# 

# 

# 

# 

# **Design Document - API**

60 Seconds

**REVISION HISTORY**

| Revision # | Author | Revision Date | Comments |
| --- | --- | --- | --- |
| 1.0 | Full Team | 2/20/2021 | Initial Document |

**TABLE OF CONTENTS**

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

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

**Document Overview……………………………………………………………6**

**Front End……………………………………………………………………….7**

**Back End………………………………………………………………………..12**

**References………………………………………………………………………15**

**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.

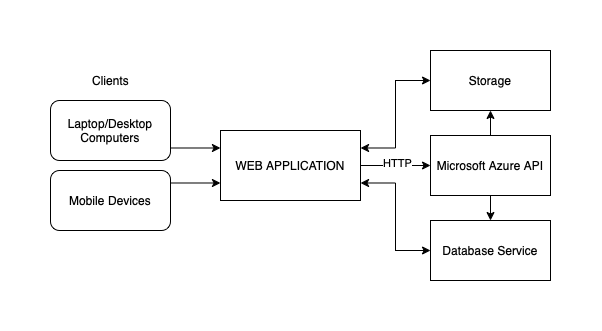
User interaction within the app will revolve around an interactive in-app calendar. This is where the user’s daily recordings will be housed, acting as the user’s ‘dashboard’. The in-app calendar will appear on the screen, and the user can interact with it.

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**

The document will break down the application into the front-end and back-end and further describe each file and planned function in terms of JSDoc documentation standards. Each file will be a heading, with functions below listed as bullet points.

**FRONT-END DEVELOPMENT**

**App.js**

* @class App.js is the main component in React and acts as a container for all other components.

**Header.js**

* @class Header.js is the header component that will be present on all pages of the app. It will contain the navigation bar with links to the About, User Profile, Calendar, Recording pages in addition to housing the Login and Logout button components.
  + /\*

@params {About, User, Recorder, Calendar, Login, Logout }, the components to be located/linked to in the header

@returns {<Header/> JSX Object}, a React JSX object containing the HTML for the header

\*/

const Header = ({About, User, Recorder, Calendar, Login, Logout}) => {

return (<header className=’header’>...</header>);

}

**About.js**

* @class About.js will be a simple page displaying information about the project, listing the developer team, and providing a contact email.
  + /\*

@returns {<div/> JSX Object}, containing the static front-end HTML for the About page.

\*/

const About = () => {

return(<div>...</div>);

}

**User.js**

* @class User.js will be a page displaying information about the user currently logged into the application using the Google Identity Services.
  + /\*

@params {firstName, lastName, emailAddress, picture, email}

@returns {<div/> JSX Object}, containing the static front-end HTML for the User Profile page. It pulls information passed in from the current sign-in session authenticated using the Google Identity Services OAuth session.

\*/

const User = () => {

return(<div>...</div>);

}

**Login.js**

* @class Login.js will be a component featuring the “Sign-in with Google” button functionality implemented with Google Identity Services. The button will be displayed directly on the navbar and will lead to the OAuth sign in process. It will only be visible if the user is not logged into the application.
  + /\*

@returns {<GoogleLogin>} object to render in the header component.

\*/

const Login = () => {

return(<GoogleLogin…></GoogleLogin>);

}

**Logout.js**

* @class Logout.js will be a component featuring a “Sign-out” button implemented with Google Identity Services that signs the user out of the application and ends the user session.
  + /\*

@returns {<GoogleLogout>} object to render in the header component when a user is logged in successfully

\*/

const Logout = () => {

return (<GoogleLogout…></GoogleLogout>);

}

**Calendar.js**

* @class Calendar.js will be a page in the application that houses the calendar component. The calendar itself will display a singular month at a time and will be made up of individual day components, which are displayed in turn when a user clicks on a given day.
  + /\*

@params {Day}

@returns {<><Day …/><>}, a React fragment that allows the calendar to return multiple days

\*/

const Calendar = (Day) => {

<Day…/>

}

**Day.js**

* @class Day.js is the component for a singular day within the Calendar component. This component will provide all information on the current date and time.
  + /\*

@returns {<url>}a Url that allows user to exporting audio files

\*/

const GetAudioFile = (day) => {

…

const file = day.recording

…

return {<url></url>}

}

**Recorder.js**

* @class Recorder.js is a page in the application that houses the recording component. The recording component will only be active if a user has not already used up their given sixty seconds pool of recording time each day.
  + /\*

@returns {<div>...</div>}, React JSX object containing the audio recording UI.

@import {Audio} from ‘expo-av’, Import audio library from expo-av package.

\*/

async function startRecording() {

…

const {recording} = await Audio.Recording.createAsync(

Audio.RECORDING\_OPTIONS\_PRESET\_HIGH\_QUALITY

);

setRecording(recording);

}

async function stopRecording(){

setRecording(undefined);

await recording.stopAndUnloadAsync();

const uri = recording.getURI();

}

**Playback.js**

* @class Playback.js is a component that will be housed within both the Recorder component and the Day component. In the former, it will allow the user to playback the recording they just made before uploading, and in the latter it will allow a user to playback a previous day’s recording that they are viewing on the Calendar page.
  + /\*

@import {Audio} from ‘expo-av’, Import audio library from expo-av package.

@param {string} uri, the uri returned from the database hosting the .mp3 file

\*/  
async function playSound(uri){

const { sound } = await Audio.Sound.createAsync(

require(uri)

);

setSound(sound);

await sound.playAsync()

}

**Searchbar.js**

* @class Searchbar.js is a component that will be present on the Calendar page. It will allow users to search their previously created recordings by name or tag. The component will query the database and return the ids of each recording and its associated day.
  + @function searchBar(calendar)
  + @summary front-end search bar that the user will interact with and input either a recording name, date, or tag
  + Points to Searchresults.js for search results
  + @return query
    - /\*

@returns {<query>} object that was entered in by user within the rendered search bar.

\*/

const Searchbar= () => {

return(<searchbar></searchbar>);

}

**SearchResults.js**

* @class SearchResults.js is a component that will be present alongside the Searchbar.js component. It will house results returned from the given search query and provide links to the corresponding Day components that match the search entry.
  + @function searchResults(query)
  + @summary displays a list of results that contain the entered query information, returns empty string if results are empty.
  + @return calendar object containing all results containing query
    - /\*

@returns {<calendar>} object that was generated by search that contains all tag calendar days consisting of the given query.

\*/

const Searchbar= () => {

return(<calendar></calendar>);

}

**Tagging.js**

* @class Tagging.js is a child component of the Recorder.js class and a sibling of SearchResults.js and Searchbar.js. This class will contain the features that allow the app to automatically generate certain tags to a recording such as approximate length, date, location, etc. as well as allowing the user to add their own custom tags to ease searching within the application.
  + @function createTag(recording)
  + @summary create auto tags for date, duration (short, moderate, long), location as well as allow input for tag generation from user
  + @return tag object
    - /\*

@returns {<tag>} object that was generated by auto tagging or user entered tags

\*/

const autoTag= () => {

return(<autoTag></autoTag>);

const userTag= () => {

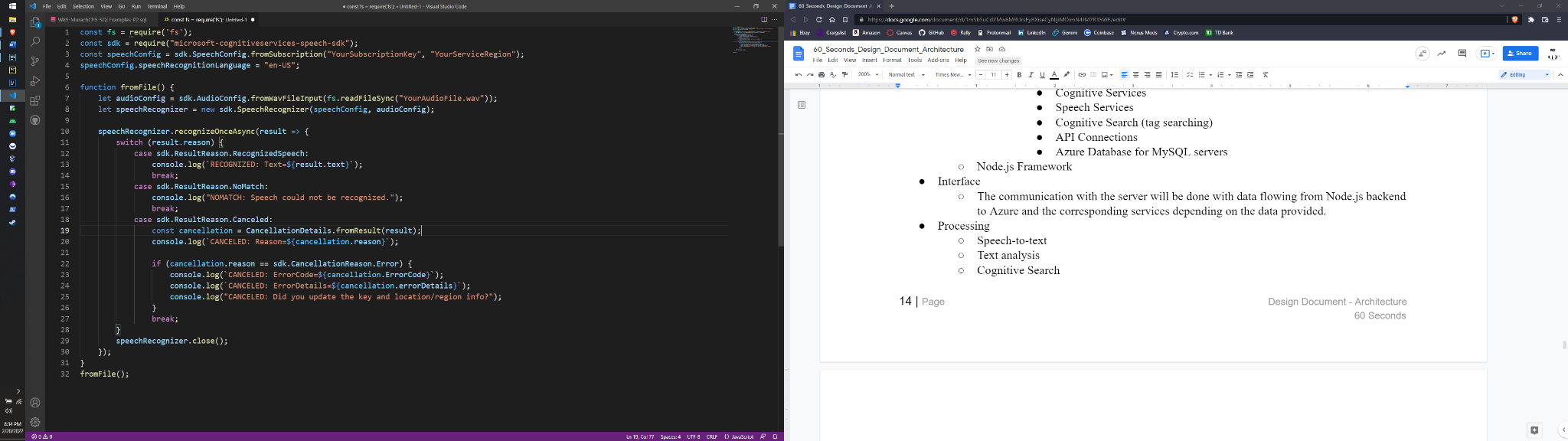
return(<userTag></userTag>);

}

**BACK-END DEVELOPMENT**

**Microsoft Azure App Services**

Services (provided by Microsoft) imported/scripted into the framework of choice (JavaScript/Node/React), implemented in development



**SpeechRecognition.js File**

Speech-to-text from Speech service, a.k.a speech recognition, enables real-time and batch transcription of audio streams into text. With additional reference text input, enables real-time pronunciation assessment and gives speakers feedback on accuracy/fluency of spoken audio.

* @summary: a script that will take an audio file, and return the contents of the audio file as a string, or will simply log them to the console
  + @param: audioFile - audio file that is returned from node.js audio component
* @param: skd - will create an instance of the Microsoft Azure Cognitive Services
* @param: speechConfig - will create an instance of the speech configuration service from the subscription, and will set the recognized language
* @function: fromFile() - this function will contain if/else statements and will return a string or log the result to the console, depending on how it is implemented
  + Dependent on the result of the speech recognition
  + @param: audioConfig - an instance of the speech configuration service with the audio file input name input as a string
  + @instance: speechRecognizer - object of Microsoft’s speech recognizer
  + @param: result - the result of the recognition instance
    - @param: recognizedSpeech
      * @summary: speech was recognized, returns the results as a string
    - @param: NoMatch
      * @summary: speech was not recognized, return null
      * @throws: “NoMatch: Speech could not be recognized”
    - @param: Canceled
      * @throws: CancellationReason:
      * @summary: if the key and location are invalid.

**Server.js File**

* @summary: this file will handle all incoming requests from the client and redirect them to the appropriate channels
* @function: onRequest() - function to handle raw HTTP requests to the server
  + @param: request
  + @return: null
* @function: onCreateAccount() - adds a new user to the database through either Google or in-house accounts
  + @param: request
  + @return: null
* @function: onDeleteAccount() - deletes the requesting user’s account
  + @param: request
  + @return: null
* @function: onLogin() - verifies logins through either Google or in-house accounts, and redirects to success or failure states based on result
  + @param: request
* @function: onUploadClip() - takes uploaded clips and stores and indexes them in the database
  + @param: request
  + @return: null
* @function: onRequestUserPage() - fetches all relevant resources from the requesting user’s database entries and sends them to the client
  + @param: request
  + @return: null
* @function: onDeleteClip() - deletes a given clip or clips requested by the user
  + @param: request
  + @return: null

DatabaseInterface.js File

* @summary: this file handles all direct calls to the database. It will mostly be used by the Server.js file to retrieve and store resources relating to user requests
* @function: storeClip()
  + @param: tags (array)
  + @param: file (file)
* @function: retrieveClipLink()
  + @param: clipID
  + @return: link (String) - returns a CDN link to the requested clip
* @function: createUser()
* @function: deleteUser()

**REFERENCES**

<https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/get-started-speech-to-text?tabs=windowsinstall%2Cterminal&pivots=programming-language-javascript>

<https://docs.expo.dev/versions/latest/sdk/audio/>

<https://www.npmjs.com/package/react-google-login>

<https://cloud.google.com/identity-platform>