CSC131

Software Engineering

Instructor: Doan Nguyen, Ph.D

Summer Session I - 2018

Project Report

of

"Student Attendance Tracker"

(Code-WebApp:https://github.com/limmil/Tracker)

(Code-iOS:https://github.com/StrawHatAaron/CSc131/)

Team Name: American Digital Project Management Systems

Aaron Miller | Derrek Gass | Pawan Chandra | Minquan Li | Shayan Tom Amir

Due date: 07/2/2018

Table Of Contents

List Of Tables

Table No. Table Name **Page No.**

List Of Figures

Figure No. Figure Name **Page No.**

Chapter 1

Project Goals

American Digital Project Management Systems' goal of this project is to design an application that will allow educators to track the attendance of their students with a Web or iOS App that stores attendance data automatically and saves class time. By using Google Sheets as the database many teachers will be able to use and easily keep track of the attendance of their students. This system can help teachers and students make better use of class time.

Chapter 2

Problem Statement

Currently, the California State University, Sacramento does not have a tool in place for teachers to log their students' attendance automatically. Although attendance points are stored digitally, teachers across all departments are forced to track attendance by hand. In some cases classes are only 50 minutes long and over 10% of the class time can be spent tracking students' presence. This disruptive process is prone to error and takes away from valuable class time thus decreasing the overall quality of the class.

Chapter 3

Proposed Solution

The proposed system allows students to be able to access a website or iOS app to log their daily attendance with a unique key that expires within 15 minutes. The key will be provided by the teacher in the classroom at the start of the class. Students will need to access the system in the beginning of class, and will be prompted for their student identification number along with a key generated (or manually entered) by the educator. The iOS application requires that the user be in direct proximity of the room in which the class takes place. Once verified, the educator will be able to view and edit the attendance information by accessing a centralized spreadsheet containing the student's attendance logs. This spreadsheet may then be exported as a CSV file and imported into Canvas where it could be calculated with other student grades.

Chapter 4

Functional and Nonfunctional Requirements

Below we have 10 requirements. 7 Functional and 3 Nonfunctional. For our Functional Requirements we have labeled 1-5 in a chronological order in which they were finished. Doing it this way made sense because each of these requirements needed their preceding ones to be made first in order for it to be properly tested. For example, if the application were to generate a random key before the user was able to enter any information in then we would be developing at high risk and most likely would have wasted time. Thankfully our teams incremental steps to a finished product were clear and we avoided these types of pitfalls.

The next Functionalities 6 and 7 were set up first thing when the Google Sheet was created. These were easily implemented and provided a good back bone for the beginning of the back end testing.

Identifier	Priority	Description
REQ_1	5	The user shall be able to enter their student ID and key and optional comment to log attendance.
REQ_2	4	The system shall generate a random key for the professor to provide students.
REQ_3	5	The system shall verify that the student ID and key match the daily class information.
REQ_4	5	After successfully logging attendance, the system shall provide the student with a receipt for confirmation.
REQ_5	5	If the student enters incorrect information, the system shall reject the logging of attendance and provides an error message.
REQ_6	4	Teacher shall be able edit students' attendance manually via the GoogleSheet.
REQ_7	4	Teacher shall be able to view report of student attendance for each class via the GoogleSheet.

Table 1: Functional Requirements for Student Attendance Tracker

Next, for the nonfunctional requirements our team is focused on addressing security because, we don't want students skipping class. There are several measures that can be taken to prevent this so our team addressed the ones that we felt would be most crucial. In a future version we could more security like only allowing one sign in per MAC address a day.

In our requirements, we have added security as the main non-functional requirement. The reason is that the client requires attendance logging to be an honest account of the students who arrived that particular day, on-time. Students have an incentive to cheat, or collude with a friend, if given the opportunity, so the system need to make sure that only one student per record is in class and on time (REQ_8, REQ_9, and REQ_10). This is handled through the implementation of hashing algorithms, proximity tests (geolocation), and a submission timer.

Identifier	Priority	Description
REQ_8	3	Security: System Shall use the SHA256 algorithm to store the teachers password
REQ_9	4	Security: System shall allow daily key to be valid for no longer than 15 minutes.
REQ_10	1	Security: The system shall verify that the student is approximately within (50 feet) of the classroom in the iOS App.

Table 2: Nonfunctional Requirements for Student Attendance Tracker

Chapter 5 Methodology

Our team decided to go with a incremental and agile model. Some of us were very experienced in programming. While some of us were not. Aaron has used several APIs provided by Google that are used for both Android and the iOS some of them being. JavaScript, HTML5, CSS, Swift, and Java.

Methodology and Tools Used
Agile Methodology (1week Iterations, Iteration Retrospectives, Product Backlog)
Deployed Student Attendance Tracker on local Tomcat Server
Implemented Google Spreadsheet REST API in our application
Java Servlets, HTML5, CSS, Swift Storyboard, JSP, Obj. C, XML, JavaScript
Version Control: Open Source Git

Table 3: Methodologies Used for Student Attendance Tracker

Because of all the speed of development of our project (it being a shortened summer session), our team opted for following agile methodology as much as it was possible. During our development, we used a product backlog in order to define all the tasks that needed to be done and prioritized what tasks needed to be done first. In addition to the backlog, we had weekly iterations, where our goal was to deliver working code that could be used to demo to a potential customer. At the end of each iteration, we had a retrospective that allowed our team to improve our process by analyzing what is going well and what can be improved. Section 12 discusses this in more detail.

Chapter 6 System Diagram

As pictured in Figure 1, we have implemented the Model View Controller architecture for our mobile and web applications. There are three main components to our application. The first component, the view, is the graphic user interface. In our application, the student or teacher directly interacts with this component to input information such as student ID, class key and comments. This component was coded in HTML and CSS for our web application, as well as Storyboard and XIB for iOS. The second component is the controller, which provides the application logic through the Java Servlet. For our web application, this was written using Java. For our iOS app, this was written in Swift and Objective-C. Through the UI, the student/teacher can send data to the controller and model in many ways that can create a variety of view

outcomes, such as a student being late or entering the wrong ID and Key combination. The view also provides the teacher with a class key and confirmation receipt. When the student issues a request through the UI, the UI sends this request to the controller, which then sends information to the final component, the model. This component directly communicates with the database and uses HTTP protocols GET, POST and PUT to perform changes to the Google Sheet. In our implementation, we used the Gradle and Cocoa Pod libraries to develop with the Google Sheet v4 API. Another benefit of using this type of architecture is that the code is separated into three components, and as long as the interfaces are well defined, a change in one component should have little effect on the other two components.

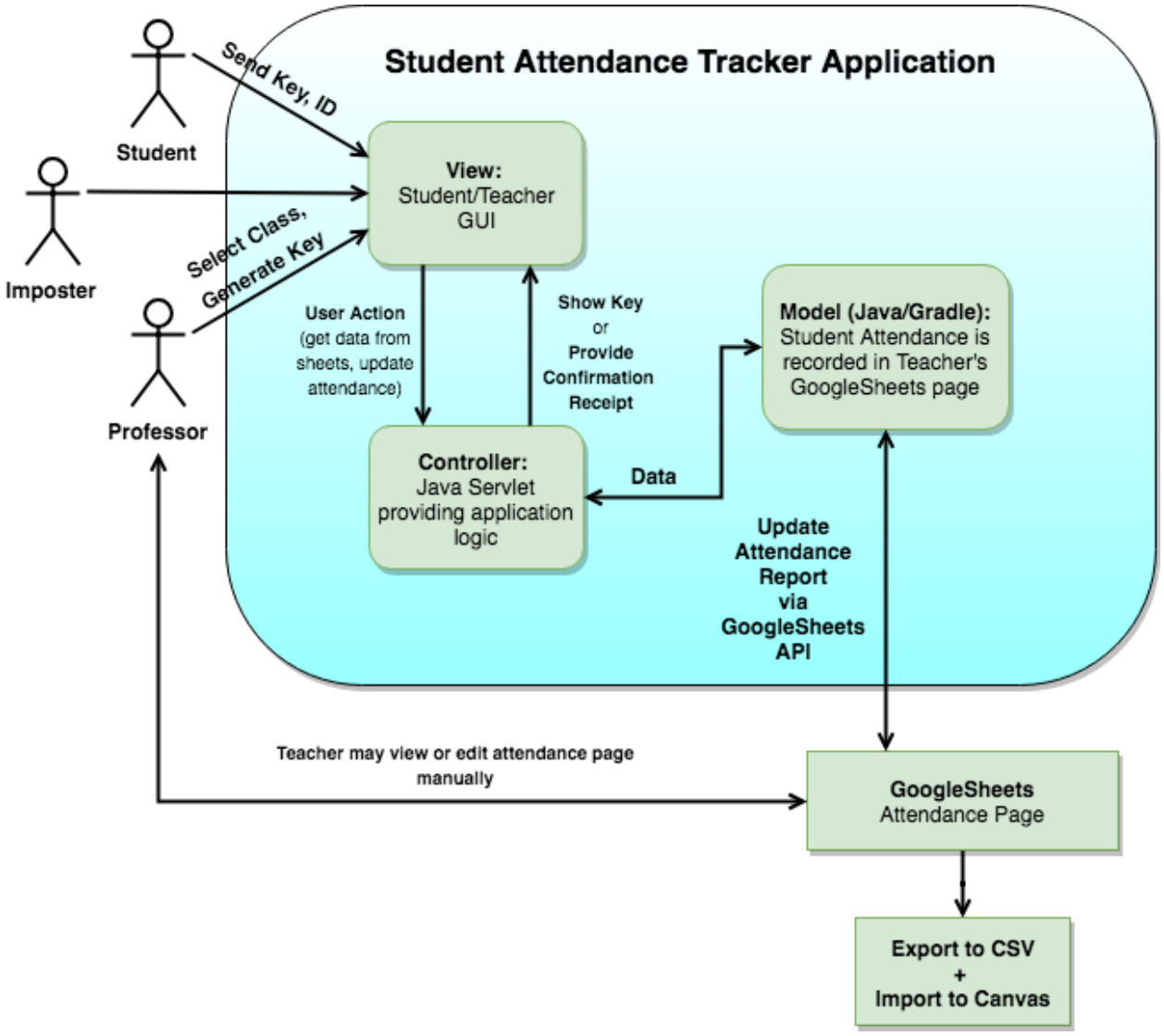


Figure 1: System Diagram for Student Attendance Tracker (Web App)

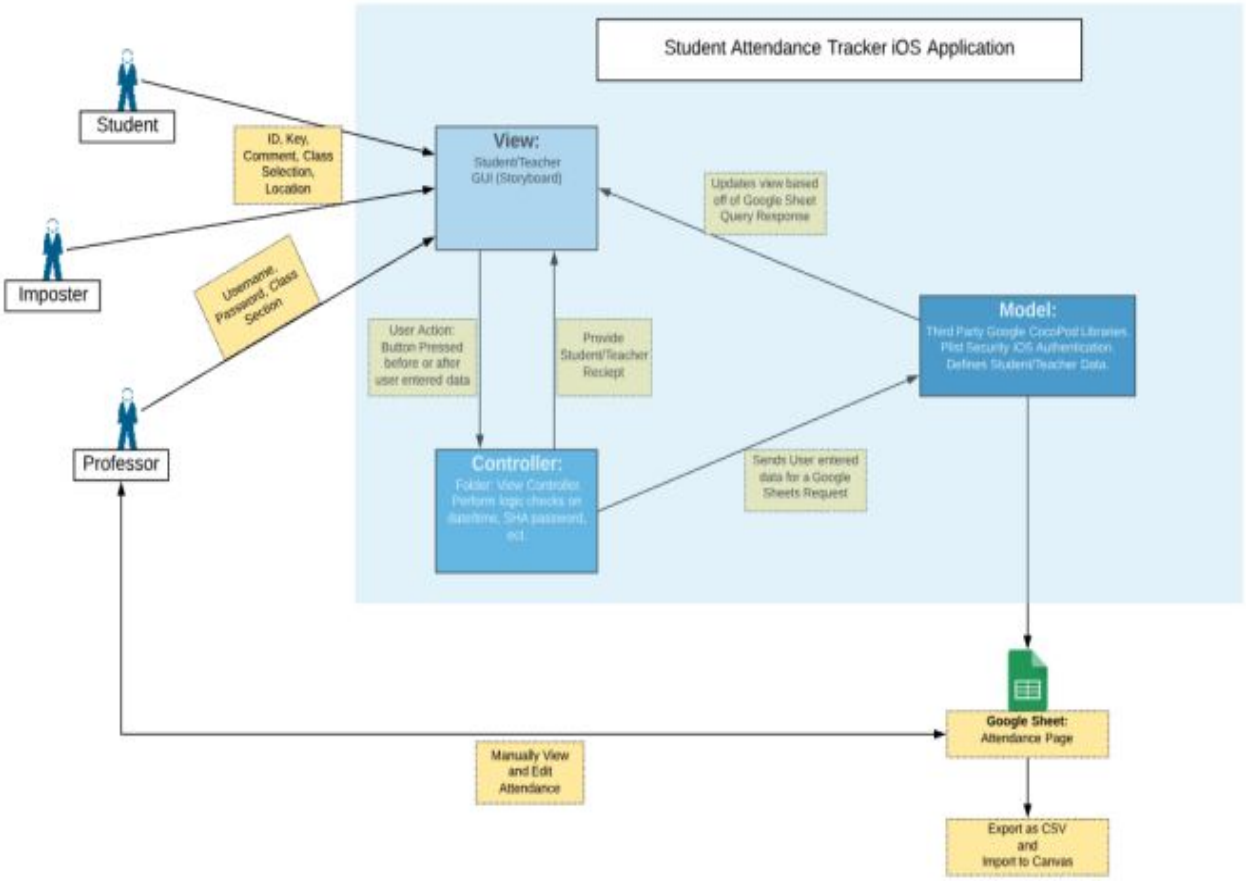


Figure 2: System Diagram for Student Attendance Tracker (iOS App)

Chapter 7 State Diagram

Pictured below is a diagram of our application flow for the Student interface. The user begins by accessing our application through either a web browser or mobile device. On the opening view, students are asked to enter a student ID and class key. Once the user has entered the requested information, a report is generated that either displays a unsuccessful submission of attendance in the case of invalid entry, or a receipt page displaying confirmation that their attendance has been marked in the Google Sheets.

Displayed in Figure 3 is the application flow for the teacher interface.

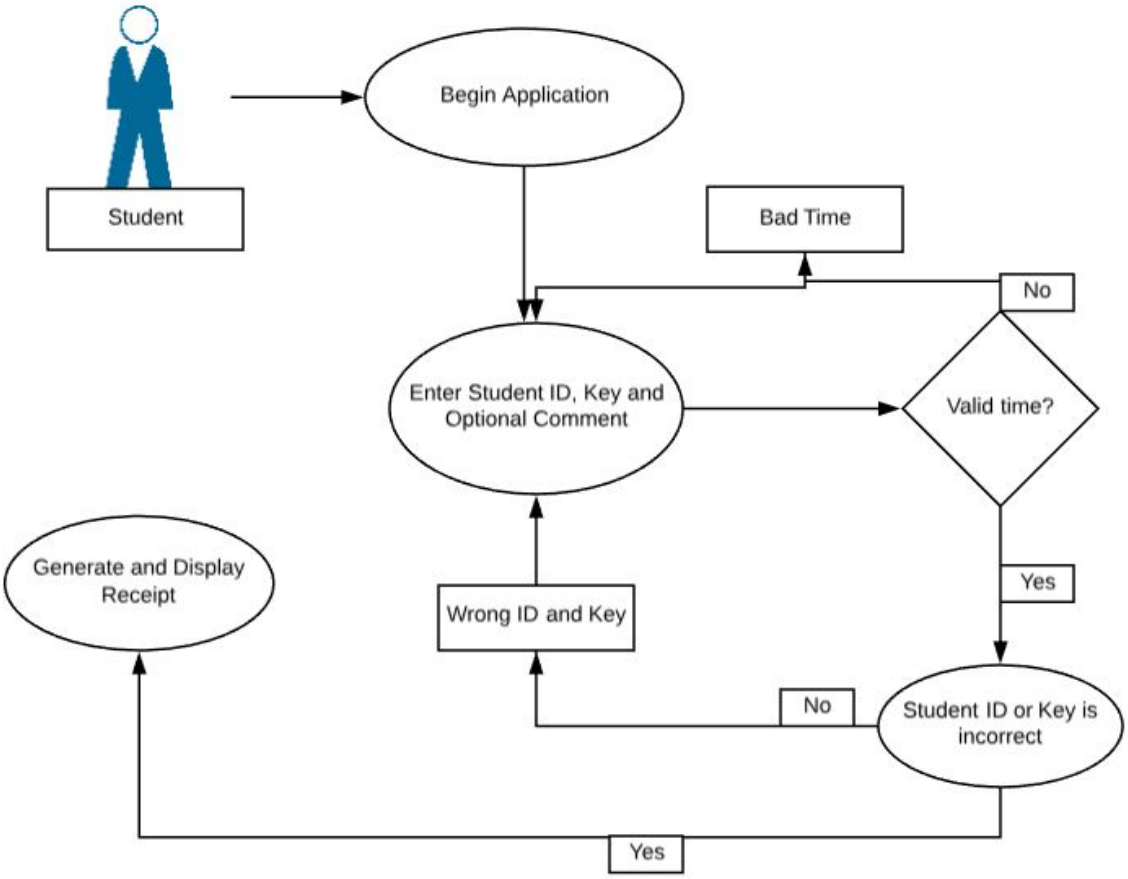


Figure 3: State Diagram for Student Attendance Tracker (Student)

For our next diagram the role of the teacher is represented. When the teacher accesses the app they will first have to enter their username, password, and select the class that they are going to take attendance for. If the class section already has attendance for that time they will be notified as such. If the teacher incorrectly entered their username and password they will be

notified in that way as well. If the submission is valid then the google sheet will be propagated with the current date, time and random key. During this operation the teacher will be notified that they have successfully signed in and will be given a copy of their key along with the amount of time that the students have left to sign in.

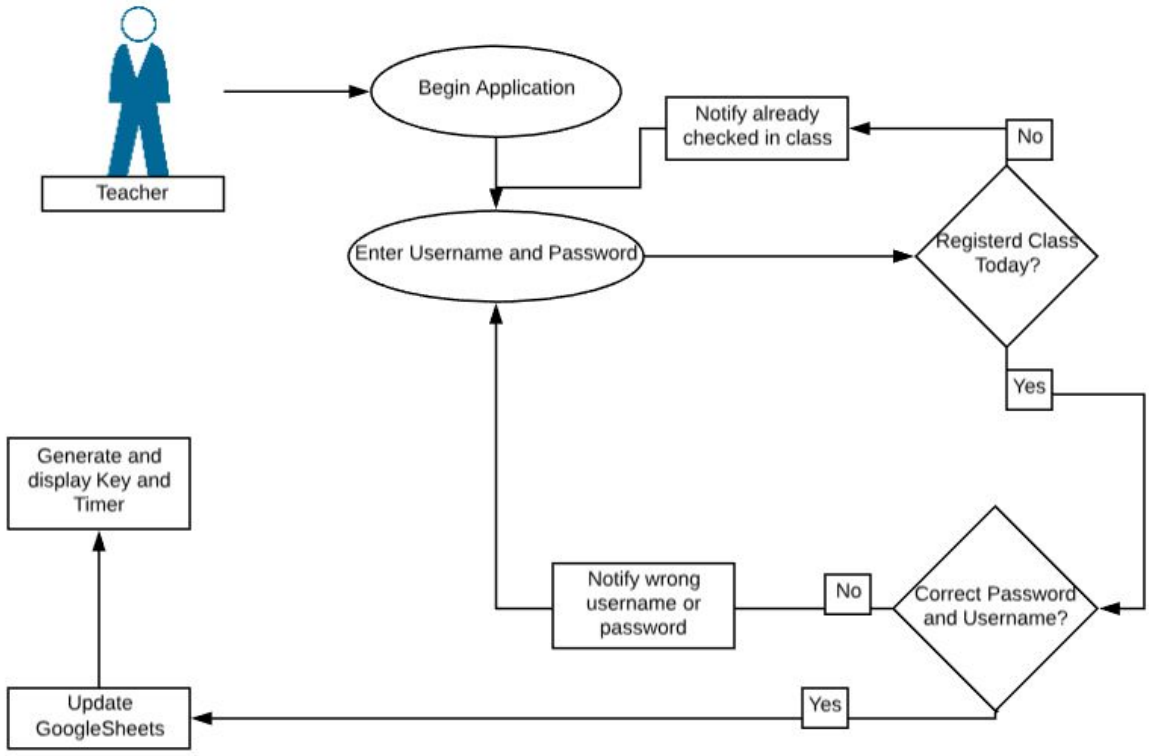


Figure 4: State Diagram for Student Attendance Tracker (Teacher)

Chapter 8 Entity Relationship Diagram

The Entity Relationship model is an abstract data model that defines an information structure which can be implemented in a database. Entities have relationships with other entities in a database. They are generally drawn as boxes (entity) that are connected by lines, often with special signifiers on each side to show the type of relationship (one-to-one, one-to-many, or many-to-many). The student attendance tracker application has several key elements to this data model. A student has many attendance records, and may be enrolled in many sections. A course may have many sections, and each section may only have one teacher. However, a teacher may teach many sections. This is helpful for sorting through relevant fields in the database.

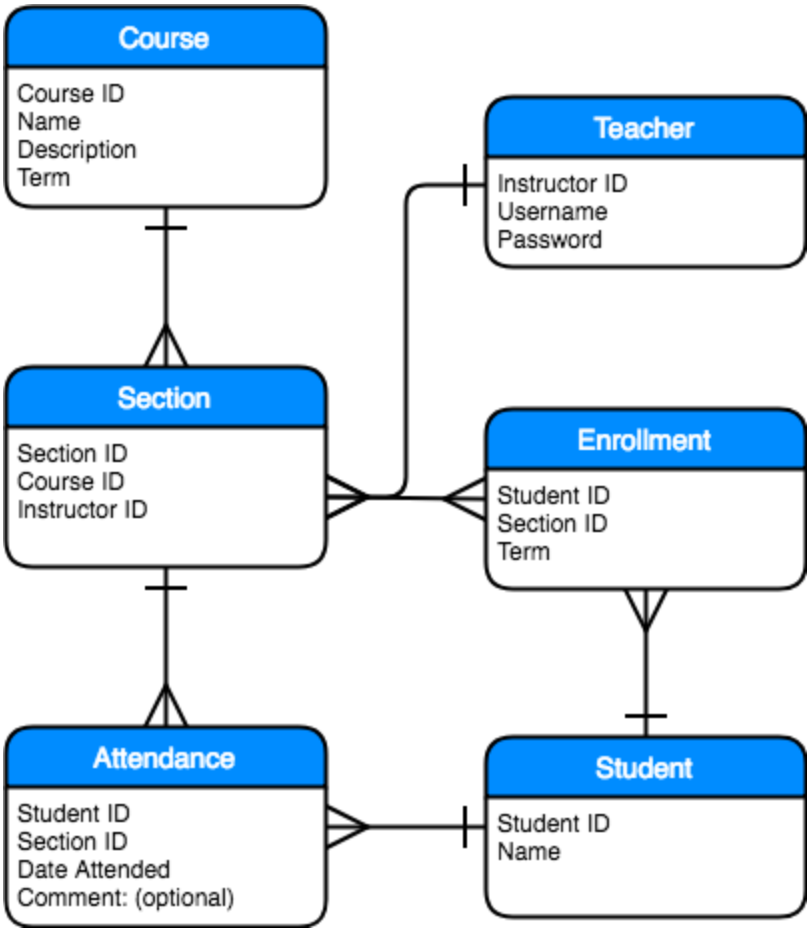


Figure 5: Entity Relationship Diagram for Student Attendance Tracker

Chapter 9 Functional Requirement Specification & UseCase Diagrams

Actors: An actor specifies a role played by a user or any other system that interacts with the subject. There are primary actors and secondary actors or supporting actors. During our use case, both students and the teacher will be a primary actor. Our supporting actors will be the external system that validates and exports data to the google sheet.

Stakeholders: Because of the nature of this project (being a class assignment for CSC131), the main stakeholders are the developers: Derrek Gass, Shayan Tom Amir, Pawan Chandra, Minaquan Li, and Aaron Miller. Another stakeholder is our professor, Doan Nguyen, who will be responsible for assessing our application.

Use Cases: Below are our use cases that identify, clarify, and organize system requirements. These use cases are generated from our product backlog, which is a mechanism used in Agile methodology that help us track our progress. The first figure is the use case diagram, which gives the reader a summary of our use cases. Then, we have a table with more details on what each use case entails.

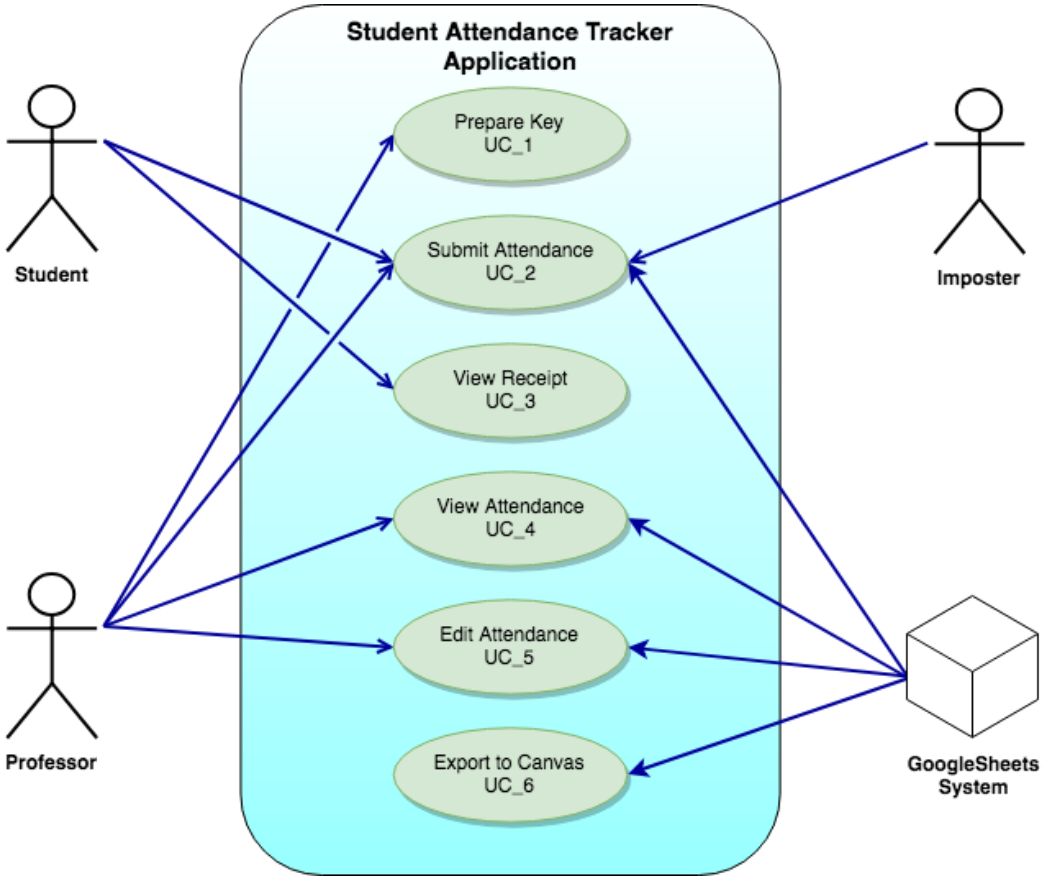


Figure 6: UseCase Diagram for Student Attendance Tracker

Use Case #	Use Case Name	Use Case Details
UC_1	Prepare Key	**<Actor>** Professor **<Goal>** To set a key for students to enter to prove they were in class. **<Pre Conditions>** Professor must login to confirm administrative role. **<Post Conditions>** Student inputs the key along with other necessary inputs and the attendance is recorded. **<Description>** The professor prepares a key that needs to be entered in order for students to submit a log of attendance. **<Sequencing Steps>** 1. Professor opens up the attendance application and logs in. 2. Professor selects the current course. 3. Application generates a random key. 4. Professor provides the key to the students to input to take attendance.
UC_2	Submit Attendance	**<Actors>** Student **<Goal>** To submit a attendance log for a class on a particular day. **<Pre-Condition>** Attendance key was provided to the students by the professor.

		<Post Condition> 1. Student gets sent to a confirmation page, acknowledging that the attendance has been recorded. 2. Online attendance Google Sheet has been updated. **<Description>** The student, provided they have a valid student ID and key, can log their attendance. **<Sequencing Steps>** 1. Student opens up the attendance tracking request webpage 2. Student inputs their ID and key and submits the attendance tracking request. 3. Student is taken to a confirmation page, acknowledging that the attendance has been recorded or rejected if invalid.
UC_3	View Receipt	**<Actor>** Student **<Goal>** To provide students with visual confirmation of successful submission of their attendance. **<Pre Conditions>** Student has logged attendance with a valid ID and key. **<Post Conditions>** Student inputs the key along with their ID and the attendance is recorded. **<Description>** Only the student can participate in this process. The student, if they have been successful (valid ID and key) are shown a receipt with details of their attendance log for that particular day after Google Sheets has been updated. **<Sequencing Steps>**

		1. Student successfully submits their attendance. 2. Receipt is shown.
UC_4	View Attendance	**<Actor>** Professor **<Goal>** To view the record of attendance for each class. **<Pre Conditions>** Professor must have access to the Google Sheet URL. **<Post Conditions>** The professor will be able to visually confirm attendance. **<Description>** The professor can manually view the attendance records in browser via direct URL access. **<Sequencing Steps>** 1. Professor enters Google Sheet URL into their browser. 2. Professor can view the attendance record in browser window.
UC_5	Edit Attendance	**<Actor>** Professor **<Goal>** To edit the record of attendance for each class **<Pre Conditions>** Professor must have access to the Google Sheet URL. **<Post Conditions>** The spreadsheet will display updated values and autosaved corrections. **<Description>** The professor can manually edit the attendance records in

		browser via direct URL access. **<Sequencing Steps>** 1. Professor enters Google Sheet URL into their browser. 2. Professor can manually edit the attendance record in browser window for special cases.
UC_6	Export to Canvas	**<Actor>** GoogleSheets **<Goal>** To export the Google Sheet attendance record as CSV and import into Canvas through API integration. **<Pre Conditions>** Google Sheets and Canvas are properly configured to communicate.. **<Post Conditions>** Teacher can view attendance grades on Canvas directly. **<Description>** GoogleSheets will export its contents to Canvas to view directly in the dashboard. **<Sequencing Steps>** 1. GoogleSheets exports CSV file. 2. Canvas imports data from CSV file. 3. Professor can view attendance record directly.

Table 4: Functional Requirement Specification of UseCase Diagrams for Student Attendance Tracker

Chapter 10 Sequence Diagram

The sequences diagrams found below provide what the system does in a chronological order for each the student and the teacher. The First diagram is for the student. Below we can see that the layers of the system that communicates with itself is comprised of the Attendance Form, Servlet, Google Sheet API, and Database. If the student properly submits there information the system will present them with a confirmation receipt that they may feel free to show to the teacher.

If the student does not properly enter the correct info they will be prompted in the appropriate fashion. In order to complete the process below they will have to try again and complete this correctly. The system will have to do this several time per class period depending on how many students have made it to class.

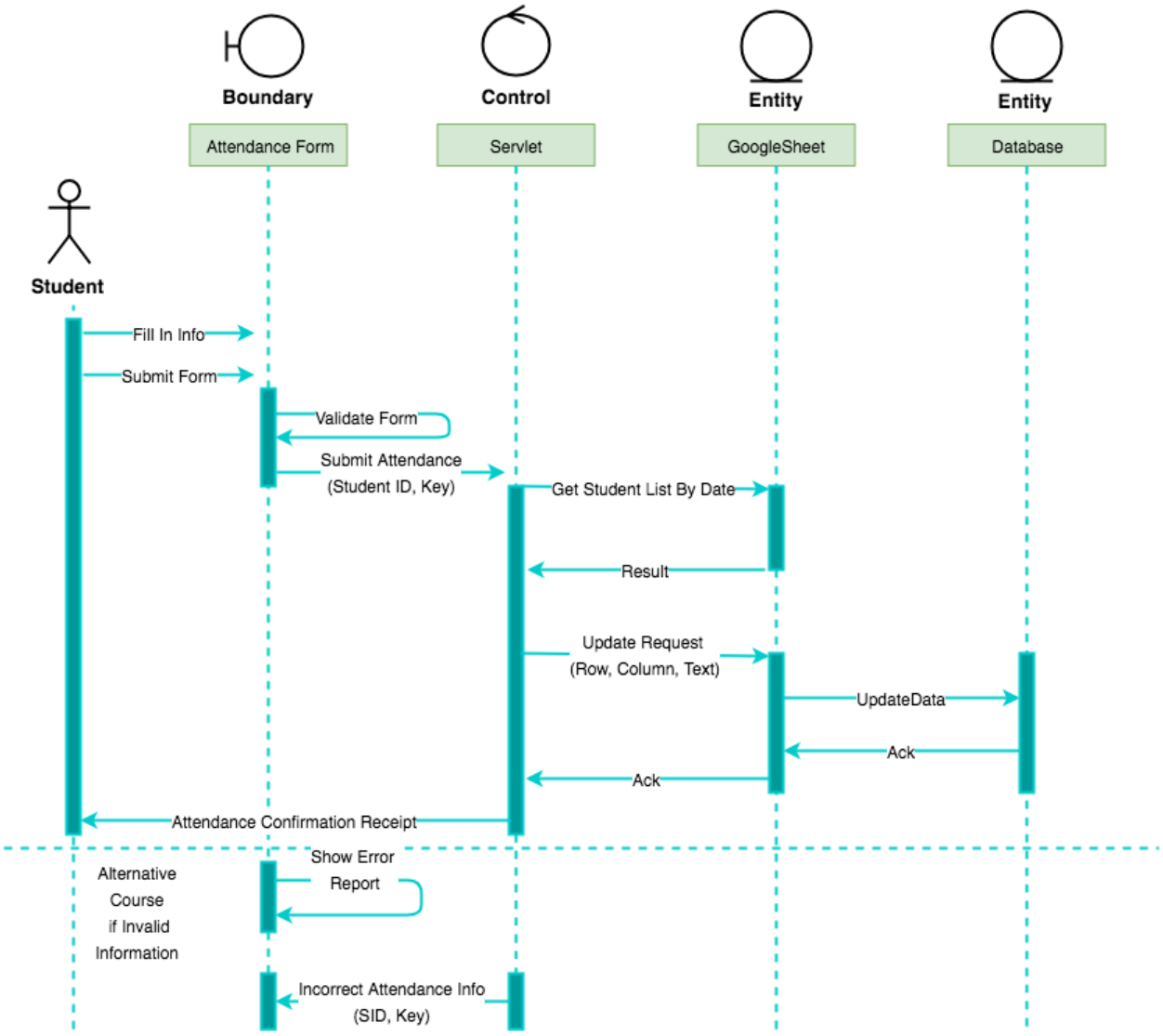


Figure 7: Student Sequence Diagram for Student Attendance Tracker

The sequence diagram for the teacher has lot of the same parts that were used for the

student. Here we can see again that if the teacher was to enter invalid credentials they would have to try again to validate their form. If they enter the correct information then they will be prompted with a receipt that provides them conformation they have completed the steps properly and that there students may now sign in.

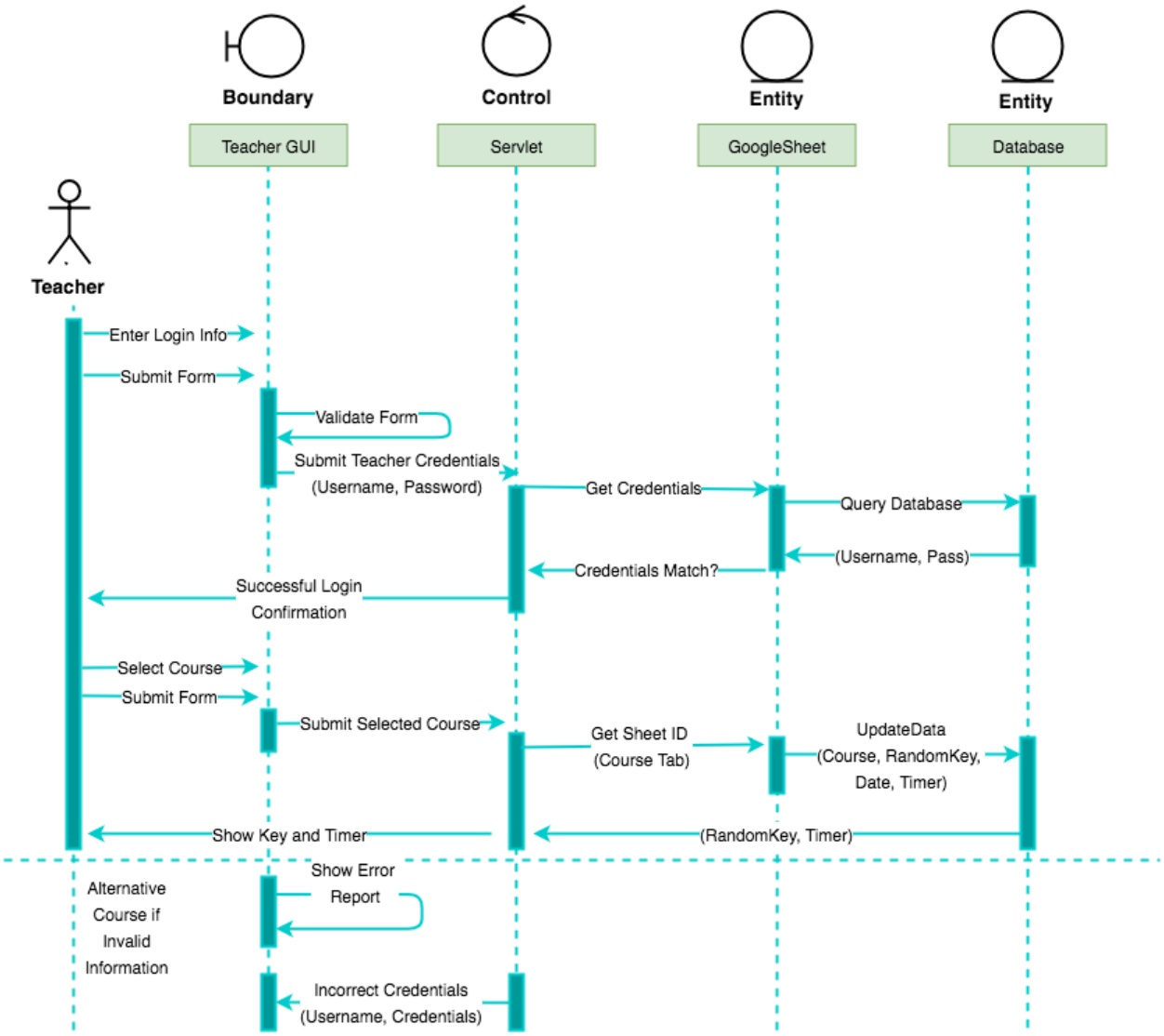


Figure 8: Teacher Sequence Diagram for Student Attendance Tracker

Chapter 11 Class Diagrams and Interface Specification

Our group built on top of the Google Sheets v4 API provided by Google. For the Website SheetsQuickstart was used in the java files, which was created by Software Engineers at Google. (example chapter 14). Directly beneath we may see the files for the Apps. First is the Web files. After that is the iOS files.

Web Files

- **QuickServlet.java** is a class that provides the application logic for our program. This both serves different updates to the view for the user interface, while interacting with SheetsQuickstart.java to communicate with the database.
- **SheetsQuickstart.java** is used to read and write data to and from the Google Sheets spreadsheet. This is essentially the model that enables QuickServlet.java to request updates to the spreadsheet.

iOS Files

- **HomeVC.swift** - This is the controller for the home page that tracks location and decides whether or not to present the student or professor login page.
- **ProfessorRecieptVC.swift -** Handles displaying the teacher's recipet for confirmation.
- **ProfessorSignInVC.swift -** Communicates with the 'ProfessorNetwork' and provides the logic checks to make sure that the professor's entered data is valid.
- **StudentRecieptVC.swift -** Presents the reciepts for the reciept for the student.
- **StudentSignIn.swift -** Handles logic for the student's entered data based off of what the 'StudentNetwork' class returns.
- **CellHelper.swift -** Helps the system find the cell to edit.
- **ProfessorNetwork.swift -** Handles the backend operations for the professor.
- **StudentNetwork.swift -** Handles the backend operations for the students.
- **MapTracker.swift -** Takes cares of tracking the student and implements the delegates needed to track students.
- **Reachability.swift -** Decides if the network is reachable or not.
- **ShowAlert.swift -** Gives the custom iOS alert and saves about 5 lines of code every time it was used.

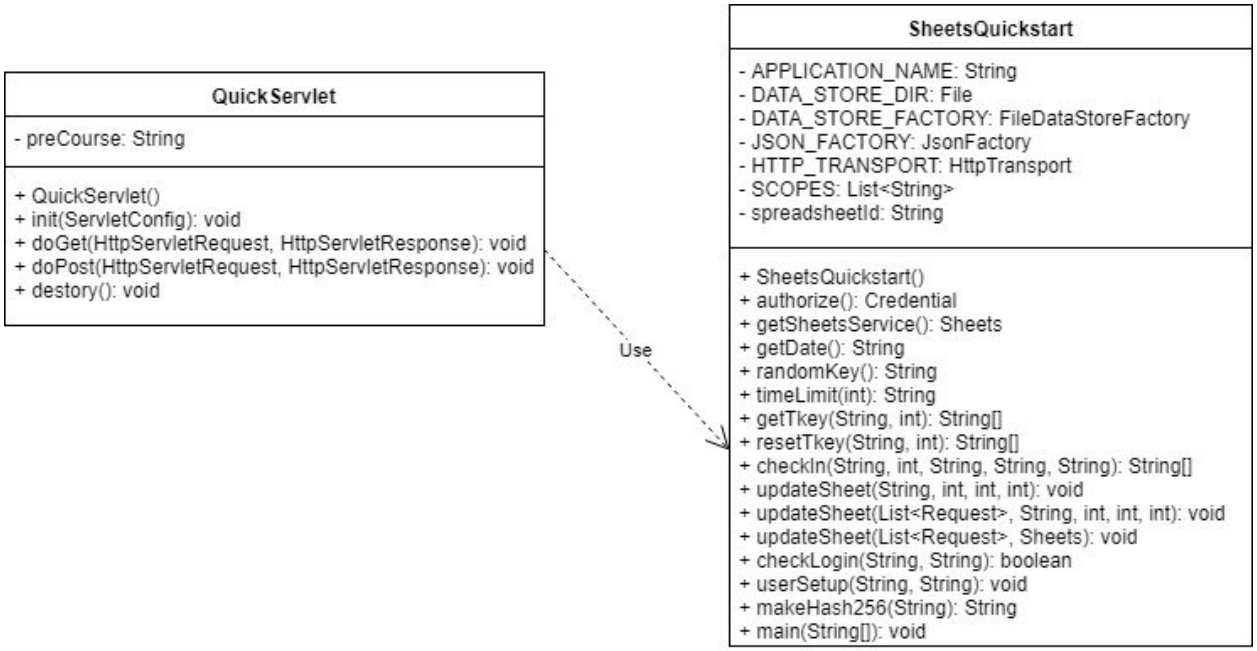


Figure 9: Class Diagram for Student Attendance Tracker (Web App)

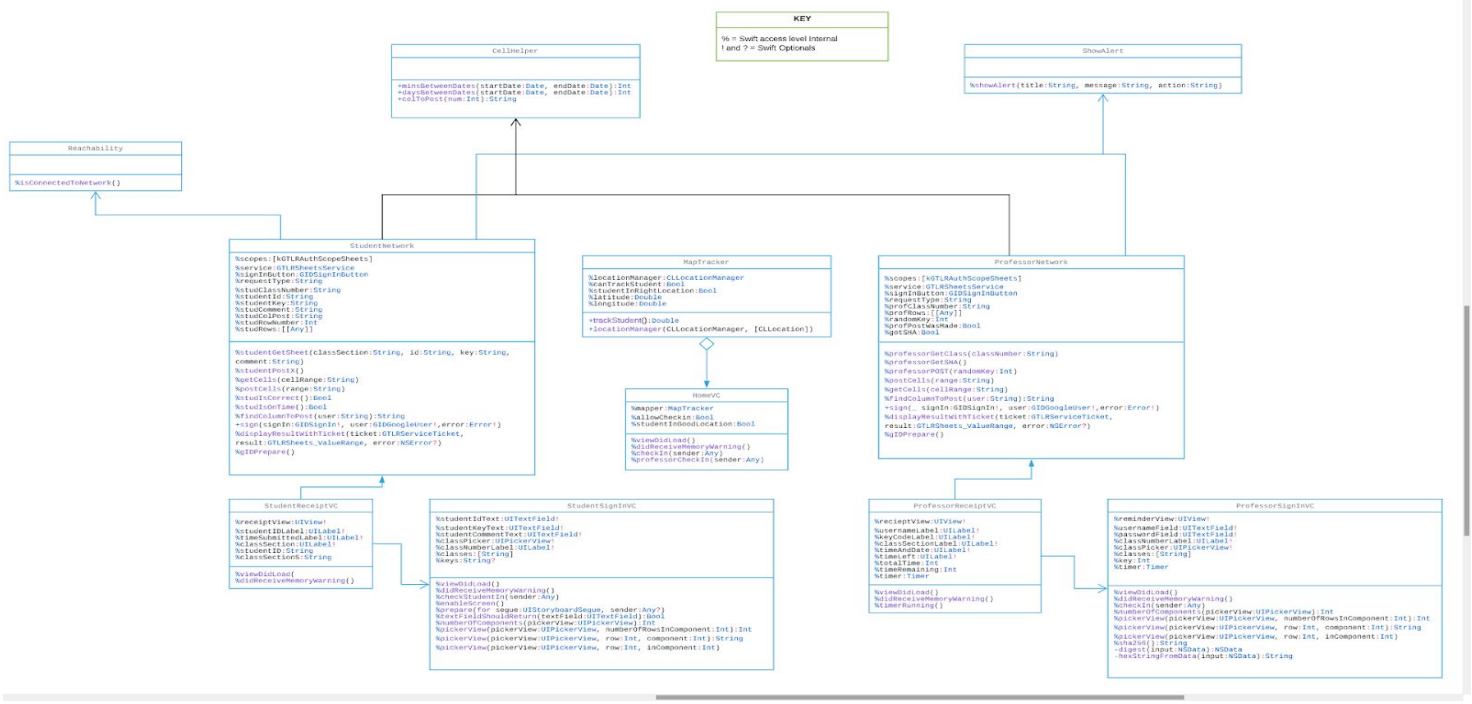


Figure 10: Class Diagram for Student Attendance Tracker (iOS App)

Chapter 12 Traceability matrix

Below is a traceability matrix, which shows the the relationship between the use cases and the requirements:

	UC_1: Prepare Key	**UC_2:** Submit Attendance	**UC_2:** View Receipt	**UC_4:** View Attendance	**UC_5:** Edit Attendance	**UC_6:** Export to Canvas
REQ_1: Student ID/Key/ Comment Log			**X**	**X**		
REQ_2: Random Key	**X**					
REQ_3: Verify ID and Key		**X**				
REQ_4: Confirmation Receipt			**X**			
REQ_5: Reject Intruder		**X**				
REQ_6: Edit Report Manually				**X**	**X**	
REQ_7: View Report				**X**	**X**	**X**
REQ_8: One Login Per Student		**X**				
REQ_9: Time Limit		**X**				
REQ_10: Proximity		**X**				

Table 5: Traceability Matrix for Student Attendance Tracker

Chapter 13 Agile Methodology

Given the time frame and the complexity of the application, we had decided to use Agile Methodology. There are four major principles in the Agile Methodology:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration
- Responding to change over following a plan

Based on these principles, we were able to successfully adjust and meet the set desired goals. We also took the incremental and iterative approach to keep better track of time. Incremental process is one in which software is built and delivered in pieces. Each piece, or increment, represents a complete subset of functionality. An iterative process is one that makes progress through successive refinement.

13.1 Iteration/Incremental Planning

Iteration #	Task Description	Deliverable	Retrospective	Iteration Manager
Iteration # 0	Be able to execute the sample code provided by the Google Sheet Quick Start guide. Set up our own Google sheet and create json file.	Project Proposal Writing data to our Google Sheet	**Set Up**	Pawan
Iteration # 1	Set up Apache Tomcat and prepare servlet to pass data to back-end. Use JavaServer Pages to display and pass data to the QuickStart	Front end and back end communication. Key generator Comment section for student	**REQ_1:** Student ID/Key/ Comment Log **REQ_2:** Random Key	Derrek
Iteration # 2	Validate professor and student credentials by reading from Google	Course Selection (drop down bar)	**REQ_3:** Verify ID and Key	Min

	sheets.	Display and validate random key. Insert date Check timer. Student ID validation	**REQ_8:** One Login Per Student **REQ_4:** Confirmation Receipt	
Iteration # 3	Add timer, manual entry, and receipt functions.	Students will not be able to take attendance after 15 minutes. Professor is able to manually change date, key, or attendance through Google Sheets	**REQ_5:** Reject Intruder **REQ_6:** Edit Report Manually **REQ_7:** View Report **REQ_9:** Time Limit	Tom
Iteration # 4	Geo-Location(iOS)	Determine the students latitude and longitude.	**REQ_10:** Proximity	Aaron

Chapter 14

Existing Servlet Application

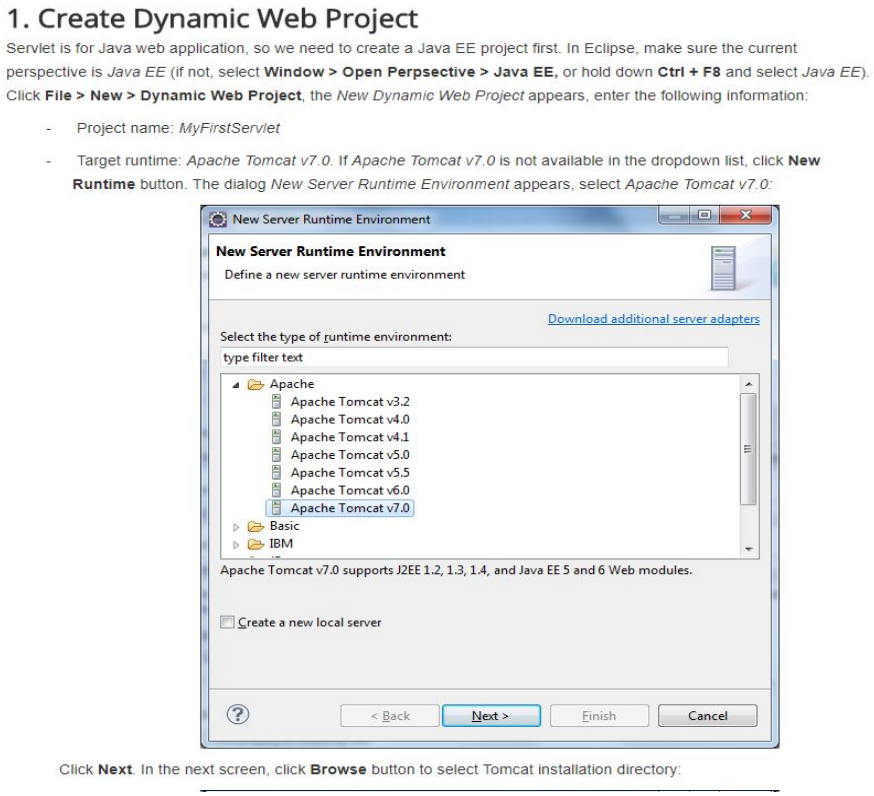
In order to get a rough Idea on how to set up a web application, we used the quick start guide provided by Google in the Google Sheet API setup page. There were many languages choose from such as Java, Ruby, Python etc., each with a specific documentation on how to get started. After selecting Java, there were just 3 steps to get the project started"

- Create Credentials
- Set up Gradle builder
- Use sample code to check for error.

Creating and setting up credentials allowed us to implement the Google Sheet API from Eclipse. To prepare the project, the quick start also documented how to set up Gradle. Gradle is

an open-source build automation system that builds upon the concepts of Apache Ant and Apache Maven. After setting up gradle, Google provided us with a sample code to test for functionality. The code was very similar to the application we are developing which made it easy to manipulate and use it as a starting point.

To deploy the Java Servlet and JavaServer Pages(JSP), we used Apache Tomcat 7.0. Codejava.net has a very well detailed guide on how to set up Tomcat with Eclipse. The guide also gave examples on how JSP files implement Java and what the XML page controls. Like the Google Sheet quick start, there was also a sample code provided to test out the implementation of Apache Tomcat. Here is a screenshot of the guide.



Chapter 15

External Interface

In this section we describe key interfaces used to initiate the servlet application, the Google Sheet API and Apache Tomcat. The Google Sheet API has a well written quickstart guide that provided instructions on how to set up builders(Gradle) and the specific client.JSON file. The API also had a dedicated forum with helpful links to specific methods used in the application. Apache Tomcat

15.1 Spreadsheet API

Using the SpreadSheet API v4 we also used a Google authentication API inside of the iOS application. The amount of documentation provided for the Java Servlet was very helpful and really speeds up the development process. There were examples for all sorts of API requests that represented POST, PUT, GET. The was also a way to to a 'batchUpdate' where multiple sheets can be updated in one request. Unfortunately our team did not get the time to implement this convenient feature. We found that using any web server language that it supported or using Android Java provided lots of helpful documentation.

For Aaron developing the iOS application was a lot of work for one man and unfortunately Google did not provide any documentation for Swift. Thankfully the Quickstart showed me how to GET from the Spreadsheet. But It took me almost two full entire 8+ hours days of trial and error. One of the reason this took so long was because google claims that if you use Oauth 2.0 correctly you can send a basic network request filled with JSON using the Swift 4 standard library. He did this but the response from Google's server still told him he had invalid credentials. After a late night of trial and error he decided to move on. This led to him creating reading the general documentation provided by Google. Since there was no stack overflow telling him how to POST he decided to make one here:

https://stackoverflow.com/questions/51015176/how-can-i-post-a-value-for-the-google-sheets-api-v4-in-swift-3-and-4/51015806#51015806

So that others may benefit from his work.

It is easy to setup but worth noting that Scopes must be provided by the developer to the SpreadSheet v4 API. By using the Google sheet as a database. This means that when the App is first used be it Web or iOS someone can sign in to google to provide credentials and make for better authentication.

This Google SpreadSheet API is a very useful tool and database that Google provides to the public for free. Many times this would cost businesses lots of time and money to have a server host their database at all times all around the world with high speed. Also for other branches of business that need to use spreadsheets are major benefactors of this.

Chapter 16

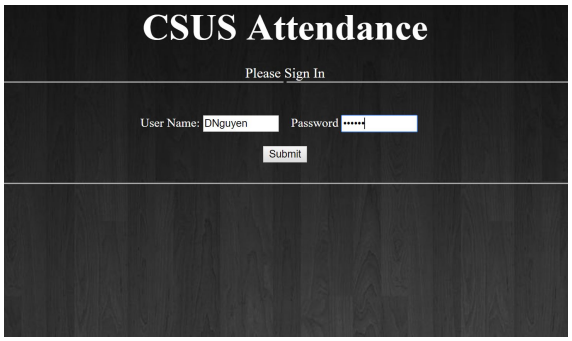
User Interface Design & Implementation

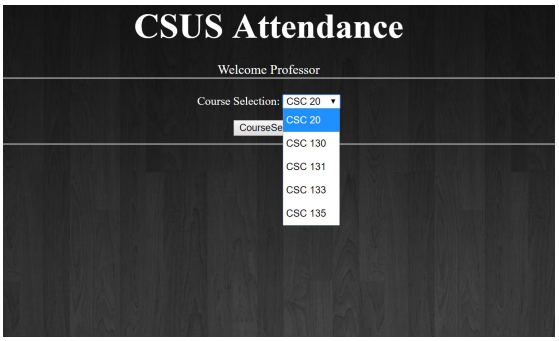
The user interface design and implementation details for the web application are discussed below. We implemented our Attendance Tracking app using Google Spreadsheet REST API as the database..

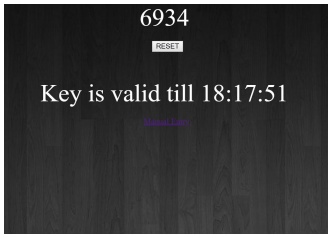
16.1 Detailed Screenshots of UI

In this section, we have collected the screenshots of our web application. We have captured all the screenshots for the teacher UI and the student UI below.

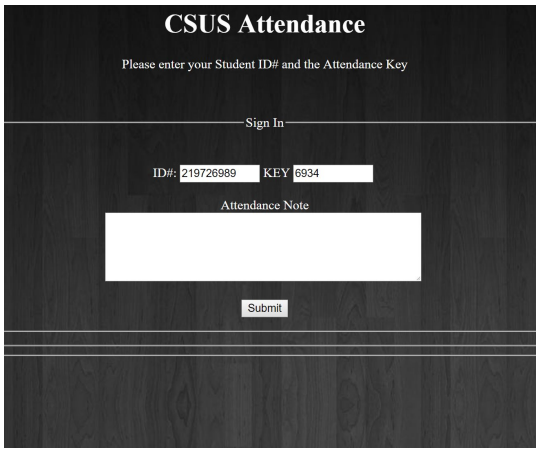
Teacher's Web UI:

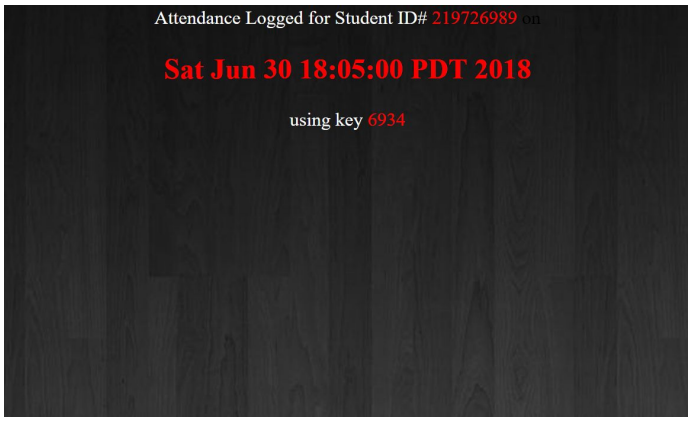




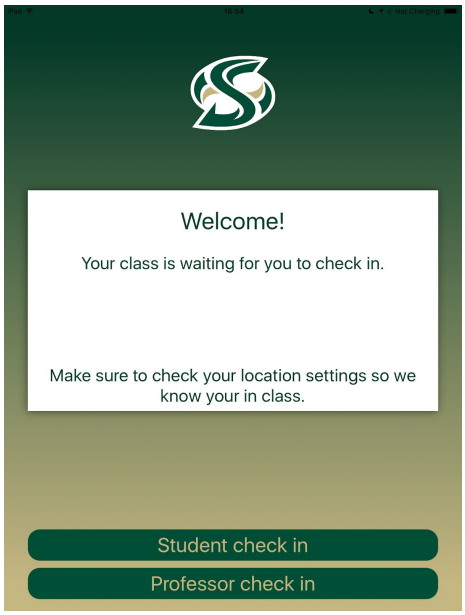


Student's Web UI:

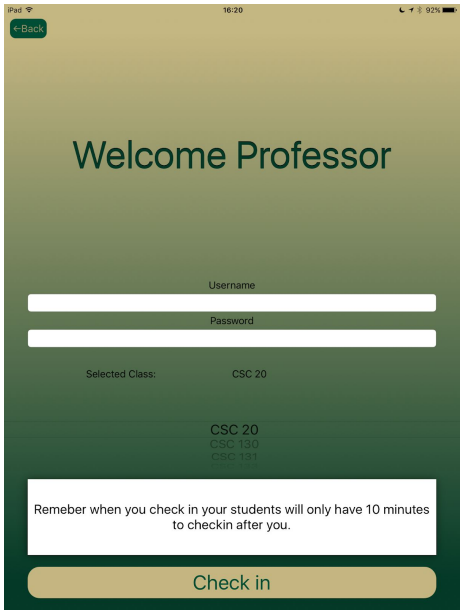


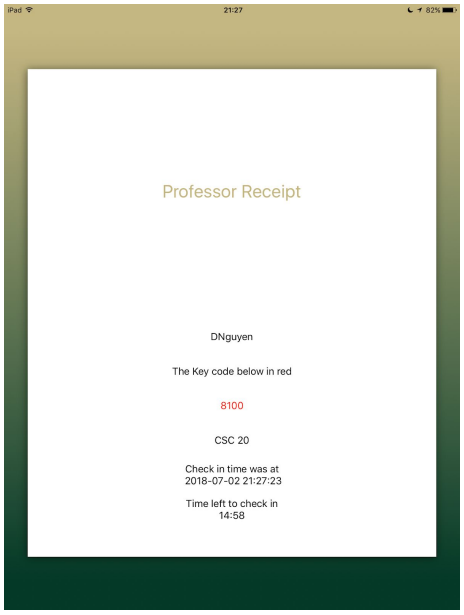


Home Screen iOS:

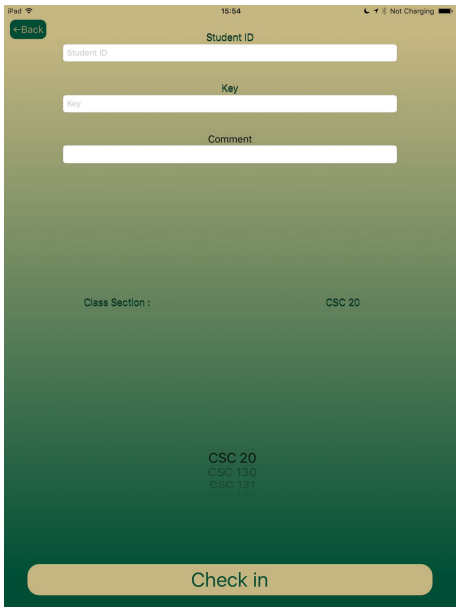


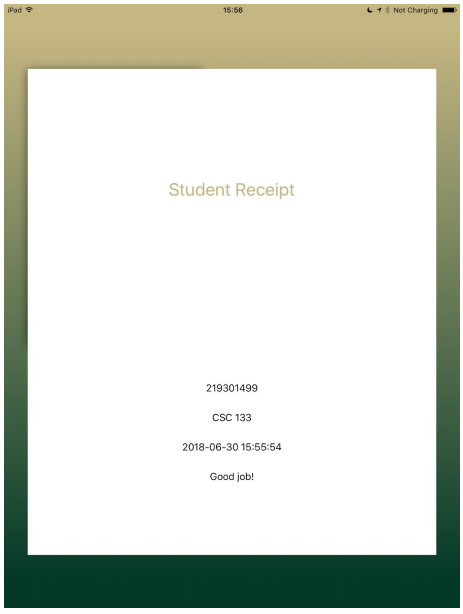
Teacher's iOS UI:





Student's iOS UI:





Teacher's Login Screen and Student's Attendance Screen

The login page will check for the teacher's valid login credential. It will say invalid

credential if the wrong username or password was entered. The teacher has to login in and then select a course. And then a random key will be generated for today's date. If any student tries to take attendance before the key and timer is generated, the student's page will reject the ID and key entry. The students will be given an URL in the beginning of the semester for the class to take their attendance with their student ID and key for when the key is valid. The student's UI will also check for valid student ID and key entry. It will say invalid credential if the ID does not match any of the ones from the Google Sheets. If the student were able to take the attendance, the web application will then display a message saying attendance logged.

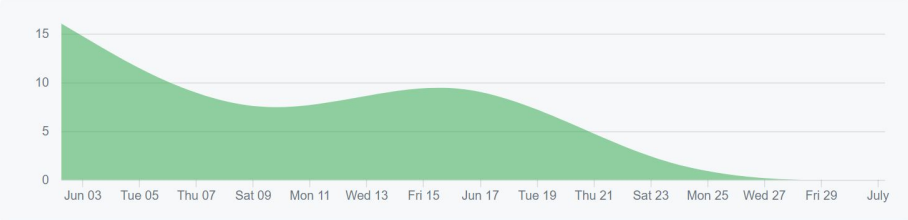
16.2 Implementation Details

The web application is being handled by the Servlet controller which uses the Google Spreadsheet API. The API allows the web application to use the online spreadsheet to store and manage data. It is easy to use and quick to set up. We used a quick-start template from Google to setup the basic read and write and then we go from there.

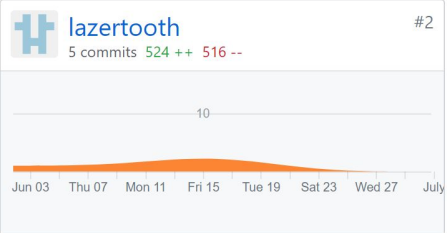
16.3 Github Network Graph

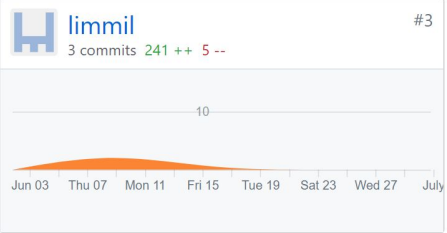
We used the very well known version control Github to maintain our code. Github definitely offers a lot of cool features for working as a group. However, most of us did not know how to use Github, and everything became messy in the original repository. Somehow, we ended up using p-chandra's Github account to make changes to the code.

Here is a graphical representation of our code contribution at Github, but we were using one account for most of the work.


15
10
5
0
Jun 03 Tue 05 Thu 07 Sat 09 Mon 11 Wed 13 Fri 15 Jun 17 Tue 19 Thu 21 Sat 23 Mon 25 Wed 27 Fri 29 July


p-chandra
#1
25 commits 3,385 ++ 2,196 --
10
Jun 03 Thu 07 Mon 11 Fri 15 Tue 19 Sat 23 Wed 27 July


lazertooth
#2
5 commits 524 ++ 516 --
10
Jun 03 Thu 07 Mon 11 Fri 15 Tue 19 Sat 23 Wed 27 July


limmil
#3
3 commits 241 ++ 5 --
10
Jun 03 Thu 07 Mon 11 Fri 15 Tue 19 Sat 23 Wed 27 July


8
6
4
2
0
May 27 Tue 29 Thu 31 June Jun 03 Tue 05 Thu 07 Sat 09 Mon 11 Wed 13 Fri 15 Jun 17 Tue 19 Thu 21 Sat 23 Mon 25 Wed 27 Fri 29 July


StrawHatAaron
#1
13 commits 291,808 ++ 228,699 --
5
May 27 Thu 31 Jun 03 Thu 07 Mon 11 Fri 15 Tue 19 Sat 23 Wed 27 July

Chapter 17 - Testing

There were many test cases implemented within each method and making sure that we minimized coupling and maximize cohesion. We also had an alpha release to test for bugs and and check for additional functionality that should be developed. The beta release was mostly for code refactoring and documentation. More information on testing provided below.

17.1 Data Validation

Besides course selection, every input the user provides is a string or a integer casted to string. The cells in the Google sheets are a two-dimensional array of objects and because string in Java string is also an object, the validation process wasn't too difficult. To test for teacher credentials, such as username and password, came down to a simple .equals method called from the string library and multiple if else condition statements. If either the username or the password in the teacher login page did not match our database, the login page will display "Invalid Credentials". By comparing two strings, we eliminated empty field, misspelling, and random character test cases. String comparison must be identical which also eliminates case sensitivity. In the backend, we implemented many print statements as flags indicating which methods were executed. This helped with data validation, especially in our alpha release since we were able to narrow down exactly where the issues occured. The student validation process was done the same way. If the student entered incorrect key or id, the system will detect it and display invalid credentials.

Test #	Description	Steps	Expected Output	Actual Output	Pass/Fail
1	Enter empty test case for teacher login.	At the login page, leave the fields empty and select submit.	A new page will open up notifying user that the credentials enter is invalid	Same as that of Expected Output	**PASS**
2	Enter invalid credentials	At the login page, enter either a wrong password or username and select submit	A new page will open up notifying user that the credentials enter is invalid	Same as that of Expected Output	**PASS**

17.2 Usability Testing Test Report

We have tested our application's usability. In this type of testing we see for usability issues encountered while performing use cases with the running application.

Following is the detail Test Report for Usability Testing.

1. Testing work that has been done:

- Usability testing for all components of Student Attendance Tracker.

2. System Under Test:

- Student Attendance Tracker

3. Description of the system under test:

- Student Attendance Tracker is a web application where a teacher can select their class and generate a random key. Then students can log their attendance by submitting their student ID and the provided key within the time limit. The system will log this information automatically and provide the student receipt of successful submission of attendance.

4. The application includes following components:

- Select Class, Generate Key
- Enter Student ID, Key
- Log Attendance

Testing Summary for Usability Testing

The student attendance tracker is designed to be user friendly with a minimalist mindset displaying only what is necessary. The application is accessible through any browser and iOS devices.

Features Tested:

- **Teacher Login:** UI display issues, Usability, Ease of use.
- **Random Key**: UI display issues, Usability, Ease of use.
- **Student Login:** UI display issues, Usability, Ease of use.

Exit Criteria

Exit criterion is used to determine whether a given test activity has been completed or not.

Testing Requirements before sign off:

- All planned test cases have been executed
- Testing should be performed on applicable browser available
- All functionalities have been tested thoroughly from functionality and usability perspective
- Test Results have been documented thoroughly with steps to reproduce and detailed issue documentation.

Detail usability report (with test cases) is as below.

Usability Testing:

Guidelines to read Usability Test Cases

Column Header	Implication
Component	Name of the Component
Feature	Identify each feature/function
Test Case Procedure	Steps for Test Case
Usability Issue	Result of the Testing
Web Browser	Browser: Modilla_Firefox/Google_Chrome/Safari_iOS
Recommendation	Additional comments on the execution
Usability Issue Severity	Severity Ranges: 0 - no severity, 5- high severity
Test PASS/FAIL	Information about test results, whether passed of failed.

Usability Test Cases

Test-Case #UT_1	
Component	Generate Key
Feature	Create a random key generator (1000-9999)
Test Case Procedure	Professor logs in Select course section Random key generated and stored in database Display key to students
Usability Issue	None
Web Browser	Chrome
Recommendation	None
Usability Issue Severity	0
Test PASS/FAIL	**PASS**

Test-Case #UT_2	
Component	Student Attendance Receipt
Feature	Creates a receipt with username, key, and date of when attendance was logged.
Test Case Procedure	Student enters ID Student enters key provided by the professor Enter a comment if needed and select submit
Usability Issue	None
Web Browser	Chrome
Recommendation	None

Usability Issue Severity	0
Test PASS/FAIL	**PASS**

17.3 Functionality Testing Test Report

We have tested our application's functionality based on the use cases and focusing on the requirements of the client. The test should verify if the program is functional and make sure the expected output matches the specification of the clientele.

Following is the detail Test Report for Functionality Testing.

1. Testing work that has been done:

➢ Functionality testing of all components of Attendance tracker.

2. System Under Test:

Student Attendance Tracker

3. Description of the system under test:

The system shall validate user input and perform all the correct methods to display the necessary output. Checking for empty cell in the database, validating student ID, and marking the correct tab based on course selection

4. The application includes following components:

➢ **Enter Student ID**

➢ **Enter Key**

➢ **Generate Receipt**

Testing Summary for Functionality Testing

The Student Attendance Tracker must be efficient and fast with the least amount API calls and coupling. Validation must be precise and tested for multiple cases.

Features Tested:

- **Teacher Login:** Validation, Key Generator, API Calling, Match Expected vs Actual results.
- **Random Key**: Validation, Key Generator, API Calling, Match Expected vs Actual results.
- **Student Login:** Validation, Key Generator, API Calling, Match Expected vs Actual results.

Exit Criteria

Exit criterion is used to determine whether a given test activity has been completed or not.

Testing Requirements before sign off:

- All planned test cases have been executed
- Testing should be performed on applicable browser available
- All functionalities have been tested thoroughly from functionality and usability perspective
- Test Results have been documented thoroughly with steps to reproduce and detailed issue documentation.

Functionality Testing:

Guidelines to read Functionality Test Cases

Column Header	Implication
Component	Name of the Component
Feature	Identify each feature/function
Test Case Procedure	Steps for Test Case
Test Case Results	Results - Actual
Expected Results	Results - Expected
Web Browser	Browser: Mozilla_Firefox/Google_Chrome/Safari_iOS
Defect Found (YES/NO)	Additional comments on the execution
Severity	Severity Ranges: 0 - no severity, 5- high severity
Test PASS/FAIL	Information about test results, whether passed of failed.

Functionality Test Cases

Test-Case #FT_1	
Component	Course Selection
Feature	Drop down bar that lists professors course section. Each course has independent data and tab within the database Upon selection, key, timer, and date are printed onto that specific course tab within the database.
Test Case Procedure	Select any one of the course(s) available. Check tabs for date and key entry. Match Key to the front end.
Test Case Results	Correct tab is located and the key is printed in the designated cell. Key is displayed to the student for use.
Expected Results	Same as Test Case Results
Web Browser	Chrome
Defect Found (YES/NO)	NO
Severity	0
PASS/FAIL	**PASS**

Chapter 18 History of Work & Current Status

History of Work: In order to get an idea of how we separated and completed the work for this project, please see the following sections:

➢ Section 13.1: Iteration Planning will give you an overview of what we accomplished in each iteration

➢ Section 16.3: Github Network Graph will give you an idea of the code commits and merges that were done during our development

Current Status: The current status of of work is as follows:

➢ Completed development of all the Functional Requirements

➢ Completed Functional Testing on All Components of the application

(Mozilla Firefox, Google Chrome, iPhone 6 Safari)

➢ Completed Usability Testing on All Components of the application

(Mozilla Firefox, iPhone 4s Safari browsers)

Chapter 19

Future Steps OR Extensions to our work

Servlet:

Currently, our application generates a report in the form of a google spreadsheet. A possible extension would be:

- Integration with Canvas for professors to have the generated .csv file exported to Canvas

Currently, our application verifies the student using his/her ID and the given key. An additional security feature would be:

- Geolocation verification to ensure that only a student within a 20 foot radius of the classroom is allowed to log attendance

Currently, our application only provides the teacher with a running total of the each student's attendance. An additional feature for the future in this regard would be:

- A grade percentage for each student based on the the current attendance score, along with a warning feature to notify teachers of students who have missed a set

amount of class (+3) or have missed a consecutive number of class sessions.

iOS:

Currently, our application generates a report in the form of a google spreadsheet. A possible extension would be:

- Integration with Canvas for professors to have the generated .csv file exported to Canvas

Currently, our application supports iOS version 10+ . We would like to expand on this by providing:

- An Android version to provide this application to a much larger user base.

Currently, our application is in English (US) and purely in written form. A possible future addition would be:

- Several additional languages, as well as accessibility options for those poor of vision or having color blindness.

Chapter 20

Lessons Learned

With the development of our attendance tracker project, our team gained significant experience in the software development process, more specifically the:

- **Agile Methodology:**

We chose the agile methodology in developing our software. Because of the time constraint (6 weeks), we felt that this was the most approach this project. We frequently interviewed the customer to address specific issues and make sure that each requirement is being met. The team focused less on documentation, and more on the implementation of the software. Every week, approximately, was one iteration in which all of the team members worked concurrently to gain simultaneous knowledge of the code.

- **Documentation experience (UML, Use Case Diagrams, System Diagrams, Class Diagrams):**

Despite following the agile methodology, this project was documentation intensive and served as a great learning tool for high level documentation and how it helps the development process as a whole.

- **Servlet application using Java:**

This was our team's first time creating a servlet web application using Java 8.0.

- **Google REST API: Spreadsheet API, Drive API**

We implemented our attendance application using Google Drive REST API and Google Spreadsheet REST API.

- **Git Version Control:**

While we did not exclusively use Git Version Control throughout the development process, we many of our team members gained knowledge regarding Git using Github, in order to speed up the development process and provide a smooth workflow between team members.