# **Software Development Plan**

60 Seconds

**REVISION HISTORY**

| Revision # | Author | Revision Date | Comments |
| --- | --- | --- | --- |
| 1.0 | Full Team | 2/6/2021 | Initial Plan |

**Table of Contents**

**System Overview**………………………………………………………………………………………..4

System Block Diagram…...……………………………………………………………………..5

Features and Requirements…………………………………………………………………….5

**Activities**…………….......………………………………………………………………………….....6

Requirements Gathering………………………………………………………………………6

Top-Level Design……………………………………………………………………………….6

= DetailedDesign…………………………………………………………………………………7

Testing………………………………………………………………………………………….10

Bug Fixing……………………………………………………………………………………..10

Optimization……………………………………………………………………………………10

**Tasks** ……………………………………………………………………………………………….... 11

Planning/ Elaboration Phase………………………………………………………………….11

Implementation Phase………………………………………………………………………...12

Testing Phase…………………………………………………………………………………..13

**Schedule** ……………………………………………………………………………….…………….14

Elaboration Phase…………………………………………………………………………….14

Implementation Phase………………………………………………………………………..15

Testing Phase………………………………………………………………………………….15

**Milestones**……………………………………………………………………………………………16

**Development Environment** ……………………………………………………………………….16

**Version Control** ………………………………………………………………………………..……17

**System Overview**

60 Seconds is a ***web-based application*** designed to ***record short, daily, audio recording segments***. It will be compatible with desktop/laptop computers via a web browser. Daily, a user will ***record audio*** segments ***up to 60 seconds*** in length. Their daily recording time is ‘pooled’ similar to ‘swipes’ on Tinder. Once the user ***exceeds their daily limit of 60 seconds***, they will be ***unable to record more content***. A user ***WILL*** be able to record ***multiple segments daily***, so long ***total recording time does not exceed 60 seconds.***

For example, in a single day, Johnny records 10 seconds of audio, and 20 seconds of audio; Johnny recorded 30 seconds of audio. If Johnny records any more audio that day, his ***recording CANNOT exceed 30 seconds***, or his ***limit will have been reached*** for that day. Once a limit is reached, the user must ***wait 24 hours to record more content***. The audio recordings that a user makes ***MUST be saved*** for the ***current day***. Recordings cannot be made for days in the past or the future. For example, if today is Wednesday, any recordings Johnny makes will all be stored under Wednesday.

Once audio segments are recorded, they’ll be ***available for the user to review prior to uploading*** (similar to Snapchat’s review of photo/video before sending). Once a recording is made, the application will ***automatically include default tags*** that will be ***stored with the recording***. These tags will be ***produced based on the recording’s metadata*** *(location, date, time, length.)* While the user reviews their audio segment, a description of the recording can be included. Additionally, the application will ***produce a speech-to-text transcription*** of the recording’s contents, which can be edited. When the user uploads the audio recording, this data will be stored alongside it in the database.

Implementation of this project will involve ***React (a JavaScript based front end framework), Node (a back end JavaScript runtime environment) and Amazon AWS (an API hosting service.)*** The most unique feature of this application will be the ***60 seconds Calendar interface***, an interactive in-app calendar where the user’s daily recordings will be housed, acting as the ***account ‘dashboard’***. This in-app calendar will appear on the screen, and the user will have the ability to interact with it.

* The ***current month is displayed*** after logging in to the application
* The user can ***navigate through months*** by clicking the left/right arrow keys
* Individual days can be accessed, displaying that day’s information
  + The audio segment(s) recorded that day by the user, if a recording was made
    - Playback will be available for each recording that day
    - A speech-to-text transcription display of the contents of the audio
    - Default (and custom) tags, a description, and any other information

For organizational purposes, the application will ***produce default tags based on the audio recording’s metadata*** (location, date, time, length). Users will have the option to ***add custom tags*** to their recordings with ***user-defined keywords***. Finally, a ***search bar*** will allow users to ***search for recordings*** based on their created tags, providing ***easy access to user-defined*** recording labels. Audio recordings will be available for download as an MP3 file

**System Block Diagram**



Figure 1: A simple system block diagram of the application as a whole

**Figure 1** represents 60 second’s ***system block diagram***. Users with an active internet connection access 60 seconds via a client system (laptop/desktop). The web application will ***communicate with Amazon AWS’s*** various hosted services, including storage solutions, database services, and ***Natural Language Processing (speech-to-text)*** transcription services.

1. **Clients:** front-end part of the application. Will be developed using React, and will act as the application’s user interface. The user will record audio, edit the postings, and save them to the database using default (optional custom) tagging.
2. **Storage:** user information will be stored using remote storage via the cloud. Information will be called from storage using various commands. Each day a user records/uploads audio, that recording, along with any tagging, the transcription of the text-to-speech, and any other data, will all be stored here.
3. **API:** Amazon AWS will serve any and all API calls, mainly the transcription service, which will take the user audio segment, and return a transcription of the audio content. This transcription returns as a json file, which can be converted to a string to be saved to storage.

**Features and Requirements**

* Account Management
  + User registration
  + User login, utilizing Google’s Google Sign-in tool for security and ease of access
  + Local user profile management and customization
* Content Storage and Delivery
  + Store audio recordings, website content, user content in a scaling Amazon AWS cloud storage solution
  + Index each audio recording within a database for easy access and organization
  + Exporting audio for users in an MP3 format
* Content Tagging
  + Allowing users add tags to custom content they upload
  + Allowing users to search and parse through their recordings based on tags
* Audio Recording
  + Record and store audio for user to playback
  + Requires access to user device’s microphone
* User Interface
  + Calendar-based organization of recordings
* Audio Analysis
  + Speech to text transcription of audio recordings using Amazon AWS’s Speech to Text service.
* Secure Coding Practices
  + Secure login, HTTPS, Password protection, content management.

**Activities**

**Requirements Gathering**

In order to begin development on our application, our team needs to do a lot of research into the different development areas we plan to touch on. These areas include the Amazon AWS cloud suite, database management, authentication and account management via Google, React, and Node. We plan to spend a sufficient amount of time researching and learning about these technologies during the early phases of this project so that we’re able to work quickly and effectively once we get to the development stage.

### **Top-Level Design**

1. Create sample web app in React featuring multi-page layout with tabs for navigation
2. Connect React front-end app with the backend, connect to Amazon AWS database and cloud storage solution
3. Implement user login system using Google Sign-in authentication service
4. Implement an interface for audio recording within the app, store audio recordings made in the storage bucket and index in database to associate them with the creator
   1. Create a calendar interface for accessing user recordings
5. Create an interface for searching through user recordings based on date/tag queries
6. Implement a playback screen for selected past recordings where users can listen to previous recordings, view/read a transcription of the recording, and download the recording to their device

### **Detailed Design**

1. **Create sample web app in React featuring multi-page layout with tabs for navigation**
   1. Site navigation will include:
      1. Default/Sign-on Page
         1. Default page when accessing the application for the first time
         2. Offers users the opportunity to sign into the application with their Google Account via the Google Sign-in authentication service.
      2. Home/Recording Page
         1. The “main” page of the application features the audio recording interface.
         2. Users will see a sixty second countdown as they are making their recording.
         3. After completing their recording, they can play it back and choose to submit it as their daily entry or to re-record.
      3. Calendar/Search Page
         1. The page where users can access their past recordings.
         2. Users can scroll through and interact with a calendar to view and playback their recordings]
         3. Users can also utilize the search bar at the top of the page to enter a specific date or to search for a specific tag they associated with a past recording.
      4. Account Page
         1. Users can view their profile which will contain certain specific data about their account
            1. Name
            2. Email address
            3. Profile photo
            4. Date of account creation
            5. Number of recordings made
         2. Users can quickly and securely sign out of their account via the Google Sign-on authentication service.
   2. **Connect React front-end app with the backend, connect to Amazon AWS database and cloud storage solution**
      1. Communicating via the API calls to connect the front-end to the back-end
   3. **Implement user login system using Google Sign-in authentication service**
      1. Upon visitation of the application, present users with the ability to sign-in to the application using Google’s Sign-on authentication service.
      2. User’s names, email addresses, and profile photos associated with their Google accounts will be used throughout the application.
   4. **Implement an interface for audio recording/playback within the app, store audio recordings made in the storage bucket and index in database to associate them with the creator**
      1. The main page of the application will feature an interface for users to record their daily oration. Users will be able to start recording and be presented with a sixty second countdown of time remaining before the recording ends. Users can choose to end the recording draft early by hitting the “stop” button.
      2. After recording, users will be presented with a playback interface for their new recording. Users have the option to listen to the full recording draft, fast-forward/rewind, and pause the playback.
      3. After the user is presented with the option to playback the recording, they will have the option to finish and submit the recording draft or to discard the recording draft and re-record.
         1. Upon submission of the draft, the user can choose to create or use previously-created tags to associate with the recording for later ease-of-access.
         2. A draft that is submitted will be uploaded to the storage solution and indexed by the database to associate the recording and its metadata to the user.
      4. Alongside the upload of the recording to the storage solution, a database entry will also be made containing the following information:
         1. Link to access the recording from the cloud storage solution
         2. Metadata associated with the recording
            1. Author/owner of the recording
            2. Date of recording
            3. Tags associated with the recording
   5. **Create a calendar interface for accessing user recordings**
      1. Users will be able to access their recordings by interacting with a built-in calendar and visiting the date of a recording they created.
      2. The calendar will be shown one month at a time, with a small dot located on each day there is a recording to indicate that a recording was made on each day.
      3. Users can tap on a day that is associated with a recording to enter the playback screen for that recording.
   6. **Create an interface for searching through user recordings based on date/tag queries**
      1. At the top of the calendar interface will be a search bar
      2. Users can utilize the search bar to access recordings if they know the date of a recording they find
      3. The search bar also accepts previously created “tags” to query a the list of a user’s recordings for audio files associated with that tag.
   7. **Implement a playback screen for selected past recordings where users can listen to previous recordings, view/read a transcription of the recording, and download the recording to their device** 
      1. After selecting a recording from the calendar/search interface, users will be presented with a playback screen.
      2. Users can play, pause, and “scrub” through the recording to listen to the audio from a given day.
      3. Below the recording playback buttons will be a transcript of the words spoken in the recording.
         1. The transcript will be generated utilizing the Amazon AWS Speech to Text service.
      4. Users can download the MP3 file of the selected recording from the cloud storage solution to their device.

### **Testing**

1. **Unit Testing**
   1. For unit testing on both the client and server side, we will use the Jest testing framework. This is a robust testing framework for Node applications that’s actually included with React by default. We think that this is going to be the perfect unit testing solution for our project.
2. **Manual Testing**
   1. Manual testing will be crucial for this project because unit testing alone will not be sufficient for catching bugs that occur during the usage of this application. To test for these kinds of issues, we will need to manually use the application in a variety of different ways to ensure that there are no edge case bugs that produce unexpected behavior. We will then report/fix any issues we find using the methodology described below.

**Bug Fixing**

For tracking and dealing with software bugs, we plan on using Rally’s built in bug tracking system. This system will allow us to keep track of bugs (defects) in any given iteration easily. Once we know what bugs our software has, we can cleanly assign bug fixes to different team members on an as-needed basis.

**Optimization**

Since the end result of our project will be a graphical interface, optimization will mostly be in terms of improving the user experience. To do this, we will examine our project to see which actions might be difficult for a new user to grasp.

## 

## 

## **Tasks**

**Planning/ Elaboration Phase (PP)**

The planning/elaboration phase is arguably one of the most important parts of the development process. It is important for all the team members to properly define, research, and understand the project and its scope before the actual implementation begins.

| **#** | **Task** | **Design of Unit X** | **Estimated**  **Effort** | **Finish Date** | **Assigned Individual(s)** | **Successor Tasks** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Group Project Proposal | None | Whole Group | 1/24/2022 | Everyone |  |
| 2 | Requirements Document |  | Whole Group | 2/2/2022 | Everyone |  |
| 3 | Software Development Plan Document |  | Whole Group | 2/6/2022 | Everyone |  |
| 4 | Design Document Part 1: Architecture |  | Whole Group |  | Everyone |  |
| 5 | Design Document Part 2: API |  | Whole Group |  | Everyone |  |
| 6 | Test Procedure Documents |  | Whole Group |  | Everyone |  |
| 7 | Test Report Documents |  | Whole Group |  | Everyone |  |
| 8 | “Hello World” |  | Whole Group | 2/12/2022 | Everyone |  |

### 

### 

### **Implementation Phase (IP)**

| **#** | **Task** | **Design of Unit X** | **Estimated Effort** | **Finish Data** | **Assigned Individual(s)** | **Successor Tasks** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Create Node.js Back-end Server |  | One Person | 3/5/2022 | N/A |  |
| 2 | Create React.js Front-End Server |  | One Person | 3/10/2022 | N/A |  |
| 3 | Initial User Interface |  | One Person | 3/14/2022 | N/A |  |
| 4 | Barebones Client-Server Communication |  | One Person | 3/14/2022 | N/A |  |
| 5 | Google Authentication Implementation |  | One Person | 3/13/2022 | N/A |  |
| 6 | Audio Recording |  | One Person | 3/18/2022 | N/A |  |
| 7 | Audio Storage |  | One Person | 3/22/2022 | N/A |  |
| 8 | Audio Delivery |  | One Person | 3/26/2022 | N/A |  |
| 9 | Content Tagging (Not Including Audio Analysis) |  | One Person | 4/2/2022 | N/A |  |
| 10 | Audio Analysis |  | One Person | 4/6/2022 | N/A |  |
| 11 | Content Tagging Via Audio Analysis |  | One Person | 4/11/2022 | N/A |  |

### 

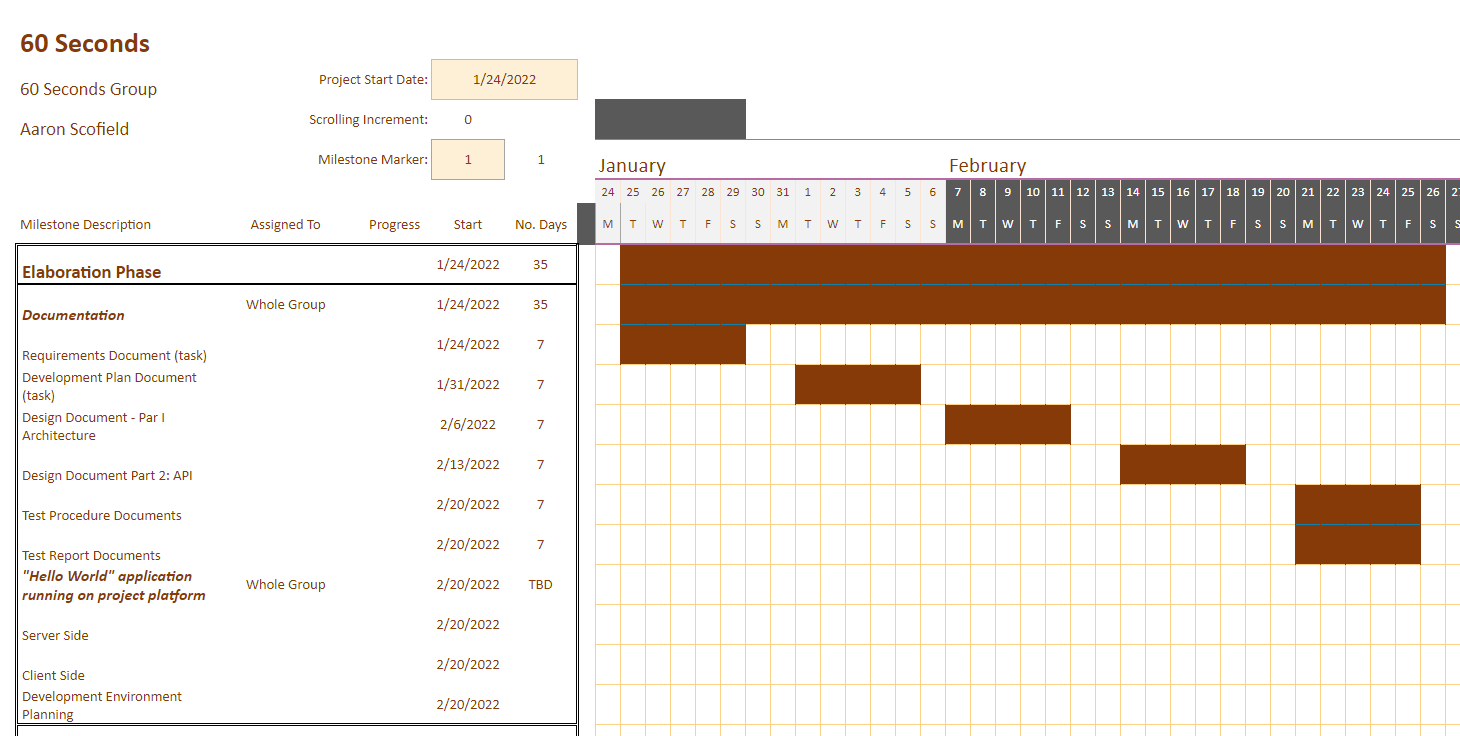
### **Testing Phase (TP)**

| **#** | **Task** | **Design of Unit X** | **Estimated Effort** | **Finish Data** | **Assigned Individual(s)** | **Successor Tasks** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Making sure that our server configuration is correct |  | One Person |  | N/A |  |
| 2 | Making sure that the back end and front end servers communicate correctly |  | One Person |  | N/A |  |
| 3 | Making sure our initial UI is bug free |  | One Person |  | N/A |  |
| 4 | Making sure the account system is working correctly |  | One Person |  | N/A |  |
| 5 | Making sure audio is stored and retrieved correctly |  | One Person |  | N/A |  |
| 6 | Making sure content tagging is working correctly |  | One Person |  | N/A |  |
| 7 | Making sure audio analysis is working as intended |  | One Person |  | N/A |  |
| 8. | Making sure the final version of our UI offers a good user experience |  | One Person |  | N/A |  |
| 9. | Making sure the final version of our UI is bug free |  | One Person |  | N/A |  |

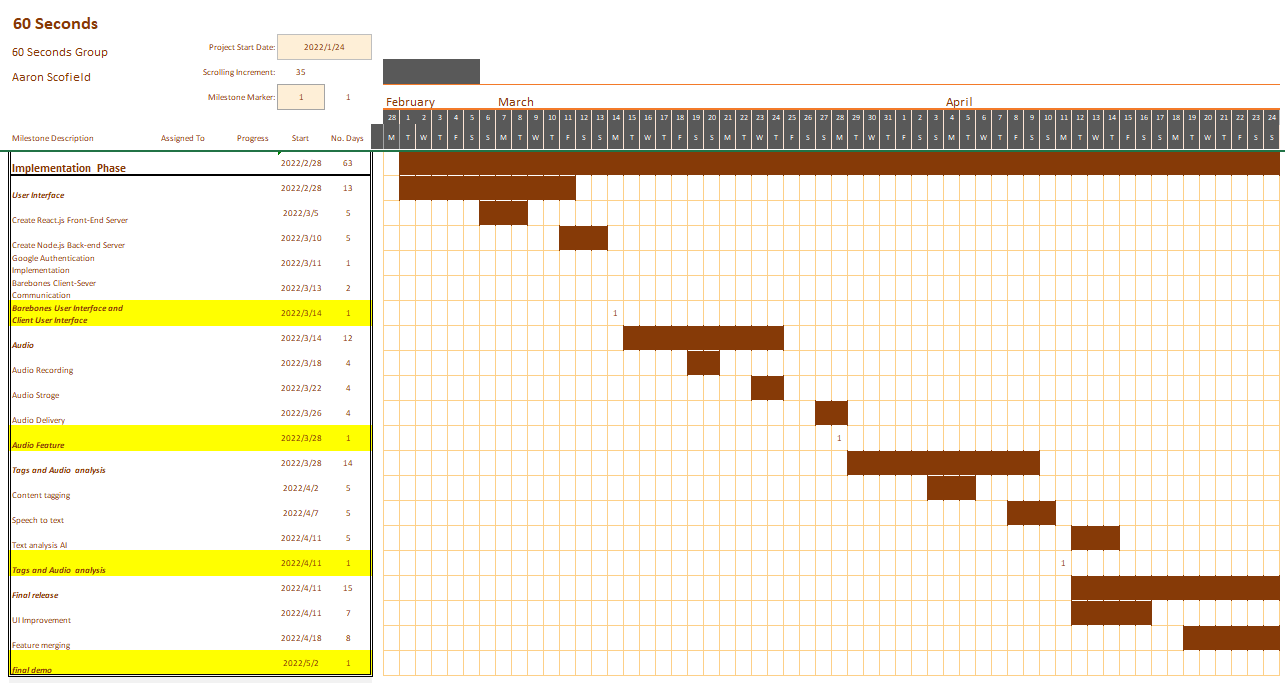
## 

## **Schedule**

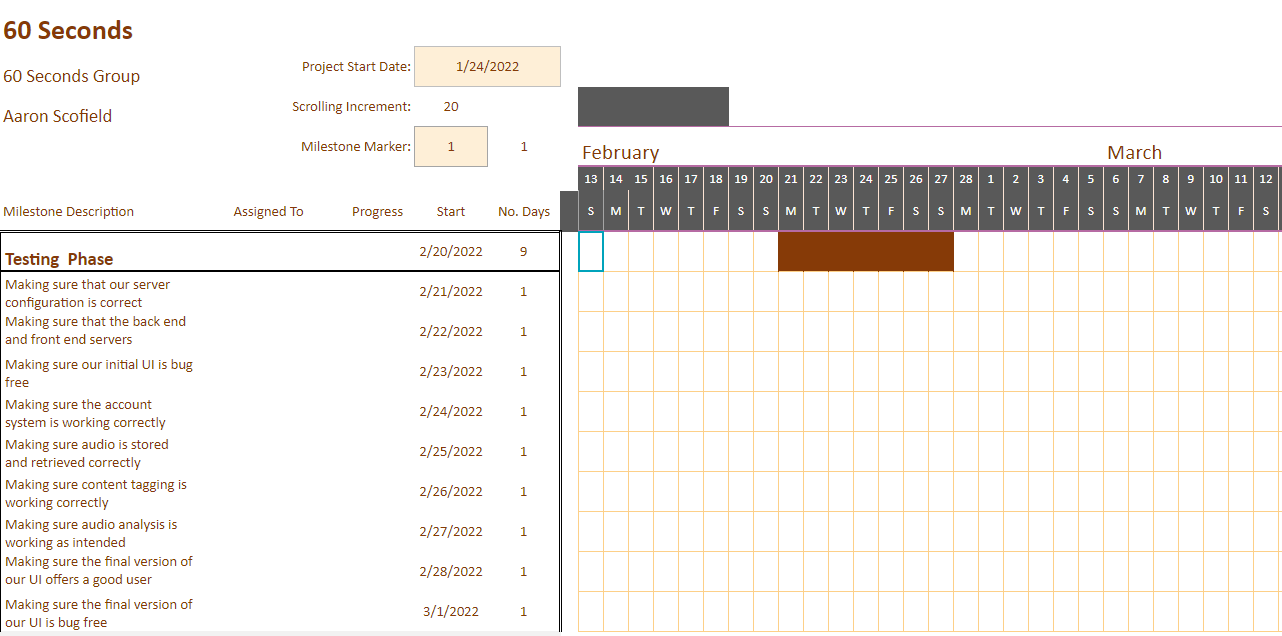
**Elaboration Phase**

****

**Implementation Phase**



**Testing Phase**

****

# 

# **Milestones**

| **Deliverables** | **Date** |
| --- | --- |
| * Barebones User Interface (Version 1.0.1) * Client Server Communication (Version 1.0.2) * Account Management (Version 1.0.3) | 3/14 |
| * Audio recording (Version 1.1.1) * Audio storage and delivery (Version 1.1.2) | 3/28 |
| * Improved UI (Version 1.2.1) * Content tagging (Version 1.2.2) * Audio analysis (Version 1.2.3)   + Speech to text   + Text analysis AI (Version 1.2.4) | 4/11 |
| Full Featured Project (Version 2.0.0) | 4/25 |

**Development Environment**

**Required Hardware**

* User information will be stored remotely, server will send/receive user data
* Device with working microphone for testing audio recording

**Required Software**

* Amazon AWS
* Amazon AWS Resource Group hosting all resources
* Amazon AWS App Service hosting web application
* Bootstrap 3
* Node.js
* Expo CLI mobile application
* HTML 5, CSS 3, React 17, JavaScript
* GitHub client connected to team repository
* NPM install for React
* Amazon AWS Database for MySQL servers
* Amazon AWS Speech to Text
* Amazon AWS Text Analytics
* Expo av
  + Records to .mp3 file

**Selected IDE**

* Visual Studio Code IDE 1.64
  + ES7 + React/Redux/React-Native extension
  + HTML CSS Support extension

**Compilers**

* Node.js
* NPM start

**Test Tools**

* Expo CLI for Mobile app testing
* Jest JavaScript test runner
* React Testing Library

**Build Tools**

* Node Package Manager (npm)
* ESLint

## 

## **Version Control**

Version Control Tool: Github/Git Repositories

The primary IDE that will be used by the group members will be Visual Studio Code. This IDE will be utilized due to its high compatibility to Git/Github version control systems. Similar to previous semesters, there will be a main branch, which will act as the ‘finalized part’ of each feature. Each feature/component will have its own sub-branch, and within each branch, team members will contribute to development of each feature/component. Once a feature has been tested, and runs successfully, the group member will make a pull request for the other group members to review. When it fulfills the requirements, and has gone through testing, the other group members will accept it, at which point it merges with the main branch. This will be done for each task for each group member. The initial version of this project will be the ‘Hello World’ implementation, which will be Version 1.0.0. Each subsequent release will be numbered according to the Deliverables section of this document.