Design Specification

For

Majorizer: The Student's Personal Assistant

Version 1.0

Prepared By: Jared Heidt, Jeremy Dewey, Josh Gillette, and Andrew H. Shaw

Team Rocket Inc.

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this software design document is to provide a description of Majorizer in both a class-perspective and data-perspective maner. There are several audiences that may find this document useful. The first of which being the Majorizer software development team themselves. This document outlines the design and vision of the project as a whole, which will be useful to reference as the project develops to understand of project internals and maintain constant design principles. Additionally, this document is also made as a resource for external entities, such as the client or a company that intends to interact and understand Majorizer's code. In conclusion, this document will outline the structure and design of the components that make up Majorizer.

1.2 Scope

Majorizer's scope almost exclusively resides within the college environment.To describe the product itself, Majorizer is a student's software solution that connects a student and advisor while also organizing various details of a student's academic life. For instance an advisor and student will have access to a student's majors/minors, past/present/future course information, required courses, grading history, and GPA through Majorizer. Majorizer's goal is simple, to make both a student's and advisor's academic life easier. All users should benefit from Majorizer, as the solution that Majorizer offers is replacing a series of paper forms, emails, and conversations shared between an advisor and student, obviously a flawed system. Majorizer's ability to centralize all of the data and actions between an advisor and student is its greatest strength.

1.3 Overview

As stated, this design document will give an overview of the design of all major elements of Majorizer, mainly being broken down to system architecture, database design, and interface design. First, it will provide a description of both the system itself and the system architecture. In this architecture design all of the classes and its elements will be stated. Secondly, it will give a detailed overview of the data design. This includes a description of the database, as well as a data dictionary that lists all significant data types in Majorizer. Lastly, the human interface design will describe an overview of the user interface design of Majorizer. Finally, a requirements matrix will be provided that lists requirements and associated items with it.

2. System Overview

The Majorizer application will be an Android phone application developed in Android Studio. There will be several software components required to fulfill the specified requirements. Such components include a database, a user login manager, user account data structures, and a material design user interface. The database for this phone application will be a NoSQL Google Firebase Database. The database is structured in

JSON format. The NoSQL database was chosen over a traditional SQL database because of its advantages with scalability and speed.

To take advantage of NoSQL, the database must be structured with modularity. All attributes concerning users, courses, majors, and minors are structured in in the database. The layout of the database will be explained in Section 4. The types of users in this application will include undergraduate students, graduate students, advisors, and admins. Each user class will have similar data attributes but will be allowed to perform only a limited amount of actions.

The Android phone running the Majorizer application and the Google Firebase hardware will be the two pieces of hardware considered in the software design. Networking functionality from database queries is abstracted by the Android phone.

3. System Architecture

3.1 Architectural Design

The main modules of this application are the Account class and its subclasses, the Course class and its subclass, the Notification class and its subclass, the Search class, and the Schedule class.

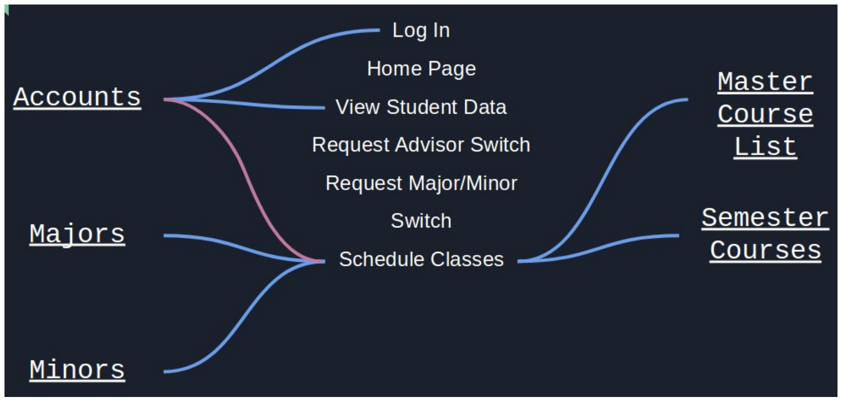
The Account class and its subclasses interact with all the other mentioned classes to present queried data about the user from the database. The Account class is also responsible for interacting with these other classes to update data. The Course class and its subclass are responsible for representing a class that a user can search or enroll in. The Search class will be used when the user uses the search bar on the interface. The Search class will examine the users entered prefix and return a list of results. The Schedule class will be an API for the Account class to view their schedule and request to make changes to it. The Notification class will have data members of notifications which will be "sent" to an Account object.

The Student class, Admin class, and Advisor class will all inherit from the Account class. The Undergraduate class and the Graduate class inherit from the Student class. The SemesterCourse class inherits from the Course class. The action that the Account subclasses can perform are shown below:

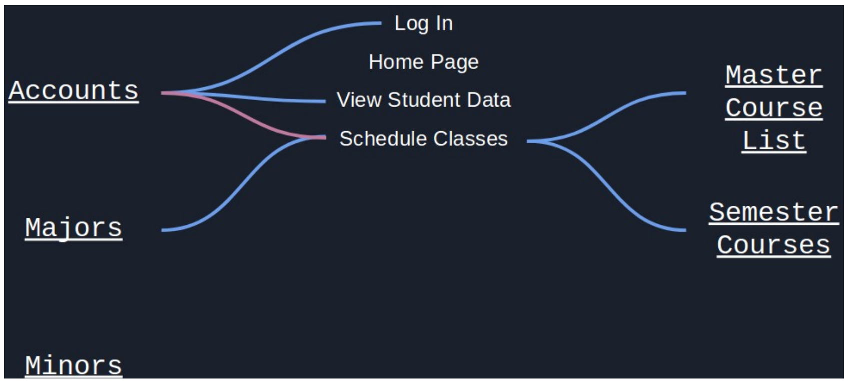
KEY FOR FIGURES BELOW:

```
**BLUE - pulling information from database

**RED - pushing information to database

**PURPLE - both pushing & pulling information
```

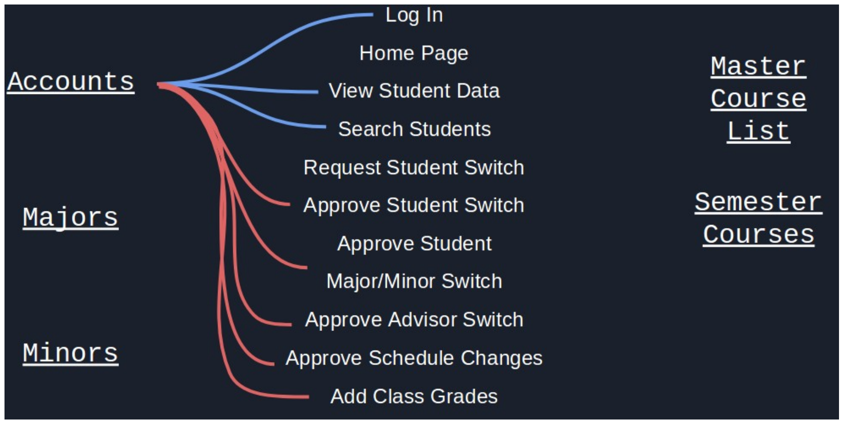
Undergraduate Student Actions Diagram



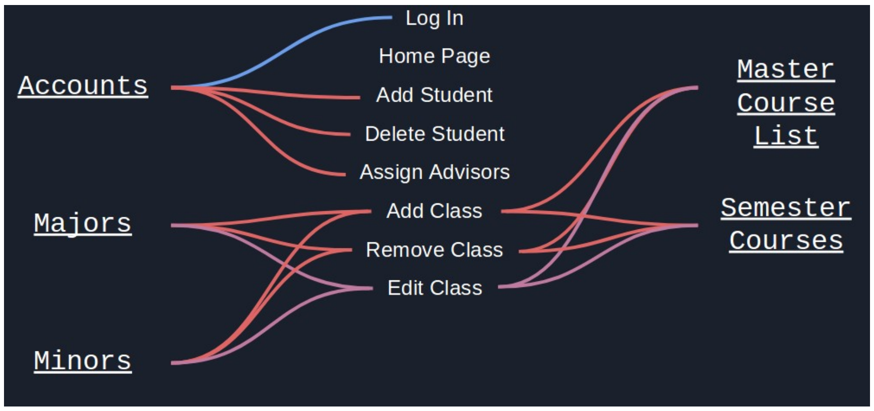
Graduate Student Actions Diagram



Advisor Actions Diagram



Admin Actions Diagram



3.2 Decomposition Description

3.2.1 "Account" Class

"Account" is the main parent class for the four types of accounts being used throughout the system. The properties of the class, "Account" are are also possessed by all of it's subclasses. All user types are consistently interacting with the program, and each user type is a subclass of “Account”.

“Account” Subclass(s):

- ❑ `Student - used by undergraduate and graduate students`
- ❑ `Advisor - used by academic advisors to the students`
- ❑ `Administrator - used by 1-2 system experts`

Whenever any user’s account is created, an “Account” object is created and it is assigned an identification number unique to that account, a username, a password, an account type, and a boolean value, “locked”, to tell whether or not a user’s account has been locked out.

“Account” Data:

```
ID - string (only contains digits)
Username - string
Password - string
accountType - AccountType (enum: ERROR,UNDERGRAD, GRAD,
ADVISOR, ADMIN)
Notifications - List<Notification>
```

The methods of “Account” are mostly used for data retrieval or generic functions required by all accounts. The two outliers are `receiveNotification()` and `sendNotification( Notification )`. They both send and receive notifications (stored in our database) respectively. They both take a `Notification` object as an argument. The information regarding the details of the notifications are stored in the `Notification` objects. Below they are listed with their respective return types.

“Account” Methods:

```
     setID(final string):void

     setUserName(final string):void

     setFirstName(final string):void

     setLastName(final string):void

     setAccountType( AccountType (enum: ERROR,UNDERGRAD, GRAD,
ADVISOR, ADMIN) ):void

     getID():string

     getUserName():string
     getFirstName():string

     getLastName():string

     getAccountType():AccountType (enum: ERROR,UNDERGRAD, GRAD,
ADVISOR, ADMIN)

     addNotification(final Notification):void

     getNotifications():List<Notification>

     removeNotification(final int):void
```

3.2.2 “Student” Class

“Student” is a subclass of “Account” (*sec 3.2.1*). It is used by undergraduate and graduate students. It inherits the properties of “Account”.

“Student” Parent(s):

- ❏ ***Account (3.2.1)***

“Student” Subclass(es):

- ❏ ***Undergraduate (3.2.3)*** - used by undergraduate students
- ❏ ***Graduate (3.2.4)*** - used by graduate students

The “Student” class has data and methods that are required for every student at a university. That includes items such as a list of every class that a student is currently enrolled in or a map of their advisor(s).

“Student” Data:

```
advisors - Map<string,string>
coursesPrevTakenList - List<ClassData>
coursesCurTakingList - List<Course>
```

“Student”s also use multiple methods that would be required of both types of students at a standard university.

“Student” Methods:

```
getAdvisorName(string username): string

getAdvisors(): Set<string>

numCoursesTaken(): int
```

```
numCurCoursesTaking(): int

getPrevCourseInformation(int courseIndex): ClassData

getCurCourseInformation(int courseIndex): Course

getCreditsTaken(): int

getCreditsTaking(): int

getGPA(): string

addAdvisor(final string username, final string fullName):
void

addCoursePrevTaken(final string courseName, final string
courseCode, final string courseGrade, final iInteger
numCredits): void

addCoursesCurTaking(final string courseName, final string
courseCode, final Integer numCredits): void
```

3.2.3 “Undergraduate” Class

“Undergraduate” is one of the subclasses of “Student” (*sec. 3.2.2*). It will only be used by undergraduate students. It shares the properties of any *Student* with “Graduate” (*sec 3.2.4*) and the properties of “Account” (*sec 3.2.1*) with all other users.

“Undergraduate” Parent(s):

- ***Student (3.2.2)***

The “Undergraduate” class also has data and methods specific to itself. Undergrad students are assigned the *Student* and *Account* data members as well as data such as *Major*, *curSchedule*, *Advisor, etc*. Below is all of the data members that the *Undergraduate* will hold.

“Undergraduate” Data:

```
Major1 - string
Major2 - string
Minor1 - string
Minor2 - string
```

Additionally, the *Undergraduates* have many methods specific to them. They will be able to view as well as request specific changes to their own current schedules, their advisors, and their majors and minors.They will only be able to view their course history, and their identity (first name and last name). Shown below are these methods.

"Undergraduate" Methods:

```
setMajor1(string major1): void
setMajor2(string major2): void
setMinor1(string minor1): void
setMinor2(string minor2): void
getMajor1(): string
getMajor2(): string
getMinor1(): string
getMinor2(): string
```

3.2.4 "Graduate" Class

Graduate is another subclass of *Student* (*sec 3.2.1*). The *Graduate* objects are very similar to the *Undergraduate* (*sec 3.2.3*) objects in the sense that they hold very similar data types. However, the actual data is coming from a different section of the database because graduate students have a completely different curriculum than undergraduate students.

"Graduate" Parent(s):

- ❑ ***Student (3.2.1)***

Graduate objects also lack some of the data of *Undergraduate* objects. Specifically, they do not have any minor(s) while in graduate school. However they do share the rest of their data with *Undergraduate*. As mentioned earlier in this section though, some data members are holding different values. *Majors* and *Advisors* are restricted to different and possibly fewer options. The table below shows these data members.

"Graduate" Data:

```
Major - string
```

Just like the method differences with *Undergraduate*, the methods of *Graduate* are slightly different as well. They lack the methods used for viewing, editing, or requesting to change any minor(s). All other methods behave the same as *Undergraduate*. **(*See section 3.2.2)***

"Graduate" Methods:

```
setMajor(string major): void
getMajor(): string
```

3.2.5 "Advisor" Class

The *Advisor* object is a subclass of *Account* (*sec 3.2.1)*. This account type will be used by whoever the university chooses to assign as advisors for the students. This could be counselors, professors, or even a professional advisor. The advisors are the main accounts that will be approving requests (whether it be a schedule change or a major change) for students or even other advisors.

"Advisor" Parent(s):

- ❏ ***Account (3.2.1)***

Advisors are primarily going to be viewing and editing **other** accounts so they do not hold much data themselves. They have a first name, a last name, whatever major they are an advisor for, and a list of the ID's for the students they advise.

"Advisor" Data:

```
studentsMap - Map<string, Student>
Department - string
```

The methods used by *Advisor* are only used to either return back stored information or to observe and assist the academics of both *Undergraduate* (*sec 3.2.2*) and *Graduate* (sec 3.3.3) accounts. From an advisor's master map of students they are advising, a list of students can be obtained. It is also possible to view the information of any students of the advisor. An advisor of a student is also able to switch that student's advisor to a different from themselves to another if requested.

"Advisor" Methods:

```
hasStudent(string studentUserName): boolean

addAdviseeToMap(string studentUsername, Student student): void

getStudents(): Map<string, Student>
```

```
getSearchableAccounts(final List<Student> studentsToSearch,
final BaseAdapter arrayAdapter, final Context context): void

getNumAdvisees(): int

getDepartment(): string

setDepartment(string department): void

dropStudentFromAdvisees(final Student student, final Context
context): void

addStudentsToAdvisees(final Student student, final Context
context): void
```

3.3.6 “Admin” Class

The *Admin* class is the third and final subclass of *Account* (*sec 3.2.1*). The *Admin* class is very different from the other *Account* subclasses as it is not linked to a specific person. It is the only account type without a first name or a last name. It is only used by one or two users who should know how to operate and maintain *Majorizer*.

“Admin” Parent(s):

- ***Account (3.2.1)***

Admin does not have any personal information that it needs to store so it does not have any practical data members. It still does have methods as well as a “NULL” data piece.

“Admin” Data:

```
NULL - static final string
```

They access and edit all of the information that is used by all other users in *Majorizer*. They can add, edit, or remove any courses, semester course, or Account straight into our databases. The methods described below are what it uses.

“Admin” Methods:

```
unlockAccount(final string accountUsername, final Context
context): void
```

```
getLockedAccounts(final List<string> usernameList, final
ArrayAdapter arrayAdapter): List<string>

addCourseToDatabase(final List<Course> coursesToSearch, final
int position, final CourseSearchAdapter searchAdapter, final
string coursesURL, final Context context): void

removeCourseFromDatabase(final List<Course> coursesToSearch,
final int position, final CourseSearchAdapter searchAdapter,
final string coursesURL, final Context context): void

populateCoursesForSearch(final Context context, final
List<course> courseToSearch, final CourseSearchAdapter
courseSearchAdapter, final string URLinDatabase): void

createNewUndergradStudentAccount(final Context context, final
String username, final string password, final string firstName,
final string lastName, final string major1, final string
major2, final string minor1, final string minor2): void

createNewGradStudentAccount(final Context context, final string
major, final string username, final string password, final
string firstName, final string lastName): void

createNewAdvisorAccount(final Context context, final String
department, final string username, final string password, final
string firstName, final string lastName): void
```

3.2.7 “Course” Class

The *Course* class is a system class that gets interacted with by the users. The class is also a parent class to *ClassData* (*sec 3.2.8*). This is because there is other information for students taking the course that is better kept seperated.

“Course” Subclasses(s):

- ❑ `ClassData (3.2.8)`

Course could be described as a collection of data that represents the classes offered at a university. An example would be Calculus II.

“Course” Data:

```
courseName - string
courseCode - string
Credits - Int
preReq - Set<Course>
```

“Course” Methods:

```
getCourseName(): string
getCourseCode(): string
getCredits(): int
getPreRequisites(): Set<Course>
```

3.2.8 “ClassData” Class

When a *SemesterCourse* object is created, it still has the properties of it’s parent class, *Course* (*sec 3.2.6*), but it also contains the grade that a student received in it. This object is only used when a student completes a course.

“SemesterCourse” Parent(s):

- ❑ ***Course (3.2.6)***

As stated before, the only data this class contains is the grade received by a student.

“SemesterCourse” Data:

```
Grade - string
```

“SemesterCourse” Methods:

```
getGrade(): string
```

3.2.8 “Notification” Class

Notification objects are created whenever a user triggers an event that will affect other user(s) in any way. An example could be, if a user wants to change their schedule, or if a course has been edited in some way or another.

“Notification” Data:

```
header - string
Message - string
ID - string
```

Notification has standard *get* methods along with *toString* that transfers the message into a readable string.

"Notification" Methods:

```
getHeader(): string
getID(): string
getMessage(): string
toString(): string
```

3.2.9 "LoginManager" Class

LoginManager is the object utilized whenever a user tries to log into the *Majorizer* system. It is responsible for issues such as account security and account usage. For example, the same account should not be able to log in twice at the same time.

"LoginManager" Data:

```
BAD_CREDENTIALS - string
COULD_NOT_GET_ACCOUNT - string
EMPTY_FIELD - string
INVALID_CHARACTER - string
BAD_QUERY - string
MAX_LOGIN_ATTEMPTS - string
mDatabase - DatabaseReference
```

"LoginManager" Methods:

```
isUserLockedOut(final DataSnapshot, final View): boolean

incrementLoginAttempts(final DataSnapshot, final string
enteredClarksonID, final View): void

resetLoginAttempts(final DataSnapshot, final string
enteredClarksonID, final Context): static void

populateAccount(final Resources, final DataSnapshot, final
Account): void
```

```
getStudentData(final Student, final Context): void

Login(final EditText clarksonUsernameField, final EditText
passwordField): void
```

3.2.10 “RequiredCourseListManager” Class

RequiredCourseListManager is used primarily by the *student* (*sec 3.2.2*). This handles the task of presenting to the user what curriculum is still required for he/she to graduate with their intended major(s).

“RequiredCourseListManager” Data:

```
PRE_REQUISITES - string

classesNeededList - final List<Course>
courseCount - int
creditsCount - int
activity - Activity
Fm - FragmentManager
```

“RequiredCourseListManager” Methods:

```
RequiredCourseListManager(final Context, final RecyclerView,
final Student, final TextView coursesRemainingView, final
TextView creditsRemainingView, final FragmentManager)

populateClassesNeeded(final Lock mutexLock, final string level,
final string curriculum, final List<string> classesTakenList,
fianl RecyclerView classesTakenRecyclerView, final TextView
coursesRemainingView, final TextView creditsRemainingView):void

getCoursesTaken(final Student student): List<string>

getStudentMajors(final Student student): Set<string>

getStudentMinors(final Student student): Set<string>

getClassNeededListItem(final int Index): Course

addClassNeededListItem(final Course course): void
```

3.2.11 “NotificationManager” Class

NotificationManager is utilized by all accounts present in the *Majorizer* system. It is in charge of receiving *Notification*'s (*sec 3.2.8*) and presenting them to the user. Different notifications carry different messages important to the user(s). For more information, see *Notication*.

"NotificationManager" Data:

```
lOCKED_OUT_1 - string
LOCKED_OUT_2 - string
ADMIN_REFERENCE - string
LOCKED_OUT_HEADER - string
MESSAGE - string
HEADER - string
Account - Account
```

"NotificationManager" Methods:

```
removeNotification(string ID): void

getNotifications(final DataSnapshot notificationSnapshot,
Resources resources, final Account account): void

notifyAdminLockedUser(final string studentName): void
```

3.2.11 "Utility" Class

Utility is used as a standard utility class throughout Majorizer.

"Utility" Class

```
databaseAdminKey - string
databaseAdvisorKey - string
databaseUndergradKey - string
databaseGradKey - string
requiredUsernameLength - int
```

"Utility Methods"

```
hideKeyboard(Context, View): void

getAccountType(final string accountType): static
Account.AccountType

isValidUserName(final string username): static boolean
```

```
isLowerAlpha(string name): static boolean

isValidName(final string name): static boolean

getActivity(final View view): static Activity

isValidPassword(final string password): static boolean

isNumber(final string str): static boolean

isValidUndergradCourseNumber(final string courseNumber): static
boolean

isValidGradCourseNumber(final string courseNumber): static
boolean

isValidNumberCredits(final string numberOfCredits): static
boolean
```

3.3 Design Rationale

The Majorizer application will follow object oriented principles as closely as possible. To avoid redundant code, several software objects will inherit from others. In addition, every class will have methods and data members which are carefully chosen to be either private, public, or protected.

There are various pros and cons of using inheritance in a large software application. The benefits include code reusability, easier code readability, and easier bug finding. Disadvantages of inheritance include dependency between base classes and superclasses and increased runtimes when traveling through levels of inheritance.

4. Data Design

4.1 Data Overview

Data storage for *Majorizer* can be split into three distinct levels of storage; database, pre-login, and post-login. Each of these play specific and distinct roles for the app by storing different data in different levels of accessibility.

4.2 Database

All data that the app will be using is stored in a Google FireBase database. The database is a NoSQL database that uses JSON. The JSON format uses a single string to store all data. FireBase abstracts the information so that the app can query specific data rather than pulling in the entire string when the app opens. This allows for the other two levels of data storage.

The database contains four objects; Accounts, Majors, Master Course List, Minors, Notifications, and Semester Courses. The Accounts object contains objects for each account, these objects contain the following data points;

- Advisor1: type string
- Advisor2: type string
- FirstName: type string
- LastName: type string
- LoginAttempts: type int (only for non-admin account types)
- Major1: type string
- Major2: type string
- Minor1: type string
- Minor2: type string
- Password: type string
- Type: type string
- Username: type string

The Majors object contains objects for each major, which each contain objects for each course in the major. These objects contain the following data points;

- CourseID: type string
- Required: type bool

The Master Course List object contains objects for each course, these objects contain the following data points;

- Credits: type int
- Name: type string
- Prerequisites: type [string]

The Minors object contains objects for each minor, which each contain objects for each course in the minor. These objects contain the following data points;

- CourseID: type string
- Required: type bool

The Notifications object contains objects for each notification, these objects contain the following data points;

- Notification: type string
- Recipient: type string
- Request: type string
- Request Type: type bool
- Sender: type string

The Semester Courses object contains objects for each course available in the current and upcoming semester, these objects contain the following data points;

- End Time: type string
- MWF: type bool
- Professor: type string
- Room Number: type string
- Section: type string
- Semester: type string
- Start Time: type string

The to see a diagram of the database structure and examples of the JSON format can be seen in Appendices 1 and 2 respectively.

4.3 Pre-Login

When the app is opened it does an initial query for login information. This data will be used to construct a login class object, this object will store the login information as an array of tuples. Each tuple will hold a username and a password both of type string. This will be used only for the login, once a user has logged in the data will be dereferenced until the app returns to the login screen.

4.4 Post-Login

When the user has logged in the app will query their account data, all basic course and account data in the case that an admin logs in, or all course data relative to the user's majors and minors, or discipline. This data will be used to make a number of objects for data storage and usage.

The first is an object that is a sub-class of the *Account*, this stores all of the user's account data. The sub-class that is used corresponds to the type of user account logging in. These classes hold all data about the user, this is different for each account type but all sub-classes store the data outlined by *Account*.

Next, if a student logs in then a *Schedule* object is made. The data for this object is a combination of data brought in on login, specifically the student's data from *Accounts* and course data brought in from *Semester Courses*.

Next, the app creates *Course* and *Semester Course* objects. *Course* objects pull data from the *Master Course List* data pulled in on login. Similarly, *Semester Course* objects pull data from the *Semester Courses* data pulled in on login. If an admin logged in then objects of both classes are made, for advisors and students only *Semester Course* objects are made.

Finally, the app pulls in data when advisors search and select a student. This triggers a query to get that specific student's account data. This creates an *Undergraduate* or *Graduate* object using the student's data.

4.5 Setting Database Data

Data is saved in two specific instances. When an admin adds, removes, or edits a course or account the app immediately adds, removes, or edits the data in Firebase. Similarly, when all parties approve a request the relevant data is changed in the Firebase.

5. Component Design

The Account will be extended by the Student class, Advisor class, and Admin class. Psuedocode for Account class:

"Account" Methods:

```
logIn(){
     remove_logged_in_state(this.ID);
     exit_main_activity();
}
logOut(){
     add_logged_in_state(this.ID);
     enter_main_activity();// Enters main activity for user
}
receiveNotification(){
     Notification list[] = pull_notification_data(this.id);
     return list;
}
sendNotification(Notification notification){
     for(Recipent recipent : notification.getRecipents()){
          insert_notification_database(id, notification)
     }
}
```

The subclasses of the Account class may have their own implementations for some of the Account methods. All classes that inherit from the Account class will follow the same method structure to modify data members. For methods that set data, the application will connect to the database, and update the existing members of an account. For methods that get data, the application will connect to the database, retrieve the requested data from the database, then present that data to the user.

6. Human Interface Design

6.1 Overview of User Interface

The user interface of the Android app will follow material design principles to express an intuitive user experience. When the user first enters the application, they will be required to sign in using the login page. The login page will have a Student ID text area, a password text area, and a login button. The login page will also have a picture of a graduation cap to give an academic theme to the app. The user will be expected to enter the requested information and then press the submit button.

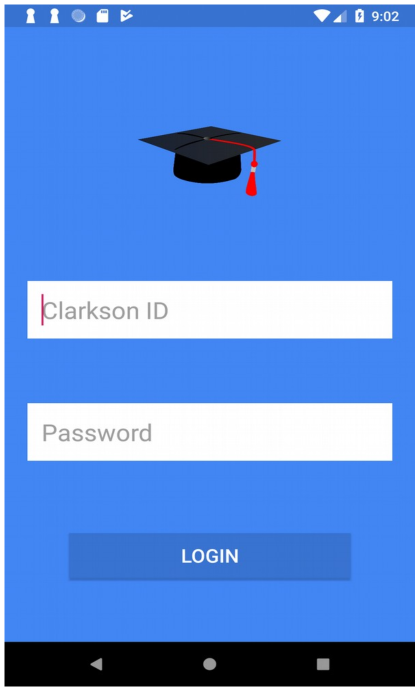
After the user enters their Student ID and correct password, they will be directed to their respected home screen. The home screen will have three tabs. Such tabs are “Home”, “Account”, and “Notifications.” The home screen will list options that the user can choose to do. These options will vary depending on the type of user logged in. For example, an admin will have the option to unlock a user’s account but a student will not have this option.

The options listed on the home page for an admin will be “Manage User Accounts” and “Modify Curriculums.” The options listed for students will be “View Academic History” and “Scheduler”. The options listed for advisors will be “Search Students”, “Manage My Students”, and “Switch Advisee.”

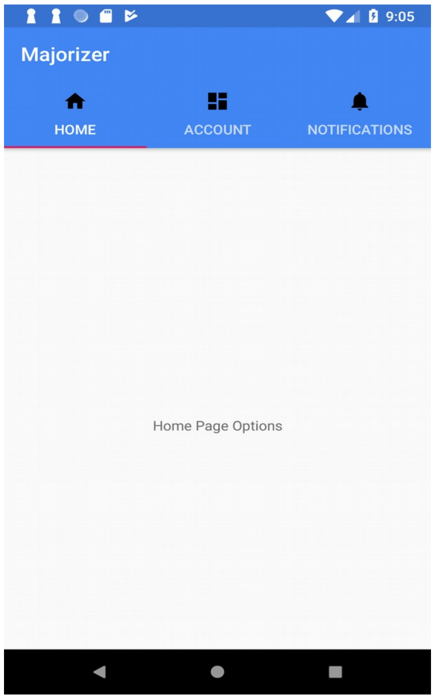
The notifications tab will contain a list of widgets which contain the notification’s information. If this list is empty, then this tab will show to the user, “No New Notifications.”

6.2 Screen Images

Login Page for all User Groups



Logged-in Page for all Users



6.3 Screen Objects and Actions

Majorizer's interface will entail a number of types of objects and actions associated with those objects. Once logged in, the user will see a toolbar which contains the home, account, and notifications objects. Selecting each object will toggle the given page associated with the toolbar object. Once a toolbar object is selected, the associated Android fragment will be shown to the user. An Android fragment is a modular section of an activity, which can be thought of as a sub-activity. Below outlines the three fragments that are present in Majorizer's layout.

HOME: This fragment will display a tile-based layout that gives the user a number of actions (one for each tile) to execute. These actions depend on the user group and are outlined below in the below table.

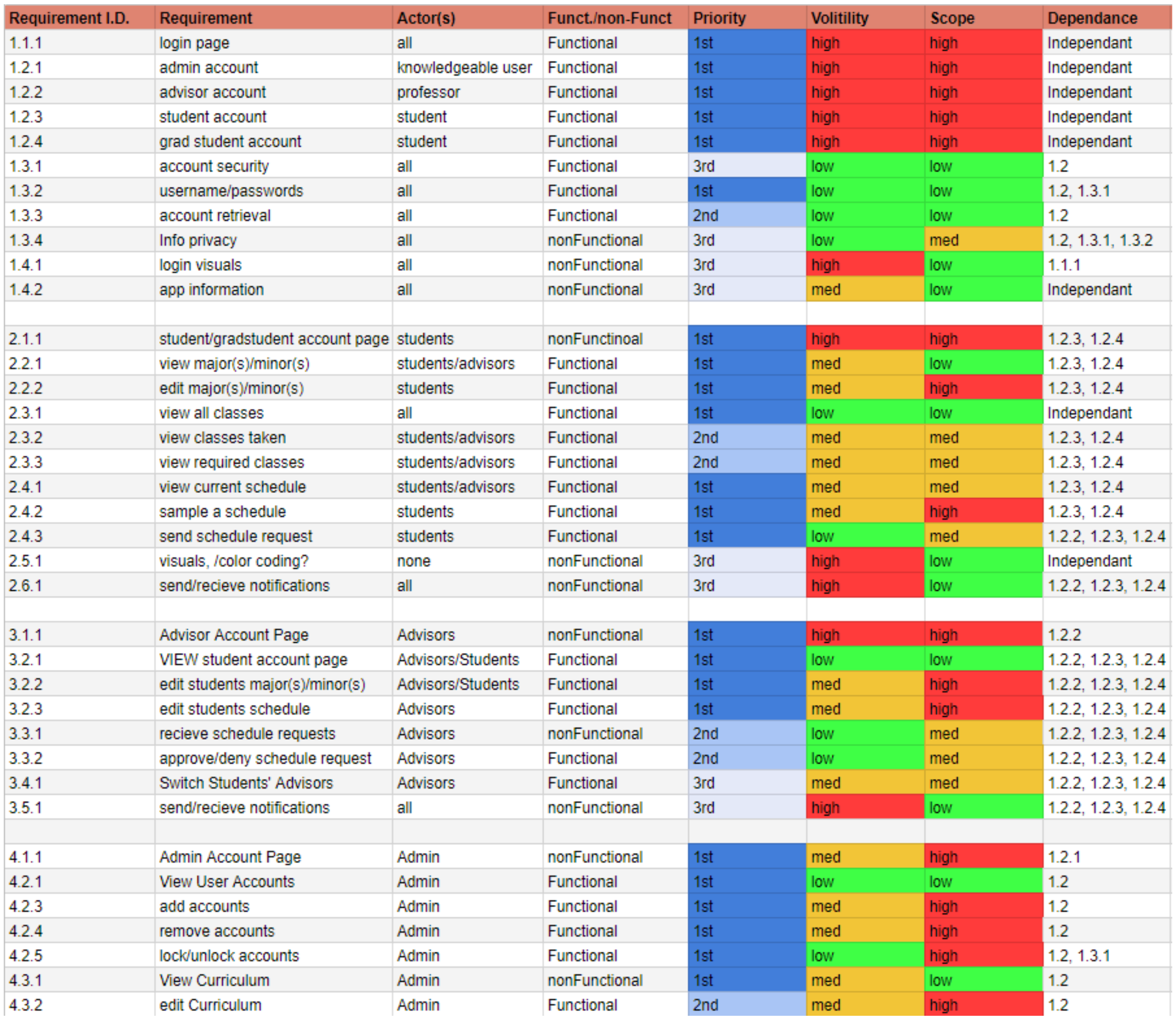
User group	Tile 1	Tile 2	Tile 3
Admin	User Accounts	Curriculum	--
Advisor	Search Students	My Students	Switch Advisee
Undergrad/Grad	Academic History	Scheduler	--

The tiles, when clicked, will launch an activity containing the data associated with the name of the tile. When required, the launched activity could launch any number of views or containers, such as additional activities, dialogs, or popups.

ACCOUNT: This fragment will give the user access to their Majorizer account. Data that will be displayed include username, password, ID, account type, and the account lock state.

NOTIFICATIONS: Under this fragment the user will be able to view their notifications. These notifications could include account changes, interactions between an advisor and student, or any instance where new information is present and notification worthy for a user.

7. Requirements Matrix



Requirement I.D.	Requirement	Actor(s)	Funct./non-Funct	Priority	Volitility	Scope	Dependance
1.1.1	login page	all	Functional	1st	high	high	Independant
1.2.1	admin account	knowledgeable user	Functional	1st	high	high	Independant
1.2.2	advisor account	professor	Functional	1st	high	high	Independant
1.2.3	student account	student	Functional	1st	high	high	Independant
1.2.4	grad student account	student	Functional	1st	high	high	Independant
1.3.1	account security	all	Functional	3rd	low	low	1.2
1.3.2	username/passwords	all	Functional	1st	low	low	1.2, 1.3.1
1.3.3	account retrieval	all	Functional	2nd	low	low	1.2
1.3.4	Info privacy	all	nonFunctional	3rd	low	med	1.2, 1.3.1, 1.3.2
1.4.1	login visuals	all	nonFunctional	3rd	high	low	1.1.1
1.4.2	app information	all	nonFunctional	3rd	med	low	Independant
2.1.1	student/gradstudent account page	students	nonFunctinoal	1st	high	high	1.2.3, 1.2.4
2.2.1	view major(s)/minor(s)	students/advisors	Functional	1st	med	low	1.2.3, 1.2.4
2.2.2	edit major(s)/minor(s)	students	Functional	1st	med	high	1.2.3, 1.2.4
2.3.1	view all classes	all	Functional	1st	low	low	Independant
2.3.2	view classes taken	students/advisors	Functional	2nd	med	med	1.2.3, 1.2.4
2.3.3	view required classes	students/advisors	Functional	2nd	med	med	1.2.3, 1.2.4
2.4.1	view current schedule	students/advisors	Functional	1st	med	med	1.2.3, 1.2.4
2.4.2	sample a schedule	students	Functional	1st	med	high	1.2.3, 1.2.4
2.4.3	send schedule request	students	Functional	1st	low	med	1.2.2, 1.2.3, 1.2.4
2.5.1	visuals, /color coding?	none	nonFunctional	3rd	high	low	Independant
2.6.1	send/recieve notifications	all	nonFunctional	3rd	high	low	1.2.2, 1.2.3, 1.2.4
3.1.1	Advisor Account Page	Advisors	nonFunctional	1st	high	high	1.2.2
3.2.1	VIEW student account page	Advisors/Students	Functional	1st	low	low	1.2.2, 1.2.3, 1.2.4
3.2.2	edit students major(s)/minor(s)	Advisors/Students	Functional	1st	med	high	1.2.2, 1.2.3, 1.2.4
3.2.3	edit students schedule	Advisors	Functional	1st	med	high	1.2.2, 1.2.3, 1.2.4
3.3.1	recieve schedule requests	Advisors	nonFunctional	2nd	low	med	1.2.2, 1.2.3, 1.2.4
3.3.2	approve/deny schedule request	Advisors	Functional	2nd	low	med	1.2.2, 1.2.3, 1.2.4
3.4.1	Switch Students' Advisors	Advisors	Functional	3rd	med	med	1.2.2, 1.2.3, 1.2.4
3.5.1	send/recieve notifications	all	nonFunctional	3rd	high	low	1.2.2, 1.2.3, 1.2.4
4.1.1	Admin Account Page	Admin	nonFunctional	1st	med	high	1.2.1
4.2.1	View User Accounts	Admin	Functional	1st	low	low	1.2
4.2.3	add accounts	Admin	Functional	1st	med	high	1.2
4.2.4	remove accounts	Admin	Functional	1st	med	high	1.2
4.2.5	lock/unlock accounts	Admin	Functional	1st	low	high	1.2, 1.3.1
4.3.1	View Curriculum	Admin	nonFunctional	1st	med	low	1.2
4.3.2	edit Curriculum	Admin	Functional	2nd	med	high	1.2

8. Testing

Task	Specific Test	Success Requirement	Tested
Database	‘Get’ queries	On call ‘get’ queries should return requested data from Firebase	YES
	‘Set’ queries	On call ‘set’ queries should set data to Firebase	YES
Login	Successful login	Correct username and password allows login	YES
	Failed login	Incorrect username or password does not allow log in and notifies user that attempt failed	YES
	Lock out	3 failed login attempts locks the user account and notifies the user of the lock, this should only unlock when an admin unlocks it	YES
	Account specific login	Verify that all of the above login tests pass for each account type	YES
Logout	From every account type attempt logout	Should log out of current account, redirect to login, and stay logged out on device	YES
Admin Account	Add account	Should add account data to the Firebase	YES
	Remove account	Should remove account data from the Firebase	YES

	Add course	Should add course data to the Firebase	YES
	Remove course	Should remove course data from the Firebase	YES
	Edit course	Should change data for chosen course	YES
	Assign Advisor	Should add an advisor ID to the students account data	YES
	Unlock user account	Should show list of locked accounts, unlocks selected accounts	YES
Search	Account Search	Should display a list of all of the accounts of the type being searched by the user	YES
		Should update the list as the user’s search criteria changes	YES
	Class Search	Should display a list of all classes available to each student type.	YES
		Undergraduate and Graduate students search from different class lists	YES
		Should update the list of classes as the user’s search criteria changes	YES
Notifications	Send Notifications	If an action permits it, a notification is sent and received by the recipient(s)	YES

	Receive Notification	If an account is the recipient of a sent message, the message will show in the user's mailbox	YES
Requests	Advisor switch request	Should send request type notifications to all advisors involved	YES
	Advisee switch request	Should send request type notifications to all advisors and advisees involved	YES
	Major switch request	Should send request type notifications to that student's advisor(s)	YES
	Minor switch request	Should send request type notifications to that student's advisor(s)	YES
Request Approval	Advisor switch approval	Should change student and advisor data, then notify every account involved	YES
	Advisee switch approval	Should change student and advisor data, then notify every account involved	YES
	Major switch approval	Should change student data, then notify every account involved	YES
	Minor switch approval	Should change student data, then notify every account involved	YES

Student Course History	Student View	The course history for students should be accurate	YES
		Should include all of a student’s courses and their respective grades	YES
	Advisor editing advisee history	Should accurately update the the History in the database	YES
Student Scheduler Tool	Find class	Should dynamically limit the list of sourse to those available, those the student has not taken, and those that the student has the prerequisites for	YES
	Add class	Should add selected courses to the visual schedule tool and removed from the course list	YES
	Remove class	Course selected in the visual schedule tool should be removed and re-added to the course list	YES
	Request Schedule	Should send a request type notification to the student’s advisor(s)	YES
	Schedule Approval	Advisor(s) approval to schedule request sends a notification to the student that their schedule was approved	YES

9. Data Dictionary

9.1 Database

Accounts: A Java object in the database containing objects for specific accounts.

Advisor1: Variable of type string.

- Data point for objects in *Accounts* (4.6.1).
- Holds the Account ID of an Undergraduate or Graduate account user’s advisor.

Advisor2: Variable of type string.

- Data point for objects in *Accounts* (4.6.1).
- Holds the Account ID of an Undergraduate account user’s possible second advisor.

CourseID: Variable of type string.

- Data point for objects in *Majors* and *Minors* (4.6.1).
- Holds the ID of a course.

Credits: Variable of type int.

- Data point for objects in *Major Course List* (4.6.1).
- Holds the number of credits that a course is worth.

EndTime: Variable of type string.

- Data point for objects in *Semester Courses* (4.6.1).
- Holds the time that a semester specific course ends.

FirstName: Variable of type string.

- Data point for objects in *Accounts* (4.6.1).
- Holds the first name of an Advisor, Undergraduate, or Graduate account user.

LastName: Variable of type string.

- Data point for objects in *Accounts* (4.6.1).
- Holds the first name of an Advisor, Undergraduate, or Graduate account user.

LoginAttempts: Variable of type int (only for non-admin account types).

- Data point for objects in *Accounts* (4.6.1).
- Holds the number of failed logins of an Advisor, Undergraduate, or Graduate account user.

Majors: A Java object in the database containing objects for each major.

Major1: Variable of type string.

- Data point for objects in *Accounts* (4.6.1).
- Holds the major of an Undergraduate or Graduate account user.

Major2: Variable of type string.

- Data point for objects in *Accounts* (4.6.1).
- Holds the possible second major of an Undergraduate account user.

Master Course List: A Java object in the database containing objects for every course.

Minors: A Java object in the database containing objects for each minor data point

Minor1: Variable of type string.

- Data point for objects in *Accounts* (4.6.1).
- Holds the possible minor of an Undergraduate account user.

Minor2: Variable of type string.

- Data point for objects in *Accounts* (4.6.1).
- Holds the possible second minor of an Undergraduate account user.

MWF: Variable of type bool.

- Data point for objects in *Semester Courses* (4.6.1).
- Holds the whether or not a semester specific course is held on Monday, Wednesday, and Friday. In the case that this is false it is given that the course is held on Tuesday and Thursday.

Name: Variable of type string.

- Data point for objects in *Major Course List* (4.6.1).
- Holds the name of a course.

Notification: Variable of type string.

- Data point for objects in *Notifications* (4.6.1).
- Holds the text being sent in a notification.

Notifications: A Java object in the database containing objects for notifications.

Password: Variable of type string.

- Data point for objects in *Accounts* (4.6.1).
- Holds the password of any type of account user.

Prerequisites: Variable of type [string].

- Data point for objects in *Major Course List* (4.6.1).
- Holds the prerequisites of a course.

Professor: Variable of type string.

- Data point for objects in *Semester Courses* (4.6.1).
- Holds the account ID of the professor teaching a semester specific course .

Recipient: Variable of type [string].

- Data point for objects in *Notifications* (4.6.1).
- Holds a list of account IDs of the users who will receive the notification.

Request: Variable of type bool.

- Data point for objects in *Notifications* (4.6.1).
- Holds whether or not the notification is a request.

Request Type: Variable of type string.

- Data point for objects in *Notifications* (4.6.1).
- Holds the type of request given that the notification is a request.

Required: Variable of type bool.

- Data point for objects in *Majors* and *Minors* (4.6.1).
- Holds whether or not a course is required for the major or minor it is in.

Room Number: Variable of type string.

- Data point for objects in *Semester Courses* (4.6.1).
- Holds the room number that a semester specific course is held in.

Section: Variable of type string.

- Data point for objects in *Semester Courses* (4.6.1).
- Holds the section number of a semester specific course.

Semester: Variable of type string.

- Data point for objects in *Semester Courses* (4.6.1).
- Holds the semester that a semester specific course is being taught.

Semester Courses: A Java object in the database containing objects for semester specific courses.

Sender: Variable of type string.

- Data point for objects in *Notifications* (4.6.1).
- Holds the account ID of the account that triggered the notification.

Start Time: Variable of type string.

- Data point for objects in *Semester Courses* (4.6.1).
- Holds the time that a semester specific course begins.

Type: Variable of type string.

- Data point for objects in *Accounts* (4.6.1).
- Holds the account type of any type of account user.

Username: Variable of type string.

- Data point for objects in *Accounts* (4.6.1).
- Holds the username of any type of account user.

9.2 Classes

Account: Sub-class to *Search Object* (4.6.2), and also super-class to all account type classes.

- Outlines variables for basic account data such as username, password, ID, account type, and the account lock state.
- Data for this class is brought in from *Accounts* (4.6.1).
- Includes methods for logging in and out, accessing user ID and account type, and sending and receiving notifications.

Admin: Sub-class to *Account* (4.6.2).

- Does not have any account specific data, only the data outlined by *Account* (4.6.2).
- Data for this class is brought in from *Accounts* (4.6.1).
- Includes methods for accessing lists user accounts and courses, adding and removing accounts, and adding, editing, removing courses for both the master list and semester list, and all methods in *Account* (4.6.2).

Advisor: Sub-class to *Account* (4.6.2).

- Consists of advisor’s first and last name, advisees, major, and data outlined in *Account* (4.6.2).
- Data for this class is brought in from *Accounts* (4.6.1).
- Includes methods for accessing account data, requesting switching a student’s advisor, and all methods in *Account* (4.6.2).

Course: Super-class to *Semester Course* (4.6.2).

- Outlines variables basic course info such as name, ID, credits, and prerequisites.
- Data for this class is brought in from *Course Master List* (4.6.1).
- Includes a method for editing its data. This is only accessible by *Admin* (4.6.2).

Graduate: Sub-class to *Account* (4.6.2).

- Consists of student’s first and last name, major and advisor info, course history, and data outlined in *Account* (4.6.2).
- Data for this class is brought in from *Accounts* (4.6.1).
- Includes methods for accessing account data, course history and schedule, editing schedule, and all methods in *Account* (4.6.2).

Notification: Super-class to *Request* (4.6.2).

- Consists of sender's ID, recipients' ID, notification text, and notification ID.
- Data for this class is brought in from *Notifications* (4.6.1) when being received or from user input and account info when being created and sent.
- Includes methods for accessing notification data and sending the notification.

Request: Sub-class to *Notification* (4.6.2).

- Consists of request type as well as the variables in *Notification* (4.6.2).
- Data for this class is brought in from *Notifications* (4.6.1) when being received or from user input and account info when being created and sent.
- Includes methods for approving and declining the request, executing requested changes, and all methods in *Notification* (4.6.2).

Schedule:

- Consists of a list of course ID's and the semester.
- Data is brought in from *Accounts* (4.6.1).
- Includes methods for adding and dropping courses, accessing current course, and verifying courses.

Search: Sub-class to *Search* (4.6.2).

- Consists of a master list of the items being searched as well as a dynamically updated list narrowed by search criteria.
- Data for this class is brought in from *Accounts*, *master Course List*, and *Semester Courses* (4.6.1) depending on what is being searched.
- Includes methods for updating the narrowed list and selecting an item in the list.

Search Object: Super-class to *Account* and *Search* (4.6.2).

- This class contains no data and has no methods.

Semester Course: Sub-class to *Course*(4.6.2).

- Consists of specific course information for courses being offered in the current semester and upcoming semester. This includes start and end times, days of the week, professor, section, room number, semester, and the variables outlined in *Course* (4.6.2).
- Data for this class is brought in from *Semester Courses* (4.6.1).
- Includes a method for editing its data. This is only accessible by *Admin* (4.6.2).

Undergraduate: Sub-class to *Account* (4.6.2).

- Consists of student's first and last name, major, minor, and advisor info, course history, and data outlined in *Account* (4.6.2).
- Data for this class is brought in from *Accounts* (4.6.1).
- Includes methods for accessing account data, course history and schedule, editing

schedule, and requesting major, minor, advisor switches, and all methods in *Account* (4.6.2).

9.3 Activities and Fragments

LoginActivity:

- Responsible for handling the processes associated with the user logging in.
- Connected to activity_login.xml
- Class will interact with the database, and provide communication when provided incorrect login credentials.
- With correct credentials, class will launch the MainActivity.

MainActivity:

- User does not interact with the MainActivity or associated activity_main directly.
- Initiates the SectionPageAdapter, ViewPager, and TabLayout that are used to form the tab layout discussed in section 6.

HomeFragment:

- Fragment associated with containing tiles to launch various pages. Differs based on user group.
- Associated layout is home_fragment.xml.

AccountFragment:

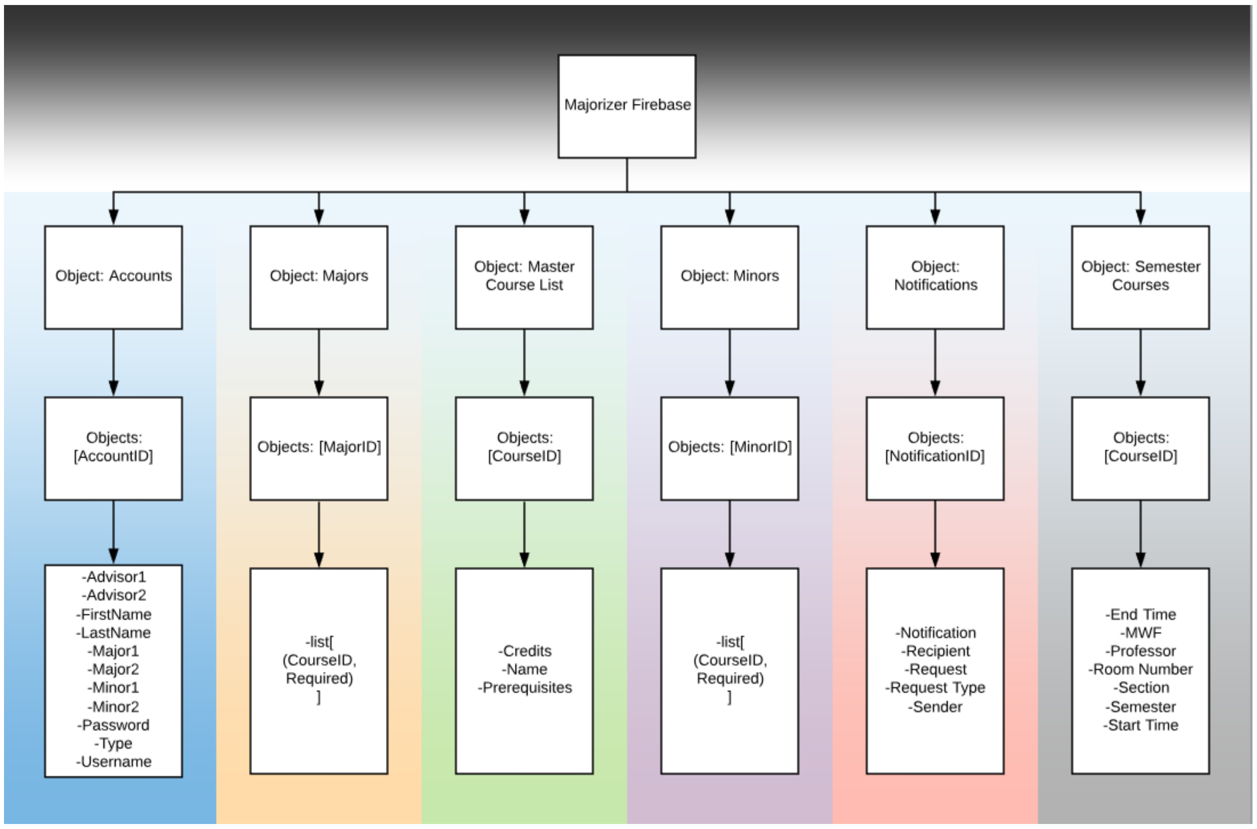
- Fragment associated with containing user account information.
- Associated layout is account_fragment.xml.

NotificationsFragment:

- Fragment associated with containing notification information.
- Associated layout is notification_fragment.xml.

10. Appendix

Appendix 1:



Majorizer Firebase
Object: Accounts
Object: Majors
Object: Master Course List
Object: Minors
Object: Notifications
Object: Semester Courses
Objects: [AccountID]
Objects: [MajorID]
Objects: [CourseID]
Objects: [MinorID]
Objects: [NotificationID]
Objects: [CourseID]
-Advisor1 -Advisor2 -FirstName -LastName -Major1 -Major2 -Minor1 -Minor2 -Password -Type -Username
-list[(CourseID, Required)]
-Credits -Name -Prerequisites
-list[(CourseID, Required)]
-Notification -Recipient -Request -Request Type -Sender
-End Time -MWF -Professor -Room Number -Section -Semester -Start Time

Appendix 2

```
{
 "Accounts" : {
   "0000000" : {
     "Advisor1" : "NULL",
     "Advisor2" : "NULL",
     "FirstName" : "ADMIN",
     "LastName" : "NULL",
     "Major1" : "NULL",
     "Major2" : "NULL",
     "Minor1" : "NULL",
     "Minor2" : "NULL",
     "Password" : "password",
     "Type" : "admin",
     "Username" : "admin"
   }
 },
"Majors" : {
  "CS" : [ {
    "CourseID" : "CS141",
    "Required" : true
  }, {
    "CourseID" : "CS142",
    "Required" : true
  } ]
},
"Master Course List" : {
  "CS141" : {
    "Credits" : 3,
    "Name" : "Intro to Computer Science I",
    "Prerequisites" : [ "" ]
  },
  "CS142" : {
    "Credits" : 3,
    "Name" : "Intro to Computer Science II",
    "Prerequisites" : [ "CS141" ]
  }
},
"Minors" : {
  "MA" : [ {
    "CourseID" : "MA131",
    "Required" : true
  }, {
    "CourseID" : "MA132",
    "Required" : true
  } ]
},
"Notifications" : {
  "00000" : {
    "Notification" : "Minor Switch Request: Math -> Physics",
    "Recipient" : "Joe Smoe's Advisor",
    "Request" : true,
    "Request Type" : "Minor Switch",
    "Sender" : "Joe Smoe"
  }
},
"Semester Courses" : {
  "CS350" : {
    "End Time" : "2:15pm",
    "MWF" : false,
    "Professor" : "Sean Banerjee",
    "Room Number" : "CAMP194",
    "Section" : "1",
      "Semester" : "F18",
      "Start Time" : "1:00pm"
    }
  }
}
```