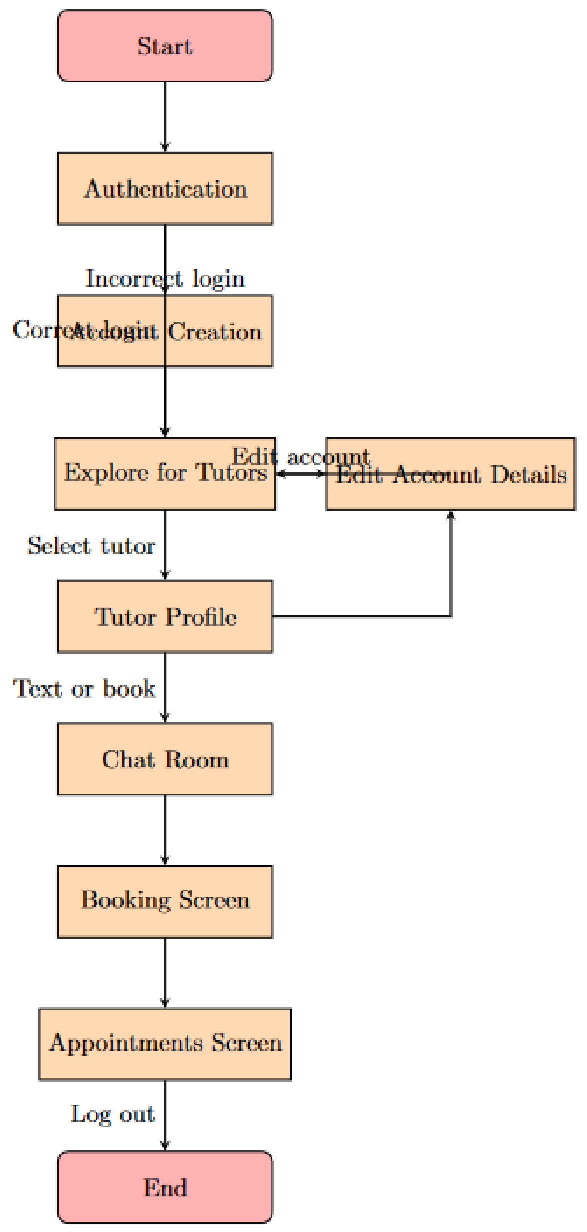
Vidhya Narayanan

CRITERION B:

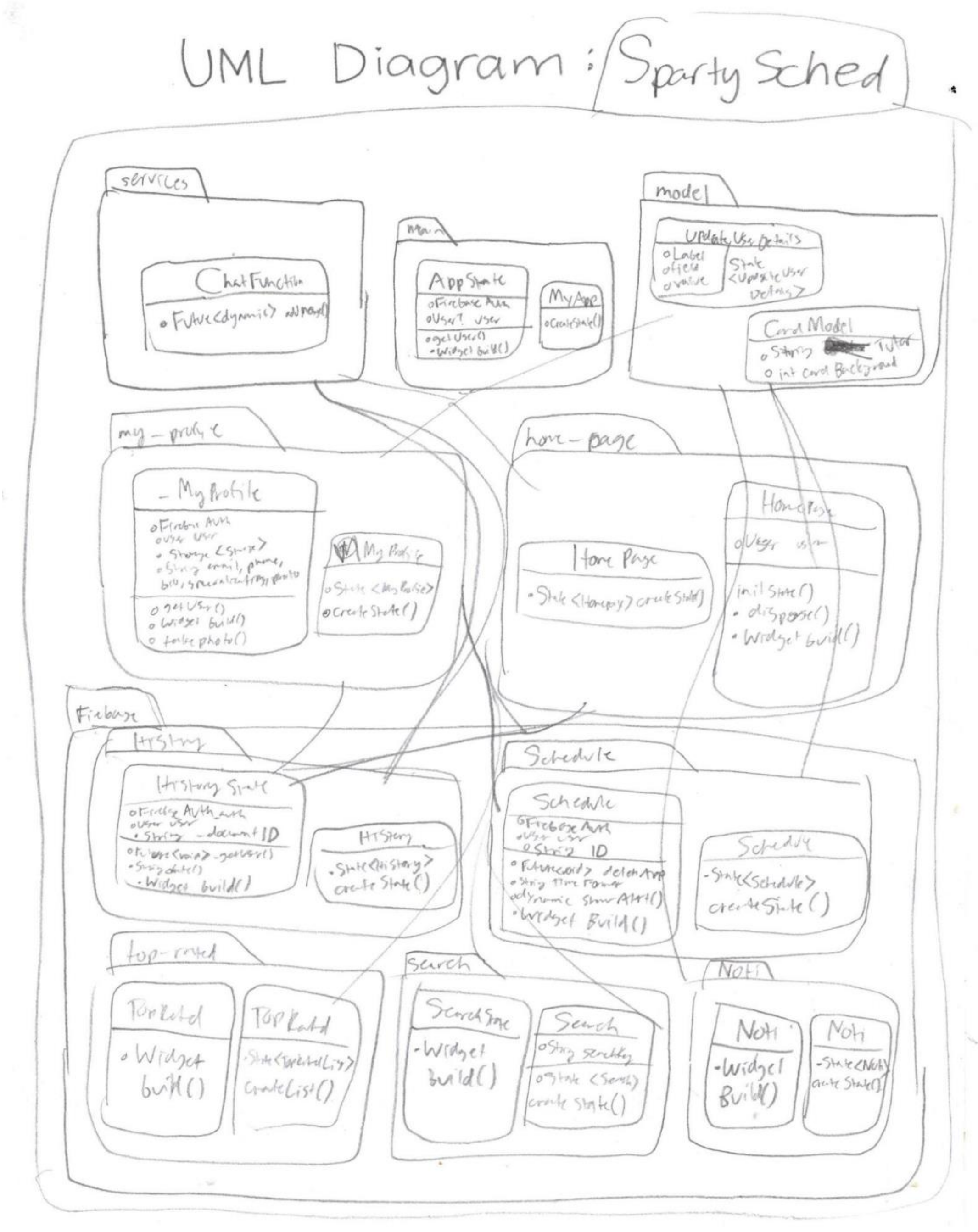
Part I: Relevant Flowcharts and Diagrams

User FlowChart:

I Intended for the flowchart to be more of a basic guideline for how I would structure my app, but the main focus of my planning was in my UML Diagram: Here is my flowchart:

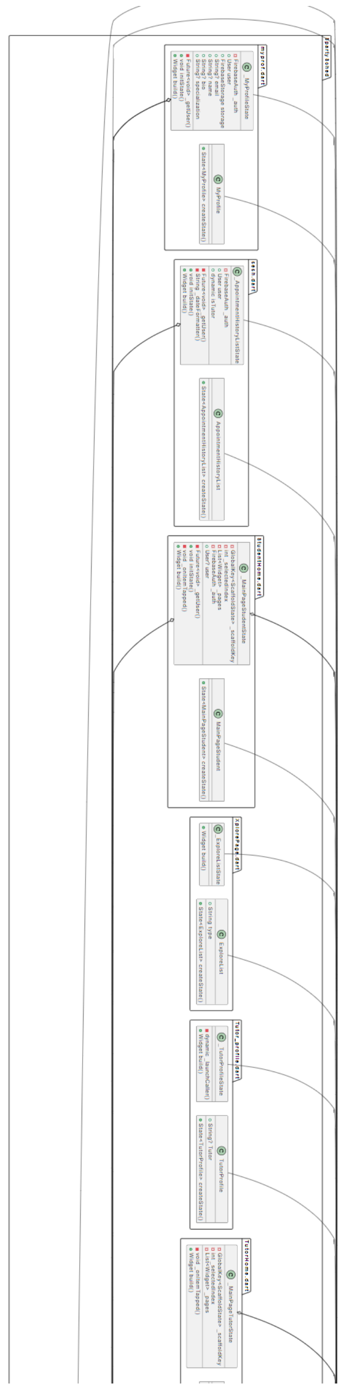


UML Diagram Version #1:



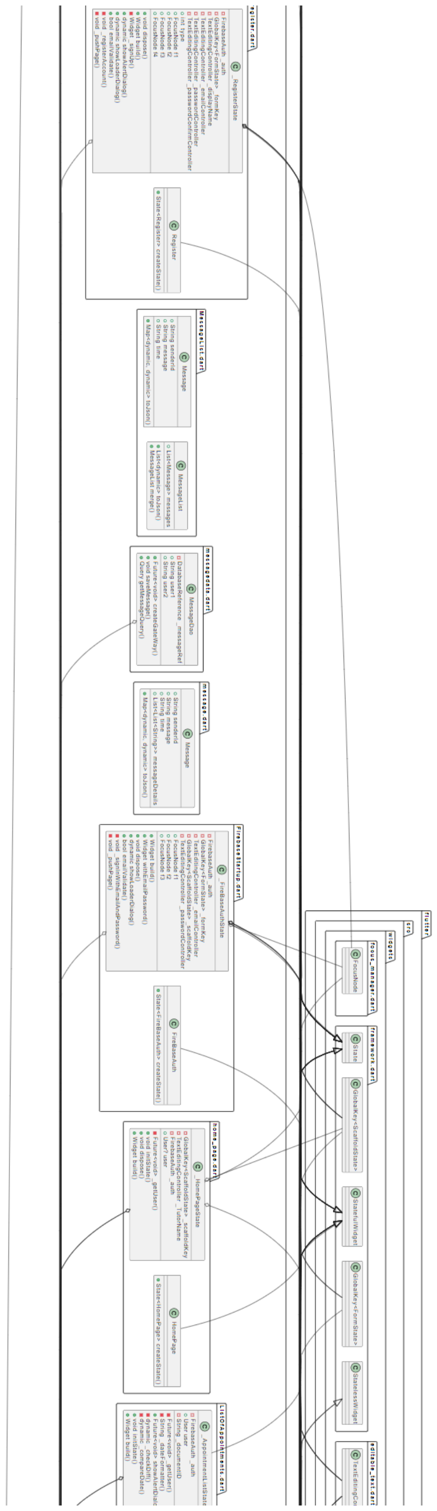
UML Diagram Version Two:

Part one:

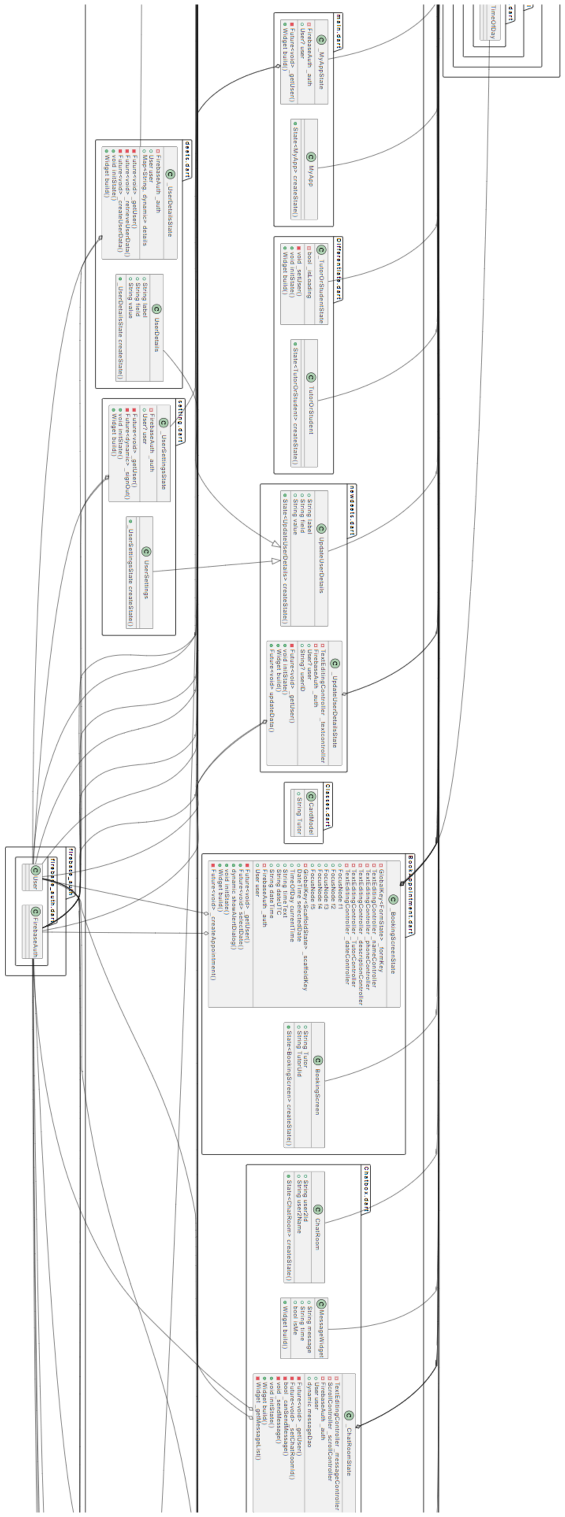


Part one:

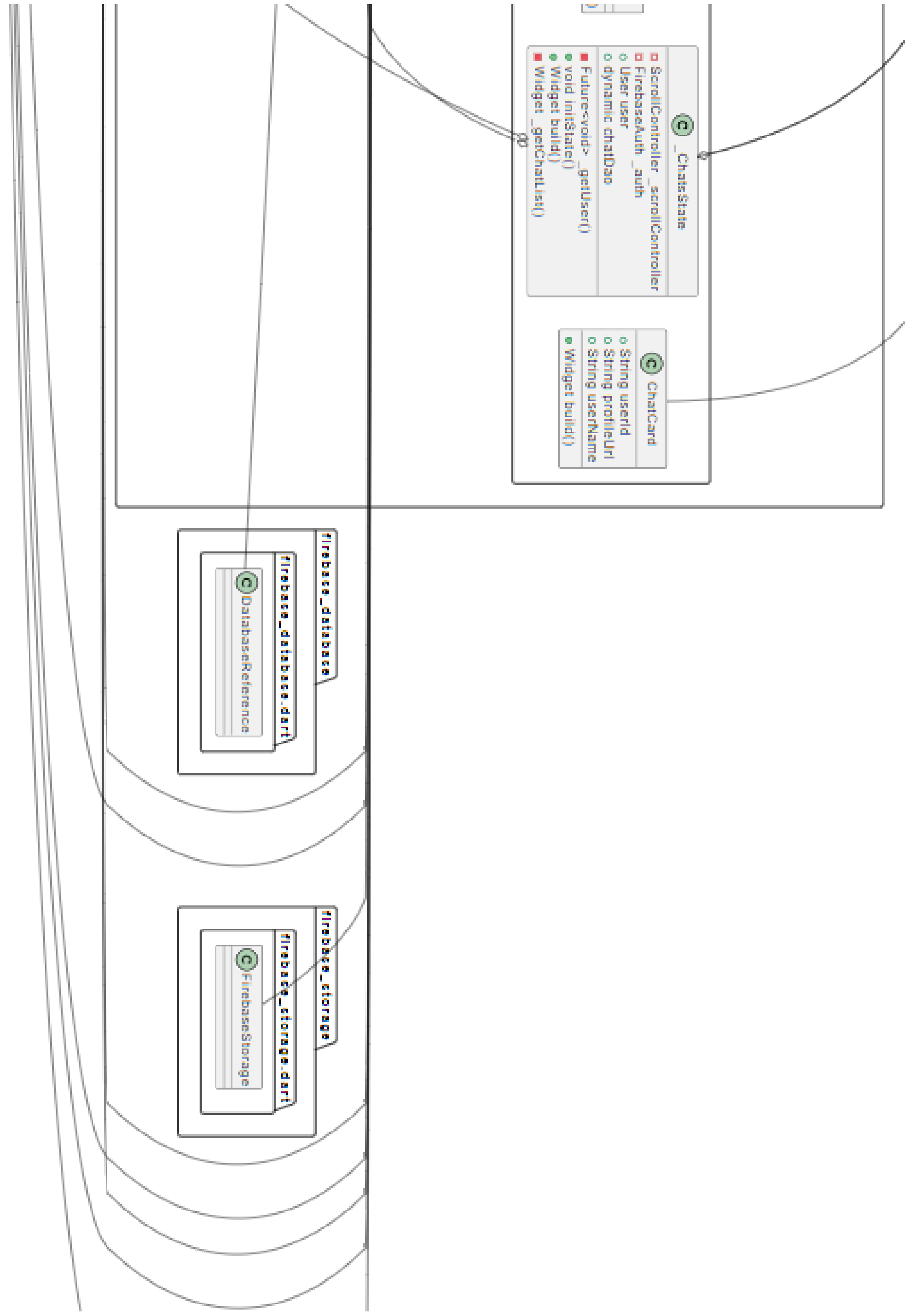
Part two:


register.dart
_RegisterState
FirebaseAuth _auth
GlobalKey<FormState> _formKey
TextEditingController _displayName
TextEditingController _emailController
TextEditingController _passwordController
TextEditingController _passwordConfirmController
int type
FocusNode f1
FocusNode f2
FocusNode f3
FocusNode f4
void dispose()
Widget build()
Widget _signUp()
dynamic showAlertDialog()
dynamic showLoaderDialog()
bool emailValidate()
void _registerAccount()
void _pushPage()
Register
State<Register> createState()
MessageList.dart
Message
String senderId
String message
String time
Map<dynamic, dynamic> toJson()
MessageList
List<Message> messages
List<dynamic> toJson()
MessageList merge()
messagedata.dart
MessageDao
DatabaseReference _messageRef
String user1
String user2
Future<void> createGateWay()
void saveMessage()
Query getMessageQuery()
message.dart
Message
String senderId
String message
String time
List<List<String>> messageDetails
Map<dynamic, dynamic> toJson()
FirebaseStartup.dart
_FireBaseAuthState
FirebaseAuth _auth
GlobalKey<FormState> _formKey
TextEditingController _emailController
GlobalKey<ScaffoldState> _scaffoldKey
TextEditingController _passwordController
FocusNode f1
FocusNode f2
FocusNode f3
Widget build()
Widget withEmailPassword()
void dispose()
dynamic showLoaderDialog()
bool emailValidate()
void _signInWithEmailAndPassword()
void _pushPage()
FireBaseAuth
State<FireBaseAuth> createState()
home_page.dart
_HomePageState
GlobalKey<ScaffoldState> _scaffoldKey
TextEditingController _TutorName
FirebaseAuth _auth
User? user
Future<void> _getUser()
void initState()
void dispose()
Widget build()
HomePage
State<HomePage> createState()
ListOfAppointments.dart
_AppointmentListState
FirebaseAuth _auth
User user
String _documentID
Future<void> _getUser()
String _dateFormatter()
Future<void> showAlertDialo
dynamic _checkDiff()
dynamic _compareDate()
void initState()
Widget build()
flutter
src
widgets
focus_manager.dart
FocusNode
framework.dart
State
GlobalKey<ScaffoldState>
StatefulWidget
GlobalKey<FormState>
StatelessWidget
editable_text.dart
TextEditingCo

Part three:



Part four:


_ChatsState
ScrollController _scrollController
FirebaseAuth _auth
User user
dynamic chatDao
Future<void> _getUser()
void initState()
Widget build()
Widget _getChatList()
ChatCard
String userId
String profileUrl
String userName
Widget build()
firebase_database
firebase_database.dart
DatabaseReference
firebase_storage
firebase_storage.dart
FirebaseStorage

Explanation of UML Diagram:

In the highest level of the app diagram, we have the classes that do not yet involve implementing the firebase. These classes exist to set up the framework where information is placed in. First is the chat function class, which simply sets up the chat function used between students and tutors if that is how they want to set up appointments. Next is the main class, which simply calls the app. The Update User class inside the model package, otherwise known as the newdeets class sets up the information screens where all users' details are displayed. The my Profile class does that for individual users in their own settings screens. The My Profile class includes variables such as name, id, and email. The three classes that are mainly used for the manipulation of the user details are the userSettings class, the userDetails class, and the updateUserdetails class. The former two classes inherit from the latter in order to process the data that users add to the application and then save it into firebase. The build widget, which is present in most classes, is simply what is used to create the UI of the app. The HomePage class simply sets up the home page users will reach after logging in. In the firebase package, we have all the classes that deal with user input and save it to firebase. These classes initialize a FirebaseAuth variable to connect them to the firebase. The history class saves the former conversations between students and tutors, as well as previously scheduled appointments that may have already passed. The Booking class deals with users in the moment of scheduling an appointment and saving those particular pieces of data in real-time. Later, these bits of info get dealt with in the FireBaseAuth class as well as the appointments class, as ultimately all the details from booking do get sent there. Because of this, these classes require significantly fewer methods and variables

Proper Credit:

Much of what I learned on how to create an updatable user profile as well as connecting it to firebase was from sources #1-3 in the appendix. Much of my UI knowledge came from sources #4-5

Data Structures:

- The Main Data structures used in these applications will be lists and maps within Firebase/Firestore. The firebase page I created will be connected to my application through the .json file of my application, which can be found in the Android Build Gradle.

TEST PLAN!

Successful Authentication + Error Handling:	The way I will test this is by first inputting an incorrect password and seeing if the app corrects me. Then, I will input a username that does not exist and see if the app corrects me, and if it does on both instances, then I know this step is working fine.
Account Creation:	In order to make sure Account Creation works, I will test first to make sure I get an error message if I input a password that doesn't match the one I originally inputted. Then, I will try putting in an email that already exists, and I should get an error message. Then, I will create accounts with both tutor and student profiles and see if it sends me to different homepages. If it does, the app is succeeding
Editing Account Details:	I will go to the settings screen and see if I am able to edit the name, bio and email for students. For tutors, I should be able to edit that as well as the category they are teaching in and the specific subject they specialize in.
Exploring for Tutors:	I will log into the one of the student accounts I have made and I will see whether the students have an explore page with text boxes that have the different categories on them. When they click on the boxes, I will check to see that they can access the tutors in those categories. Tutors must NOT have this option
Proper Tutor Profile:	I will check to see that once students click on a tutors icon from the explore page, that it includes the name of the tutor, the specific subject they specialize in, and their bio.
Chat Room:	The chat room must be fully functional. Students should be able to send messages to tutors and vice versa, and I will check this by sending messages from a student account to a tutor account, and then checking from the tutor account to see if it was received and vice versa.
Booking Screen:	In order to make sure the booking screen is functional, I will log into a user account and test to see whether I can input my name, details about the appointment, and use the datepicker to select a date. Once I do this, I will log into the respective tutor account to see whether I have received the appointment request
Appointment Screen:	After scheduling appointments through a tutor account, I will check to make sure the appointments went through in the appointment page of the Tutor accounts

Word Count: 492