Harmonic Links

Team 6 - CS 4485.0W1 Group Project

Srujana Ayyagari,

Sunaina Ayyagari,

Nate Christie,

Kevin Dang,

Alec Ibarra,

Benjamin Wowo

Supervisor: Professor Sridhar Alagar

Course Coordinator: Thennannamalai Malligarjunan

Erik Jonsson School of Engineering and Computer Science

University of Texas at Dallas

May 11, 2025

Table of Contents

Chapter 1: Introduction

1.1 Background

With the rise of music streaming platforms such as Spotify, music consumption has never been more accessible. However, the large influx of attention and artists have made organic discovery of new and niche artists harder. Recommendation algorithms, while powerful, often keep offering music similar to the user's existing preferences. Many users wish for a more interactive and exploratory experience.

To address this, Harmonic Links was developed – a simple web-based game which transforms music discovery into an interactive challenge. Inspired by the concept behind the game *Movie to Movie*, it invites users to connect artists though albums and influences to help them discover new artists.

1.2 Objectives, Scope, and Goals Achieved

Objectives & Scope

The primary goal was to build a web application for discovering music interactively. The target audience includes any music enthusiasts seeking to expand their listening tastes in a fun and engaging way.

Goals Achieved

Over the course of the project, our team successfully developed an interactive web application that transforms music discovery into a game. We implemented three distinct game modes: Daily Challenge, which presents a unique puzzle each day; Endless Mode, allowing unlimited solo play; and Multiplayer Mode, where users can compete against others in real time. To personalize the experience, we integrated authentication using both Spotify and Google OAuth, enabling users to log in securely. One of the standout features is the ability to generate Spotify playlists composed of top tracks from the artist chain users build during gameplay, which are then saved directly to their Spotify accounts. We also implemented core gameplay logic to validate artist-to-artist connections, a scoring system to encourage engagement, and a leaderboard to foster friendly competition among players.

1.3 Key Highlights

- **Artist Connection Game**: Core mechanic of the app. Users find connections between artists via shared albums.
- **Spotify Playlist Generation**: Automatically compiles a playlist of top tracks from artists in the player's connection chain.
- **Multiple Game Modes**: Offers flexibility in gameplay, from casual exploration to competitive challenges.
- **User Authentication**: Spotify and Google login for personalized features.
- **Leaderboard System**: Adds competitiveness and long-term engagement.

1.4 Significance of the Project

As the digital music space grows, helping users meaningfully explore it becomes more important. Harmonic Links aims to turn music discovery into a game accessible to all users to promote all artists, not just those favored by recommendation algorithms. By connecting artists through meaningful links and empowering users to explore beyond the mainstream, it offers a valuable alternative to passive listening habits.

Harmonic Links was developed to transform the way users discover music by turning exploration into an engaging, interactive experience. In an age where algorithm-driven recommendations dominate listening habits, *Harmonic Links* empowers users to take control of their musical journey through gameplay that is intuitive and culturally expansive.

By leveraging the Spotify API, the application draws from a vast and diverse pool of artists, ensuring representation across genres, languages, and backgrounds. This inclusivity reinforces the project's broader aim: to foster musical curiosity and celebrate global artistry. Whether connecting artists through shared collaborations or tracing paths between albums, the game invites users to deepen their understanding of music as a web of interconnected creative expressions

The significance of this project not only lies in its technical execution but also in its potential to reshape how people engage with music-making discovery more personal, playful, and inclusive.

Chapter 2: High-Level Design

2.1 System Overview & Proposed Solution

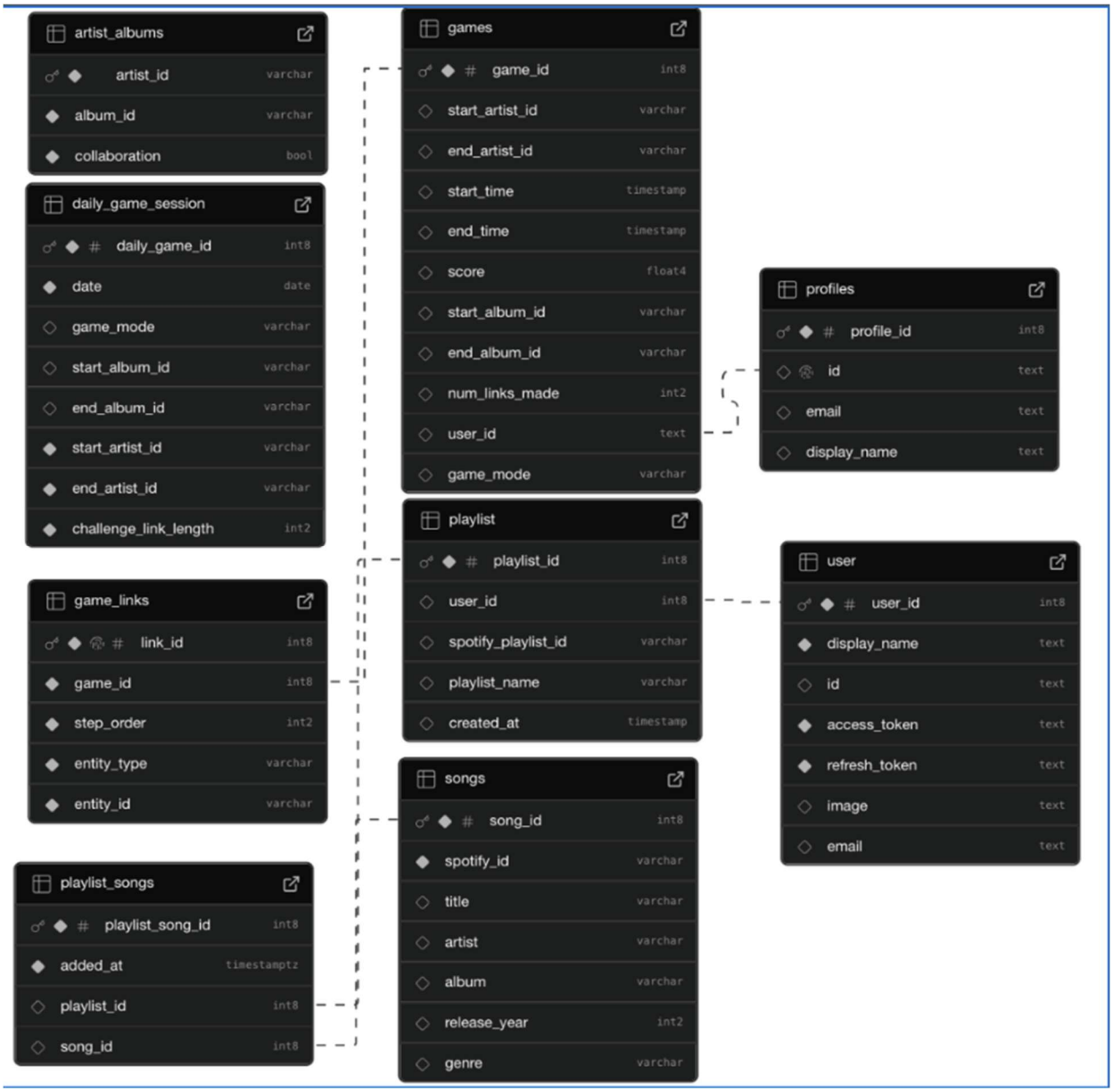
Harmonic Links is built around the Spotify Web API, which serves as the main source for artist, album, and track data. The application generates music discovery challenges by identifying a path between two artists based on shared attributes (albums, genres, or collaborations). Users navigate this path by selecting valid connections between artists until the target is reached.

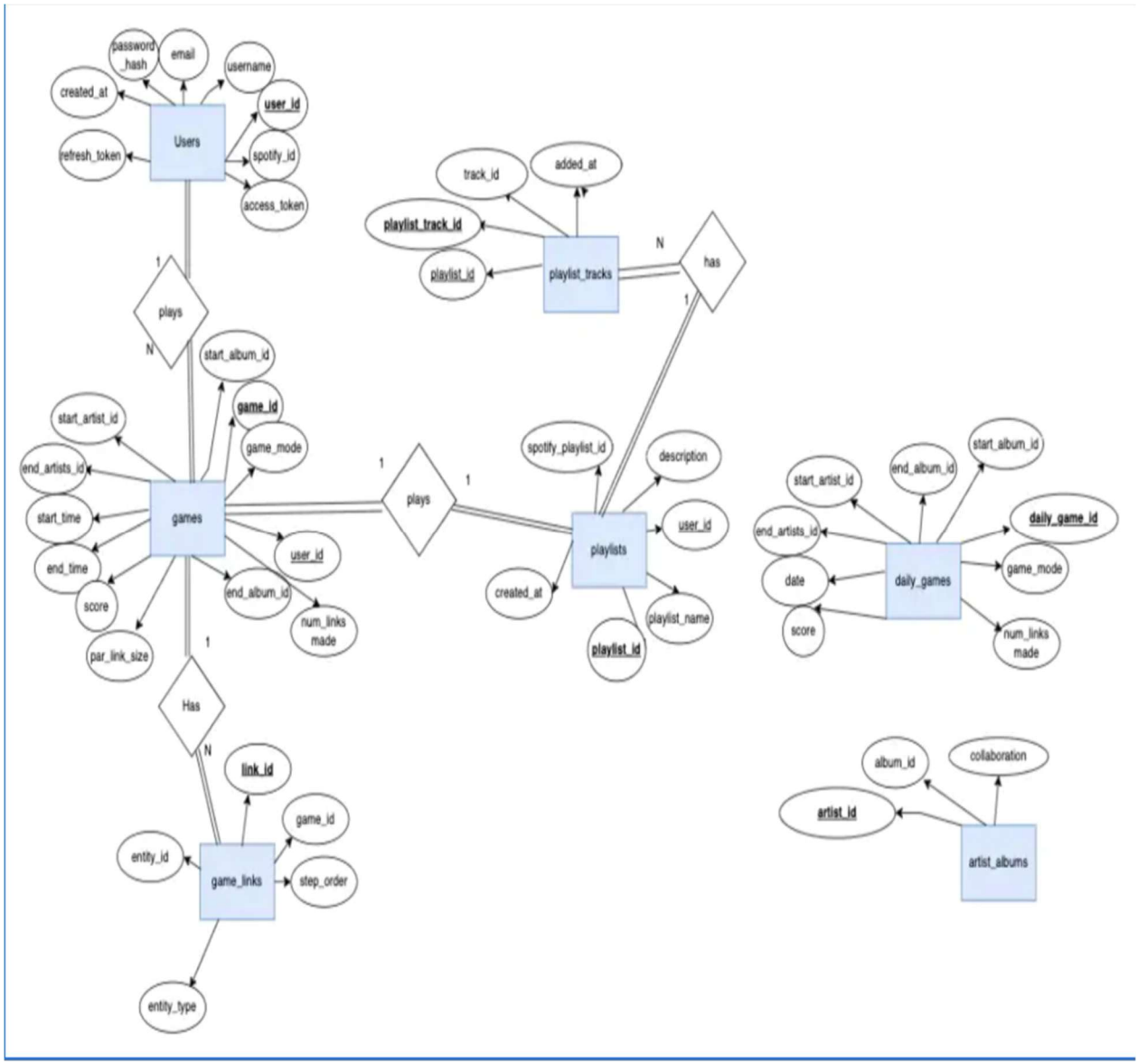
The backend is built using Next.js API routes which interface with the Spotify API to fetch real-time artist, album, and track data. To improve the performance under load and reduce API rate limits, the backend also seeds game data (e.g., start/end artist pairs). This allows for consistent challenge delivery and a smoother gameplay experience.

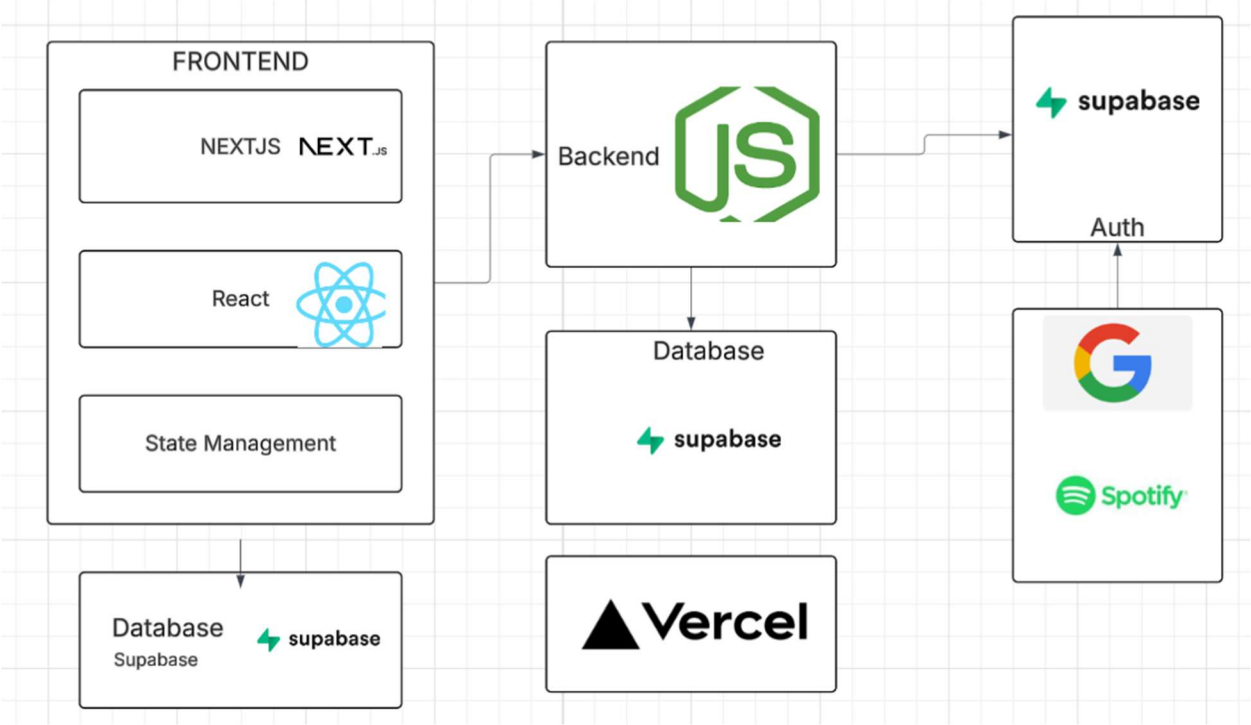
The database schema includes the daily_game_session for storing daily challenge metadata, and the leaderboard table for tracking player performance metrics in the daily, endless, and multiplayer game modes. Supabase also enables players to see when others finish the game in shared sessions in the multiplayer mode.

On the client side, the application is built with React and Tailwind CSS, providing a responsive and dynamic interface. State management is handled through local component state and React hooks while API calls are managed with native fetch requests. The Spotify API tokens are refreshed on the server side to maintain security.

2.2 Design and Architecture Diagrams




password_hash
email
username
created_at
user_id
Users
refresh_token
spotify_id
access_token
1
plays
N
track_id
added_at
playlist_track_id
playlist_id
playlist_tracks
N
has
1
start_album_id
game_id
start_artist_id
game_mode
end_artists_id
1
plays
1
spotify_playlist_id
description
start_time
games
user_id
playlists
end_time
user_id
created_at
score
end_album_id
playlist_name
num_links made
playlist_id
par_link_size
1
Has
N
link_id
game_id
entity_id
game_links
step_order
entity_type
start_album_id
end_album_id
start_artist_id
daily_game_id
end_artists_id
date
daily_games
game_mode
score
num_links made
album_id
collaboration
artist_id
artist_albums



2.3 Overview of Tools Used

- **Next.js & React**: For building an interactive and fast frontend
- **Tailwind CSS**: Styling framework for quick and responsive design
- **Node.js + Express**: Backend routing and logic handling
- **Supabase**: Open-source backend-as-a-service (BaaS) for database and auth
- **Spotify Web API**: Main data source for music metadata
- **Vercel**: Deployment platform with CI/CD pipeline
- **GitHub Projects**: Task tracking and project management
- **Figma**: Design wireframes and user flow planning
- **Discord**: Real-time team collaboration

Chapter 3: Implementation Details

3.1 Technologies Used

- **Next.js & React**: Power the interactive, dynamic frontend interface
- **Node.js + Express**: Handle server-side routing and game logic
- **Tailwind CSS**: Enables rapid, responsive design with utility-first styling
- **Supabase**: Manages authentication, database storage, and real-time functionality
- **Spotify Web API**: Serves as the primary source of music metadata and playlist creation
- **Vercel**: Hosts the application and manages the CI/CD deployment pipeline

3.2 Code Documentation

Our project is fully open-source and available on GitHub:
https://github.com/adibarra/harmonic-link

Game Logic

The game consists of three core components: gameplay, generation, and scoring — all powered by a combination of Spotify's API and Supabase.

Gameplay

Gameplay is primarily client-driven, supported by API calls to both Supabase and Spotify. Users begin a game by selecting from three available modes: daily, endless, or multiplayer. Daily games are pre-seeded and retrieved via a date-specific route, while endless and multiplayer games pull from a random pool.

The game flow consists of:

- Selecting an artist or album based on available connections
- Verifying that the selection is valid and progressing the path
- Repeating until the user reaches the target artist, completing the game

All user moves are validated on the backend to prevent invalid or repeated paths.

Game Generation

Game generation is handled via an admin-facing process and is more computationally intensive. It constructs a valid path between two artists using a Depth-First Search (DFS) algorithm. This path has a predefined length, known as the "par," which serves as a baseline for scoring.

An alternative Breadth-First Search (BFS) algorithm was tested for shortest path verification but was deprecated due to performance bottlenecks and high API latency.

Once generated, the full game data (start and end artists, par path, metadata) is stored in Supabase for quick loading and consistent player experience.

Scoring System

Scoring is handled server-side and calculated based on:

- Path length relative to the par: shorter paths earn higher scores
- Completion time: faster players receive bonus points

Upon finishing a game, the score is recorded in the database and added to the user's profile. For daily games, it is also displayed globally on the leaderboard.

User Auth

Authentication is managed through Supabase Auth, using OAuth providers such as Google and Spotify. Supabase's built-in OAuth support simplified integration with our backend and enabled secure login flows.

A PostgreSQL trigger function automatically creates a user profile the first time someone logs in via OAuth, allowing us to immediately associate scores, games, and playlists with their account.

Playlist Generation

At the end of each game, users are offered the option to generate a Spotify playlist that reflects the artists they discovered along the way.

This feature uses the Spotify API:

1. Fetches top tracks for each artist in the user's final path
2. Creates a new playlist in the user's Spotify account
3. Adds five tracks per artist to complete the playlist

Once created, the playlist is accessible via a direct link at the end of the game.

3.3 Development Process

To manage and execute the project effectively, our team followed an Agile-inspired development methodology. We utilized GitHub Projects to structure our workflow, organizing tasks across three key categories: front-end, back-end, and general team responsibilities.

Each week began with setting clear sprint goals and defining deliverables. We held bi-weekly team meetings to track progress, identify blockers, and reassign tasks as needed. Weekly feedback from our professor functioned as a sprint review, helping us refine our features and adjust priorities.

Overall, this approach enabled effective communication, promoted flexibility in responding to challenges, and ensured a consistent development pace throughout the project lifecycle.

Chapter 4: Performance

4.1 Testing and Validation

To test the API's functionality in our project we generated JSON scripts that output the return content for each API call we made. Additionally, to test the front end rendering we used our loaded our local host to test if our front end was exhibiting the expected images. Additionally, chain validation, artists/album lookup and the overall behavior of the game was tested in isolation using a custom test case to verify edge behavior.

For debugging processes, we used Console +Log-based debugging where we implemented server-side logs to trace errors across different request types. We handled async errors and displayed meaningful fallback messages in the UI to track errors. We used browser developer tools to monitor network activity, inspect component states, and resolve layout or rendering issues during development. Together, these practices allowed us to catch bugs early on and maintain stability throughout the development of the application.

4.2 Metrics Collected and Analysis

In summary, our application has very high performance scores in various regions where we measure, with a score above 95 indicating consistent speed and responsiveness worldwide. Additionally, our First Contentful Paint, Largest Contentful Paint, Total Blocking Time, Speed Index, and Cumulative Layout Shift scores were all in excellent ranges, indicating that our site loads quickly, has stable visual content, and minimal input delay.

All of these metrics were gathered using Google's Lighthouse, an open-source, automated tool for improving the quality of web pages. Lighthouse audits performance, accessibility, best practices, SEO, and more, providing a reliable benchmark for how our application performs across key user experience areas. Other metrics we used were accessibility, adherence to best practices, resource size, and number of requests made. For each of these metrics, we concluded that our site is fully accessible to all users, including people with disabilities. Our site follows all current best web practices using HTTPS, secure requests, and no outdated APIs. Our resource sizes and total number of requests were also within a reasonable range.

User Metrics

Our application was not distributed to the public because of drawbacks from the Spotify developers dashboard. This drawback prevented us from being able to allow several users to login to our app with their Spotify accounts which limited the features that users were able to use. However, as a group, we played the game on a daily basis during our testing periods. We all enjoyed playing the game, and were

pleasantly surprised to see how many new artists we were able to discover as the producers of this application.

Chapter 5: Lessons Learned

5.1 Insights Gained

As a group, we gained many valuable skills from this project. We learned how to schedule and communicate more effectively, both within our small teams and as a larger group. By leveraging each other's strengths and sharing information, we were able to collaborate more efficiently and produce a successful product. We also discovered that each of us excelled at different tasks, and delegating responsibilities accordingly proved to be a crucial step in overcoming early challenges and driving the project forward.

5.2 Lessons from Challenges

Rate Limiting

One of the major challenges we faced was rate limiting from the Spotify API. As the core of our project depended on this API, it became a significant bottleneck, particularly for implementing the endless game mode. Initially, this mode was designed to randomly generate starting and ending artists using live API calls. However, as more users began accessing the game simultaneously, we quickly hit Spotify's rate limits. To resolve this, we opted to pre-seed a collection of starting and ending artists and store them in the database. While this slightly compromises the randomness of the endless mode, it remains functionally random due to the growing pool of stored games updated daily.

Poor/Outdated API Documentation

Another issue we encountered was poor or outdated API documentation. Several Spotify API routes that would have streamlined our workflow were deprecated this year. For instance, searching albums by genre or even retrieving a list of available genres was no longer supported. To work around this, we turned to the tracks endpoint, which—though undocumented—still accepted a genre parameter. This allowed us to generate albums and artists associated with random genres.

Multiplayer

Finally, implementing a multiplayer mode proved difficult due to our initial lack of experience. Fortunately, Supabase's Realtime service, built on WebSockets, provided the necessary tools for player presence tracking and message broadcasting. By leveraging Realtime, we were able to connect players through a shared channel and enable multiplayer gameplay. Since our backend already used Supabase, this solution integrated smoothly and sets a solid foundation for future feature expansions.

5.3 Skills Developed and Improved

As a team, we developed and refined numerous skills, both technical and collaborative. On the technical side, we significantly improved our research abilities, which were essential in selecting the appropriate technologies at the project's outset, understanding Spotify’s documentation for playlist generation, and implementing the game logic. Our front-end team also enhanced their proficiency in React and Figma, which were key tools in creating a smooth and intuitive user interface. Meanwhile, the back-end team gained experience working with external APIs and designing robust game logic to support the core mechanics of the application.

Chapter 6: Future Work

6.1 Proposed Enhancements

Album-to-Album Gamemode

A new game mode where albums serve as the starting and ending points instead of artists. The core game logic would remain largely unchanged, with only the entry and goal nodes being switched from artists to albums.

Genre Selection

Currently, games are seeded randomly across all genres with no user input. Adding a genre selection feature would allow users to choose a specific genre to play with, offering a more personalized and challenging experience.

Time Attack Mode

A competitive game mode for players seeking a higher level of difficulty. This mode would include multiple difficulty levels based on the minimum number of connections required to complete the path. Players would be given a limited amount of time to reach the endpoint, and failure to do so within that timeframe would result in a loss.

iCloud Login for Apple Music

To expand beyond Spotify and increase accessibility, future development could include support for iCloud login. This would enable integration with Apple Music, allowing users to generate playlists and gameplay based on their Apple Music data.

6.2 Recommendations for Development

For continued development, integrating pgRouting could significantly improve the generation of artist and album routes. pgRouting, a lightweight graph extension, is designed to work with any graph-like data structure. Since the relationship between artists and albums is already structured in a graph format (with albums as edges and artists as nodes), using pgRouting would streamline route generation. This approach would drastically reduce the need for frequent Spotify API calls by enabling the database to handle more of the data processing. As new games are generated, the data could be progressively crawled and populated within the database, improving both efficiency and scalability.

Chapter 7: Conclusion

7.1 Summary of Key Accomplishments

By the end of this project, we successfully developed a fully functional application called Harmonic Links. The game features three distinct game modes, a playlist generator, and the ability for users to create links and discover new music. It operates on an algorithm that generates starting and ending artists, allowing users to engage with the game and complete challenges effectively.

7.2 Acknowledgements

We would like to thank Professor Alagar and Thenn for their invaluable support throughout this project. Their weekly feedback provided crucial insights, guiding us in making timely adjustments to our application. Their help enabled us to address aspects of the project we might have otherwise overlooked. We are truly appreciative of their mentorship and assistance during this process.

References

- https://developer.spotify.com/documentation/web-api
- https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/quickstart-for-projects
- https://gist.github.com/drumnation/91a789da6f17f2ee20db8f55382b6653
- https://github.com/adibarra/harmonic-link
- https://github.com/users/adibarra/projects/3
- https://github.com/vercel/next.js/tree/canary/examples/with-supabase
- https://nextjs.org/
- https://react.dev/learn
- https://ui.shadcn.com/
- https://vercel.com/
- https://www.figma.com/design/IzhyMKot3NWQaGJtgEDeFu/Harmonic-Links-Wireframe?node-id=0-1&t=jCEeQ7ONnKAbEAem-1
- https://www.w3schools.com/js/js_cookies.asp