CS360 - Software Engineering
Team 1 - Martin Benchev, Joshua Smith, Adam Hughes, Connor Springer
Client - Dr. Amal Khalifa

# Attendance Tracking System 4.0: Final Documentation

## 1. Introduction

Application Purpose

The Attendance Tracking System 4.0 is the next major release of the system after the 3.0 version. This Android application helps the user (lecturer) track students' attendance and generate reports containing useful and relevant attendance data. The application will allow the user to create an account on their local device, create semesters, organize courses within each semester, import a prepared class list, generate quick response (QR) codes that identify students, send the created QR codes to students through their (PFW) email address, take attendance using the system's QR code scanner, generate reports that list student attendance, and finally send attendance reports to the user through the user's email.

Project Goals and Objectives

1. Individual QR Code Sending & Registration
2. Per-Student Calendar View Improvements
3. Report Previews
4. Pre-day Absence & Attendance Reports
5. UI Improvements

Project Constraints

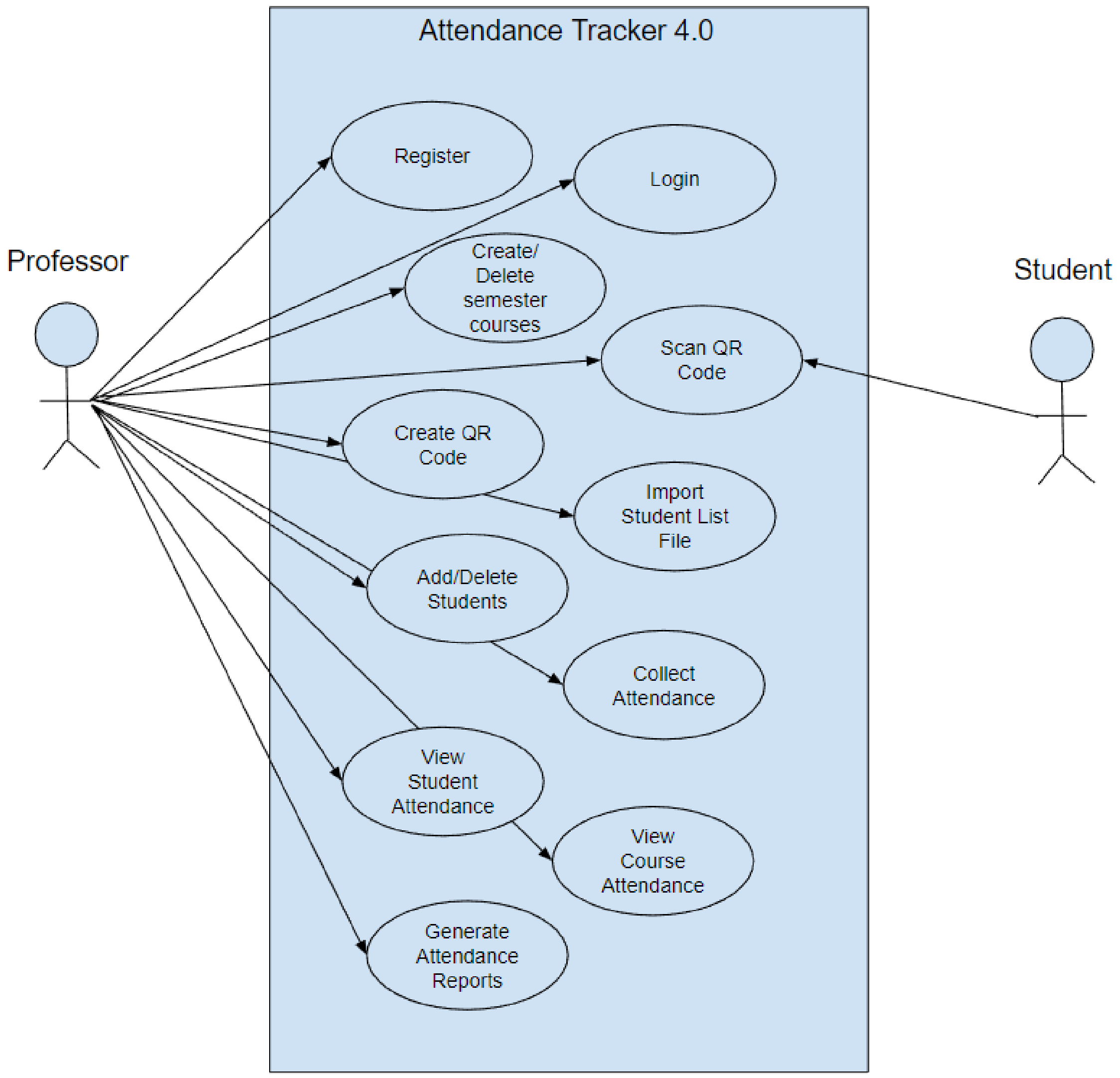
- Time Constraint - One college semester (4 months) to complete the 4.0 Version.
- Data Security - Internally stored data needs to remain secure as the system is worked on.
- Version 4.0 - Code has been built on by many past teams, so documentation and code has varying amounts of quality.
- Unfamiliar Software - Needed to learn new software and programs such as Android Studio, Gradle, and XML files.

## 2. Project Glossary

A. **Attendance Tracker:** The main application that records students' attendance using a barcode scanner to scan QR codes generated for each student.
B. **Database:** An organized collection of logically related data.
C. **Barcode Scanner:** An auxiliary application that scans QR codes corresponding to student names and writes the details associated with the scanned QR codes to a file.
D. **Quick Response (QR) Code:** A unique bitmap that stores the encoded information of a string into the black and white areas of an image. QR codes are used to store student attendance information in the database.
E. **Bitmap:** A mapping from some domain (such as an image) to bits.
F. **Comma-Separated Values (CSV) File:** A text file that separates data values by commas. This format is used to easily import and export data in spreadsheets and databases. In the attendance tracker, class list information can be imported from CSV files, and attendance data can be generated as CSV files.

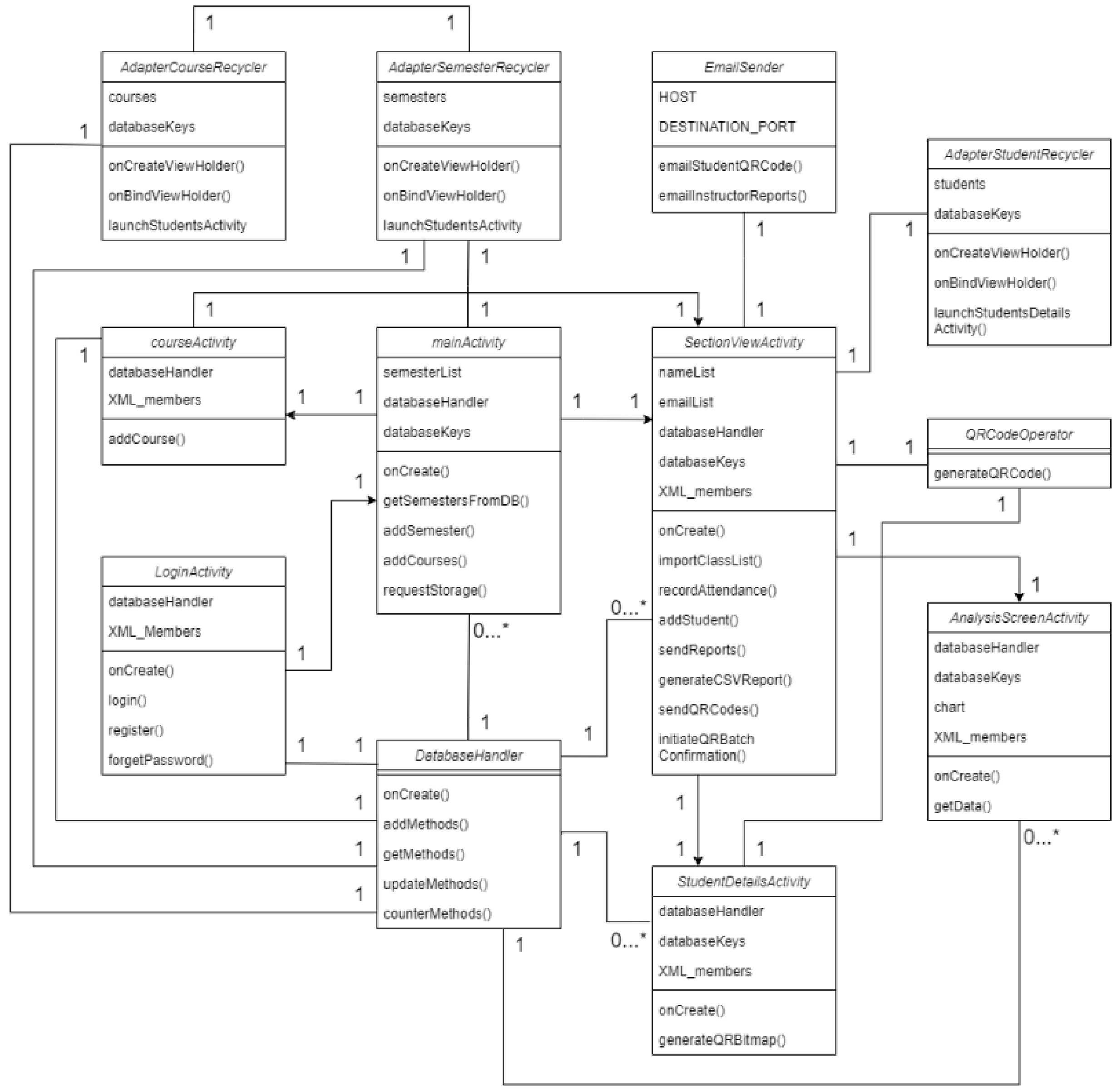
## 3. Models & Descriptions of Implementation Models

### A. Use Case Diagram



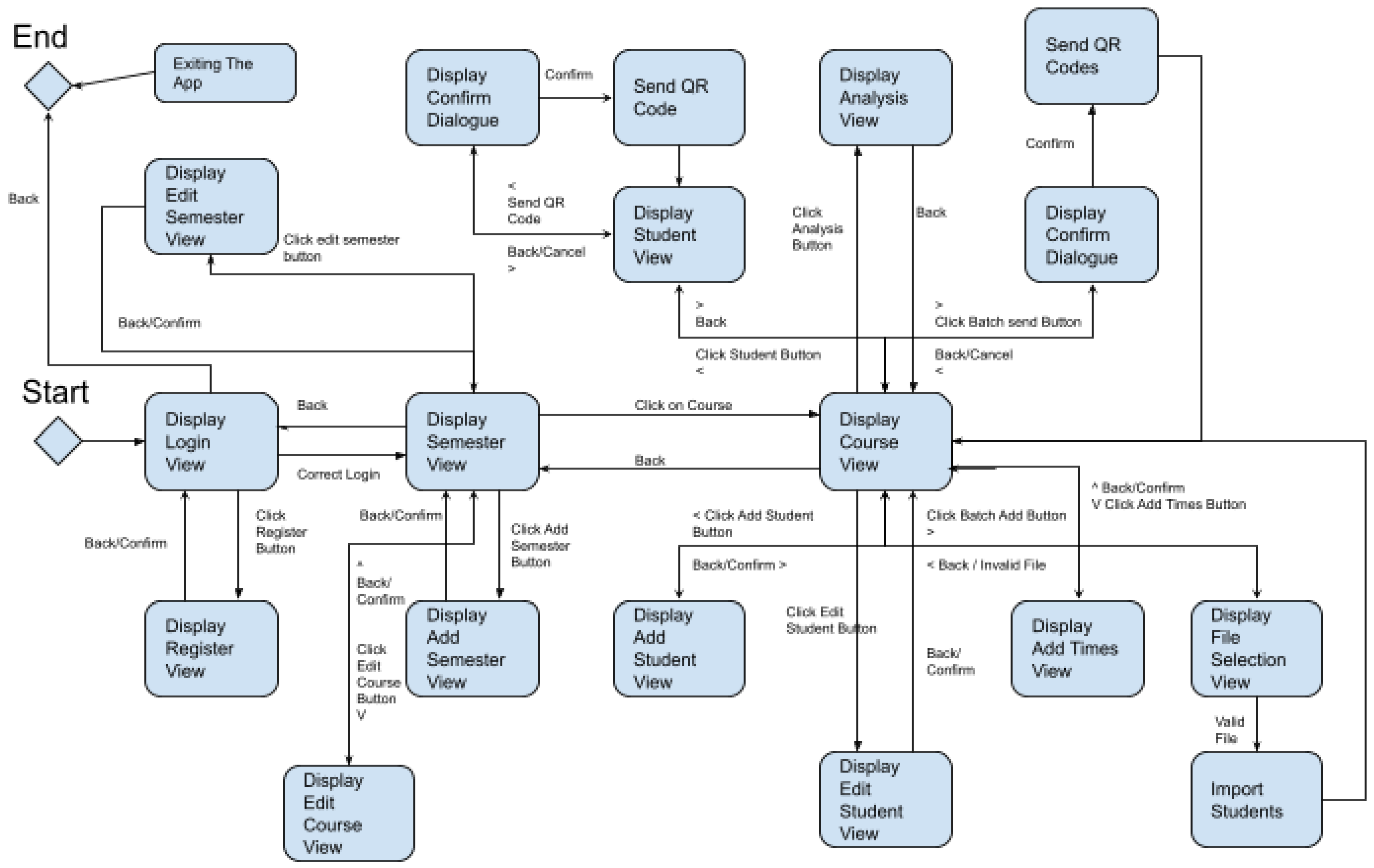
**Description:** This use case diagram shows what actors will do with the system. The actual User of the application (the Professor) has access to the vast majority of the functionality, while the Student only interacts with the system by scanning their QR codes in.

## B. Class Diagram



**Description:** The class diagram shows the actual classes in the Attendance Tracker's implementation, and their relationships in terms of ratios. Most of the classes developed for the application have a strict 1:1 ratio, but some notable exceptions include the relation between the DatabaseHandler and StudentDetailsActivity, where one StudentDetailsActivity can spawn anywhere from zero to an infinite amount of DatabaseHandlers.

## C. State Model



**Description:** This state model shows the flow of interactions between the user and the application, starting from the login page, and branching out to every possibility from there, until all possible interactions with the application are enumerated and described.

## 4. Important Modules

### A. LoginActivity

a. onCreate()
   i. Parameters: Bundle savedInstanceState
   ii. Functionality: Initializes the activity's attributes and objects such as XML views and widgets, data structures, and drives the activity's functions.
   iii. Postcondition: The user's display will contain all of this class's views and widgets, and it will have all of the functionality listed in the following methods.
   iv. Returns: void

b. loginClicked()
   i. Parameters: none
   ii. Functionality: Takes user input from email and password EditText objects and checks credentials with the database.
   iii. Postcondition: User will be brought to MainActivity screen if credentials are confirmed. Otherwise, a message will display that the entered credentials are invalid.
   iv. Returns: void

c. registerClicked()
   i. Parameters: none
   ii. Functionality: Displays a screen overlay that allows the user to input information. The user may press the back button to return to the login screen.
   iii. Postcondition: The registration screen will be displayed to the user.
   iv. Returns: void

d. registerSubmitClicked()
   i. Parameters: none
   ii. Functionality: Adds a new user to the database if all fields are entered and the email address has not been used before on the current device.
   iii. Postcondition: If the entered email address is unique to the database and all fields are entered, the system will display a message that the new user has been registered. If the email is not unique, a message will display that the user was not added successfully. If all fields are not entered, a message will display and the user will not be added.
   iv. Returns: void

## B. MainActivity

a. onCreate()
   i. Parameters: Bundle savedInstanceState
   ii. Functionality: Initializes the activity's attributes and objects such as XML views and widgets, data structures, and drives the activity's functions.
   iii. Postcondition: The user's display will contain all of this class's views and widgets, and it will have all of the functionality listed in the following methods.
   iv. Returns: void

b. getSemestersFromDB
   i. Parameters: String email
   ii. Functionality: Retrieves the semester list from the database using the user's email as the primary key.
   iii. Postcondition: An array list of String objects representing semester names will be returned to be used in the recycler view display.
   iv. Returns: ArrayList

c. addSemester()
   i. Parameters: String email
   ii. Functionality: Displays a popup that allows the user to insert a new semester and up to five courses in that semester.
   iii. Postcondition: A semester and any number of courses in that semester will be added to the database. The display will be updated with the semester and courses. Any errors will prompt an error message to the user.
   iv. Returns: void

d. addCoursesToDB()
   i. Parameters: ArrayList courses, String semesterName, String userEmail
   ii. Functionality: Works as a part of addSemester() to add courses to the database.
   iii. Postcondition: After a semester is created with courses, those courses will be added to the database.
   iv. Returns: Boolean

e. requestStoragePermissions()
   i. Parameters: none
   ii. Functionality: Requests access to the storage of the user's device.
   iii. Postcondition: System will be able to access storage on the device for input and output of data.
   iv. Returns: void

C. CourseActivity
   a. onCreate()
      i. Parameters: Bundle savedInstanceState
      ii. Functionality: Initializes the activity’s attributes and objects such as XML views and widgets, data structures, and drives the activity’s functions.
      iii. Postcondition: The user’s display will contain all of this class’s views and widgets, and it will have all of the functionality listed in the following methods.
      iv. Returns: void
   b. addCourse()
      i. Parameters: String userEmail, String semesterName
      ii. Functionality: Adds a single course to the semester in the database and on the recycler view.
      iii. Postcondition: The semester will display the new course and the database will reflect the addition.
      iv. Returns: void

D. SectionViewActivity
   a. onCreate()
      i. Parameters: Bundle savedInstanceState
      ii. Functionality: Initializes the activity’s attributes and objects such as XML views and widgets, data structures, and drives the activity’s functions.
      iii. Postcondition: The user’s display will contain all of this class’s views and widgets, and it will have all of the functionality listed in the following methods.
      iv. Returns: void
   b. editMeetingTimes()
      i. Parameters: none
      ii. Functionality: On its first use, displays a popup window that allows the user to input the days and times this course meets. Once the user has added days and times to the database, the system will not allow changes. The button that calls this method will display “View Times”. Calling this method again after the database contains meeting times for this course will only have a button to dismiss the popup. Any changes made will not be saved at this point.
      iii. Postcondition: Meeting days and times will be inserted into the database reflecting what the user inputs. Clicking the button again will display what the database holds.
      iv. Returns: void
   c. initiateAnalysisScreen()
      i. Parameters: none
      ii. Functionality: Brings the user to the analysis screen.
      iii. Postcondition: The system will have navigated from SectionViewActivity to AnalysisScreenActivity.
      iv. Returns: void

d. initiateBatchRequest()
    i. Parameters: none
    ii. Functionality: Opens the dialog on which the user chooses whether or not to send a batch of QR codes, as a result of pressing the QR code sending button.
    iii. Postcondition: The QR codes are sent, or an exception is thrown
    iv. Returns: void
e. initiateQRBatchConfirmation()
    i. Parameters: none
    ii. Functionality: Generates a confirmation popup that appears after QR codes are sent to all students using the batch qr code sending button.
    iii. Postcondition: The popup is shown on screen, and waits to be closed.
    iv. Returns: void
f. importClassList()
    i. Parameters: Uri uri
    ii. Functionality: Allows the user to select a CSV file from external storage. The method will read through the file, getting students from a line's fields and adding students to the database and student list.
    iii. Postcondition: The database and student list are updated with the imported students.
    iv. Returns: void
g. makeScanOptions()
    i. Parameters: String prompt
    ii. Functionality: Constructs the options for the QR code scanner (used when launching the barcodeLauncher).
    iii. Postcondition: Returns ScanOptions that have been configured for the QR code scanner.
    iv. Returns: ScanOptions
h. recordAttendance()
    i. Parameters: String studentEmail
    ii. Functionality: Returns true if 1) a student with the given student email exists in the database and 2) the date present for the student was successfully recorded in the database. This returns false otherwise.
    iii. Postcondition: Records the current date as a date present for the given student in the database (as long as it is valid).
    iv. Returns: boolean
i. generateReports()
    i. Parameters: None
    ii. Functionality: Displays a pop-up window allowing the user to enter a date that will be used in report generation. This allows the user to input the date either through the keyboard or through a date picker. The pop-up disappears if the user selects “cancel” or “accept.” If the user selects “accept,” then attendance reports will be sent to the user’s email.

iii. Postcondition: The pop-up disappears, and if the user selected “accept” with a valid date, then report generation is started, and reports are sent to the user’s email.

j. writeReports()
   i. Parameters: Date chosenDate
   ii. Functionality: Creates attendance reports based on the date in the database. These files are stored in the device as CSV files. The reports are also mailed to the user via the user's email.
   iii. Postcondition: Attendance reports are stored in the device’s application storage and emailed to the user.
   iv. Returns: void

k. sendReports()
   i. Parameters: File… reportFiles
   ii. Functionality: Constructs an email with the CSV attendance report files and sends it to the user’s email.
   iii. Postcondition: Emails containing attendance report files are sent to the user’s email.
   iv. Returns: void

l. generateQrCodeBatch()
   i. Parameters: None
   ii. Functionality: Generates QR codes based on students in the database and stores them to the device's application data. Then, if the user specified that it wants to send QR codes to students, it sends QR codes to them via their emails. iii. Postcondition: QR codes are stored in the device’s application storage, and (if specified) are sent to students via their emails.
   iii. Returns: void

m. sendQrCodes()
   i. Parameters: String studentName, String studentEmail, File qrCode
   ii. Functionality: Constructs an email with the student’s QR code and sends it to the student’s email.
   iii. Postcondition: An email is sent to a student with the given QR code. iv. Returns: void

n. addStudent()
   i. Parameters: none
   ii. Functionality: If the fields are entered and the email is not in the database already, then the inputted student information is added to the class list.
   iii. Postcondition: The student will be added to the class list on the screen and in the database.
   iv. Returns: void

## E. StudentDetailsActivity

a. onCreate()
   i. Parameters: Bundle savedInstanceState
   ii. Functionality: Initializes the activity's attributes and objects such as XML views and widgets, data structures, and drives the activity's functions.
   iii. Postcondition: The user's display will contain all of this class's views and widgets.
   iv. Returns: void

## F. DBHandler

a. onCreate()
   i. Parameters: SQLiteDatabase db
   ii. Functionality: Generates the database and all of the tables that will be used within the database using SQL statements. The database is only built if it does not already exist in the applications data.
   iii. Postcondition: The database will be built on the device's storage.
   iv. Returns: void

b. onUpgrade()
   i. Parameters: SQLiteDatabase db, int oldVersion, int newVersion
   ii. Functionality: Allows the programmer to update tables. This method is called every time a request to the database has been made.
   iii. Postcondition: If the database schema has been upgraded, and version control has been properly maintained, the database will be upgraded to represent the changes made from this method.
   iv. Returns: void

c. checkEmailExist()
   i. Parameters: String email
   ii. Functionality: Query the database to check if the email exists on the user table. iii. Postcondition: True will be returned if the email exists, false will return if the email does not exist.
   iii. Returns: Boolean

d. checkEmailPassword()
   i. Parameters: String email, String password
   ii. Functionality: Authentication method for user credentials when logging in. The system will query the database for a match with the arguments.
   iii. Postcondition: Will return true if there is a match and false if there is not a match.
   iv. Returns: Boolean

e. Add Methods
   i. Parameters: Keys to access desired table
   ii. Functionality: Will add the respective entity to its corresponding table as long as the primary key constraint is not violated.

   iii. Postcondition: The entity will be added to its corresponding table and return a Boolean referring to whether the addition was a success or failure.
   iv. Returns: Boolean

f. Get Methods
   i. Parameters: Keys to access desired table
   ii. Functionality: Retrieves the specified entity/entities in the method name.
   iii. Postcondition: The database will be queried to retrieve all tuples that match the query.
   iv. Returns: String/ ArrayList/ ArrayList>/ ArrayList

g. updateSemesterName()
   i. Parameters: String oldSemesterName, String newSemesterName, String userEmail
   ii. Functionality: Updates a semester's oldSemesterName with newSemesterName.
   iii. Postcondition: The semesters table will have an updated tuple where the updated semester name is replaced with the new name.
   iv. Returns: void

h. updateCourseName()
   i. Parameters: String oldCourseName, String newCourseName, String semesterName, String userEmail
   ii. Functionality: Updates oldCourseName with newCourseName.
   iii. Postcondition: The courses table will have an updated tuple where the updated course name is replaced with the new name.
   iv. Returns: void

i. updateStudentName()
   i. Parameters: newStudentName, String course, String semester, String userEmail, String studentEmail
   ii. Functionality: Updates the name of the student identified with the inputted studentEmail.
   iii. Postcondition: The student's name will be updated with newStudentName.
   iv. Returns: void

j. updateStudentEmail()
   i. Parameters: String oldEmail, String newEmail, String course, String semester, String userEmail
   ii. Functionality: Changes the email address of a student with newEmail.
   iii. Postcondition: The students table will have an updated tuple where the updated student email is replaced with the new email.
   iv. Returns: void

k. Delete Methods
   i. Parameters: Keys to access desired table
   ii. Functionality: Deletes the corresponding entity from its table using its primary key.

iii. Postcondition: The table that the entity was on will no longer contain the tuple corresponding to the deleted entity.
iv. Returns: void

l. Count Methods
   i. Parameters: none, or keys corresponding to desired table
   ii. Functionality: Counts the number of occurrences of a particular query. This could be the size of a table or the size of a cursor that queries with a parameter list.
   iii. Postcondition: The size of the query will be returned
   iv. Returns: long

## G. EmailSender

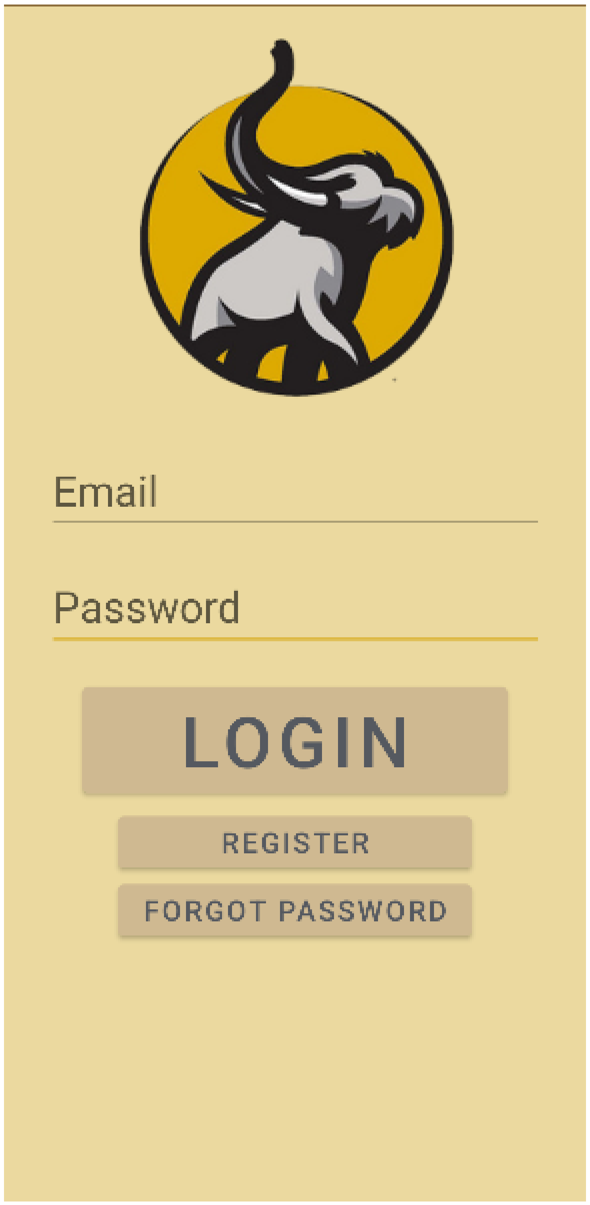
a. Constructor()
   i. Parameters: none
   ii. Functionality: Sets the session properties required to send an email.
   iii. Postcondition: Email session properties set and credentials authenticated.
b. emailStudentQRCode()
   i. Parameters: String sectionName, String studentName, String studentEmail, String userName, File qrCode
   ii. Functionality: Constructs a message to the student and sends the specified file (QR code) to the specified student email.
   iii. Postcondition: Email is sent to the student containing the QR code and a simple message.
   iv. Returns: void
c. emailInstructorReports()
   i. Parameters: String semesterName, String sectionName, String userEmail, String userName, File... reportFiles
   ii. Functionality: Constructs a message to the instructor and sends the specified files (CSV attendance report files) to the specified user email.
   iii. Postcondition: Email is sent to the user containing three attendance report files and a simple message.
   iv. Returns: void

## H. QRCodeOperator

a. generateQRCode()
   i. Parameters: String studentEmail
   ii. Functionality: Encodes the given student's email as a QR code (2D bitmap) and returns the generated QR code.
   iii. Postcondition: QR code that decodes to the given student’s email is generated and returned.
   iv. Returns: Bitmap

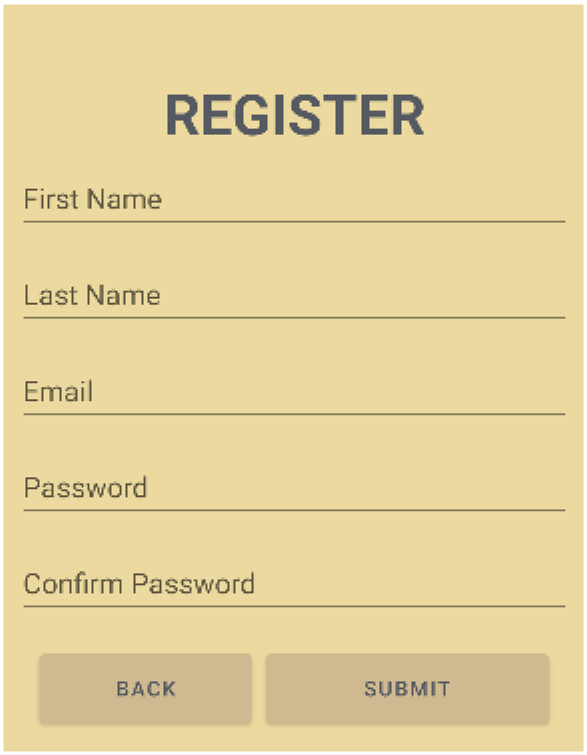
## 5. Demonstration

### A. Opening Screen



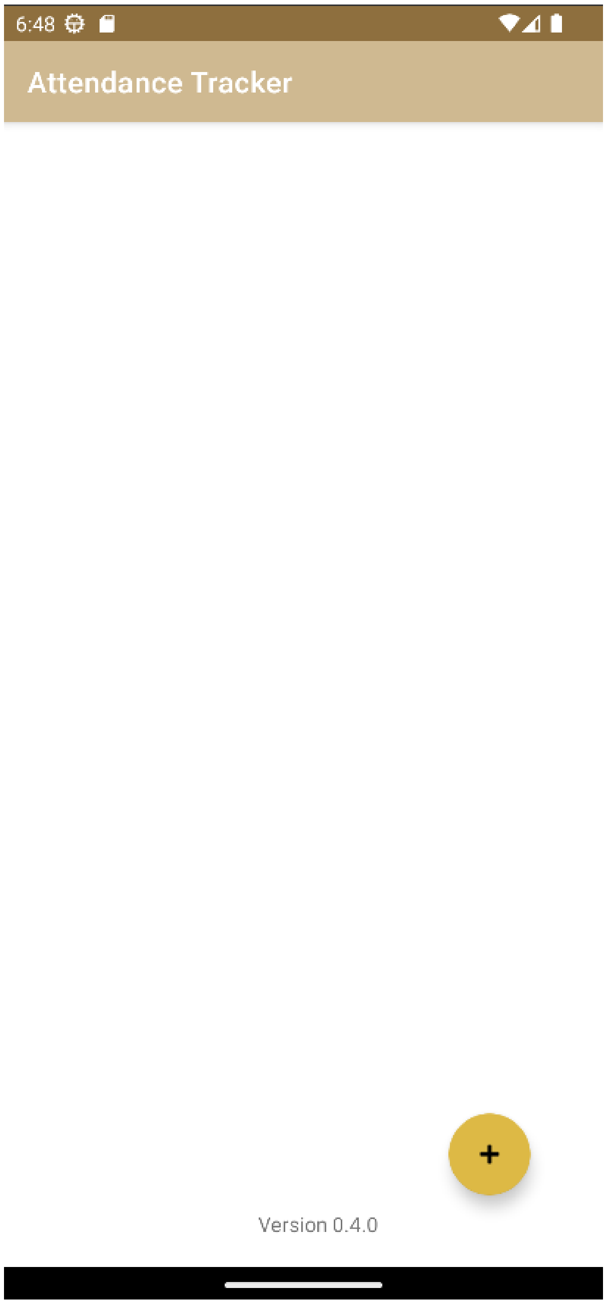
a. Users are prompted to either Login or Register an account.
b. Logging in requires an email and password that is within the system database.
c. Clicking the register button takes the user to the registration page.
d. Clicking login with proper account info will take the user to the course page for that account.

B. Register Screen

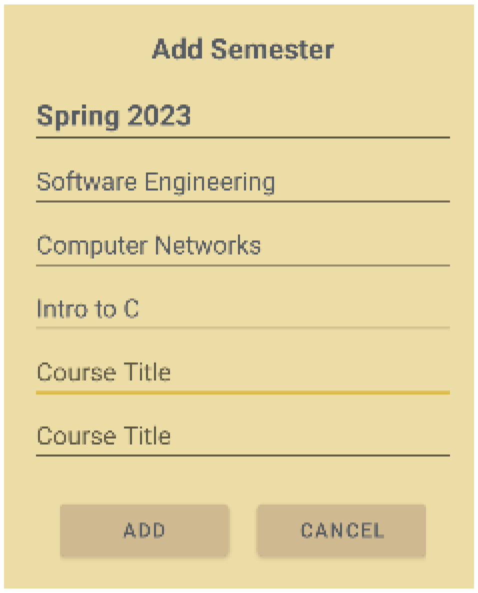


a. Users can create accounts.
b. Accounts require a first name, last name, email, and password.
c. Account information is stored locally in the device's storage.
d. Successfully registering an account takes the user to the login page, where they can re-enter the information to log into the account's course screen.

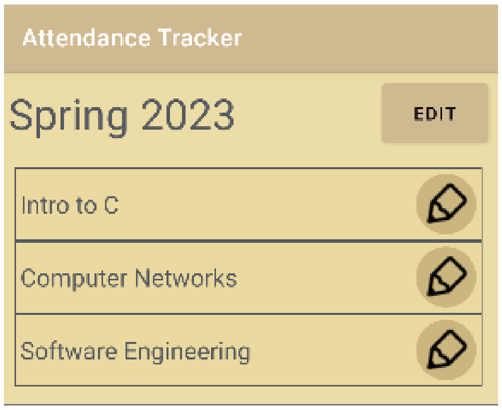
C. Semester Creation Screen
   a. Add Semester Screen



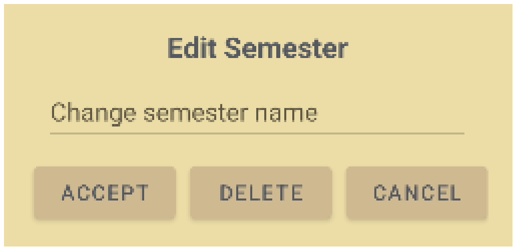
   b. Users can add a semester by clicking on the add button in the bottom right corner.
   c. The add button will take the user to a screen where they can set a semester name and up to 5 courses attached to that semester.



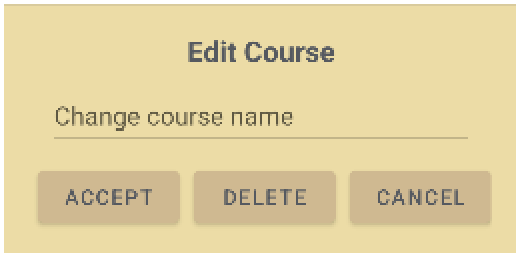
d. After adding a semester, the courses are added to the semester screen.



e. The user can edit a semester by clicking on the edit button by the semester name.

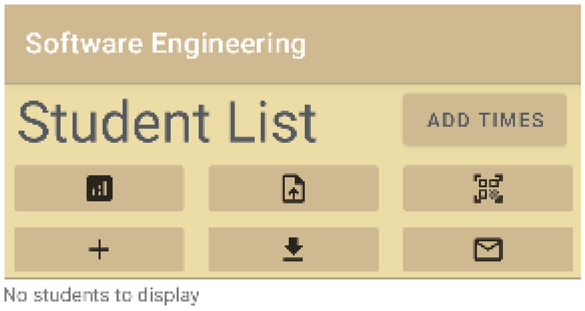


f. The user can also edit a course by clicking on the edit button by the course's name.

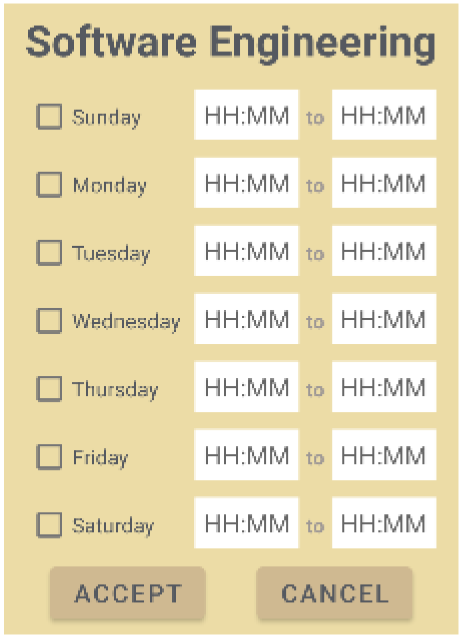


g. Clicking on the name of a class will take the user to that class's Course View page.

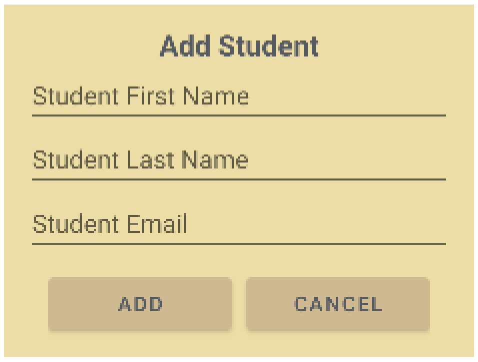
D. Course View Screen

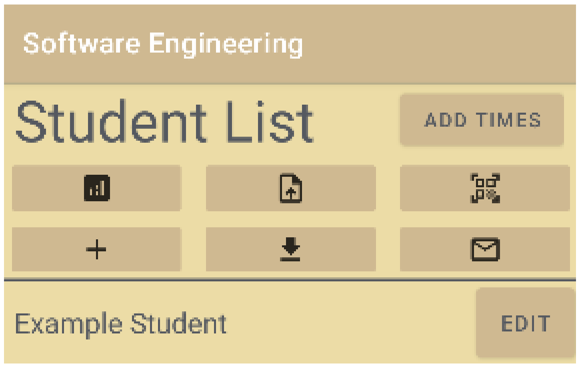


a. By clicking on the Add Times button in the top right corner, the user can type in the time that the class is in session for each day of the week.

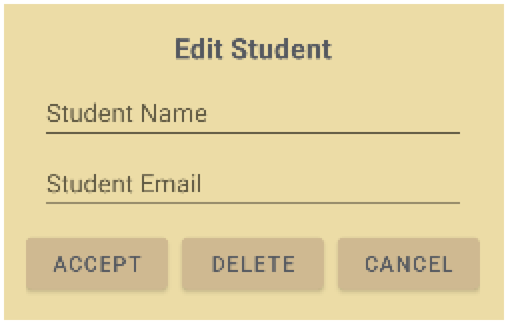


b. By clicking the Add Student button in the bottom left of the button grid, the user can add a student to the course with their first name, last name, and student email.

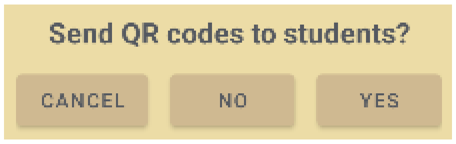




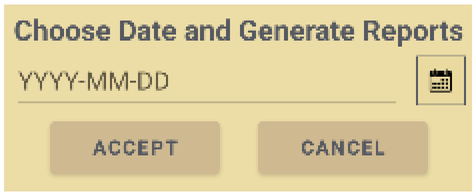
c. By clicking the edit button by the student’s name, the user can edit their name and email.



d. The user can batch send QR Codes to all students’ emails by clicking the Send QR Codes button.



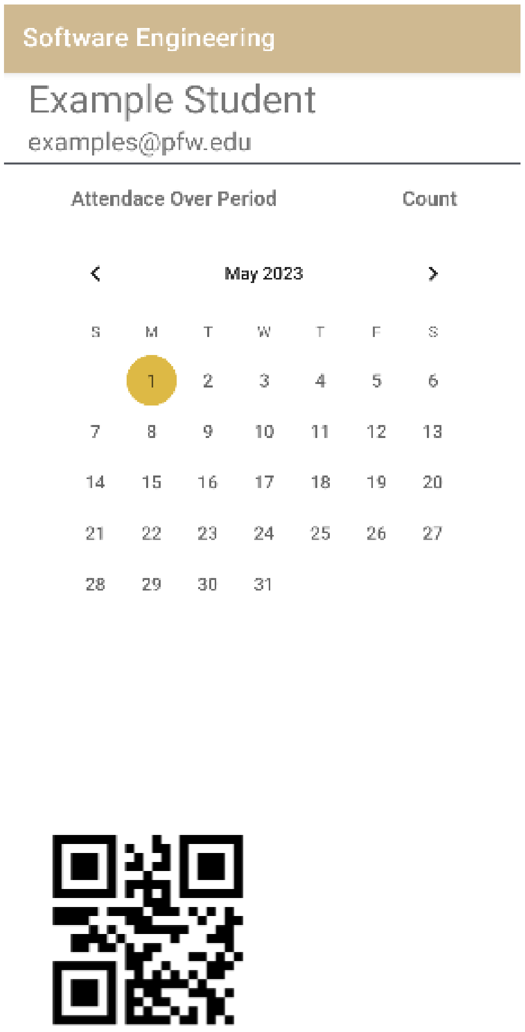
e. The user can generate reports by clicking the Generate Reports button. These reports are created depending on the given date and sent to the attendance tracker email.



f. By clicking on the Import Class List button, the user can import a predetermined class by selecting the CSV file in their local storage.

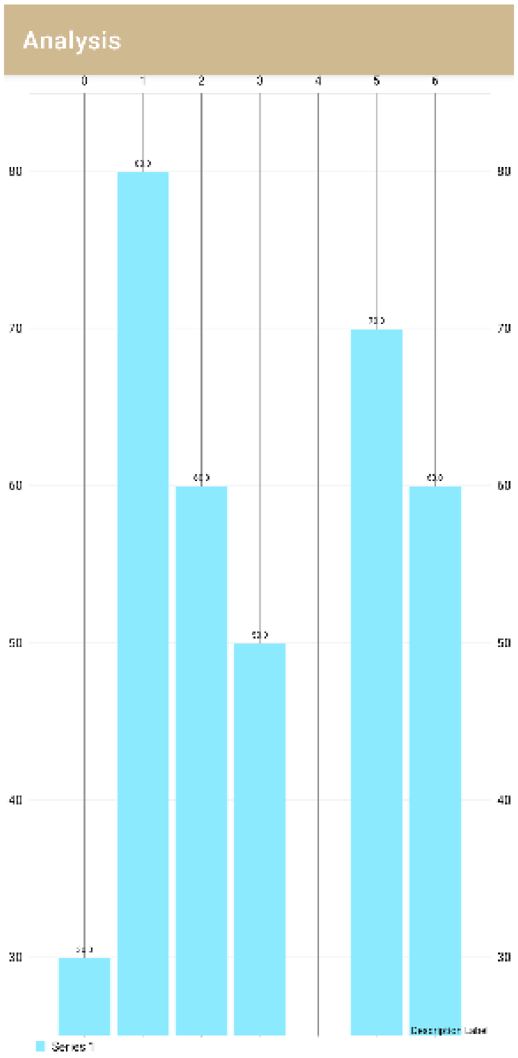
E. Student View

a. The user can access the Student View screen for a particular student by clicking on their name.



F. Analytics Screen

   a. The user can access the placeholder analytics screen by pressing the Analytics Screen button on the Course View page.



G. QR Code Scanner Screen
   a. The user can go to the placeholder QR Code Scanner screen by clicking the Scan button. Any student QR codes that are visible to the scanner will be identified as that student and will log the student as attended for that day of class.

## 6. Testing

- The cases laid out in the 3.0 documentation were ran again to ensure that our changes did not break the existing functionality of the program. These have been documented below for convenience. Additionally, we ran a series of tests relevant to the changes we made in the 4.0 edition, which are the first two tables.

## Test Coverage Report (Test Cases):

| **Test Case ID** | **Test Case** | **Pre-condition** | **Test Steps** | **Test Data** | **Expected Result** | **Actual Result** | **Status Pass/ Fail** |
|---|---|---|---|---|---|---|---|
| Import_Student_Class_List_Test 7.5 | Importing a singular student to a class list | 1.) Students are registered for the course.<br>2.) Student has a valid PFW email | 1.) Enter student PFW Email with and first and last name. | CSV file with student data (first, last, email) | Student successfully added | Student successfully added | Pass |
| Import_Student_Class_List_Test 7.5 | Importing a singular student to a class list | 1.) Students are registered for the course.<br>2.) Student has a valid PFW email | 1.) Enter invalid student PFW Email with and first and last name. | CSV file with student data (first, last, email) | Unsuccessful addition of students. Invalid Email | Unsuccessful addition of students. Invalid Email | Pass |
| Import_Student_Class_List_Test_7.6 | Importing a singular studnet to a class list | 1.) Students are registered for the course.<br>2.) Student has a valid PFW email | 1.)Enter incomplete fields.<br>2.)Click enter | CSV file with student data (first, last, email) | Unsuccessful addition of students. Please fill all fields | Unsuccessful addition of students. Please fill all fields | pass |

| **Test Case** | **Test Case** | **Pre-condition** | **Test Steps** | **Test Data** | **Expected** | **Actual** | **Statu** |
|---|---|---|---|---|---|---|---|

| **ID** | | | | | **Result** | **Result** | **s Pass/ Fail** |
|---|---|---|---|---|---|---|---|
| Generate_QR_Code_Test. 6.5 | Sending QR code to a singular student | 1. Students are registered for the course. 2. Student has valid credentials | 1.) Enter student PFW Email 2.) Click Send | Semester, course, and student email | Qr code successfully sent | Qr code successfully sent | Pass |
| Generate_QR_Code_Test. 6.6 | QR Code confirmation dialogue when sent | 1. ) student information is entered into boxes. 2.) Students are registered for the course. | 1.) Hit send | Semester, course, and student email | Qr code successfully sent | Qr code successfully sent | pass |
| Generate_QR_Code_Test. 6.6 | QR Code confirmation dialogue when sent | 1. ) invalid student information is entered into boxes. 2.) Students are registered for the course. | 1.) Hit send | Semester, course, and student email | Qr code not sent. Check student input data | Qr code not sent. Check student input data | pass |
| Generate_QR_Code_Test. 6.7 | QR Code confirmation dialogue when sent | 1. ) Valid student information is entered into boxes. 2.) Students are not registered for the course. | 1.) Hit Send | Semester, course, and student email | Qr code not sent. Student not found | Qr code not sent. Student not found | pass |
| Generate_QR_Code_Test. 6.8 | QR Code confirmation dialogue when sent | 1. ) No information is entered into the boxes | 1.) Hit Send | Semester, course, and student email | Please enter all required fields. | Please enter all required fields. | pass |

| Test Case ID | Test Case | Pre-Condition | Test Steps | Test Data | Expected Result | Actual Result | Status Pass/Fail |
|---|---|---|---|---|---|---|---|
| Report_Test 2.0 | Generate reports for course with no students added | 1) Have no students registered for course selected | 1) Click the button with paper and arrow up image 2) Input valid date 3) Select accept to generate reports | 2022/12/12 | Reports generated with empty list | Reports generated with empty list | Pass |
| Report_Test 2.1 | Generate reports for course with invalid date format | 1) Go to course screen of any created crouse | 1) Click the button with paper and arrow up image 2) Input invalid date 3) Select accept to generate reports | 20221212 | Invalid date format | Invalid date format | Pass |
| Report_Test 2.2 | Empty date field | 1) Go to course screen of any created crouse | 1) Click the button with paper and arrow up image 2) Select accept to generate reports | "" | Invalid date format | Invalid date format | Pass |
| Report_Test 2.3 | Generate reports for course with students added | 1) Go to course screen of any created crouse with students added | 1) Click the button with paper and arrow up image 2) Input valid date 3) Select accept to generate reports | 2022-12-12 | Reports generated successfully and save to system file | Reports generated successfully and saved to system file | Pass |
| Report_Test 2.4 | Generate reports and send to user | 1) Go to course screen of any created course with students added | 1) Click the button with paper and arrow up image 2) Input valid date 3) Select accept to generate reports | 2022-12-12 and registered user email | Reports generated successfully and email sent | Report generated successfully and email sent | Pass |

| Test Case ID | Test Case | Pre-Condition | Test Steps | Test Data | Expected Result | Actual Result | Status Pass/Fail |
|---|---|---|---|---|---|---|---|
| Semester_ Creation_Test 3.0 | Enter valid semester name | 1) Be in semester screen | 1) Click on + button<br>2) Input valid semester name<br>3) Leave course fields empty<br>4) Click add button | Fall 2022 | Fall 2022 successfully added | Fall 2022 successfully added | Pass |
| Semester_ Creation_Test 3.1 | Enter duplicate semester name | 1) Be in semester screen | 1) Click on + button<br>2) Input duplicate semester name<br>3) Leave course fields empty<br>4) Click add button | Fall 2022 | Could not add Fall 2022 | Could not add Fall 2022 | Pass |
| Semester_ Creation_Test 3.2 | Enter empty semester name | 1) Be in semester screen | 1) Click on + button<br>2) Click add button | "" | Could not add semester | Could not add semester | Pass |
| Semester_ Creation_Test 3.3 | Enter symbols in semester name | 1) Be in semester screen | 1) Click on + button<br>2) Input symbols<br>3) Click add button | !@#$%^&*()_+ | !@#$%^&*()_+ successfully added | !@#$%^&*()_+ successfully added | Pass |
| Semester_ Creation_Test 3.4 | Enter valid semester name and course name | 1) Be in semester screen | 1) Click on + button<br>2) Input valid semester name<br>3) Inter valid course name in course field<br>4) Click add button | Fall 2022 , CS360 | Semester and course successfully added | Semester and course successfully added | Pass |

| Test Case ID | Test Case | Pre-Condition | Test Steps | Test Data | Expected Result | Actual Result | Status Pass/Fail |
|---|---|---|---|---|---|---|---|
| Course_ Creation_Test 4.0 | Enter valid course name | 1) Be in course screen | 1) Click + button 2) Input valid course name 3) Click accept button | CS360 | Course CS360 successfully added | Course CS360 successfully added | Pass |
| Course_ Creation_Test 4.1 | Enter empty course name | 1) Be in course screen | 1) Click + button 2) Click accept button | "" | Course name cannot be empty | Course name cannot be empty | Pass |
| Course_ Creation_Test 4.2 | Enter symbols in course name | 1) Be in course screen | 1) Click + button 2) Input symbols in course name 3) Click accept button | !@#$%^&*()_+ | Course successfully added | Course !@#$%^&*()_+ successfully added | Pass |
| Course_ Creation_Test 4.3 | Enter duplicate course name | 1) Be in course screen | 1) Click + button 2) Input duplicate course name 3) Click accept button | CS360 | Could not add course | Could not add course | Pass |
| Course_ Creation_Test 4.4 | Enter course name with 50 characters | 1) Be in course screen | 1) Click + button 2) Input course name 3) Click accept button | CS36000000000000000000000000000000000000000000000000 | Course successfully added | Course successfully added | Pass |

| Test Case ID | Test Case | Pre-Condition | Test Steps | Test Data | Expected Result | Actual Result | Status Pass/Fail |
|---|---|---|---|---|---|---|---|
| Scan_QR_ Code_Test 5.0 | Scan blank Image | 1) Device with camera 2) Be in course screen | 1) Click on button with QR image to activate scanning 2) Point camera at blank image | Blank image | No QR code scanned | No QR code scanned | Pass |
| Scan_QR_ Code_Test 5.1 | Scan an invalid image | 1) Device with camera 2) Be in course screen | 1) Click on button with QR image to activate scanning 2) Point camera at invalid image | Invalid image (Not a QR code) | No QR code scanned | No QR code scanned | Pass |
| Scan_QR_ Code_Test 5.2 | Scan an invalid QR Code | 1) Device with camera 2) Be in course screen | 1) Click on button with QR image to activate scanning 2) Point camera at invalid QR code | Invalid QR code | QR code scan unsuccessful | QR code scan unsuccessful | Pass |
| Scan_QR_ Code_Test 5.3 | Scan valid QR code | 1) Device with camera 2) Be in course screen | 1) Click on button with QR image to activate scanning 2) Point camera at valid QR code | Valid QR code (QR code generated by application) | QR code scanned successfully and data stored | QR code scanned successfully and data stored | Pass |
| Scan_QR_ Code_Test 5.4 | Scan QR code continuousl y | 1) Device with camera 2) Be in course screen | 1) Click on button with QR image to activate scanning 2) Point camera at valid QR codes sequentially | Valid QR codes (QR codes generated by application) | QR code scanned successfully without existing scanning mode | QR code scanned successfully without existing scanning mode | Pass |

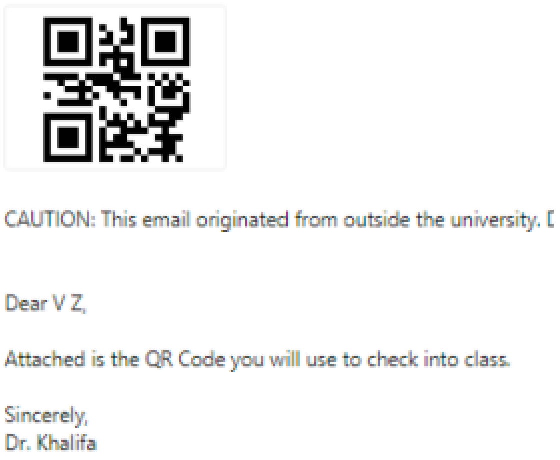
| Test Case ID | Test Case | Pre-Condition | Test Steps | Test Data | Expected Result | Actual Result | Status Pass/Fail |
|---|---|---|---|---|---|---|---|
| Generate_ QR_Code_ Test 6.0 | Generate bulk QR code | 1) Be in course screen<br>2) Multiple students added in course selected | 1) Click on button with mail image to activate QR code generate<br>2) Select yes in pop up window | Semester, course, and student list | QR codes generated | QR codes generated | Pass |
| Generate_ QR_Code_ Test 6.1 | Write generated QR code to filesystem | 1) Be in course screen<br>2) Students added in course | 1) Click on button with mail image to activate QR code generate<br>2) Select yes in pop up window | Semester, course, and student list | QR codes written to filesystem on device | QR codes written to filesystem on device | Pass |
| Generate_ QR_Code_ Test 6.2 | Send QR codes by email | 1) Be in course screen<br>2) Students added in course with valid credentials | 1) Click on button with mail image to activate QR code generate<br>2) Select yes in pop up window | Semester, course, and student list | QR codes generated and sent by email | QR codes generated and sent by email | Pass |
| Generate_ QR_Code_ Test 6.3 | Error sending QR codes by email | 1) Be in course screen<br>2) Students added with invalid credentials | 1) Click on button with mail image to activate QR code generate<br>2) Select yes in pop up window | Semester, course, student list, invalid credential (student email example.com) | QR codes generated and not sent by email | QR codes generate and not sent by email | Pass |
| Generate_ QR_Code_ Test 6.4 | Write generated QR code to full filesystem | 1) QR generated<br>2) File system full | 1) Click on button with mail image to activate QR code generate<br>2) Select yes in pop up window | Semester, course, and student list | QR codes generated and not stored to files system | QR codes generated and not stored to files system | Pass |

| Test Case ID | Test Case | Pre-Condition | Test Steps | Test Data | Expected Result | Actual Result | Status Pass/Fail |
|---|---|---|---|---|---|---|---|
| Import_ Student_ Class_ List_ Test 7.0 | Import CSV file with one student | 1) CSV file with properly format student data (first, last, email) on local device storage | 1) Click on button with arrow down image 2) Select CSV file from local storage | CSV file with student data (first, last, email) | Student successfully added from CSV file | Student successfully added from CSV file | Pass |
| Import_ Student_ Class_ List_ Test 7.1 | Import CSV file with multiple students | 1) CSV file with properly format students data (first, last, email) on local device storage | 1) Click on button with arrow down image 2) Select CSV file from local storage | CSV file with students data (first, last, email) | Students successfully added from CSV file | Students successfully added from CSV file | Pass |
| Import_ Student_ Class_ List_ Test 7.2 | Import CSV file with no student added | 1) CSV file with no students on local device storage | 1) Click on button with arrow down image 2) Select CSV file from local storage | CSV file with no student data ("", "", "") | Error reading from file | Error reading from file | Pass |
| Import_ Student_ Class_ List_ Test 7.3 | Import CSV file with invalid format of student data | 1) CSV file with invalid format of student data (first, last, email, favorite color) on local device storage | 1) Click on button with arrow down image 2) Select CSV file from local storage | CSV file with invalid format student data (first, last, email, favorite color) | Student successfully added from CSV file with the first three fields (first, last, email) | Student successfully added from CSV file with the first three fields (first, last, email) | Pass |
| Import_ Student_ Class_ List_ Test 7.4 | Import CSV file with missing last name | CVS file on device with student data missing last name | 1) Click on button with arrow down image 2) Select CSV file from local storage | CSV file with missing data first, email | Student successfully added from CSV file | Student successfully added from CSV file | Pass |

## 7. Tools Used

- **Java:**
    - Our programming language of choice. Its object oriented design and high level features make implementation relatively intuitive and self-documenting.
- **Android Studio:**
    - The official integrated development environment for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development.
- **Android Virtual Device (AVD):**
    - A configuration that defines the characteristics of an Android phone, tablet, Wear OS, Android TV, or Automotive OS device that you want to simulate in the Android Emulator.
- **Device Manager:**
    - An interface you can launch from Android Studio that helps you create and manage AVDs
- **SQLite:**
    - A library that implements a self-contained and serverless SQL database engine. Replaced TinyDB, which was used in the previous version.
    - This is useful for highly personalized apps where the user will not need to share data with another device or person

- **Outlook API:**
    - The Outlook API is used to send out the QR codes to emails via the attendance.tracker.pfw@outlook.com email address.
    - The credentials for the email address are as follows:
        - **Email:** attendance.tracker.pfw@outlook.com
        - **Password:** PurdueFortWayneAttendanceTracker123
        - **Application Password:** hpoaviagzwckefbz
        - **Country:** United States
        - **Date of Birth:** January 1, 2000
        - **Verification Emails:** bradea02@pfw.edu, bencmd01@pfw.edu
        - **Verification Phone Number:** (260) 271-9655
    - An example email sent by the application looks like this:



CAUTION: This email originated from outside the university.

Dear V Z,

Attached is the QR Code you will use to check into class.

Sincerely,
Dr. Khalifa

- **ZXing ("Zebra Crossing") library:**
    - ZXing ("zebra crossing") is an open-source, multi-format 1D/2D barcode image processing library implemented in Java, with ports to other languages
- **GitHub**:
    - An online software development platform used for storing, tracking, and collaborating on software projects.
    - It enables developers to upload their own code files and to collaborate with fellow developers on open-source projects.
    - It uses Git, a distributed version control system, to keep track of project modifications.
- **Barcode component reader (QR code scanning and history):**
    - The Barcode Scanner is an open-source library from ZXing that allows an Android device with imaging hardware to scan barcodes or 2D barcodes and retrieve the data encoded.

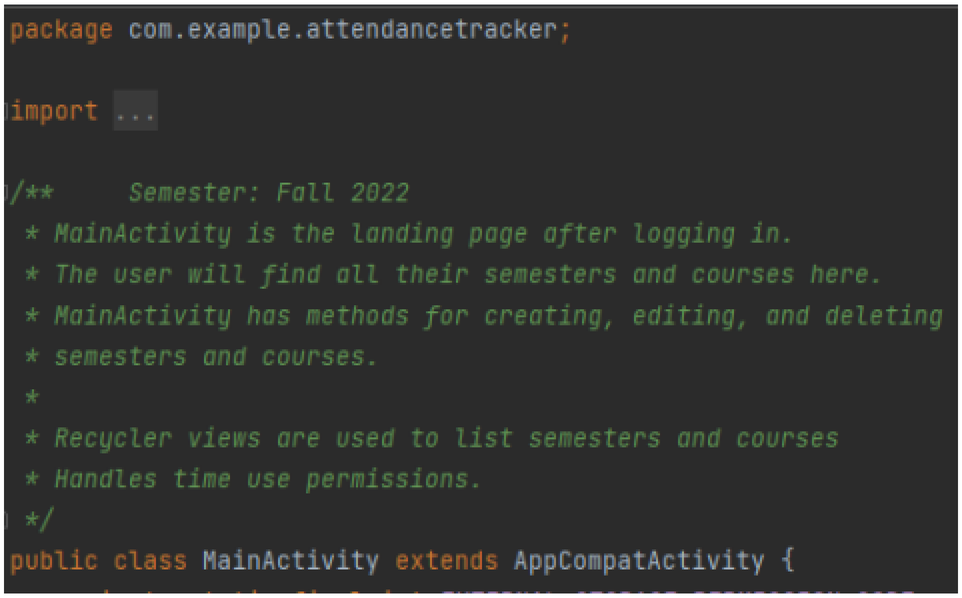
## Read Me for Deployment:

1. To deploy the Attendance Tracker application, Android Studio must be installed, and the project must be downloaded from the git repository.
    a. See this link to download Android Studio: https://developer.android.com/studio
2. From here, you can choose to either use a physical device attached via USB or a built-in emulator.
    **a. For a physical device:**
        i. A USB cable should be used to connect the physical device to the computer that Android Studio runs on.
        ii. Developer Options will need to be enabled on the Android device in order to put the Attendance Tracker application on it (see device manufacturer's steps for enabling this feature).
            1. See this guide for enabling Developer Options: 39 https://developer.android.com/training/basics/firstapp/running-app
        iii. Once Developer Options has been enabled, enable USB Debugging.
        iv. Run the Attendance Tracker application on the Android device. It will download the application to the device, where the device can now safely use the app, with or without a connection to Android Studio.
        v. When the app is first run, the lecturer may get a pop-up dialog asking for certain permissions (e.g., read and write storage, network activity), which must be enabled to run the application.
    **b. For a built-in emulator:**
        i. Extract the project zip to a location of your choosing
        ii. Open android studio, and navigate to the 3 dots by the New Project | Open | Get from VCS.
        iii. From there, select "Import Project"
        iv. Navigate to the extracted files and **select the attendance_tracker_system** directory (It should have an android logo instead of a regular folder symbol).
        v. Navigate to Device Manager in the top-right.
        vi. Choose a device to emulate, Android Studio has a decent selection. We suggest the Resizable (Experimental) device with the Tiramisu system image. This may take a while to download.
        vii. On the next page, press finish.
        viii. Press the play button in the top right, the field by it should show your chosen device.
        ix. From there, it should start running.

## 8. Good Programming Techniques

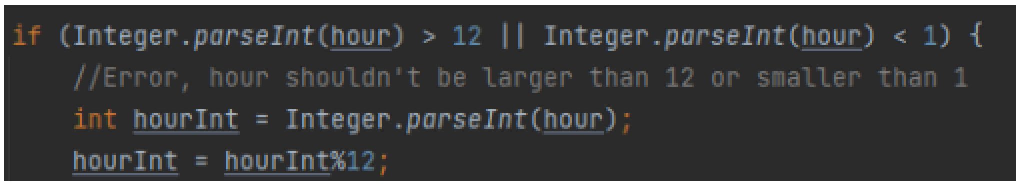
A. Comments

a. Header comment sections were added to the beginning of the main working classes in the project package. These explain the purpose of the class and any important information that is necessary to understand the class further.



```
package com.example.attendancetracker;

import ...

/**     Semester: Fall 2022
 * MainActivity is the landing page after logging in.
 * The user will find all their semesters and courses here.
 * MainActivity has methods for creating, editing, and deleting
 * semesters and courses.
 *
 * Recycler views are used to list semesters and courses
 * Handles time use permissions.
 */
public class MainActivity extends AppCompatActivity {
```

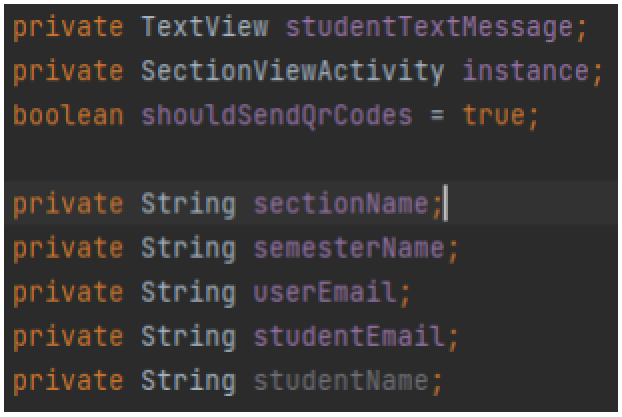
b. The code was written to be self documenting; however, this is not always reasonable for any number of reasons. When this is the case, comments are added to explain the following blocks of code.



```
if (Integer.parseInt(hour) > 12 || Integer.parseInt(hour) < 1) {
    //Error, hour shouldn't be larger than 12 or smaller than 1
    int hourInt = Integer.parseInt(hour);
    hourInt = hourInt%12;
```

B. Variable Declaration

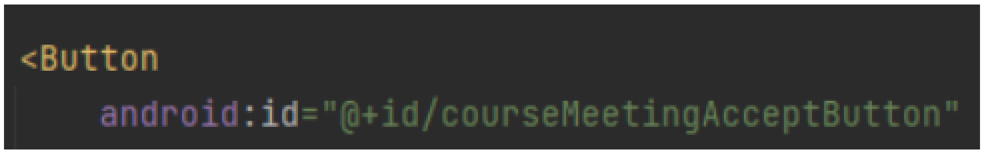
a. Variables were chosen carefully to help identify the purpose it held, and in some cases variables were named to identify their location and relevance in the code.



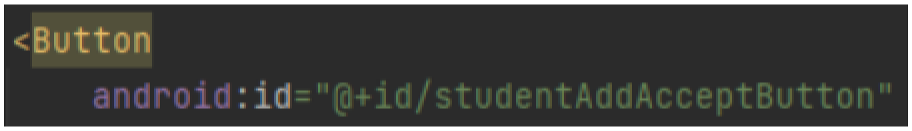
```
private TextView studentTextMessage;
private SectionViewActivity instance;
boolean shouldSendQrCodes = true;

private String sectionName;
private String semesterName;
private String userEmail;
private String studentEmail;
private String studentName;
```

b. XML id attributes were written to specifically identify its purpose and the activity that it is a part of. This was particularly important to distinguish different widgets and views that held the same functionality but were in different locations

*Two "Accept" buttons in two different dialogs*



```
<Button
    android:id="@+id/courseMeetingAcceptButton"
```



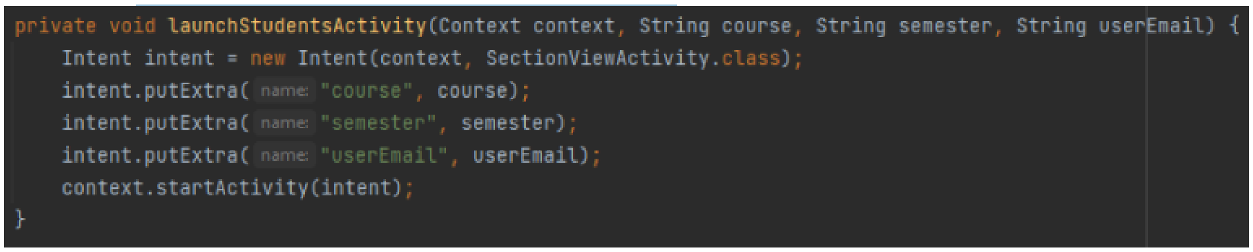
```
<Button
    android:id="@+id/studentAddAcceptButton"
```

C. Code Reuse
   a. Given the nature of the project, much of the code reuse consisted of reusing code from imported libraries. Many methods and other modules had specialized uses that did not allow for general reusability. However, code reusability was always considered when creating these modules.
   b. Helper methods were created within classes that could be used outside of its context. For example, the parseTime method in SectionViewActivity takes any String of numeric characters, including the colon ':' symbol, and converts it to an Integer array with two members, hours as [0] and minutes as [1]. This allows input Strings from EditText boxes to be converted so that it can be used with the database. This method is not coupled with the database directly, so this method could be used with any situation requiring the conversion of different String times to a tokenized Integer representation.
   c. QRCodeOperator class is the helper class that creates QR codes. The parameter in the constructor method is a String called studentEmail. This parameter name represents the intended input, but the class can create a QR code representing any String. This allows the class to be reusable for other purposes, such as a URL.

D. Data Hiding/ Encapsulation

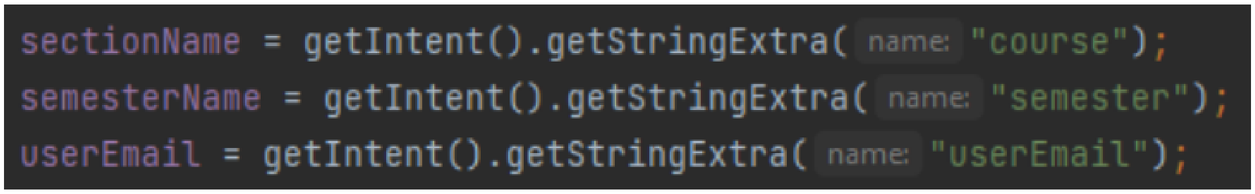
   a. Each class has different private attributes within it and methods that act on them within the class. In the context of this project, communication between classes of the application are done through Android intents. Therefore, information between classes in the Attendance Tracking System are shared explicitly and programmatically. Intents use a key-value pair system to access information from the previous activity.

*Launching next activity with intent extras*



```
private void launchStudentsActivity(Context context, String course, String semester, String userEmail) {
    Intent intent = new Intent(context, SectionViewActivity.class);
    intent.putExtra( name: "course", course);
    intent.putExtra( name: "semester", semester);
    intent.putExtra( name: "userEmail", userEmail);
    context.startActivity(intent);
}
```

*Accessing intent extras using their corresponding keys*



```
sectionName = getIntent().getStringExtra( name: "course");
semesterName = getIntent().getStringExtra( name: "semester");
userEmail = getIntent().getStringExtra( name: "userEmail");
```

E. Cohesion/Coupling

   a. All modules within our project have at least procedural cohesion. Some modules perform a series of actions related by the procedure to be followed by the product.
   b. All modules within our project have at least common coupling. Some modules have write access to global variables; however, modules do not have write access simultaneously.

## 9. Team Member Contribution

| Team Member | Requirements | Design | Presentations and Documents | Implementation | Testing |
|---|---|---|---|---|---|
| Martin Benchev | X | X | X | X | X |
| Joshua Smith | X | X | X | X | X |
| Adam Hughes | X | X | X | X | X |
| Connor Springer | X | X | X | X | X |