**Architecture and Design Document**

**StudiBuddi**



Hunter Davis

Oscar Gepiga

Tymee Kong

Tanner Mindrum

Shihong Sun

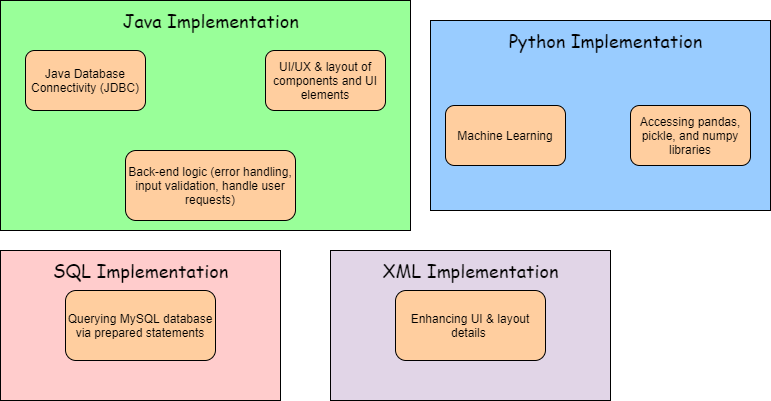
CECS 491A

Jean Tsao

4/11/2020

**Diagrams**

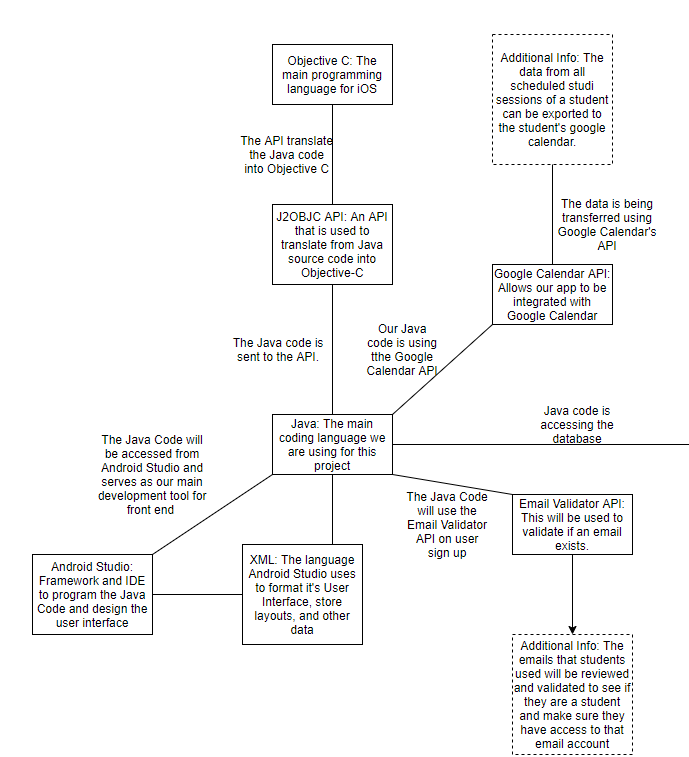
***Language diagram***



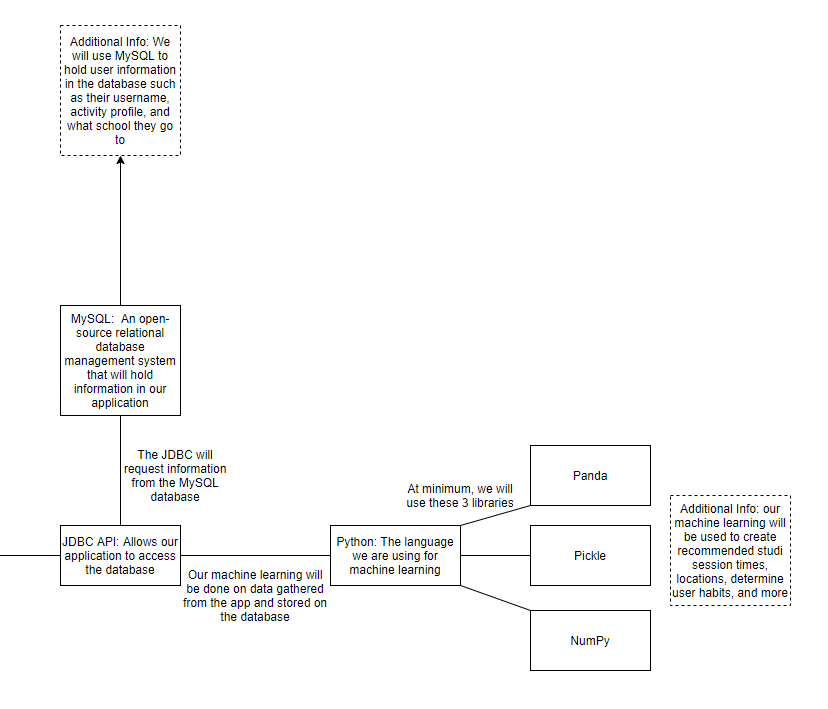
***Reasoning for each language choice***

* **SQL via MySQL**
  + SQL will allow us to query a MySQL database we are using to store necessary user and application data. The data stored in this database could be anything from a user’s student email address to tracking all the set study session times for specific users or to simply storing information about popular study locations on campus. For us to load this data into the app and for users to save information, we must use SQL to read and write from the MySQL database. We have used both the MySQL Workbench to directly write SQL queries to the database and prepared statements within our main software via JDBC.
* **Java**
  + We are using Java for almost every aspect of programming in our application. First, Java is being used to create the front-end design of our application. Since our app is being built for Android, the front-end design, logic, and UI/UX of our app can be built through Java code. Second, Java will be used for any back-end programming. This includes error handling, user input validation, taking user’s inputs and correctly performing what they request, any logic needed to make the app function will be done in Java. Lastly, we are using Java via the Java Database Connectivity (JDBC) library. JDBC allows us to embed SQL queries in our main software classes. We use this Java library to read and write directly to the database in real time.
* **Python**
  + We will incorporate Python programming into our application for machine learning purposes. As a group, we have limited experience with machine learning, so we are not exactly sure of what ways we’ll be specifically using Python. However, we do know that we would like to implement a machine learning model that will detect a user’s patterns, mainly their studying habits (e.g. who they study with, where, how long, what subjects), and automatically create study sessions based off of those patterns. To accomplish this task, we believe we’ll need work with Pandas, Pickle, and NumPy libraries.
* **XML**
  + In addition to building the front-end of our application using Java, XML will also be used to design the UI of our application. Certain aspects of the layout of our program (e.g. centering the text boxes for the login page or multi-tabbed page layout) can be made using XML. Android Studio and XML work nicely together to provide applications with deeper access to editing layouts, so whenever we may need to clean the look and feel of our app, it is possible we will use XML to achieve those tasks.

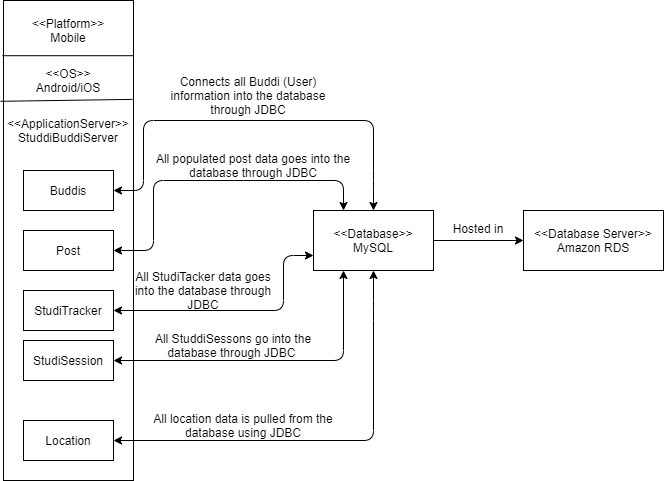
***Interface diagram***



***Interface diagram continued***

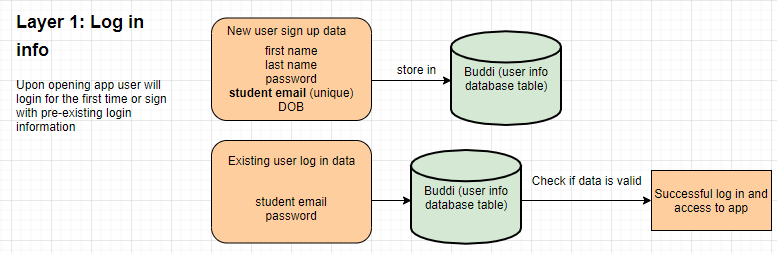


***Deployment Diagram***

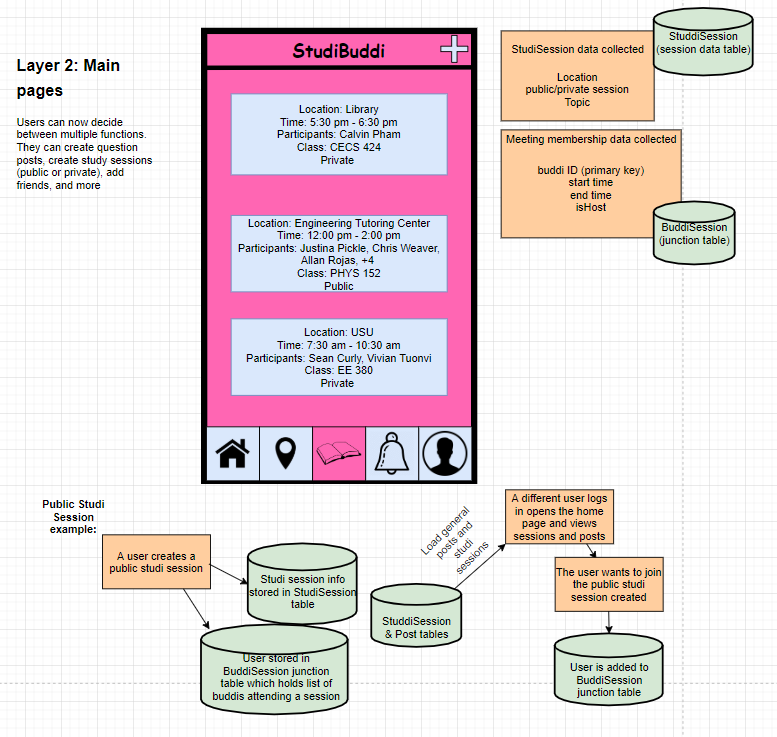


***Information flow for primary data items***

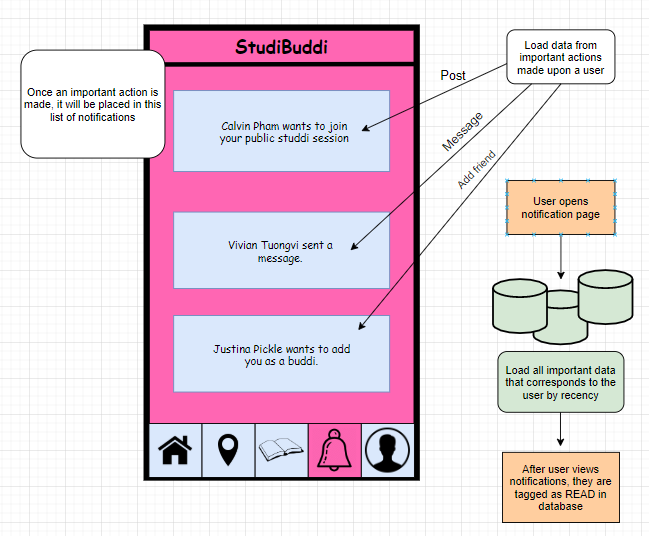
*User login data flow*



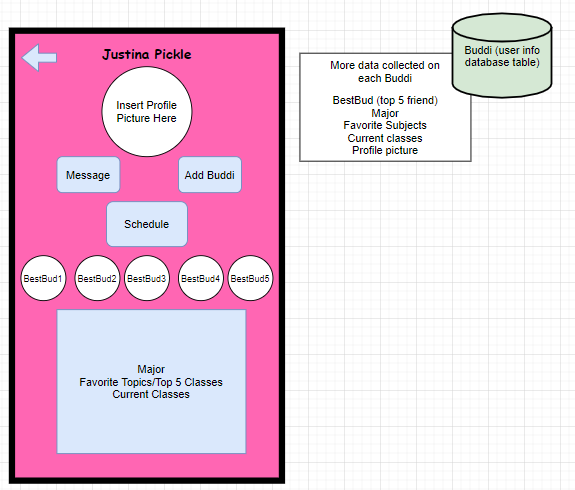
*Studi Sessions data flow*



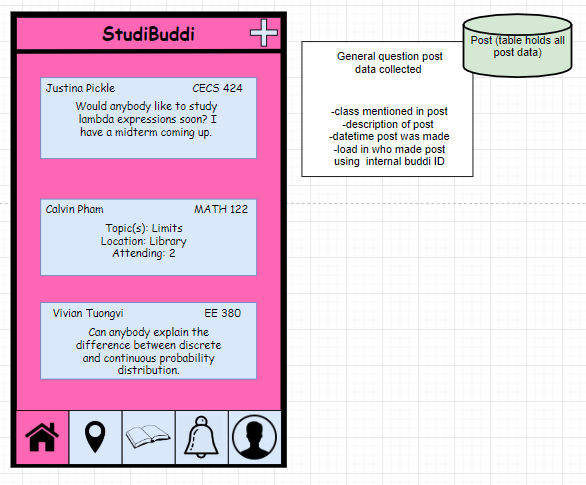
*Notifications data flow*



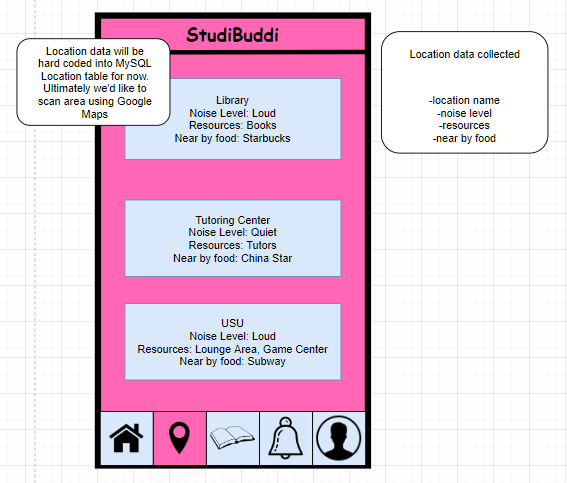
*User details data flow*



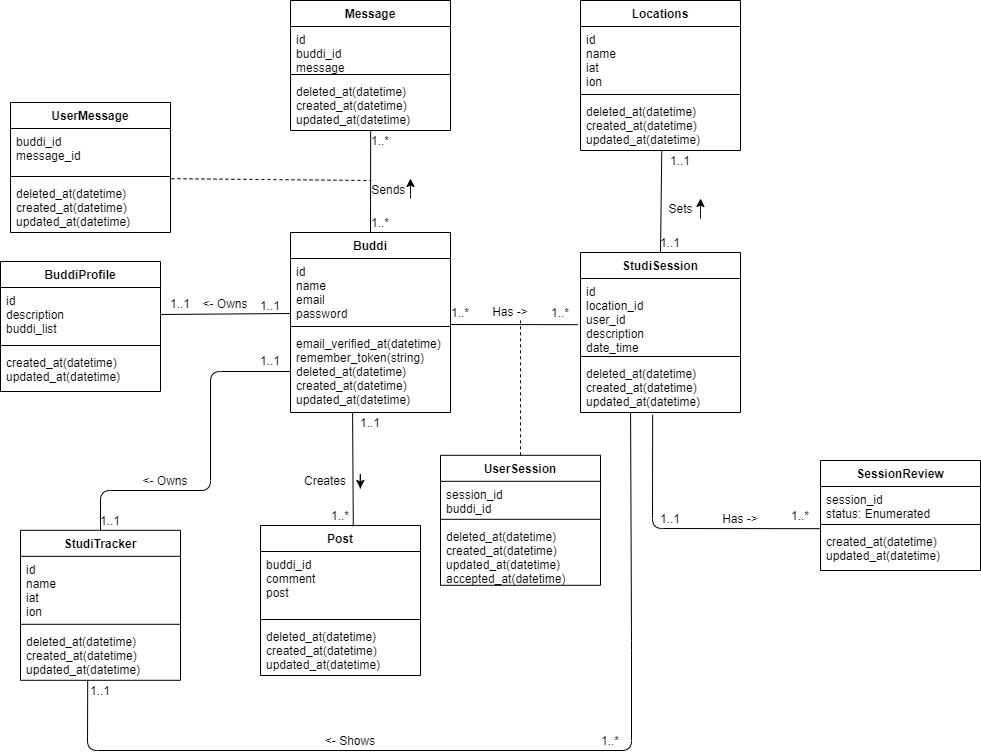
*General question posts*



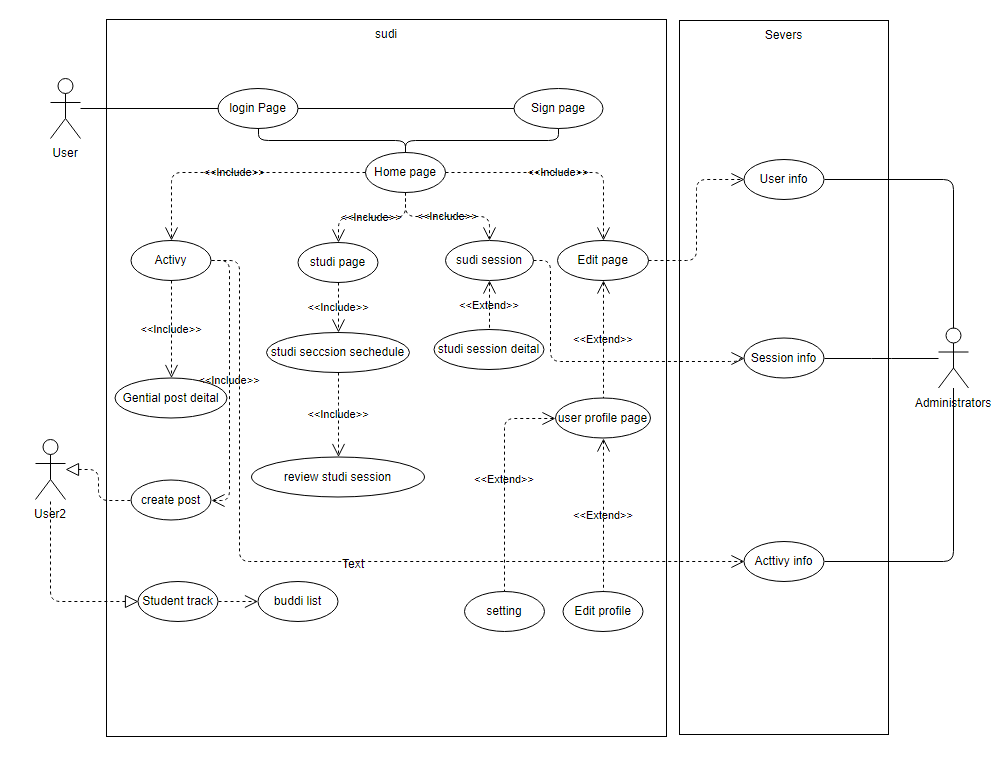
*Locations*



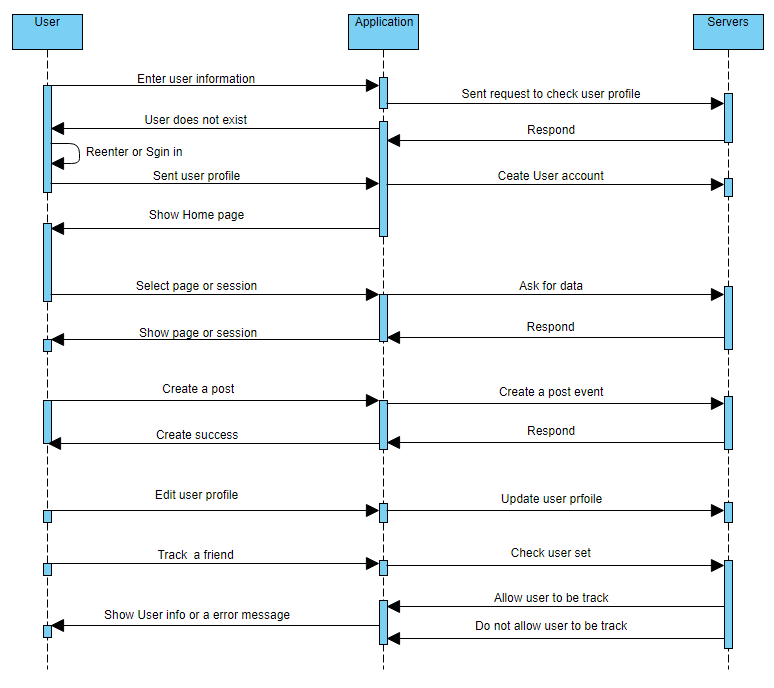
***Class Diagram***



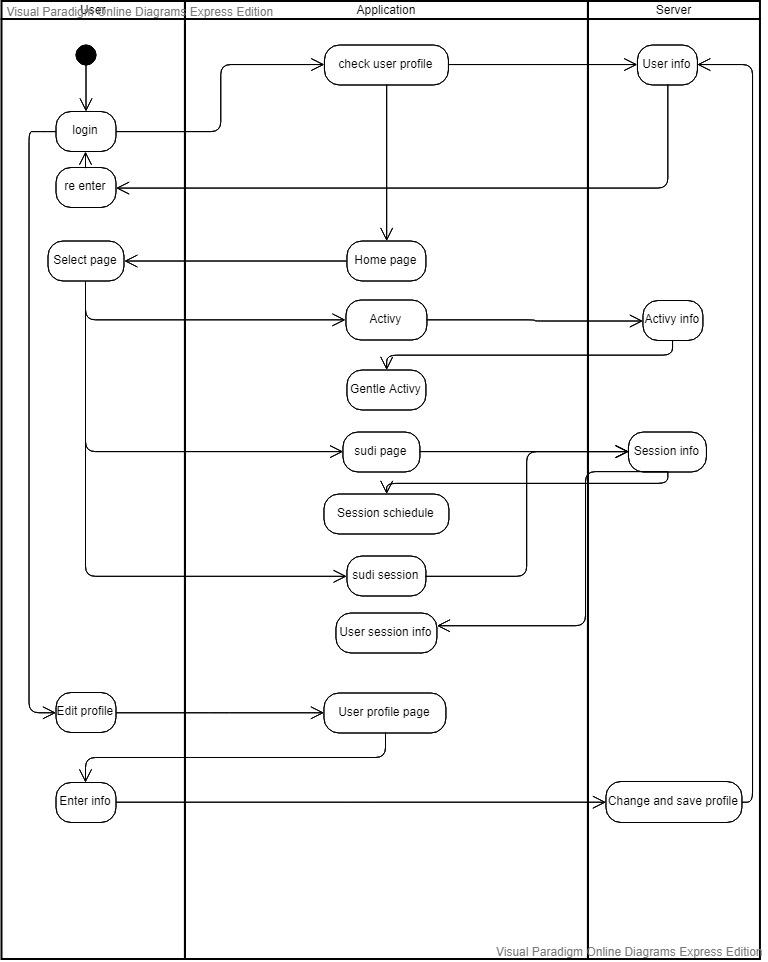
***Use case Diagram***



***Sequence Diagram***



***Activity diagram***



**Tradeoff Analysis**

***Major Architecture Patterns Employed***

* **Event-Driven Architecture**
  + We are using Event-Driven Architecture because this type of architecture is best with applications that have user interfaces. StudiBuddi will be mainly interacted with through the user interface. It will constantly wait for user input to trigger an event. For example, if a user wants to create a studi session with another user, they have to tap certain buttons on the application that allow them to create a studi session. Furthermore, using an Event-Driven Architecture would make it easier to scale. Once we start finishing the basic functionalities of the application, it will be easier to add other functions to the application. This could also be easier in the future when more users start to use our program.
  + Some drawbacks to using an Event-Driven Architecture is that if there is high coupling among modules, it can create problems when adding new features to the application. This could also lead to troublesome error handling when many modules are used on the same event trigger.
* **Client-Server Architecture**
  + We are using a client-server pattern because StudiBuddi will be an online application that allows users to connect with each other. There are many features that we want to add such as posting about subjects that students are struggling with and allowing other students to comment on the posts. These types of features need to send messages to the server so all users using the application can see the same amount of content that the server sees.
* **Event-Bus Pattern**
  + We are using the Event-Bus Pattern because this is a useful architecture when developing for Android Studio. There are four major components in this architecture, the event source, event listener, channel, and event bus. The event source is similar to components on a user interface with buttons. Once they are interacted with, they have an event listener that corresponds with whatever click is made. The event sources then send messages to particular channels using an event bus. Furthermore, this type of architecture could be used to send notifications to users that are subscribed to certain channels and are able to listen to the channels. If this occurs, we could send notifications from our application to the users.

***Mobile vs Web Tradeoff Analysis***

|  |  |  |  |
| --- | --- | --- | --- |
| Criterion | Weight | Mobile | Web |
| Our Technical experience | 15 | 10 | 5 |
| Primary Market Research Results | 30 | 25 | 15 |
| User Base (for our demographic) | 30 | 25 | 15 |
| Accessibility | 10 | 10 | 8 |
| Desired Features | 15 | 12 | 9 |
| Total | 100 | 82 | 52 |

For the tradeoff of Mobile vs Web, we heavily took our primary market research and demographic into account. We understand that a quick and efficient study session planning software would be much more useful in the hands of students as a mobile app rather than a webapp that would be more focused on a user planning their sessions through a laptop, where the quick streamlined experience would be slowed down. On a mobile phone, we could tailor this experience to busy students and provide ease-of-use to the end user.

***Component Choice Tradeoff Analysis***

1. **Android vs iOS Tradeoff Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| Criterion | Weight | Android | Apple |
| Our Technological Experience | 20 | 15 | 5 |
| Accessibility | 10 | 8 | 7 |
| Audience | 10 | 7 | 8 |
| Timeline | 5 | 3 | 4 |
| Software Features | 20 | 15 | 10 |
| Hardware Features | 25 | 23 | 20 |
| Performance | 10 | 8 | 8 |
| Total | 100 | 79 | 65 |

In our analysis of Android vs iOS, we took into consideration the devices of our teammates and our technological experience. We have some experience in Android development, and we figured any person in the class could run an android emulator on their laptops and be able to try our platform. We are planning on using an API to port to iOS, but we figured to start development in Android for the time being.

1. **Amazon RDS vs Microsoft SQL Server**

|  |  |  |  |
| --- | --- | --- | --- |
| Criterion | Weight | Amazon RDS | Microsoft SQL Server |
| Pricing | 25 | 20 | 10 |
| Management | 25 | 23 | 20 |
| Compatibility | 25 | 22 | 18 |
| Durability | 25 | 20 | 20 |
| Total | 100 | 85 | 68 |

With our servers we have chosen to use Amazon RDS. One of our main criteria was the pricing. Amazon RDS’s service is based on how much is used and includes a free year of use. Microsoft SQL Server has pricing options that are too expensive for a whole product that we are not going to use all of. Another important criterion was the management of the databases as Amazon RDS edges Microsoft SQL Server by providing metrics for the databases as well as allowing us to run the database through the cloud. In terms of durability Amazon RDS also beats Microsoft SQL Server by providing automated backups with the ability to go back into previous instances of the database. Lastly Amazon RDS offers compatibility with many database technologies just in case there is a scenario where we have to switch databases.

***Database Tradeoff Analysis***

|  |  |  |  |
| --- | --- | --- | --- |
| Criterion | Weight | MySQL | Firebase |
| Design | 20 | 17 | 10 |
| Model | 20 | 18 | 12 |
| Community | 5 | 5 | 1 |
| Standardization | 20 | 18 | 10 |
| Schema | 5 | 2 | 3 |
| Flexibility | 20 | 10 | 13 |
| Experience | 10 | 10 | 0 |
| Total | 100 | 80 | 49 |

In analyzing these two Database platforms, we allocated weight to the importance to our team. We ultimately decided to use MySQL largely to our team’s experience in developing using the database. We also wanted to emphasize a full stack development process throughout this class, to give ourselves experience in this type of development. MySQL works in our advantage because we can model tables that follow our class design. While MySQL isn’t as flexible as Firebase, it will work well in our program because it will act as the main source of data for our program and we will rarely have to change the database with the way we are designing our application. Also, MySQL allows us to use JDBC in our application, which will be an easy way to query and add data into our main database.

***Server Architecture Tradeoff Analysis***

|  |  |  |  |
| --- | --- | --- | --- |
| Criterion | Weight | Server | Serverless |
| Pricing | 10 | 6 | 9 |
| Networking | 25 | 23 | 18 |
| 3rd Party Dependencies | 15 | 15 | 11 |
| Environments | 20 | 16 | 18 |
| Timeout | 15 | 13 | 10 |
| Scale | 15 | 14 | 14 |
| Total | 100 | 87 | 80 |

For our Server vs. Serverless tradeoff analysis our criteria is pricing, networking, 3rd party dependencies, environments, timeout, and the scale. In general if we were to go serverless, it would be cheaper. However, with Amazon RDS giving a free year of server usage this criteria isn’t as important to us. In terms of networking, serverless architecture wasn't scored as high because serverless functions can only be used as private API’s. For this to be implemented we would have to establish an API Gateway. For 3rd party dependencies our project relies on many libraries that aren’t built into the languages we are using. If we chose to go serverless we would have to package these large libraries into the application itself, which is something we do not want to do. Serverless architectures have the advantage when setting up different environments. With serverless architecture there will only need to setup one environment. Having a server architecture is superior in terms of timeout because computing in a serverless environment has a hard 300-second timeout limit. We do not anticipate our application to have complex or long-running functions however to be safe we prefer having the server architecture. Lastly the scaling process for a serverless architecture is automatic and seamless but has a lack of control. This lack of control is something that we do not want personally in our senior project.

***Framework Tradeoff Analysis***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criterion | Weight | Android Studio  (Java, Kotlin) | React Native (Javascript) | Xamarin  (C#, .NET) |
| Ease of Use | 15 | 10 | 10 | 8 |
| Accessibility | 20 | 15 | 15 | 12 |
| 3rd Party Libraries | 15 | 15 | 12 | 8 |
| Experience | 20 | 18 | 10 | 15 |
| Language | 15 | 15 | 10 | 10 |
| Performance | 15 | 15 | 15 | 12 |
| Total | 100 | 88 | 72 | 65 |

For our Framework analysis, we considered using Android Studio, React Native, and Xamarin to develop our application on. Language and our development experience came into play as some of our main factors in choosing, as well as being native in android was a plus for us. This allows us a lot of freedom in android-specific usages and implementations of UI, multithreaded tasks, and interfacing with the Android Operating System.

***Language Tradeoff Analysis***

|  |  |  |  |
| --- | --- | --- | --- |
| Criterion | Weight | Kotlin | Native Java |
| Ease of Use | 15 | 10 | 13 |
| Accessibility | 20 | 10 | 15 |
| 3rd Party Libraries | 20 | 14 | 18 |
| Experience | 25 | 0 | 20 |
| Performance | 20 | 14 | 20 |
| Total | 100 | 48 | 86 |

For our Language Tradeoff, we decided overall to pursue native Java programming. Our team is highly familiar with Java, and we would like to do a full stack implementation using it. We know our application will have lots of functionality built in this way, and we have native access to Android’s device peripherals, native android UI and strong performance. Another strong point for native Java development is access to third party libraries and plugins to improve usability and accessibility for users of our app.

***Extra Credit Tradeoff Analyses***

**Machine Learning Model Tradeoff Analysis**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criterion | Weight | Logistic Regression | Classification and Regression Trees | multi-layer perceptron |
| Speed | 10 | 8 | 9 | 5 |
| Accuracy | 35 | 30 | 28 | 30 |
| Skill Estimate | 10 | 8 | 8 | 5 |
| Repeated Estimates | 15 | 12 | 10 | 13 |
| Distribution of Estimates | 15 | 12 | 10 | 10 |
| Central Tendency | 15 | 12 | 12 | 12 |
| Total | 100 | 82 | 77 | 75 |

For our machine learning model tradeoff analysis, we choose three classical models of popular models to compare with. Since none of our team members has full experience in Machine learning topics, it is good to start our pick from what we need it to do. Per our design of the program, we need our model to rank the weight of each session and give the user a time schedule to work with. Meaning, we need our machine learning to recognize the study habits and patterns of each student. So, our model needs to recognize the importance of the session and give a weight for the session, then we can base the weight rank of each session and show the user a customized schedule based on their study patterns. Based on these requests, the result that we chose is the logistic regression model. This is a linear model usually used on voice or handwritten number recognition. The running speed and accuracy are quite high compared to most of the models, but it is also easy to understand and use with the project we have. By using this model, we could easily distinguish the importance of each session and give it an appropriate weight or learn users’ patterns, which helps us to build customized, personalized schedules. Other models could do the same, but the Logistic Regression should be the easiest to use and the fastest.

**Cloud Decisions**

For our app, we decided to host the server with Amazon RDS, giving us a cloud platform for our database that we can all access as developers. This database will grow over time as we add more functionality to the user and the data that needs to be held such as user preferences, friends list, comments, etc. We already have extensive plans on how we are going to build up our database in terms of user profiles and are planning on implementing that by code release 2.

**Server-Side Validation vs Client Side Validation Tradeoff Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| Criteria | Grading | Server Side Validation | Client Side Validation |
| Security | 60 | 60 | 40 |
| User Experience | 40 | 30 | 40 |
| Total | 100 | 90 | 80 |

For Server-Side vs Client Side validation we only care about the security and the user experience. Client-side validation would be useful as it allows a better user experience through its quick response time. When using Client-Side Validation, all the inputs will be checked through the client meaning there will be no constant round trips to the server saving time and network traffic. However, as an app that is user based, we need to be more secure for the malicious user. If we were to use Client-Side Validation, a malicious user can easily manipulate their way into our server and database.

**Security Decisions**

For security of our app, we have considered many things. Mainly, we discussed the safety of user’s emails and passwords in our database. We are planning on using a method to hash these passwords, so they are not visible to anyone with access to the database, and also for the database itself to be encrypted. This is something we will address before we deploy the app to the app store and research before CECS 491B.

**Logs/Monitoring Decisions**

We plan on adding in CECS 491B live monitoring of our app through our own database and through the Google Play Store. The Google Play Store has lots of functionality provided in monitoring users and app usage. Once we deploy the app these tools will immediately be available to us. We also planned on adding a live monitoring interface for us developers, to be able to pull certain statistics from the database and monitor app usage that way. This will also lead into our machine learning models, as these can help us visualize where in the application, we want to implement our models on.

**CI/CD Choices**

Our team will be doing continuous integration and delivery, but not continuous deployment when developing this app. Currently we are using an agile-like development process to add features on a daily and weekly basis, but these features won’t be added to the actual app immediately. We also want to enable automated testing for our development and that is something we will add in 491B.

**Risk Management**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Description | Mitigation Scheme | Severity Level | Date of Identification | Status |
| 1 | Machine learning Model selection | Analyze the data and find which model is most fit to use | Low | 3/7/2020 | Not started |
| 2 | Student identity check | Student Email check, maybe student ID check in future | Medium | 3/7/2020 | Not started |
| 3 | User privacy protection | Encrypting our database and using a salt password | High | 3/7/2020 | Not started |
| 4 | User safety protection | Allowing the user to share location and status to an  Emergency contact | High | 3/7/2020 | Not started |
| 5 | User account recovery | Answering security questions or have a code be sent to their phone number | High | 3/8/2020 | Not started |
| 6 | Filtering appropriate posts | Analyze posts that a user can create and determine if that post is appropriate | High | 3/8/2020 | Not started |
| 7 | User profile vulnerability | Allows users to make their profile private so only friends can view their profile. | High | 3/8/2020 | Not started |
| 8 | Project acceptance | Deciding whether or not our project will be accepted by the public. | High | 4/11/2020 | In progress |