Waveform Project Plan

Elevator Pitch

Waveform is a music reviewing platform where users can share their insights on tracks, albums, EPs, and artists, and get tailored recommendations based on what genres and music they tend to rate highly. Waveform is designed to foster a community of music enthusiasts through users sharing their thoughts on various music genres and albums, and helping people find new music to listen to based on what they enjoy, as well as giving critics a place to spotlight their reviews and publications about music.

Personas

Emma Carlisle - The Music Enthusiast

- **Age**: 27
- **Occupation**: Freelance Writer
- **Location**: Brooklyn, NY
- **Motivations**: Emma loves diving deep into indie and experimental music. She uses Waveform to find hidden gems, write detailed reviews, and connect with like-minded fans.
- **Challenges**: Struggles with platforms that prioritize algorithms over user-driven content and often finds existing sites lacking in-depth reviews.
- **Needs**: Easy to use tools for creating long-form reviews, features for engaging with followers, and personalized music recommendations.

Dylan Harris - The Casual Listener

- **Age**: 22
- **Occupation**: College Student
- **Location**: Colorado suburbs
- **Motivations**: Dylan wants quick access to trending songs and artist suggestions while occasionally sharing short reviews.
- **Challenges**: Overwhelmed by endless options on streaming platforms, and prefers a simple, intuitive interface for quick music discovery.
- **Needs**: Filters for genre and mood-based browsing, a mobile-friendly design, and a quick review option.

Michael Simmons - The Critic

- **Age**: 34
- **Occupation**: Professional Music Critic
- **Location**: Los Angeles, CA

- **Motivations**: Michael uses Waveform to share in-depth critiques and establish credibility with a wider audience.
- **Challenges**: Frustrated by platforms that fail to highlight professional reviews and lacks tools for formatting and analytics.
- **Needs**: A dedicated section for critic reviews, media embedding options, and tools to track engagement metrics.

Functional Requirements

- **Home Page**
  - Showcase a dynamically updating section for trending reviews and trending albums on the homepage right off the bat.
- **About Page**
  - Include an overview of Waveform, and its usage to connect music lovers through thoughtful reviews.
  - Add links to social media and other helpful links.
- **Explore Page**
  - Allow users to browse and filter reviews by genre, artist, or ratings.
  - Provide curated lists of trending and highly rated music tracks and albums.
- **Review Submission**
  - Enable users to write, edit, and delete reviews of tracks and albums.
  - Allow ratings, art, genres, and tags for better categorization.
- **Recommendation System**
  - Offer personalized music recommendations based on user preferences and review activity.
  - Include a “Pick For Me” mode with random suggestions for adventurous listeners.
- **Profile Pages**
  - Display user profiles with their reviews, favorites, and followers/following lists.
  - Include a bio section where users can share a text blurb talking about themselves.
- **Critic Reviews Section**
  - Highlight professional reviews separately from user-generated content to maintain credibility.
- **Community Engagement**
  - Allow users to comment on reviews and start discussions.
- **Admin Panel**
  - Grant admins the ability to manage user accounts, moderate reviews, and feature specific content on the homepage.
- **Mobile-First Design**
  - Ensure the platform is optimized for mobile devices, providing seamless navigation and features.
- **Authentication**
  - Allow users to register, log in, and manage their accounts securely.

Non-Functional Requirements

- **Performance**
  - Pages should load within 2 seconds, even with high traffic.
- **Validation and Security**
  - Include robust form validation for all inputs (e.g., registration, review submission).
  - Encrypt sensitive data and ensure secure communication using HTTPS.
- **Stability and Bug-Free Operation**
  - Conduct rigorous testing at every development stage to minimize bugs and ensure a smooth user experience.
- **Documentation**
  - Provide a comprehensive README file with setup instructions and configuration details for developers.
- **Customizability**
  - Provide customizable dark/light themes for users.
- **Responsive Design**
  - Maintain consistency and usability across devices, including desktops, tablets, and smartphones.
- **Scalability**
  - Design the system to handle future growth, including expanding the database and increasing user activity.
- **User Feedback Integration**
  - Implement a feedback page to collect suggestions and continuously improve the platform and gather and fix bugs.

Low-Fidelity Mobile-First GUI Mockup

The GUI Mockup can be viewed within the “figma-mockup” folder in the project Github Page.

Use Cases

- As a user, I can edit or delete my reviews.
- As a user, I can rate songs and albums without leaving a detailed review.
- As a user, I can search for songs, albums, or artists using a search bar with autocomplete functionality.
- As a user, I can filter reviews by genre, rating, or release year to find content that matches my preferences.
- As a user, I can see a history of the reviews I’ve written on my profile page.
- As a user, I can view a feed of activity from the users I follow, including their reviews and comments.
- As a user, I can see which artists, songs, or albums are currently trending on the platform.
- As a user, I can receive notifications when someone comments on my review or follows my profile.

- As a user, I can like or comment reviews to show appreciation for other users' insights.

- As a critic, I can view analytics about the reach and engagement of my reviews to track their impact.
- As a critic, I can link my reviews to external sites or social media for additional visibility.

- As an admin, I can flag inappropriate reviews, comments, or user profiles for moderation.
- As an admin, I can manage user roles, promoting users to critics or demoting critics if necessary.
- As an admin, I can feature specific reviews, playlists, or users on the homepage.
- As an admin, I can generate reports on site activity, including most-reviewed genres and trending artists.

Database Diagrams (PostgreSQL)

(NOTE: This is subject to change!)

Users Table

- **user_id: SERIAL, PRIMARY KEY**
- **username: VARCHAR(50), UNIQUE, NOT NULL**
- **password: VARCHAR(255), NOT NULL**
- **email: VARCHAR(100), UNIQUE, NOT NULL**
- **created_at: TIMESTAMP, DEFAULT CURRENT_TIMESTAMP**

Reviews Table

- **review_id: SERIAL, PRIMARY KEY**
- **user_id: INT, FOREIGN KEY → Users(user_id), NOT NULL**
- **song_id: UUID, FOREIGN KEY → Songs(song_id)**
- **album_id: UUID, FOREIGN KEY → Albums(album_id)**
- **review: TEXT, NOT NULL**
- **rating: NUMERIC(3, 1), CHECK (rating BETWEEN 1.0 AND 10.0)**
- **created_at: TIMESTAMP, DEFAULT CURRENT_TIMESTAMP**

Comments Table

- **comment_id: SERIAL, PRIMARY KEY**
- **review_id: INT, FOREIGN KEY → Reviews(review_id), NOT NULL**
- **user_id: INT, FOREIGN KEY → Users(user_id), NOT NULL**
- **comment: TEXT, NOT NULL**

- **created_at: TIMESTAMP, DEFAULT CURRENT_TIMESTAMP**

Songs Table

- **song_id: UUID, PRIMARY KEY**
- **title: VARCHAR(100), NOT NULL**
- **artist_id: UUID, FOREIGN KEY → Artists(artist_id), NOT NULL**
- **album_id: UUID, FOREIGN KEY → Albums(album_id)**
- **release_date: DATE**
- **genre: VARCHAR(50)**

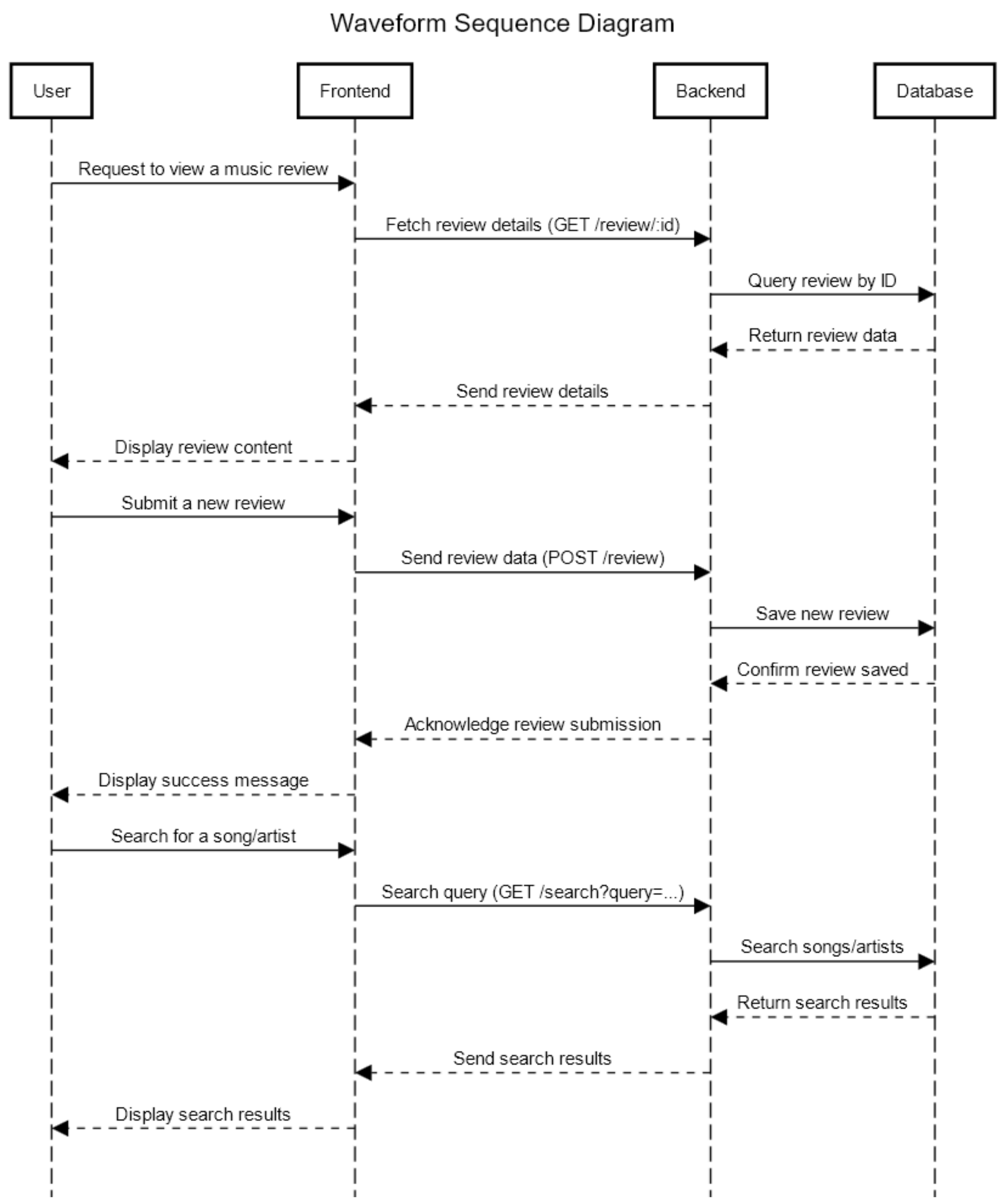
Albums Table

- **album_id: UUID, PRIMARY KEY**
- **title: VARCHAR(100), NOT NULL**
- **artist_id: UUID, FOREIGN KEY → Artists(artist_id), NOT NULL**
- **release_date: DATE**
- **genre: VARCHAR(50)**

Artists Table

- **artist_id: UUID, PRIMARY KEY**
- **name: VARCHAR(100), NOT NULL**
- **bio: TEXT**

Sequence Diagram for Reviews and Searching



JSON Web Service Endpoints

(NOTE: This is subject to change!)

Authentication and User Management

- **POST /api/register**
  Input: { username: String, password: String, email: String, firstName: String, lastName: String }
  Output: { success: String, userId: Int }

Reviews

- **POST /api/review**
  Input: { songId: String, albumId: String (optional), reviewContent: String, rating: Number (1-10), userId: Int }
  Output: { success: String, reviewId: Int }
- **GET /api/review/{reviewId}**
  Output: { reviewId: Int, userId: Int, songId: String, albumId: String (optional), reviewContent: String, rating: Number, timestamp: String }
- **PUT /api/review/{reviewId}**
  Input: { reviewContent: String, rating: Number (1-10) }
  Output: { success: String }
- **DELETE /api/review/{reviewId}**
  Output: { success: String }

Comments on Reviews

- **POST /api/review/{reviewId}/comment**
  Input: { userId: Int, commentContent: String }
  Output: { success: String, commentId: Int }
- **GET /api/review/{reviewId}/comments**
  Output: [{ commentId: Int, userId: Int, commentContent: String, timestamp: String }]
- **DELETE /api/comment/{commentId}**
  Output: { success: String }

Songs

- **POST /api/song**
  Input: { title: String, artist: String, albumId: String (optional), releaseDate: String, genre: String }
  Output: { success: String, songId: String }
- **GET /api/song/{songId}**
  Output: { songId: String, title: String, artist: String, album: String (optional), releaseDate: String, genre: String, averageRating: Number }

Albums

- **POST /api/album**
  Input: { title: String, artist: String, releaseDate: String, coverImage: String (URL), genre:

String }
Output: { success: String, albumId: String }

- **GET /api/album/{albumId}**
Output: { albumId: String, title: String, artist: String, releaseDate: String, genre: String, averageRating: Number, tracklist: [{ songId: String, title: String }] }

Artists

- **GET /api/artist/{artistId}**
Output: { artistId: String, name: String, genre: String, biography: String, albums: [{ albumId: String, title: String }], songs: [{ songId: String, title: String }] }

Users

- **GET /api/user/{userId}**
Output: { userId: Int, username: String, reviews: [{ reviewId: Int, songId: String, albumId: String (optional), rating: Number }] }

Recommendations

- **GET /api/recommendations**
Input: { userId: Int }
Output: [{ songId: String, title: String, artist: String, genre: String, reason: String }]

Architectural Stack

- **Frontend**: React
- **Backend**: Node.js with Express
- **Database**: PostgreSQL
- **Hosting**: Azure for front/backend, or Vercel, or Locally hosted (to be determined)

Testing Plan

I plan to use a few testing frameworks. **Jest** will be used for the model layer to test data structures and database operations, confirming that everything runs smoothly. For the controller layer, **Nock** will be used to test API endpoints, ensuring correct handling of inputs and outputs, along with proper exception management. In the view layer, **Cypress** will be used to test the user interface, making sure that all interactive elements like buttons and links work as intended.

User Acceptance Test Plan (Subject to change, but generally how the flow will go)

1. **Login:**
    - Username: Test-User
    - Password: testpassword236
2. **Setup:**
    - When logged in, browse through the page and ensure you can see various reviews and songs. Then go to your profile with the profile button.
    - When at your profile, set a bio.
3. **Making Review:**
    - From here, click on the review button to create a review. This will take you to the review creation page.
    - You will have the option to create a new one or use an existing song or album. Search for “Test & Test 2 - Test Song” and select it when it pops up. This will fill in the data.
    - Fill in “8” for the rating, and “This is a test review.” for the review. Set the visibility to public, then click submit.
    - You should be sent to the song page, and see your review on the page.
4. **Song Page:**
    - While on the song page, check to see if you can see friend reviews directly under your review. You should see a review from “Test-User-2” that gave the song a 9/10 with the review “Awesome song!”.
    - You should also be able to see critic reviews.
    - Click on the Test-User-2 name to go to their profile, and attempt to change their bio. You should not be allowed to do this.
5. **Recommendations**
    - Go to the recommendation tab, and view the recommendations available. These recommendations are dynamically created, so they won’t be exactly the same every time, but ensure that none of them are “Test & Test 2 - Test Song” as you shouldn’t be recommended songs you’ve already heard.
    - There should be a song, EP, album, and artist section here.
6. **Creating an album entry**
    - Click on the review button to create a review.
    - This time, click on the “add new” option to add a new thing. Select album.
    - Add 2 artists, “Test” and “Test 3”, and the album name as “Test LP”.
    - Add genres “EDM” and “Trap”, set the release date as “1/13/2025”, and click finalize. Do not worry about setting an album art.
    - Finally, create an album review for this album. Give it the rating “7” and a review “This is my new album review, and here’s a review!”
    - You should be redirected, and see the album page fully setup with your review included as the first one.