# 

# 

# **Test Procedure Document**

60 Seconds

**REVISION HISTORY**

| Revision # | Author | Revision Date | Comments |
| --- | --- | --- | --- |
| 1.0 | Full Team | 2/21/2021 | Initial Document |

**TABLE OF CONTENTS**

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

* System Block Diagram……….…………………………………………….5
* Features and Requirements….……………………………………………...6

**Document Overview……………….……………………………………………...7**

**Unit Test….……………..………………………………………………………….7**

* Client………………….…………………………………………………….7
* Server……………………………………………………………………….8

**Integration Testing……………………………………………………………..….9**

**Acceptance Testing………………………………………………………….……11**

**SYSTEM OVERVIEW**

60 Seconds is a web-based application used for recording short, daily audio segments, and will be available for both mobile devices, and desktop/laptop computers via a web browser. A user will record daily audio segments, the maximum sum of recording time is 60 seconds. Their daily recording time is ‘pooled’ similar to ‘swipes’ on Tinder. Once the user exceeds a total recording time sum of 60 seconds for that day, they will be unable to record more content. A user WILL be able to record multiple segments a day, so long as the sum of their playtime does not exceed 60 seconds.

For example, in a single day, Johnny records a 10 second, and two 20 second segments. The sum of content for the day is 50 seconds. If Johnny records another audio segment, it cannot exceed 10 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.

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). While users review their audio segment, they will be able to add a description, tags, dates, and/or pictures as well as a transcript of what is said during the recording. When the user uploads their audio segment, this additional data will be included, is indexed and associated with the user, and then stored in a database.

As a web and mobile application, the implementation of this project will involve React, a JavaScript-based front-end framework, in addition to Node, a back-end JavaScript runtime environment. The application will utilize the Microsoft Azure cloud for hosting these services.

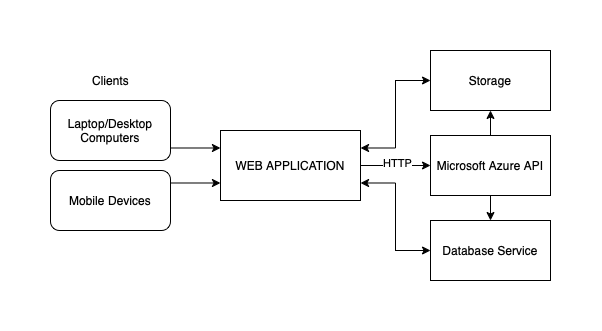
User interaction within the app will revolve around an interactive in-app calendar. This is where the user’s daily recordings will be housed, acting as the user’s ‘dashboard’. The in-app calendar will appear on the screen, and the user can interact with it.

On the in-app calendar, the current month will appear when the user first opens the mobile app or web app. The user can navigate through months by swiping left or right on mobile, or clicking the left or right arrows on the web application. Individual days can be tapped/clicked, bringing up that day’s screen, which will contain the audio segment(s) recorded that day by the user. Each audio segment can be played, in addition to a transcript will be available on screen, powered by Microsoft Azure’s Speech to text service. The calendar day will also display any additional information provided by the user, such as tags or descriptions.

Users can tag recordings at time of creation with default tags or user-defined keywords. Date, time, and location of the recording will be available as default tags. Users can also tag audio recordings by length, specifying whether they are short or long.

In order to find a specific recording, a searching mechanism will be available via a search bar and search button. Users can search for recordings based on their created tags, providing easy access to user-defined recording labels. In addition, each audio recording will be available for download as an MP3 file and shared at the user’s discretion.

**System Block Diagram**



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

Figure 1 shows a simple system block diagram of the application. Users with an active internet connection can access the web application via a client system, which can be a mobile device or a laptop/desktop computer. The web application will communicate with the various services hosted on the Microsoft Azure cloud, including the storage solution, database service, and various other Azure services spanning from transcription to hosting.

**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 Microsoft Azure 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 Microsoft Azure’s Speech to Text service.
* Secure Coding Practices
  + Secure login, HTTPS, Password protection, content management.

**DOCUMENT OVERVIEW**

Part one of this design document, focusing on architecture, describes the software architecture in more detail and how the design requirements are to be mapped into the design. The document will be a combination of diagrams and text that describes what the diagrams are showing. The document will also detail a description of the different application components and their interfaces (ex. Client, server, database). For each component, class and sequence diagrams will be provided showing the data flow for use cases. Finally, the document will explain algorithms employed in the project, primarily focusing on the interactions between Microsoft APIs and structure and design of the database.

**UNIT TESTS**

**Unit tests**: A type of software testing where individual units or components of a software project are tested. Validates that each software code unit performs predictably. Conducted during development (coding phase) of application by the developers. Isolate a section of code to verify its correctness; may be an individual function, method, procedure, module, or object.

**Framework:** Jest

**Client**

| canFormatApp | Test: Attempt to load page given the dimensions of the window running the application.  Result: True if given dimensions are compatible; load error if incompatible |
| --- | --- |
| canRecordAudio | Test: Attempt to gain access to microphone  Result: True if access is permitted; denied permission if false |
| canGetDateTime | Test: Attempt to gather device date and time  Result: True if date and time are accessible; null if date is not accessible |
| canScrubAudio | Test: Attempt to traverse through an audio recording, this consists of play, stop, scrub functionality.  Result: True if all operations are functional; attempted function if failed. |
| canLogin | Test: Attempt login with dummy credentials  Result: True if login was successful; Failed login if false. |
| canLogout | Test: Logout of application with dummy account  Result: True if logged user is successfully logged out of session; False if user is still logged in |
| canRenderCalendar | Test: Load an instance of a calendar object using date and time on device.  Result: True if device specifications are able to render object; False if dimensions are ineligible or device date/time are invalid. |

**Server**

*Audio Processing*

| canConvertSpeechToText | Test: Attempt to run a speech to text conversion on a given audio snippet  Result: True if the snippet is successfully translated; return attempted translation instead if it’s incorrect  (may need to tweak this in practice) |
| --- | --- |
| canApplyTags | Test: Attempt to apply tags to an audio clip based on text analysis  Result: True if the correct tags are created; return attempted tagging instead if it’s incorrect  (may need to tweak this in practice) |

Database/CDN

| canConnectToDatabase | Test: Attempt to connect to the database  Result: True if the database connection can be successfully established |
| --- | --- |
| canUpdateAccountInfo | Test: Attempts to update an existing user’s account information  Result: True if records are updated successfully |
| canRetrieveAccountInfo | Test: Attempts to retrieve account information for an existing user  Result: True if account info is retrieved correctly |
| canUploadFile | Test: Attempts to upload a new audio clip  Result:True if the file is successfully uploaded |
| canRetrieveFileURL | Test: Attempts to retrieve the file URL for a given clip  Result: True if the correct URL is retrieved |
| canCreateUser | Test: Attempts to create a new user  Result: True if new user is created successfully |
| canDeleteUser | Test: Attempts to delete an existing user’s account  Result: True if the account is deleted successfully |

**INTEGRATION TESTS**

**Integration Testing**: testing of various software modules under development together, determines whether all modules function together seamlessly. Not to be confused with unit testing. Integration testing is done on groups of these modules to make sure they work together and interact properly. Usually conducted later in the project timeline.

**Use Case #1 - User wants to use the mobile application**

**Objective:** Ensure that the client, server, database, and CDN are able to work together correctly

**Description:** The client establishes a connection with the backend server after being served react files through the expo server. The client attempts to retrieve information from the backend server about existing audio clips to display on the home page.

**Expected Result:** Upon receiving a request from the client, the backend server should contact the database to retrieve the relevant information about the client user. That information, including CDN links to audio files should then be forwarded to the client where it can be displayed using react.

**Use Case #2 - User wants to upload a new audio clip to their account**

**Objective:** Test the ability of the application to correctly upload and store audio clips from end users.

**Description:** The user, who is already authenticated, launches the client from the app. They navigate the UI to record and upload a new audio clip to the server. The clip is received by the backend server, where it’s tagged, indexed in the database, and uploaded to the CDN.

**Expected Results:** The audio clip should be successfully uploaded to the CDN and indexed in the database. An audio analysis should take place, where the content is properly tagged using language processing.

**Use Case #3 - User wants to login to the application**

**Objective:** Ensure that users can login to the application, which will require coordination between the react client, the server, and the database.

**Description:** The user clicks the login button on the home screen. They choose to log in either through Google or through the built in account system. If they choose to log in with Google, they’re redirected to the “Login With Google” page where they receive an OAuth token which is then shared with the server. The server stores information about the user’s session in the database. The user is then redirected to the logged-in homepage. If they choose to log in through their 60 seconds account directly, they’re redirected to a secure login page where they can enter their credentials. Those credentials are then forwarded to the backend server, where they’re checked against the passwords stored in the database. If the login is successful, the user is redirected to the home page, and information about their login session is stored in the database.

**Expected Results:** After the login process is complete, the user should stay logged in for a reasonable amount of time. They should be able to access all portions of the application that require authentication.

**User Case #4 - User wants to play previous recording**

**Objective:** Test the ability of the application to correctly play the audio that the user selects.

**Description:** After the user successfully login, the user navigates to the calendar page and selects a date. The user will see all recordings for the day selected by the user. User clicks the play button, the backend server will retrieve the recording from cloud storage and return to the user.

**Expected Results:** The backend server should correctly retrieve the recording that the user selected.

**ACCEPTANCE TESTS**

**Acceptance Testing**: A method of software-development and testing where a system is tested for acceptability. The goal is to evaluate the compliance of the system with the business requirements and assess whether it is acceptable for delivery or not. The final phase of software testing, performed after System Testing and before system’s commercial use availability.

| **Test ID** | **Action/Steps** | **Notes/Expected Results** |
| --- | --- | --- |
| 1 | **Create an audio recording**  Open the app  Record an audio clip  Save the audio clip | Audio recording is stored on the user’s account |
| 2 | **View previous recordings**  Open the app  From the home screen, browse previous audio recordings | All recordings the user has created are available to view from their account |
| 3 | **Login to the application**  Open the app  Click “login”  Enter valid credentials | The user is successfully logged in |
| 4 | **Create a new account**  Open the app  Click “Create Account”  Create an account, optionally using Google | A new account is successfully created |
| 5 | **Process an audio clip and automatically tag it**  Open the app  Record an audio clip  Submit it for processing | The audio clip should be automatically tagged based on the text analysis algorithms we implement |
| 6 | **Implement the 60 seconds timer for a single day**  Record multiple audio segments, the sum of their length being less than or equal to 60 seconds. | 3 segments in a single day of 20 second audio clips  2 segments of 30 second audio clips  A subset of combinations to test the robustness of the application |
| 7 | **Verify that the Calendar Interface is consistent with a real Calendar**  Make sure that the Calendar days and months implemented in the application are consistent with a real life Calendar | When the application loads, the user’s dashboard will be the Calendar interface, make sure that the Calendar is highlighting the current day correctly, and also that days/months/years line up with the real calendar. |