

Awakening Documentation

by Adam Scalzo

Awakening:

Welcome to the Awakening documentation! Awakening is a simple yet effective user-subscription service web application where we grant users access to our soothing soundtracks to guide them through meditations, help them sleep, or help them stay focused! We offer monthly or yearly billing periods (with different prices), and three collections of soundtracks for various purposes.

This documentation will include information categorized in the order of:

- Front End UI Documentation
- Back End Technology Documentation
- Database Schema
- Formal Test Plan

Front End UI Documentation:

Landing Page: Basic landing page. Directs user to login or register.

Welcome to Awakening

Register

Log in

Registration Page: Where you can register, clearly. Just requires unique username.

Registration

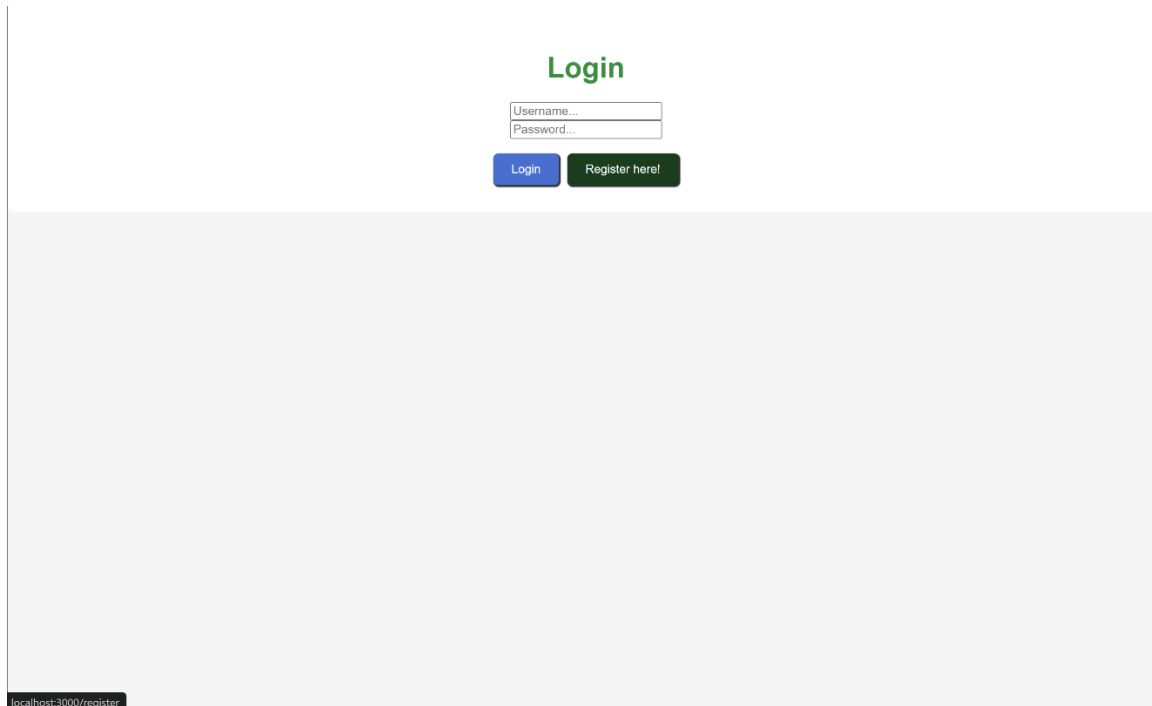
Username

Password

Register!

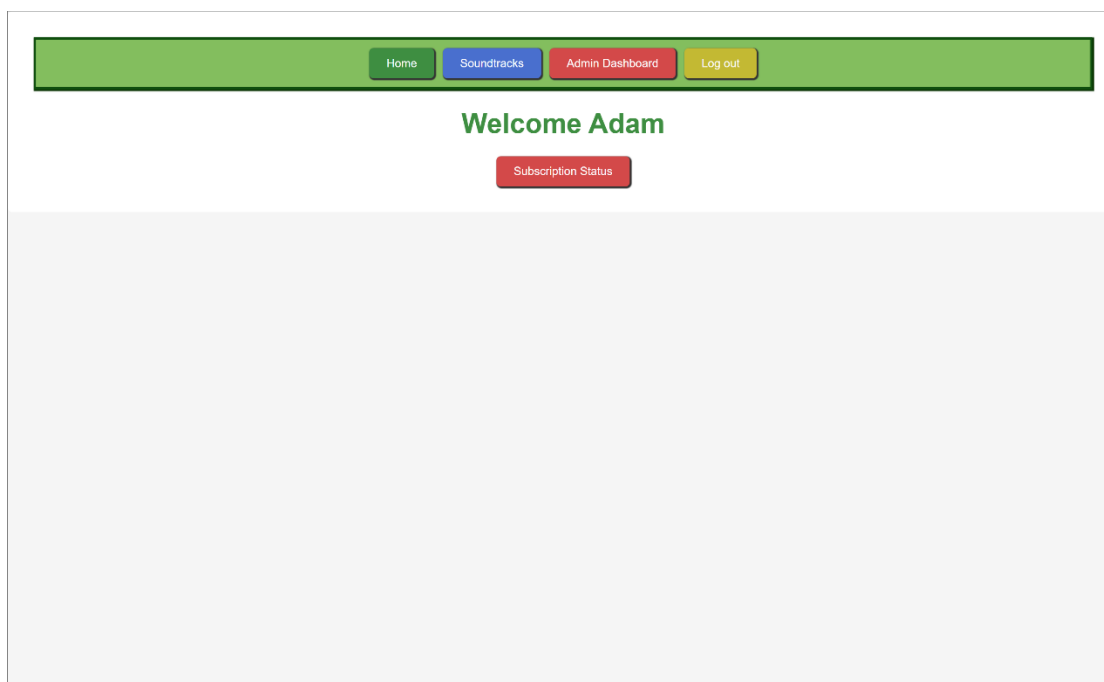
Log in here!

Login Page: Login page! Must input a valid username and password to be successfully logged in and redirected to the home page.



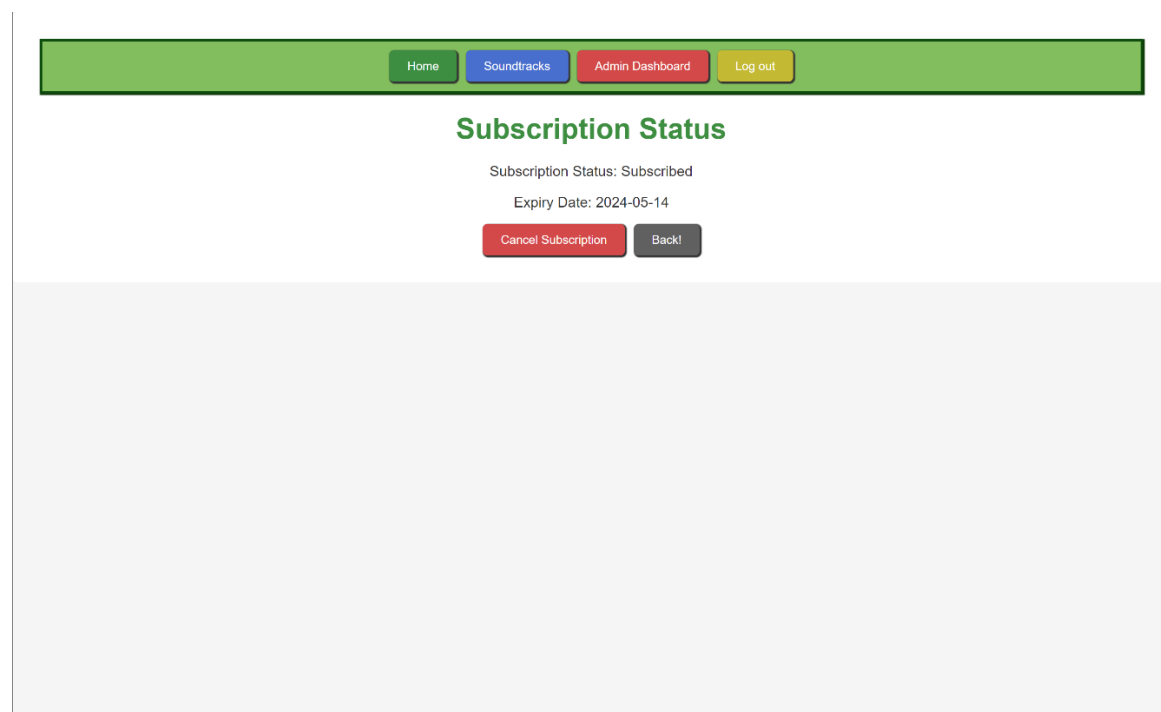
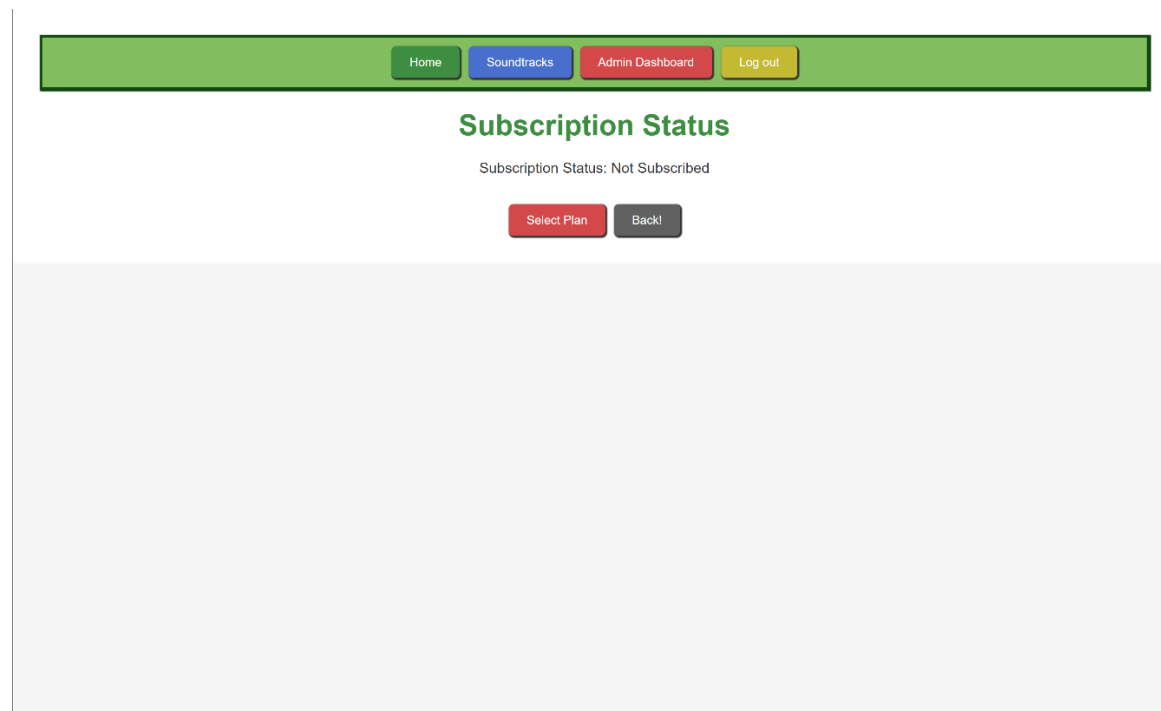
The screenshot shows a web browser window with a light gray background. At the top center, the word "Login" is displayed in a green, sans-serif font. Below it are two stacked input fields: the top one is labeled "Username..." and the bottom one is labeled "Password...". Underneath the input fields are two buttons: a blue button labeled "Login" and a dark green button labeled "Register here!". In the bottom left corner of the browser window, a small address bar shows the text "localhost:3000/register".

Home Page: Links to each of the categories of meditations, subscription page if they are unsubscribed, and some other things. General eye candy as well as main table of contents.



The screenshot shows a web browser window with a light gray background. At the top, there is a green navigation bar with a black border. Inside the bar are four buttons: "Home" (green), "Soundtracks" (blue), "Admin Dashboard" (red), and "Log out" (yellow). Below the navigation bar, the text "Welcome Adam" is displayed in a green, sans-serif font. Underneath this text is a red button labeled "Subscription Status". The rest of the page is a large, empty light gray area.

Status Page: Accessible directly from the homepage, this page will display whether or not the logged in user is subscribed. If they are, it will show the type of subscription, the expiry date, and an option to cancel the subscription. If they are not subscribed, it will simply display “Subscription: Inactive” and render a button to the subscription checkout process.



Subscriptions Page: User can select to purchase a subscription for a month, or a year.

Select your Plan!

Monthly

\$19.99/month

Full Access to:

- Sleep soundtracks
- Focus soundtracks
- Meditation soundtracks

Select!

Yearly

\$179.99/year

Full Access to:

- Sleep soundtracks
- Focus soundtracks
- Meditation soundtracks

Select!

Back!

Soundtracks Page: User can select one of the three categories of soundtracks to listen to depending on their desires. Meditate sounds, sleep sounds, and focus sounds are the current options. If selected, the user will be redirected to the category page as selected, where they can start listening!

[Home](#) [Soundtracks](#) [Admin Dashboard](#) [Log out](#)

Pick a Category!

Meditate

Moving sounds to open your eyes.

View Tracks

Focus

Binaural beats to help you focus.

View Tracks

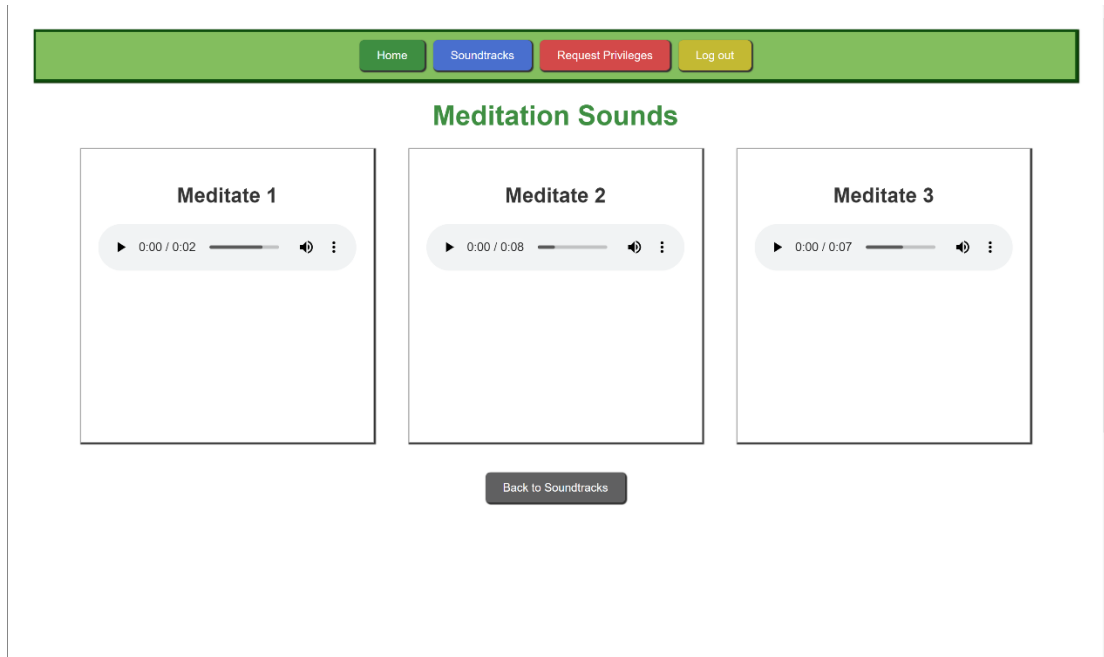
Sleep

Soothing sounds to help you sleep.

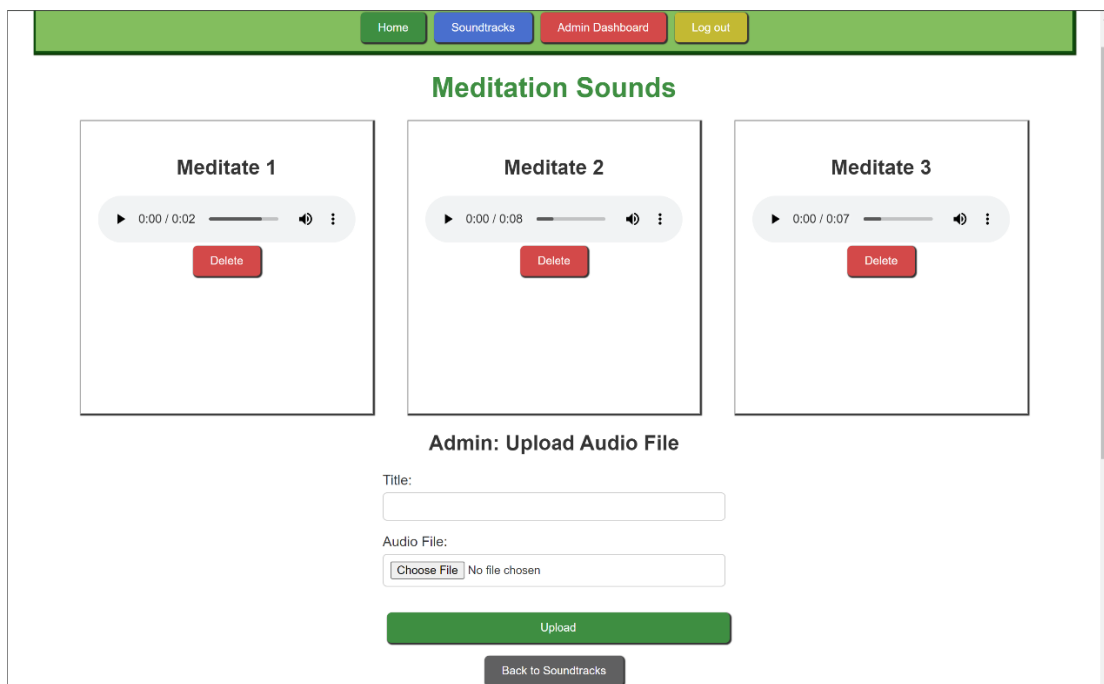
View Tracks

Back!

Focus, Meditate, and Sleep Pages: This is where our users can listen to our soundtracks! If they are an administrative user, they will be granted soundtrack adding and deleting accessibility. If not, they will just be able to listen to the current soundtracks available.



And the same page from an administrator's point of view:



Admin Request Page: Only viewable by non-admin users: This page allows users to request administrative privileges so they can access the admin dashboard. Simply for testing and building this project, I will add a password section so that it's not just 'free admin privileges'.

Admin Request Form

Become Admin

Back!

Admin Dashboard Page: Voila, useful information about all of our users! Here you will find a table with a column for each registered user. This table shows each user's user ID, username, subscription status, and admin status, followed by a 'View' button for each user. Upon clicking the view button, the administrator will be routed to the:

HomeSoundtracksAdmin DashboardLog out

Admin Dashboard

User ID	Username	Is Subscribed	Is Admin	View
127	Adam	Yes	Yes	View
128	Jeffrey	Yes	No	View
135	lilly	Yes	Yes	View
140	henry	No	Yes	View

User-View Page: Here, an admin can see the same information about the specified user as seen in the admin dashboard, except they can manually toggle the user’s admin privileges and subscription status, as well as deleting a user entirely. Beneath this section is the “subscription history” section, where information regarding the selected user’s subscription history will be pulled from the database and displayed.

[Home](#) [Soundtracks](#) [Admin Dashboard](#) [Log out](#)

User Details

User ID	Username	Subscription Status	Is Admin
127	Adam	Active	Yes
Back!	Delete User	Remove Subscription	Remove Admin

Subscription History

Start Date	Expiry Date	Plan	Payment	
4/14/2024	Indefinite	Admin	N/A	Delete
4/14/2024	5/14/2024	Monthly	\$19.99	Delete
4/11/2024	Indefinite	Admin	N/A	Delete
4/4/2024	4/4/2025	Yearly	\$179.99	Delete
4/2/2024	Indefinite	Admin	N/A	Delete
4/2/2024	Indefinite	Admin	N/A	Delete
4/2/2024	5/2/2024	Monthly	\$19.99	Delete
4/2/2024	Indefinite	Admin	N/A	Delete

Checkout Page: Just a simple page with a routing button to Stripe’s checkout process. This will either be /checkout/1 if it’s a monthly plan, or /checkout/2 if it’s a yearly plan.

Checkout for a month!

[Proceed to Checkout](#) [Back!](#)

Stripe Checkout Integration: This is Stripe's page for handling the actual transaction process. It is a standard checkout form with good error handling.

← Adam Scalzo TEST MODE

Subscribe to Monthly Subscription
\$19.99 per month
Monthly Subscription to Awakening Services

Pay with link ➡

Or pay with card

Email

Card information

1234 1234 1234 1234

MM / YY CVC

Cardholder name

Full name on card

Country or region

United States

ZIP

☒ Save my info for 1-click checkout with Link
Securely pay on Adam Scalzo and everywhere Link is accepted.

(201) 555-0123

link · More info

Subscribe

By confirming your subscription, you allow Adam Scalzo to charge you for future payments in accordance with their terms. You can

Success Page: This page just shows up to inform the user that the checkout attempt was successful, and provides a link directly back to Awakening's home page:

Payment Successful!

Thank you for your subscription.

[Back to Homepage](#)

Cancel Page: Similar to the success page, this page appears if the user decides to cancel the checkout process, and provides a link directly back to the home page.

Payment Canceled

Your subscription was not processed.

[Back to Subscription](#)

Back End Technology Documentation:

Front-end technology: Awakening is a web application created using React JS as it's main framework. React is what all of the front-end is built on, other than the obvious incorporation of CSS.

Back-end technology: For the underlying mechanics behind the website, I use a couple things:

1. Express Js (more specifically CORS Express) is used on the server's side to listen for a connection to the client and gather JSON information from the request.
2. I used a mix of localStorage flags and database retrievals to make my own system for route protection. Subscription status is volatile to change, so I always pull that from the database when needed, though things like username, userid, and admin status remain pretty static, so I just retrieve them with localStorage.
3. MySQL is my Database Management System (DBMS) of choice. The database contains all of the user specific information, like account ID #, username, password, privileges, account status, etc.
4. I used Bcrypt as my password hashing for storage in the database. Bcrypt is an industry standard hashing and salting algorithm.
5. I used a Stripe test environment for my checkout processes. I created two products: one for monthly and one for yearly. Following through checkout successfully updates the database using a webhook.

To Get Running

1. Open the awakening_client directory and run "npm start". This will bring up the client-side front-end.
2. Open the awakening_server directory and run "npm start". This will start the server.
3. Open the project directory "CSIS3126_Scalzo" and run "stripe listen --forward-to localhost:3001/webhook". This tells stripe to listen for the checkout event, and forward to the server-side "webhook" endpoint, which handles database and flag updates post checkout.

Database Schema

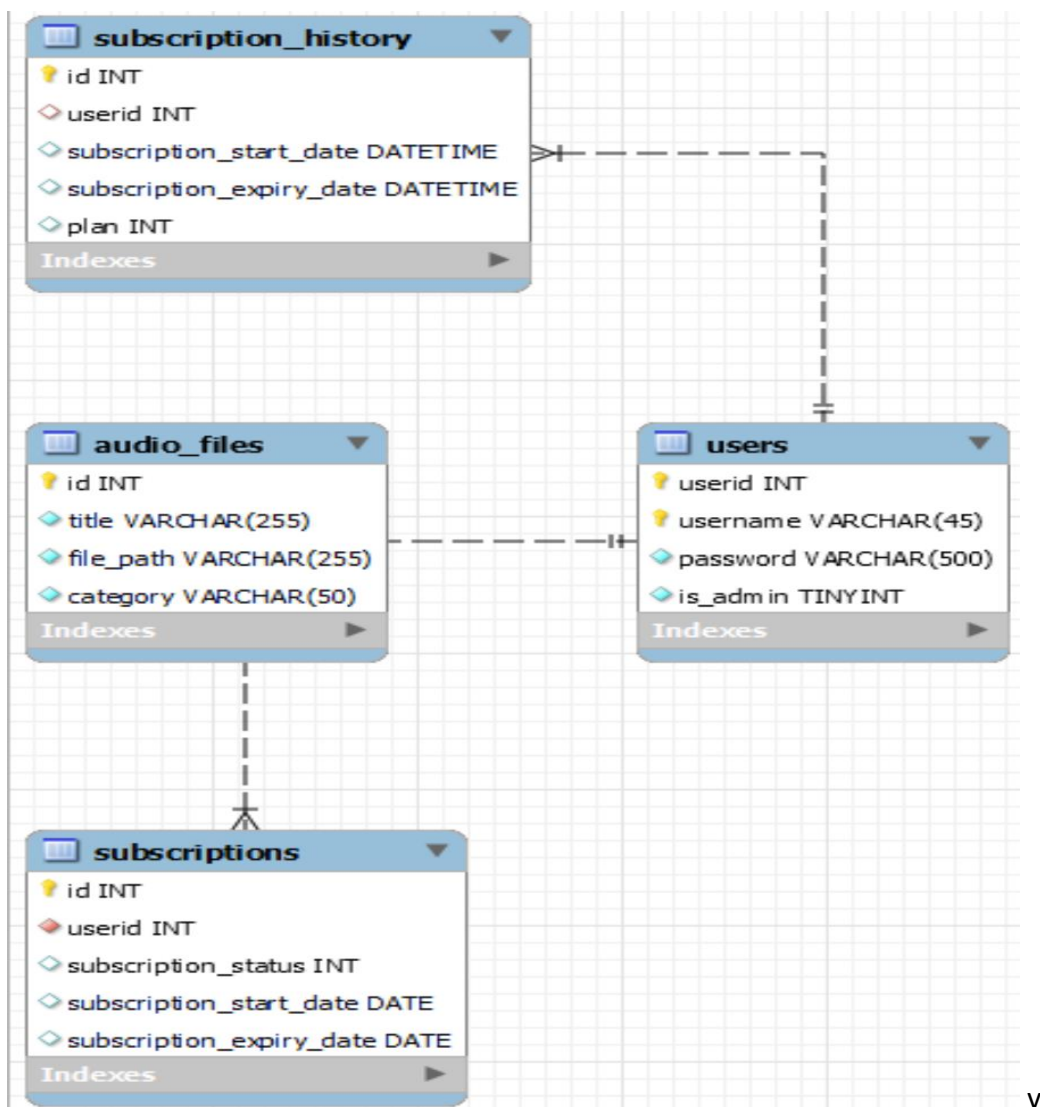
In a MySQL Database I have 4 tables:

Audiofiles: Stores the id, name, file path, and category of each audio file.

Subscription_history: Stores the id, userid, subscription start and expiry dates, and plan.

Subscriptions: Stores the id, userid, subscription start and expiry dates, and sub status

Users: Stores userid, username, password, and is_admin status.



Formal Test Plan Documentation:

Unit Tests:

For functionality of my application, the client-side has many calls to the server. I have abstracted the following server calls for organization and unit testing using Jest:

1. Fetching user
 - a. Asserted that the fetch function was properly called.
 - b. Asserted that the result was expected and matches the mocked user data.
2. Toggling admin status
 - a. Asserted that the result matches the expected user data.
3. Toggling subscription status
 - a. Asserted that the function called with correct URL and options.
 - b. Asserted that the result matches expected subscription status and user info.
 - c. Tested the error messages upon a failure to toggle subscription status.
4. Fetching subscription history
 - a. Asserted that the function was called correctly with proper URL and options.
 - b. Asserted that the history results match our mocked expectations.
 - c. Asserted a test case that failed and matched our expected error message.
 - d. Asserted a network error when the response is undefined.
5. Deleting subscription history
 - a. Assert that function was called properly with correct URL and userID.
 - b. Asserted a failure to verify error message displaying properly.
 - c. Asserted an undefined response resulting in "network error" message.
6. Registering a new user
 - a. Assert that function was called properly with correct URL and options.
 - b. Tested the case of a successful user registration,
 - c. Checked to verify that post update, the user info is in the DB.
 - d. Test case for an exception during registration/if username already taken.
7. Authenticating and logging in a user
 - a. Asserted the proper call of the function with correct URL and options.
 - b. Checking that it throws an error when login failures (i.e. no password, wrong info)
8. Deleting a user
 - a. Asserted the function being called correctly with proper URL and userID.

- b. Asserted the failure to delete a user.
 - c. Asserted a network error on an undefined response.
- 9. Creating a stripe checkout session
 - a. Assert that we properly call function with correct URL.
 - b. Assert that we create the checkout session successfully with correct plan.
 - c. Asserted that the redirectToCheckout() function works properly, and routing to Stripe works
- 10. Checking out for a one-month subscription
 - a. Asserted that it is called correctly with plan parameter.
 - b. Asserted the successful transaction, and event occurrence.
 - c. Tested the case of transaction failure.
- 11. Checking out for a year subscription
 - a. Asserted that it is called correctly with plan parameter.
 - b. Asserted the successful transaction, and event occurrence.
 - c. Tested the case of transaction failure.

End to End Tests:

I performed my end to end tests manually, as given my interface, this was more efficient and exhaustive than trying to write automated tests. I performed the following manual tests for Awakening.

1. Fully tested the registration page by registering new users, attempting to register users that already exist to get errors, attempting to register without filling in fields.
2. Fully tested the login page by trying to login with correct information, no information, and incorrect information.
3. Verified the database and flag updates after:
 - a. Subscribing via month plan.
 - b. Subscribing via yearly plan.
 - c. Subscribing manually via admin dashboard.
 - d. Unsubscribing via status page.
 - e. Unsubscribing via admin dashboard.
 - f. Toggling administrative privileges.

4. Logged into a non-admin user after using that admin user to enable their subscription services.
5. Attempted to access all of my “protected routes” without being signed in or subscribed.
6. Attempted to add non-mp3 files to my audio files directory.
7. Verified the subscription_history logs actually works.
8. Verified that removing admin status from the signed in user kicks them out of the dashboard instantly.
9. Verified that deleting the current user that is signed in kicks that user back to the login page.

I also used GitHub Dependabot to detect security vulnerabilities, and approved some of it's pull requests after reviewing them to reduce the about of vulnerabilities on my app.