# Tutor by Request

### Design and Planning Document

### 2021/03/01, version 1.0

## Document Revision History

Rev. 1.0 2021/03/01: initial version

Rev. 1.1 2021/03/26: after iteration 1

## System Architecture

**Activity/Fragment**

For our Android mobile application, we will be utilizing Activities/Fragments for the main structure of the application. These are UI-based classes that handle interaction from the user and are responsible for routing to other areas of the application. At present we have one main activity with fragments for all other parts of the program. After learning the limitations of this implementation, we’ve decided to at least have a separate activity for login and the rest of the program. Ultimately we’re considering using separate activities for login, student, tutor, and student/tutor sections. Doing so has proven to have its own set of challenges and due to time restrictions we’ve decided for the first iteration to stick to a simpler structure.

**ViewModel**

ModelView is a reusable view in our application that will allow us to store and use UI-related data. This ViewModel will interact with activities and fragments directly to give the user a different experience based on the model.

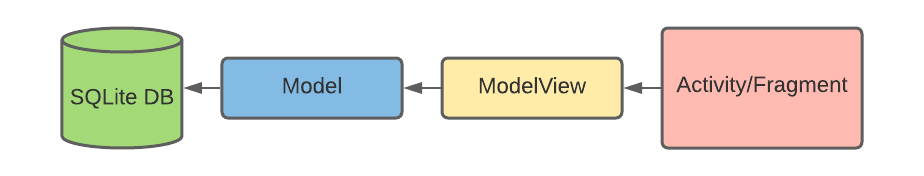
**Model**

For our architecture, we plan to keep Activities/Fragments as lean and simple as possible. We plan to utilize Model classes as the driving force of our application. Instead of storing information in Activities/Fragments, after logging in, we will populate a Model class with information about the user and this class will be used to drive the interactions within the application. This sort of architecture will result in better performance, but also better scalability.

**Database-Driven Architecture**

We’re not being allowed access to the UW database. Because of this we’re working on setting up a remote database so that the project can go forward. Because of the issues with setting up the remote database, we’re currently creating a local database on the emulator that’s populated through the main activity of the app.

Using what we defined in the areas above, we plan to have a database driven architecture, where the experience a user receives from us is based on the data we have on them in the database. Data about tutors and students will drive how each user interacts with the other and is the driving force for the application. This architecture will allow the application to to dynamically handle different types of input and interaction between users in a simple, straightforward manner.



## **Design Details**

**Frontend**

# The front end of our app will be built in Android Studio in the Java language. Android Studio gives us a GUI interface to more easily design the layout and structure of the app. We had originally left the implementation of the Frontend open ended to allow us to explore all possible options during development since we were still very inexperienced with app development and all of the possibilities. At present we have one basic main activity with many fragments branching off from it. This has worked very well for getting started and allowing us to program all of the functions that we want in the app. Moving forward we’re considering ways to make it so that there’s a more usable navigation system so that the user doesn’t have to go back to the homepage any time they want to go to a different section. The reason we currently haven’t implemented it as such is because of the challenges that come along with that.

To iterate, and clear up any confusion that may persist, at present, with the simple layout, there are simply buttons on the home page that allow the user to go to each section, My Sessions, Get A Tutor, Set Availability, etc. For the next iteration, we plan to implement a more robust navigation system that allows the user to navigate to any page to any other page (within their activity based on user type). We’re going to leave many of the details of that open for now in order to allow ourselves the freedom to explore and not marry ourselves to any one implementation. However, the two leading ideas at present are to have a consistent global navigation bar at the top of each page with buttons that navigate to each fragment or to have a tabbed activity that allows the user to navigate between the sections by swiping left or right.

**Backend**

* **Framework:**

For our framework we’ll be using Java as this and Kotlin are the main languages used in Android app development and we are all far more familiar with Java. Android Studio provides a seemingly infinite amount of libraries to assist with this.

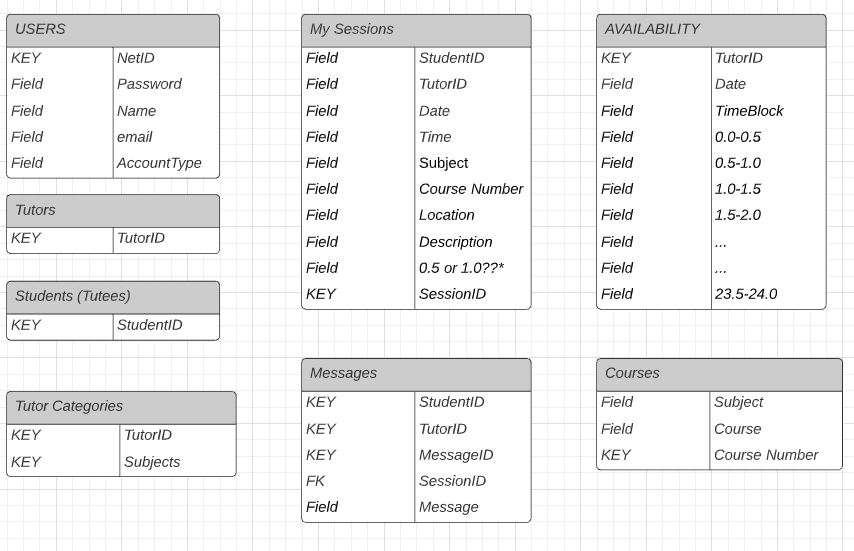
* **Hosting:**

If this app goes public, hosting will be handled by UW - Madison. For the purposes of testing we are looking into using a friend’s server or using Evan’s desktop as a server.

* **Database:**

Android Studio supports SQL commands and that is how we’ll be doing everything from setting up the database to populating and querying it.

* **Database Schema:**



* **Algorithms**
  + Searching
    - Our program will use SQL queries to search through our database to return various fields relating to our program classes.
    - For example, each student using the UW-Madison Undergraduate Learning Center (ULC) Tutoring By Request (TBR) service, has a unique NetID that is required for login.

Query: SELECT \* FROM Student\_Table WHERE NetID = : [NetID value].

This key value can be used to retrieve all student information from our database, and can be passed to various fragments and activities in order to interface with other student functions (e.g. scheduling a session between a student and tutor Student.scheduleSession(String StudentNetID, String TutorNetID, String Date, String Time, String Subject, String CourseNum, String Location, String Description)

* + Input validation
    - Login Page
      * According to ULC policy, only eligible CoE students are permitted to sign-in and use the Tutoring By Request Service. Since our team will not be given access to the UW-Madison student database at this time, we plan to simulate a database of eligible students, only allowing proper NetID/username and password matches to access the main page of the application.
      * Input validation will be done with the built libraries of Android Studio.
        + NetID will only allow for letters and numbers with a max of 20 characters.

android:maxLength="20"

android:digits="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"

* + - * + Password will allow for all characters, numbers, and special characters.
    - Get A Tutor (schedule session)
    - Student.scheduleSession
    - (String StudentNetID, String TutorNetID, String Date, String Time, String Subject, String CourseNum, String Location, String Description)  
      * Input variables to schedule a tutoring session (String Date), will first be narrowed down to display a maximum of seven days (one week) at a time. The seven days that will be presented as valid choices for the user, will be determined by the “week selection” list populated with week long date ranges corresponding to the weeks of a semester.
      * Input variables to schedule a tutoring session (String Subject, String CourseNum) will be handled by a set ListViews populated by our “Courses” database table which contains only valid input for available classes for tutoring.
      * Input variables to schedule a tutoring session (String TutorNetID, String Time) will be selected from color coded time blocks corresponding to an available tutor.
        + The available tutors will be populated by Tutor Subjects database table corresponding to the subject and course selection made previously.
        + The available time blocks, will be populated from the Availability database table corresponding to the available tutors (based on subject and course fields selections)
        + If the course and subject fields have not been chosen, the calendar will remain empty and the user will not have the option to select a time block to schedule a tutoring session
      * Input variables to schedule a tutoring session (String Description) will be input by the user once a time block selection has been made. The format of the user input will be in the form of a textbox with a character limit of 400 chars. Users will also be able to select a label corresponding to the topic they would like to focus on during the session (Homework, Project, Exam, Concepts, Review).
      * Once all fields have been completed, the user will be able to select the Confirm button to confirm the session and add the data to the Sessions database table.
      * If the user hits cancel, the user interface will return to the calendar view and the user will be able to select a different available time block for the previous subject and course entries that were made.
    - My Sessions
      * Users will be able to view their scheduled tutoring sessions based on the entries in the Sessions database table corresponding to the NetID of the current user.
      * Stretch goal: be able to send messages to the tutor of a particular scheduled session.
        + Users will be able to enter a message in a text box with a character limit of 400 characters.
        + If the user hits send, this string will be added to the Messages database table, along with the sessionID, StudentID, TutorID, MessageID.
* **Protocols**
  + The only protocol that is of concern to us is TCP/IP. This will be used to connect our app to the server that contains the database.
* **Security framework**
  + SQL Injections
    - Login
      * Whitespace not allowed for username or password.
    - Session Description
      * Wrap description text in quotes to ensure text is not interpreted as SQL query
    - Messaging
      * Wrap message text in quotes to ensure text is not interpreted as SQL query
    - Other
      * All other input for the program will be done by drop-down lists that are pre-populated with only valid input (i.e. not an SQL query injection)

## Implementation Plan

Dependency plan:

Our main functionality, scheduling of sessions, is highly interdependent and needs to be handled carefully and methodically. The main necessity as such is to get a database running as soon as possible and start implementing functionality from there. In the case that the database is not yet ready, a dummy database is also hardcoded in to test the functionality of certain tasks without the need of any other. Not all areas of the database are necessary to be functional for the interdependent tasks to start working and as a result those areas are lowered in priority or pushed to a following iteration.

**Legend:**

**Dependency:** Index Number

**Responsibility:**

Pair programming: Student 1 & Student 2

Primary/Secondary responsibilities: Student 1/Student 2

**Iteration 0: Preprogramming stage**

| **Index** | **Task** | **Difficulty**  **(1-5)** | **Priority**  **(1-5)** | **Time units** | **Dependency** | **Responsibility** |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | Setting up git repo | 1 | 1 | 1 | - | Evan/Ryan |
| 1 | Create forks | 1 | 1 | 1 | 0 | Edwin/Harrison |
| 2 | Hardcode Dummy database | 1 | 1 | 2 | - | Ryan/Azzed |
| 3 | Create code skeleton and UI Navigation Graph | 2 | 1 | 2 | - | Evan/Edwin |

**Iteration 1: Minimum Viable Product**

| **Index** | **Task** | **Difficulty**  **(1-5)** | **Priority**  **(1-5)** | **Time units** | **Dependency**  **(Index)** | **Responsibility** |
| --- | --- | --- | --- | --- | --- | --- |
| 4 | Build User model | 3 | 1 | 1 | - | Edwin/Azzed |
| 5 | Set up SQlite database | 3 | 1 | 2 | - | Ryan/Harrison |
| 6 | Define database Schema for Users | 2 | 1 | 1 | 4,5 | Ryan & Evan |
| 7 | Define database Schema for Tutor availability | 2 | 1 | 1 | 5 | Azzed & Edwin |
| 8 | Define database Schema for Sessions | 2 | 2 | 1 | 5,7 | Harrison & Ryan |
| 9 | Define Grid Class | 2 | 1 | 1 | - | Evan/Harrison |
| 10 | Build login view model | 3 | 1 | 2 | 4 | Ryan/Harrison |
| 11 | Login UI and functionality   * Verify credentials and direct User to appropriate activity | 2 | 2 | 2 | 4,5,10 | Harrison/Azzed |
| 12 | Get A Tutor page   * Functional drop down menus * Shows grid representing calendar * Updating grid based on selection * Selectable grid highlighted when touched * Ability to book time slot with a popup window to add details | 4 | 3 | 3 | 6,7 | Evan/Edwin |
| 13 | Tutor Calendar page   * Functional drop down menu * Selectable grid highlighted when touched * Ability to select time slot for availability | 4 | 3 | 3 | 6,7 | Azzed/Evan |
| 15 | Activity and fragment management   * Showing User only pages they have privileges to see * Switching between pages on click of a button | 2 | 2 | 3 | - | Harriso/Ryan |
| 16 | Testing   * Functionality * User Feedback | 2 | 2 | 5 | N/A | Edwin/Evan |

**Iteration 2: Fully Functional app**

| **index** | **Task** | **Difficulty**  **(1-5)** | **Priority**  **(1-5)** | **Time units** | **Dependency**  **(Index)** | **Responsibility (Primary/Secondary)** |
| --- | --- | --- | --- | --- | --- | --- |
| 17 | Define database Schema for Messages | 3 | 1 | 2 | 4,5,6 | Ryan/ Harrison |
| 18 | Define database Schema for Courses | 2 | 1 | 1 | 5 | Edwin/Azzed |
| 19 | Home page   * Show posts made by admin that are clickable * Show reminders of sessions that are clickable * Popup windows when clickable object is selected * Add clickable tick box to posts and update database * Add cancel appointment prompt to reminders | 2 | 4 | 3 | - | Azzed/Evan |
| 20 | My Sessions page   * Listed clickable sessions * Popup windows for sessions when clicked * textbox in popup window for messages between students and tutors * Rating Tutor after session is complete | 1 | 2 | 2 | 7,8 | Evan/Edwin |
| 21 | My Appointments page   * Listed clickable sessions * Popup windows for sessions when clicked * textbox in popup window for messages between students and tutors * Rating Student after session is complete | 1 | 2 | 2 | 7,8 | Azzed/Harrison |
| 22 | Messaging between Tutor and Student   * Ability to send messages from either My appointments page or My Sessions page * Recorded history of messages updated in database and shown on screen * Messages sent through email as well | 4 | 3 | 4 | 6,8,17 | Evan & Harrison |
| 23 | Tutor Profile page   * Prompt to add Profile picture * Prompt to add hobbies and graduation plans in textboxes * Import Major and Year from Database | 2 | 5 | 2 | 6 | Ryan & Edwin |
| 24 | Logout Button   * Logs user out to TBR | 1 | 5 | 1 | - | Edwin |
| 25 | Tutor Preferences page   * Ability to set preferences stored in database * Ability to choose courses and select proficiency stored in database * Ability to add comments for Tutee | 3 | 4 | 3 | 7 | Azzed/Evan |
| 26 | Testing   * Testing complete functionality * User feedback | 3 | 2 | 5 | N/A | Ryan & Harrison |

**Iteration 3: Stretch goals and added features**

| **Index** | **Task** | **Difficulty**  **(1-5)** | **Priority**  **(1-5)** | **Time units** | **Dependency**  **(Index)** | **Responsibility (Primary/Secondary)** |
| --- | --- | --- | --- | --- | --- | --- |
| 27 | Exclusive Messaging page for communication between tutors and students | 3 | 1 | 2 | 22 | Azzed |
| 28 | Search by tutor option | 2 | 1 | 1 | 6,7 | Harrison |
| 29 | Visual Improvements to UI | 2 | 4 | 3 | - | Ryan |
| 30 | Extensive Testing   * Testing all features in depth * User feedback | 1 | 2 | 2 | N/A | Evan & Edwin |

## Testing Plan

Much of the testing practices and procedures will be taken from Android Studios recommend testing procedures. Specifically the methods for setting up and executing unit tests and integration tests.

<https://developer.android.com/studio/test>

<https://developer.android.com/training/testing>

Unit Testing:

This should occur throughout all the iterations. Development should occur in segments with the underlying database being the first to be developed. As new features are designed and implemented new tests will be created. All back end and front end functions can be tested using JUnit extensions. Objects can be created for testing purposes and the necessary files needed to automate this process should be produced by the JUnit extension. For this we will need to develop the code to allow for testing without the full code being completed. This is easy enough to do with the front end as we can simply add functions and create unit tests whenever we add functionality. We also can add the UI modules so that independent testing of each unit can be set up using JUnit.

<https://developer.android.com/training/testing/fundamentals#small-tests>

Integration Testing:

For integration testing we will be required to ensure the separate methods within the class structure for the database all function together. The integration tests for this background data should be relatively simple and can be executed in the same testing directories as the unit tests themselves. These will mostly be internal tests that ensure all the functions implemented to allow quick access to the database are working correctly. Ensuring they don’t interfere with each will be a simple matter of combining the different segments and running the database through the automated tests prepared in the same way as the unit tests. The UI side of testing can all be accomplished using Android’s built in Espresso. This will likely occur when we have created the separated modules that we are each working on. After each section of the user interface is created and each internal fragment has been tested using Espresso we'll move on to integrating the separated activities together.

<https://developer.android.com/training/testing/ui-testing/espresso-testing>

System Testing:

As far as system testing goes this will likely be done manually as this will be the easiest way to conduct it. We will run the completed version on a variety of emulators on Android Studio and also using actual devices we hook up to it. We will use instrumentation to monitor and control activities while we run the whole system. We will also be able to implement all out test cases while running the main system to ensure total system functionality. Using the built in tools we can also write automated tests that send keystrokes and touch events to simulate all interactions under controlled test environments. Finally we will do manual testing consisting of simply running the application on various emulators and devices.

<https://developer.android.com/training/testing/ui-testing>

In this section goes a brief description of how you plan to test the system. Thought should be given to how mostly automatic testing can be carried out, so as to maximize the limited number of human hours you will have for testing your system. The purpose of this section is a reality check that your design will in fact be testable with a reasonable effort. You should specifically discuss design decisions that affect testing, and describe any test interfaces built into the system in this section. You must discuss here your plans for unit testing (approach, when are they created, when are they run), integration testing (what order), system testing (what kind), regression testing (how are you going to organize and run them).

Regression:

All the unit tests will be written concurrently with the design and implementation of the sections in code. Most will be automated and we can script them to run all at the same time. Each section will have its own test cases to run. Integration testing will occur after the internal sections are completed and tested. The thorough system testing will be a mixture of fully automated testing and manual testing. We will run the automated one periodically after every major commit. The manual testing will be much less scheduled and will occur at any time we see fit. All tests will be organized in a separate test folder within the main directory. The tests within the test folder will be organized by frontend and backend and then by section(Login, Tutor, etc)