# 

# 

# **Design Document - Architecture**

60 Seconds

**REVISION HISTORY**

| Revision # | Author | Revision Date | Comments |
| --- | --- | --- | --- |
| 1.0 | Full Team | 2/13/2022 | Initial Document |
| 1.1 | Aaron Scofield | 4/4/2022 | Updates based on progress and feedback |

**TABLE OF CONTENTS**

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

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

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

**Client Components…………………………………………………………….7**

**Server Components…………………………………………………………….14**

**Database Components…………………………………………………………16**

**References………………………………………………………………………19**

**SYSTEM OVERVIEW**

60 Seconds is a web-based application used for recording short, daily audio segments, and will be available for desktop/laptop computers via a web browser. A user will record a daily audio segment, the maximum length of recording time is 60 seconds. Once the user exceeds a total recording time sum of 60 seconds for that day, they will be unable to record more content. Once a limit is reached, the user must wait unti the next calendar day 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, tags, 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 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 AWS cloud for hosting these services.

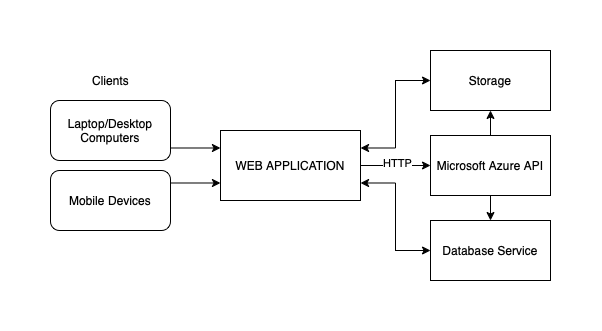
User interaction within the app will revolve around an interactive in-app calendar. This is where the user’s daily recordings will be housed, acting as the user’s ‘dashboard’. The in-app calendar will appear on the screen, and the user can interact with it.

On the in-app calendar, the current month will appear when the user first opens the web app. The user can navigate through months 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 JavaScript Web Speech API. 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.

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 laptop/desktop computer. The web application will communicate with the various services hosted on AWS, including the storage solution, and various other AWS services spanning from databases 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 AWS 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 the JavaScript Web Speech API.
* 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 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.

**CLIENT COMPONENTS**

The Client-Side of the application be based on the React framework. The Calendar interface will be structured along the date-fns package installed through NPM. Though the actual Calendar library is simple, it can be expanded upon to accommodate the entire screen. When the user first logs on to the application, the first page that will be displayed will be the Calendar Interface. Every month will be accessible, however the current month will be displayed first, and the current day will be highlighted. Each individual day within a month is interactable through a click/touch by the user. Months can be navigated through by using the left/right arrows on either side.

Table

Description automatically generated

When the user clicks on a day, that day’s page will display. Any information available from the database regarding that day will be queried and returned by the database. If it is a day that has already passed or a day that has not come yet, the ability to record audio will not be available, only for the day that it is right now. When the user clicks on the current day, any recordings that exist for the day already will be displayed, any existing time that remains will be displayed, and a button will be available for the user to record more segments.

All information about a day will be retrieved from the backend API and populated in various React components that make up the application. The recording is created on the “Record” page, where the recording itself along with its’ creator, tags, date of creation, and transcription are posted to the backend and indexed by the MySQL database hosted on AWS.

**Use Case 1**

Ben wants to make a 60 seconds account

1. Ben accesses the application page via 60seconds.io
2. Ben clicks the “create a new account” button on the home screen
3. Ben is redirected to the account creation page
4. Ben clicks on the “sign in with Google” button
5. Ben is redirected to the secure Google sign in page where he is prompted to link his Google account to 60 seconds
6. Ben accepts the terms on the Google sign in page, his Google account is now linked to a 60 Seconds account
7. He is redirected back to 60Seconds.io, now logged into his account

Diagram

Description automatically generated

*Figure 2. Sequence Diagram of the Sign Up Process*

**Use Case 2**

Cameron wants to login to his 60Seconds account

1. Cameron opens the home page on 60seconds.io
2. Cameron clicks the “sign in” button on the home screen
3. Cameron is redirected to the login page
4. Cameron clicks on the “sign in with Google” button
5. Cameron is redirected to the secure Google sign in page which immediately redirects him
6. back to the app if he’s already logged into his Google account

A session cookie is created to make his login persistent

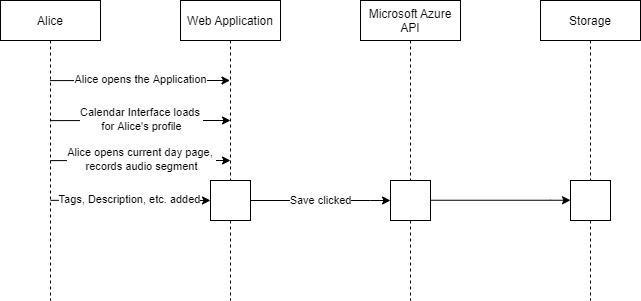
**Graphical user interface

Description automatically generated**

*Figure 3. Sequence Diagram of the Login Process*

**Use Case 3**

User Story: Alice wants to create an audio segment for the day and set a tag

1. Alice signs in to her 60 seconds account, which navigates her to the Calendar Interface
2. Alice clicks on the highlighted current day, which brings up that days page
3. Alice clicks the ‘Record’ button and records for 60 seconds
4. Once finished recording, reviews and edits the transcription of her recording, optionally tags her location or other custom tags (any text she wants).
5. Alice hits the ‘Save’ button on the day’s page, and her input (audio file, tags, etc) is sent from client to the backend of the application.
6. The backend of the application creates a record in the database for the recording, populating the date, creator, URL, tags, and transcript field automatically. 

*Figure 4. Sequence diagram of making a recording and setting a tag*

**Use Case 3**

Tom wants to export his audio recording.

1. Tom opens his computer and logs into his 60 Seconds account.
2. Tom navigates to the calendar page and selects a date.
3. Tom clicks on the audio download button on the date he selected.
4. The recording returned from the database is saved in his local storage in MP3 format

.

Diagram

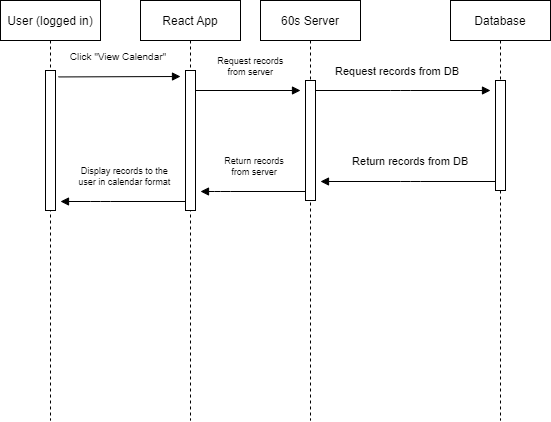
Description automatically generated

*Figure 5. Sequence Diagram of recording downlaoding*

**Use Case 5**

Jessica wants to view her previous recordings

1. Jessica logs into her account
2. Jessica opens the homepage
3. Jessica clicks on the “view calendar” button
4. Jessica is then able to click on each individual day and see the recordings she’s made as well as their respective tags and transcriptions

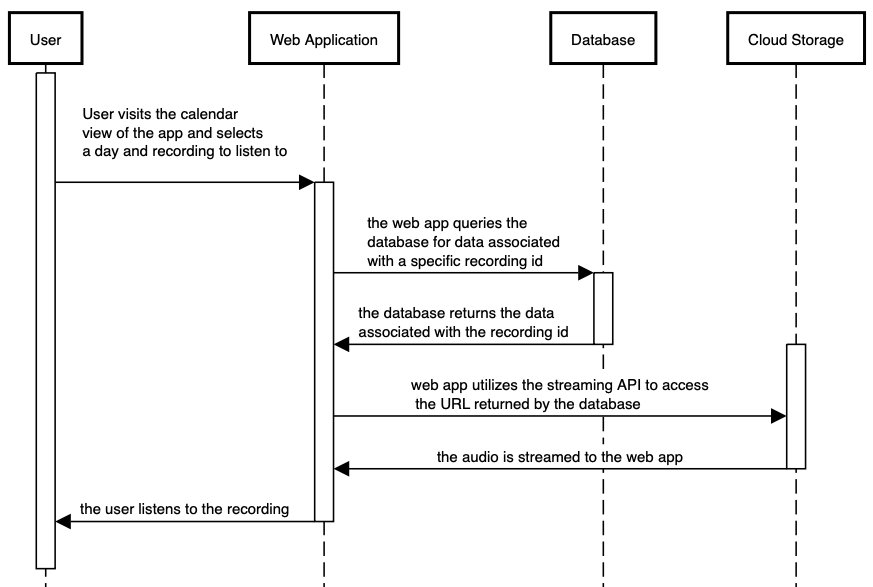
****

*Figure 6. Sequence Diagram of a User Viewing Their Calendar*

**Use Case 6**

Jeremy wants to listen to his recording from yesterday.

1. Jeremy logs into 60 Seconds
2. He visits the calendar view and selects the previous day on the calendar
3. He is presented with an audio player for the corresponding recording and
4. He selects the play button and listens to his recording.

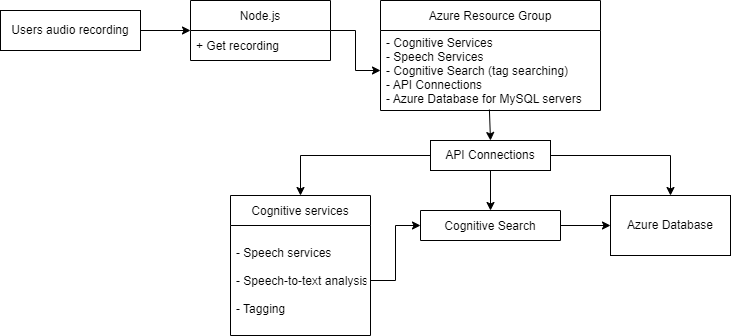


*Figure 7: Sequence diagram of a user choosing and listening to a previous day’s recording*

**SERVER COMPONENTS**

All server components for the 60 Seconds web application will be hosted within an AWS account. The initial backend for the application will be housed in a Node.js framework.

* Identification
  + 60 Seconds - Backend Server
* Type
  + Backend and server components using Node,js as a framework for application development.
* Purpose
  + Processing power for all user audio recordings and speech to text transcriptions and their corresponding APIs. This also includes the processing power for text analysis.
  + Providing an API for access to database components.
  + Login credential management and database storing.
* Function
  + Initial account creation and storing of login credentials for each access.
  + The user records audio for their day, server takes this audio file and using JavaScript Web Speech API transcription to provide what the user logged into text for user editing and manipulation.
  + After initial recording and transcription, the server makes a second API call to an AWS text analysis tool which breaks down the text transcription into designated components. This consists of Tagging dates, locations, times, and events to that audio recording.
    - Using the tags created there will be functions in place i.e. addToCalendar() to place these into their designated location within the calendar.
* Dependencies
  + AWS
    - Resource group
      * Cognitive Services
      * API Connections
      * Database for MySQL servers
  + Node.js Framework
* Interface
  + The communication with the server will be done with data flowing from Node.js backend to AWS and the corresponding services depending on the data provided.
* Processing
  + Speech-to-text
  + Text analysis
  + Cognitive Search
* Data
  + Audio recorded by the user formatted in .mp3
  + User credentials (secured)

****

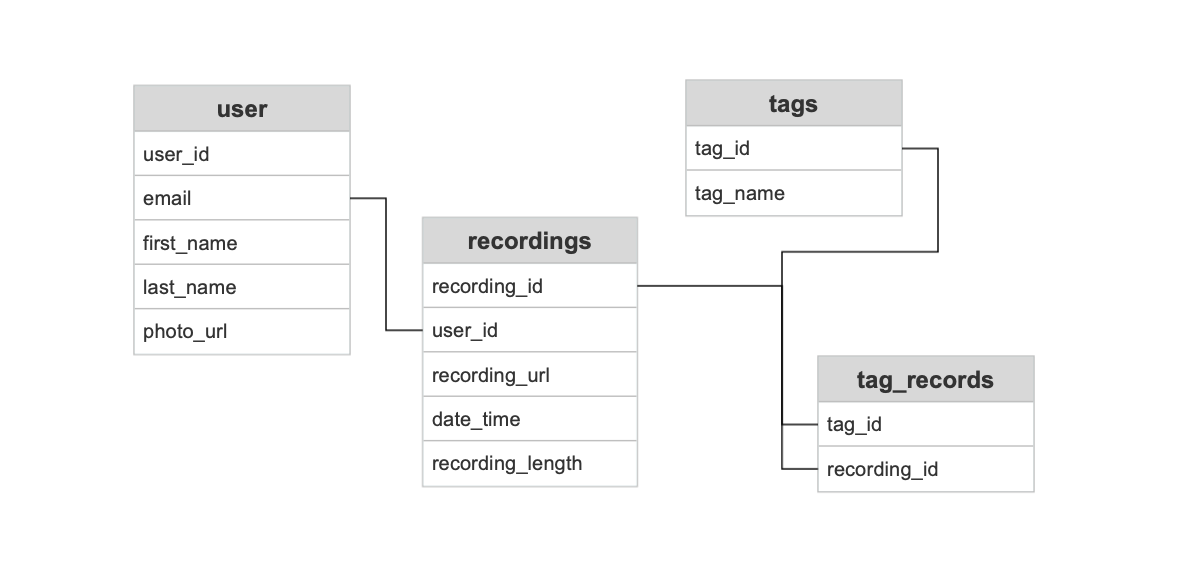
*Figure 8. Server Class Diagram*

**DATABASE COMPONENTS**

The database will structure the data and be utilized throughout the entire application. Since the application will be hosted on the AWS, we will be using a basic AWS MySQLdatabase to store data. The chosen database service provides seamless integration with the AWS EC2 instance, where the application itself is hosted. There will be multiple tables needed to store data, specifically to store the following:

* user information (username, password, etc.) defined in the login/register components.
* Tag information associated with a given recording\_id
* Recording information, including recording\_id, the creator of the recoridng, URL of the recording, and the transcription of the recording

The database objects and a description of their associated data points are described below.

*Figure 9. Entity relation diagram of the initial database design*

* Table 1: user
  + user\_id - a string representing a unique identifier for each user’s account
  + email - a string representing an email address pulled from the Google Identity Services API2 object returned at sign-on
  + first\_name - a string representing the user’s first name, pulled from the Google Identity Services API object returned at sign-on
  + last\_name - a string representing the user’s last name, pulled from the Google Identity Services API object returned at sign-on
  + photo\_url - a string representing the link to a user’s profile photo, pulled from the Google Identity Services API object returned at sign-on
* Table 2: recordings
  + recording\_id - a string representing a unique identifier for each recording created by a user
  + user\_id - the identifier of the user who created the recording, this relates each entry in the recordings table to a user
  + recording\_url - a string representing the link to the recording, located in the cloud storage solution.
  + datetime - the a datetime object representing when the recording was made and uploaded to the server
  + recording\_length - a time object representing the length of the recording in seconds and milliseconds
* Table 3: tags
  + tag\_name - a string representing the name of the user created/chosen tag
  + tag\_id - an integer uniquely identifying each tag name
* Table 4: tag\_records
  + tag\_id - an integer uniquely identifying each tag name
  + recording\_id - an integer uniquely identifying each recording

The user table holds information about a user’s account. There will be a unique entry for each user who registers with the application. User\_id is the primary key and is assigned at the creation of the entry. email, first\_name, last\_name, and photo\_url are all pulled from the sign-on object returned from the Google Identity Services API that the application will use to manage user sign-on.

The recordings table will hold data associated with each recording created by users. The primary key, recording\_id, will be used to uniquely identify each recording. Other data points include the user\_id associated with each recording, a link to the recording itself, and other metadata including the date, time, and length of recording.

The tags associated with each recording will be organized in two separate tables. The first table, entitled tags, is simply an assignment of an unique identifier to each tag name. The second table, tag\_records, associates the tag\_id to a recording\_id.

**REFERENCES**

2 <https://developers.google.com/identity/gsi/web>

https://blog.logrocket.com/react-calendar-tutorial-build-customize-calendar/