into the CU Federated Identity Service, as well as the associated course providers (which in our case is Canvas and Moodle). The user’s session is cached to disk until they log out or stop using the service. Once they have logged in, they are taken to the dashboard page which displays general info about their courses as well as their overall grades in these courses. This information is taken from both Canvas and the Computer Science Moodle using their respective APIs. From there, they can navigate to a specific course page that itemizes their grades, displaying each assignment’s name, due date, points awarded for completed assignments, and maximum points for that assignment. On each of these pages, the user is also able to click the “Logout” button at the top right side of the screen, finishing their session.

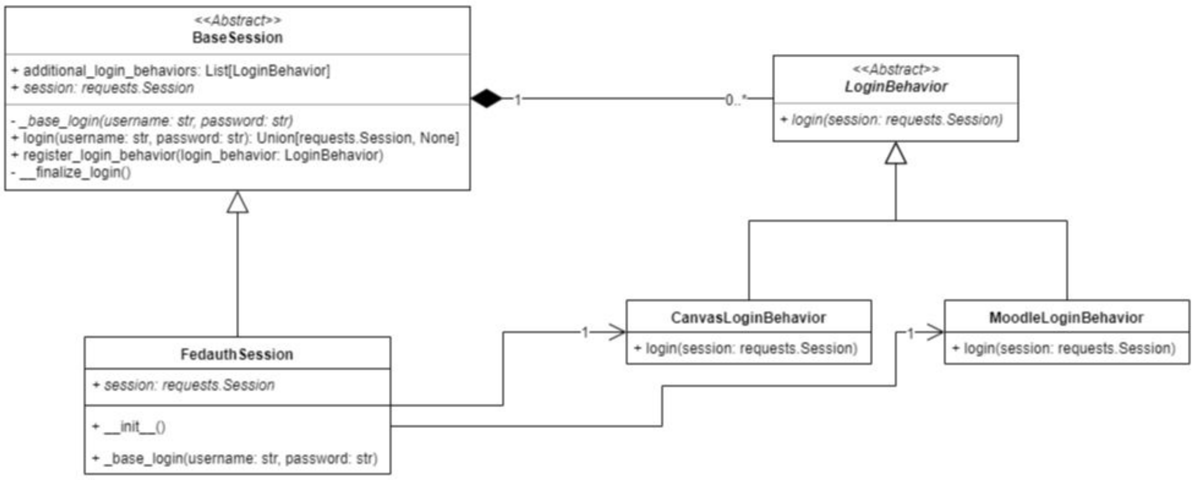
Final Class Diagram and Comparison Statement

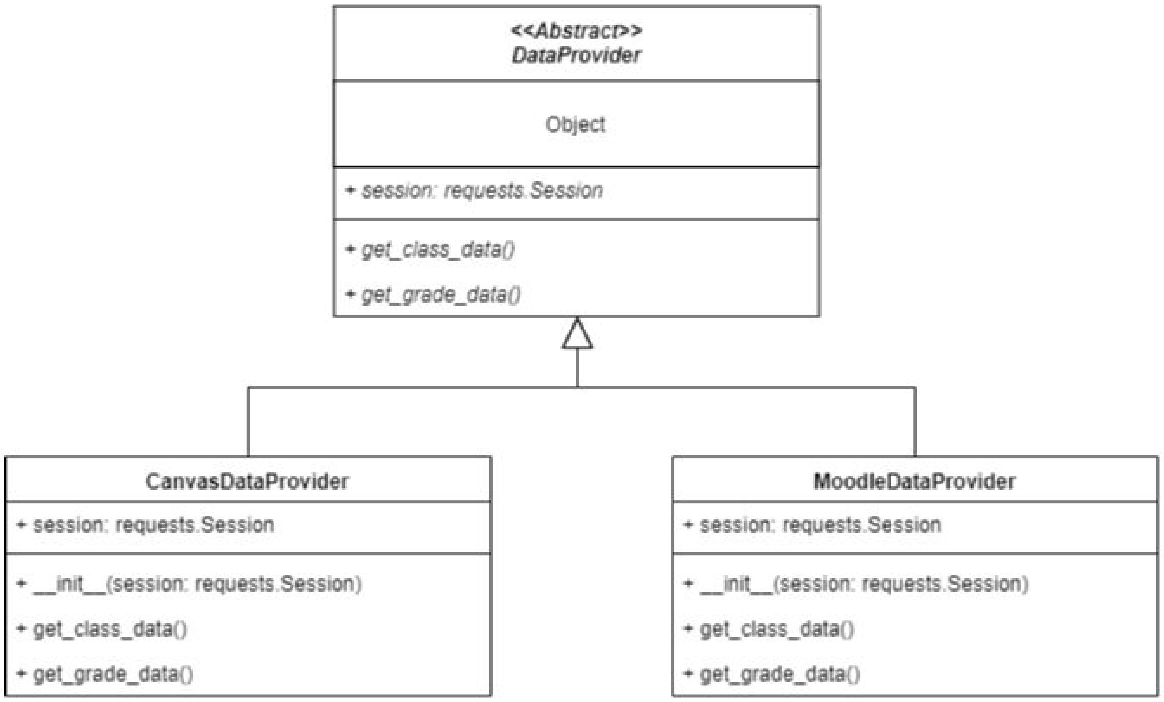
The class diagram went through a number of major changes between projects 4 and 5/6, but the changes between projects 5 and 6 were pretty minor.
Project 4 UML Diagram:

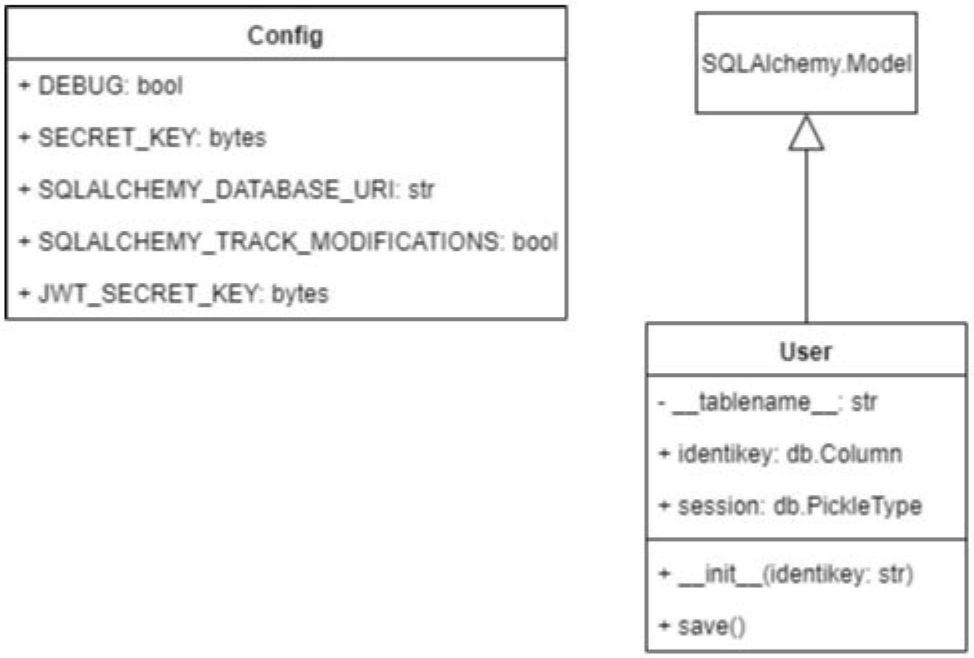


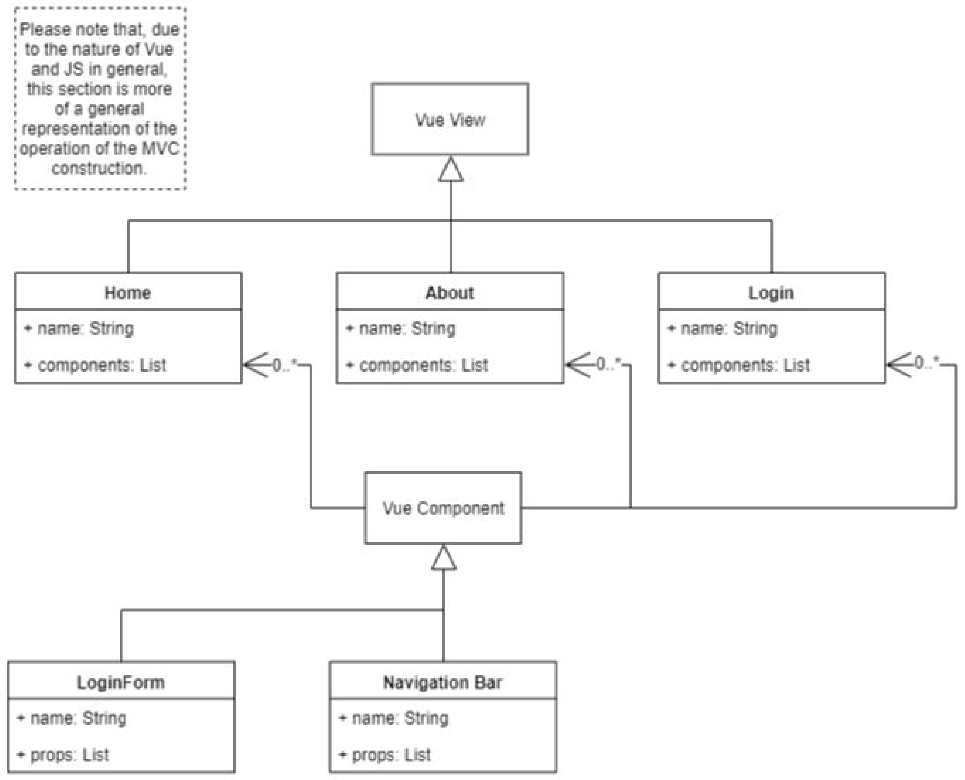
<<Abstract>>
DataProviderSession
# session: requests.Session
+ __init__()
+ login(username: str, password: str)
+ get_grades_json(): str
+ get_classes_json(): str
MoodleSession
+ login(username: str, password: str)
+ get_grades_data(): dict
+ get_classes_data(): dict
CanvasSession
+ login(username: str, password: str)
+ get_grades_data(): dict
+ get_classes_data(): dict
Vue.js
flask
PageController
+ __init__()
+ login(username: str, password:str): bool
+ request_data(username: str): bool
PageModel
+ state: dict
+ __init__(initial_state: dict)
+ set_state(data: dict): bool
+ get_state(): dict
PageView
+ __init__(initial_state: dict)
+ update(data: dict): bool
DatabaseConnector
- database_path: str
+ retrieve_record(username: str): dict
+ update_record(username: str, data: dict): bool
+ create_record(username: str, data: dict): bool
sqlite3

Project 5 UML Diagram:


<<Abstract>>
BaseSession
+ additional_login_behaviors: List[LoginBehavior]
+ session: requests.Session
- _base_login(username: str, password: str)
+ login(username: str, password: str): Union[requests.Session, None]
+ register_login_behavior(login_behavior: LoginBehavior)
- __finalize_login()
1
0..*
<<Abstract>>
LoginBehavior
+ login(session: requests.Session)
CanvasLoginBehavior
+ login(session: requests.Session)
MoodleLoginBehavior
+ login(session: requests.Session)
1
1
FedauthSession
+ session: requests.Session
+ __init__()
+ _base_login(username: str, password: str)




Config
+ DEBUG: bool
+ SECRET_KEY: bytes
+ SQLALCHEMY_DATABASE_URI: str
+ SQLALCHEMY_TRACK_MODIFICATIONS: bool
+ JWT_SECRET_KEY: bytes
SQLAlchemy.Model
User
- __tablename__: str
+ identikey: db.Column
+ session: db.PickleType
+ __init__(identikey: str)
+ save()


Please note that, due to the nature of Vue and JS in general, this section is more of a general representation of the operation of the MVC construction.
Vue View
Home
+ name: String
+ components: List
About
+ name: String
+ components: List
Login
+ name: String
+ components: List
0..*
0..*
0..*
Vue Component
LoginForm
+ name: String
+ props: List
Navigation Bar
+ name: String
+ props: List

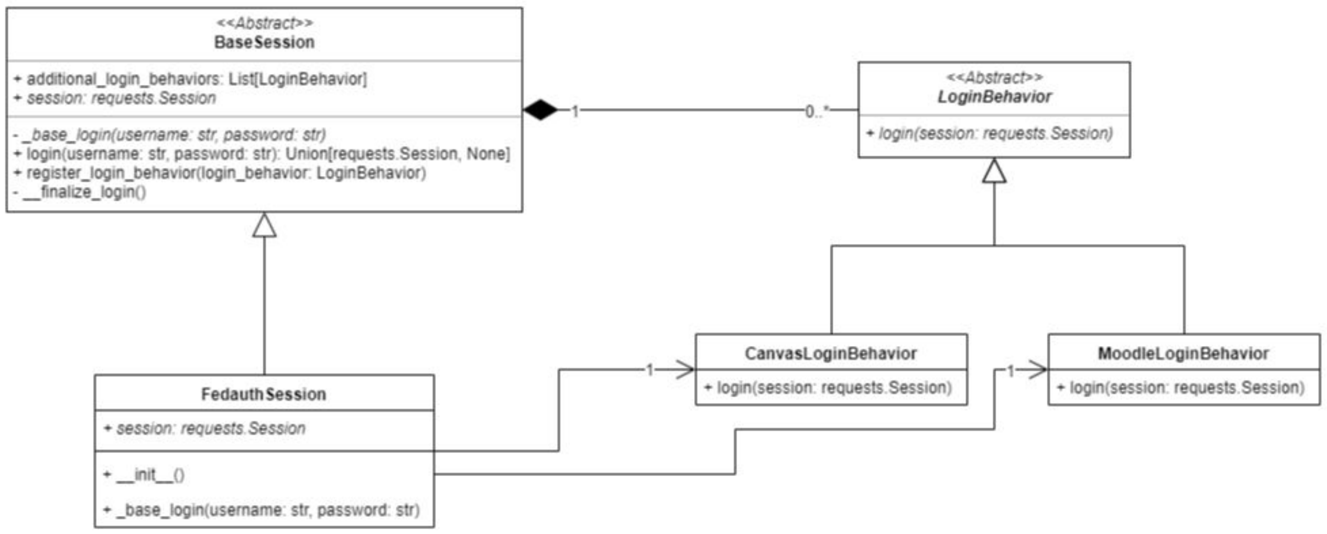
Here, we can see a number of major changes between project 4 and project 5. First, the DataProviderSession that we initially envisioned was transformed into a mix of the BaseSession, DataProvider, and LoginBehaviors. The BaseSession provides the interface for logging into the CU Federated Identity Service, which is realized by the FedauthSession class. In addition, the BaseSession class provides a method to register additional login behaviors (i.e. additional CU services), which are provided by the LoginBehavior class in the form of a strategy pattern, and realized by the corresponding classes (CanvasLoginBehavior and MoodleLoginBehavior). In this way, the BaseSession also operates as a template pattern, requiring the implementing class to implement the base_login method, but while already having defined the overall login method, which in turn calls the additional login behaviors.

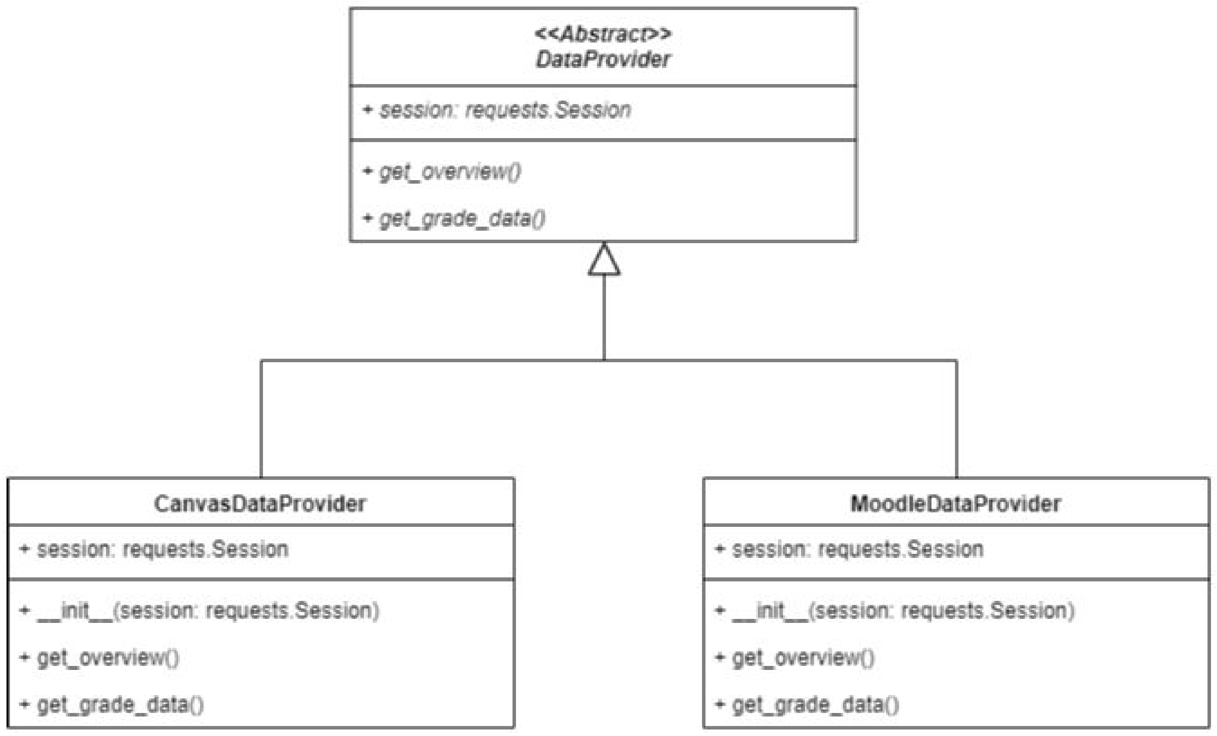
Next, and perhaps the most simple class, is the Config class, which Flask reads when it initializes itself. While normally classes that simply act as "data holders" are not encouraged, this was the simplest way to provide a startup configuration to Flask.

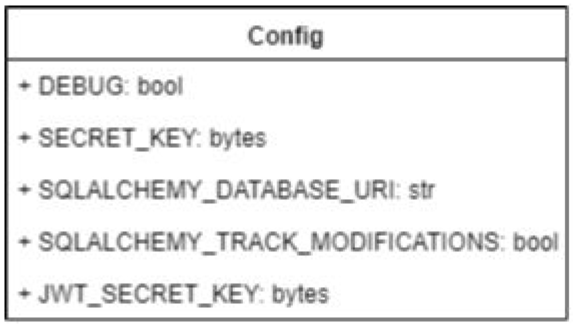
Another change between project 4 and 5 was the removal of the DatabaseConnector class. Instead of connecting directly to the database and having specific methods for record creation, we instead used an ORM. Specifically, we utilized Flask-SQLAlchemy and created a User class that extended the base model (as seen in the diagram for project 5). We utilized the Flask built-in SQLite database, though it would be easy to extend to other SQL-like databases as well. The User model contains the table name, so the ORM knows where to save it to, the user's IdentiKey as the primary key, and a pickle object so that their logged-in HTTP session can be serialized. In addition, the User model contains a save method, which saves the calling instance of that model to the database on disk.

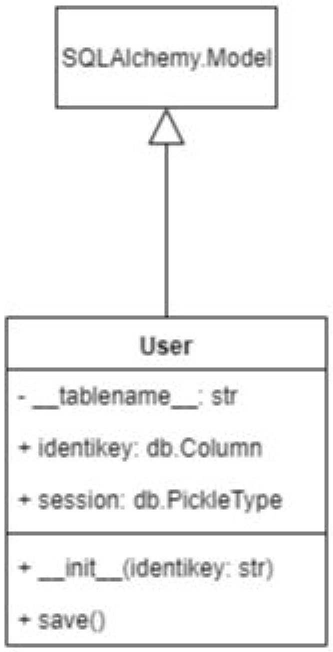
Finally, we can see the "UML" diagram for the frontend, which makes use of Vue.js. Vue.js is not necessarily a class-based architecture, which is why traditional UML diagrams don't necessarily make sense. However, Vue.js does make heavy use of the MVVM, or the Model–view–viewmodel, design pattern. At its core, Vue.js is made up of components. A component can define a view, and each view can also include multiple sub-components. Each component has data elements, a name, props, methods, and a list of sub-components. Additionally, for our project we leveraged Vue-Router, which delivers a single-page application experience. That is, the page is never actually reloaded and instead the content on the page is dynamically re-written every time. As such, the page itself is the ViewModel, the View is whatever component is being rendered, and the model is that view's data.

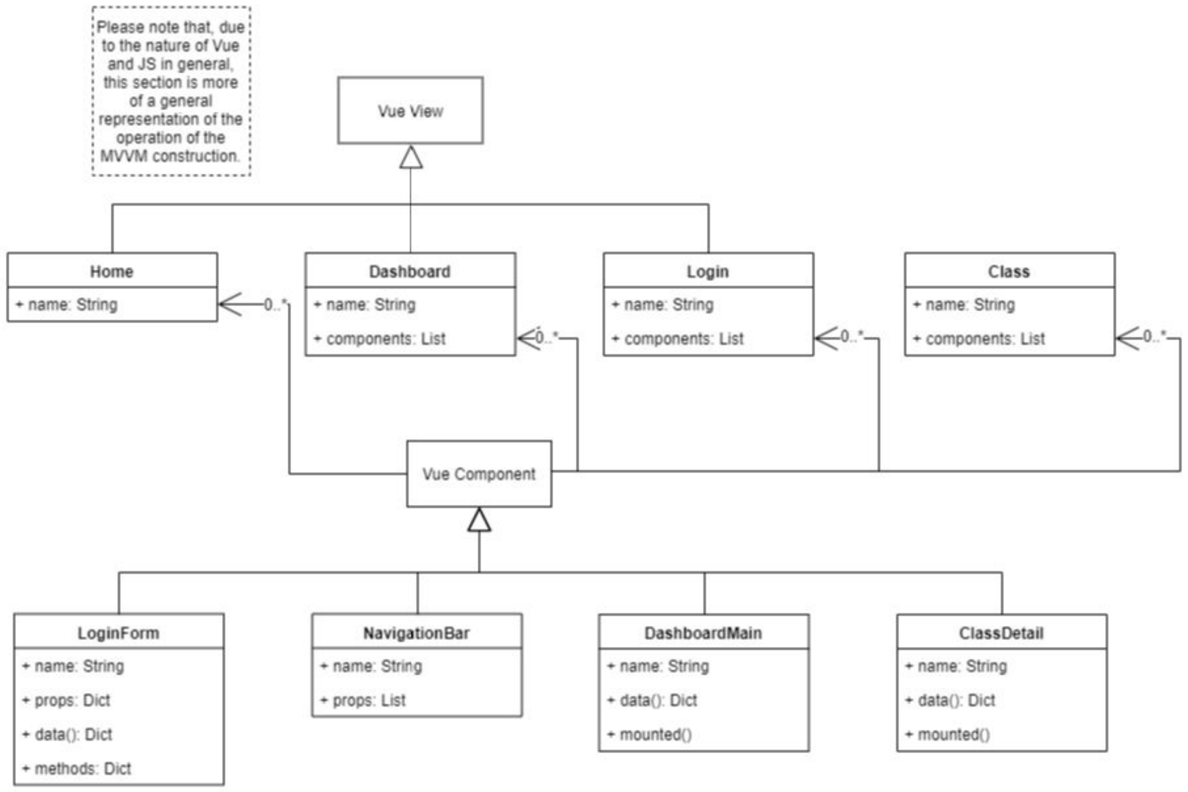
The Project 6 UML diagram is shown below:


<<Abstract>>
BaseSession
+ additional_login_behaviors: List[LoginBehavior]
+ session: requests.Session
- _base_login(username: str, password: str)
+ login(username: str, password: str): Union[requests.Session, None]
+ register_login_behavior(login_behavior: LoginBehavior)
- __finalize_login()
1
0..*
<<Abstract>>
LoginBehavior
+ login(session: requests.Session)
FedauthSession
+ session: requests.Session
+ __init__()
+ _base_login(username: str, password: str)
1
CanvasLoginBehavior
+ login(session: requests.Session)
1
MoodleLoginBehavior
+ login(session: requests.Session)




Config
+ DEBUG: bool
+ SECRET_KEY: bytes
+ SQLALCHEMY_DATABASE_URI: str
+ SQLALCHEMY_TRACK_MODIFICATIONS: bool
+ JWT_SECRET_KEY: bytes


SQLAlchemy.Model
User
- __tablename__: str
+ identikey: db.Column
+ session: db.PickleType
+ __init__(identikey: str)
+ save()


Please note that, due to the nature of Vue and JS in general, this section is more of a general representation of the operation of the MVVM construction.
Vue View
Home
+ name: String
Dashboard
+ name: String
+ components: List
Login
+ name: String
+ components: List
Class
+ name: String
+ components: List
0..*
0..*
0..*
0..*
Vue Component
LoginForm
+ name: String
+ props: Dict
+ data(): Dict
+ methods: Dict
NavigationBar
+ name: String
+ props: List
DashboardMain
+ name: String
+ data(): Dict
+ mounted()
ClassDetail
+ name: String
+ data(): Dict
+ mounted()

Here, we see very few changes in the UML diagram between project 5 and project 6. The get_class_data method defined by the DataProvider class was renamed to get_overview, and more views and components were added to the Vue section. Overall, the following design patterns were used:

- Strategy pattern - The BaseSession class delegated additional login behaviors to the concrete instances of the LoginBehavior class
- Template pattern - The BaseSession requires that the user implement the base_login method in concrete classes, but the login method is defined. It calls the base_login method that the user implemented and then calls any additional login behaviors registered (the strategy pattern above)
- MVVM pattern - The Vue.js model, especially as a SPA, directly uses the MVVM pattern, where the ViewModel is the page/DOM, the view is a Vue component, and the model is the data elements of those components.
- Decorator pattern - While not explicitly shown in the diagram, the backend code makes extensive use of the native Python decorators to wrap functions and provide additional functionality and authentication guards.
- State Pattern - While we did not leverage this very much, the Vuex module provides a consistent state interface for the frontend.

Third-Party code vs. Original code Statement

The following third-party frameworks were used in our project:

- Node.js - JavaScript runtime environment that serves as the base for Vue.js.
- Vue.js - An open-source MVVM frontend framework built on top of Node.js specializing in "single-page" applications and progressive web development.
- Vuetify - A material design component framework for Vue.js
- Vue Auth - A JWT library for Vue.js
- Axios - An HTTP client for Node.js
- Flask - A lightweight, backend web framework built on Python.
- Flask-JWT-Extended - A JWT library for Flask
- Flask-Caching - A caching library for Flask
- Requests - An HTTP library for Python
- webargs - A Python library for parsing and validating HTTP requests
- Flask-SQLAlchemy - A Flask ORM for SQL databases.

The above lists the most important dependencies. However, it does not list dependencies of dependencies or other smaller dependencies. For a full list of all dependencies used, the requirements.txt and the package.json can be queried for Python and JavaScript, respectively.

The login form was adapted from this Vuetify example.

The data table displaying the overall class list was adapted from this GitHub issue.

The Flask initialization and factory pattern was adapted from this blog post.

The Vue Auth and the Flask-JWT-Extended documentation was consulted extensively to setup authentication on the frontend on the backend, and boiler plate code was copied and adapted from the aforementioned documentation.

Overall, however, save for any code that was pre-generated by Vue.js and Flask, the remainder of the code is original.

Statement on the OOAD process for your overall Semester Project

- Shifting goals - After initially choosing to design a game for this project, the overwhelming scope of that task became apparent and we eventually switched to our current course info aggregator design. This pushed us back by about a week or so, but the transition went smoothly and we adapted quickly to the new project.
- Team cohesion - Our group faced some difficulties in building a cohesive team dynamic. This resulted in some repeated work, miscommunications, and other minor squabbles. Ultimately, we learned from our mistakes and gained back our footing in order to put together a more cohesive project.
- Unexplored territory - While the group was generally familiar with Python and JavaScript which underlined our backend and frontend frameworks respectively, learning the Flask and Vue.js frameworks themselves turned out to be a more daunting challenge. With Vue's strict xml-like configuration files and Flask's specialized sub-libraries for caching, web communication, and more, taking time to learn these frameworks was a must.