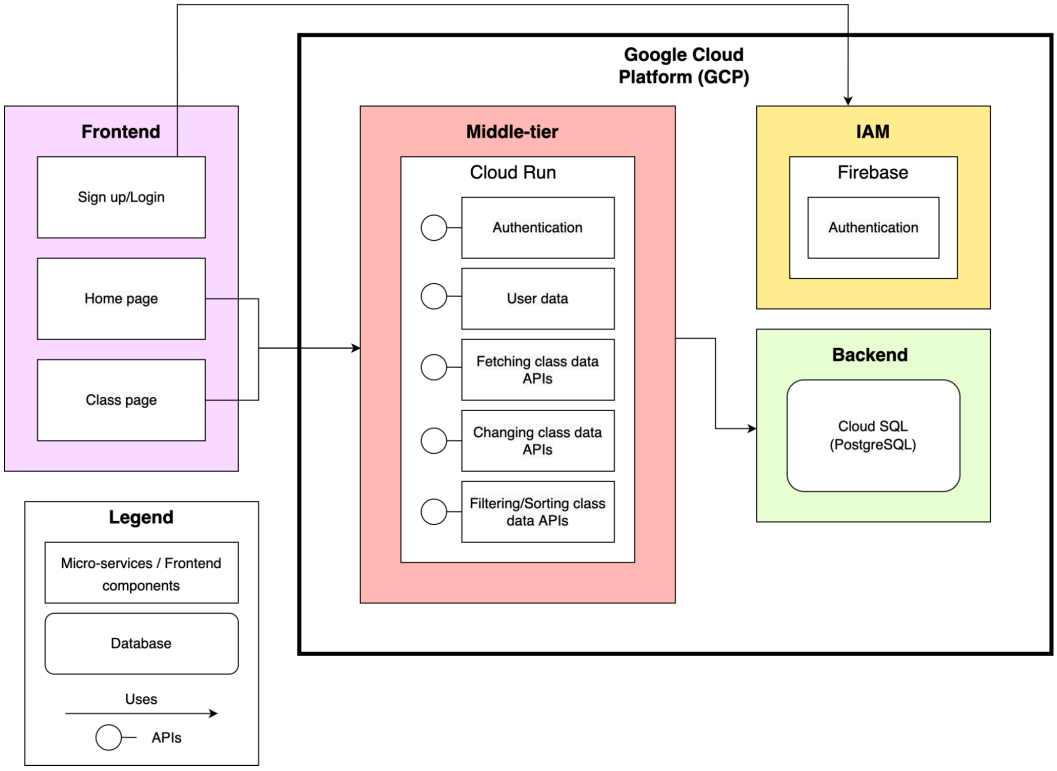

For the frontend of this application, I chose to use the Flutter framework because it allows for easy cross-platform app development. This means that although I decided to release my MVP on the IOS platform, I could release the same app to another platform such as Android with the majority of the code unchanged. This is a crucial capability that other similar technologies do not possess. Additionally, flutter is coded in the Dart language, which is extremely similar to Java, one of the coding languages that I am familiar with. This prior knowledge enabled me to adapt to the Flutter framework very quickly.

For the middle tier of this app I used Python because of its popularity and the fact that I wanted to expand upon the prior knowledge that I had with the language so that I could utilize it in this real world application. I am interested in learning Machine Learning in the future and Python is the de facto language for that. As for the technology of this tier, I chose Cloud Run, which is provided by the Google Cloud Platform (GCP), to run my web services. Cloud Run allows you to run a container, a vessel containing all

of your code and its dependencies, in a serverless environment. By eliminating the use of a physical server and hosting in the cloud instead, I achieved a more scalable and cost efficient middle tier.

For the backend of my application I selected Cloud SQL to manage my PostgreSQL database. This service is also provided by GCP, and is a robust and reliable way of storing and fetching data. I chose to use a relational database because of its scalability, and the fact that it met my requirements for the data that I would be working with.

In regards to the structure of this application, I opted for the three-tier architecture shown in the diagram because it offers the three key advantages of Scalability, Security, and efficient development.This architecture enable my app to support large number of users because both the middle tier and the backend Cloud SQL can be easily scaled to support increasing load. Scaling is also more cost efficient because the two tiers can be scaled separately.

Another reason I settled on this architecture is the possibilities of enhanced security that it provides. The frontend does not communicate directly with the backend database because it must go through the middle tier first. This structure enables stronger security measures to be put in place regarding user data. For example, if the frontend code were to be hacked, an additional layer of security could prevent the backend and hence the user data from being exposed. Lastly, the three tiered architecture provides for much quicker development, and updates. Due to their separated nature, the frontend and middletier can be worked on and updated independent from one another as long as the APIs signature does not change. For example, if a bug arises in the logic of the program when a user tries to filter their class data, I can simply edit the API that controls this function in my middle tier code, deploy the new container to Cloud Run, and the issue will be fixed. Without the tiered architecture, I would need to re-deploy the entire app to the app store and await a reviewing process which could take days or weeks longer. Likewise, when making edits to the frontend and deploying the new version to the app store, the process would take less time because just the frontend code is being deployed instead of the middle tier code being packaged along with it.