TRACKIFY

Your Spotify music journey visualized.

ABSTRACT

The Spotify App Tracker is a full-stack web application designed to fetch, store, and display personalized user data from Spotify. The primary objective is to analyze user listening habits by leveraging the Spotify Web API, providing insights into their top tracks, most played artists, and recently played songs.

The backend is developed using Node.js and Express, with PostgreSQL as the database, hosted on Railway and managed via DBeaver. Spotify’s OAuth 2.0 is implemented for secure user authentication. Upon successful login, the app fetches user data such as profile details, top tracks, top artists, and recently played tracks, which are then stored in the PostgreSQL database.

The outcomes of this project include:

* Seamless Spotify authentication
* Persistent user data storage
* API routes to retrieve and serve Spotify user data
* An extensible backend architecture suitable for future features like user dashboards and visual analytics

TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| S.No | Title | Page No. |
| 1 | Introduction | 1 |
| 2 | System Analysis | 1 |
| 3 | System Design | 1-2 |
| 4 | Implementation | 2-3 |
| 5 | Results & Discussions | 3-4 |
| 6 | Conclusion & Future Scope | 4 |
| 7 | References | 4 |

INTRODUCTION

**Background of the Project**

Music is a key part of daily life, and with streaming services like Spotify, users can access a vast library of tracks. However, users lack visual and intuitive access to personal listening data. Trackify is a web application that allows users to log in with their Spotify account and visualize their listening trends.

**Objectives**

* Integrate Spotify API to access personal listening data.
* Display top tracks, recently played songs, and top artists.
* Store listening data into a PostgreSQL database for extended use.

**Scope of the Project**

Trackify offers users the ability to analyse their music habits, understand preferences, and explore trends using an interactive, modern web interface.

SYSTEM ANALYSIS

**Problem Statement**

Users often do not have a dedicated tool to view and analyse their personal Spotify listening history and trends.

**Functional Requirements**

* Spotify login and authorization.
* Display currently playing track, top tracks, recently played songs, and top artists.
* Store retrieved data in a backend database.

**Non-Functional Requirements**

* Responsive and intuitive user interface.
* Secure access token handling.
* Persistent data storage using Railway-hosted PostgreSQL DB.

SYSTEM DESIGN

**ER Diagram:**

**Entities:**

• Users • Tracks • Artists • Playback History

**Relationships:**

* Users can have multiple top tracks, artists, and recently played entries.

**Schema Design**

| **Table** | **Columns** |
| --- | --- |
| users | user\_id (PK), display\_name, email |
| top\_tracks | track\_id (PK), user\_id (FK), track\_name, artist\_name, album, image\_url, timestamp |
| recently\_played | play\_id (PK), user\_id (FK), track\_name, artist\_name, played\_at, image\_url |
| top\_artists | artist\_id (PK), user\_id (FK), artist\_name, image\_url |

**SQL Table Structure**

A screenshot of a computer program

AI-generated content may be incorrect. A screenshot of a computer program

AI-generated content may be incorrect.

IMPLEMENTATION

**Technologies Used**

* **Frontend:** HTML, CSS, Bootstrap, JavaScript
* **Backend:** Node.js, Express.js
* **Database:** PostgreSQL (hosted on Railway)
* **Tools:** Railway (for deployment), DBeaver (for DB management)

**System Architecture**

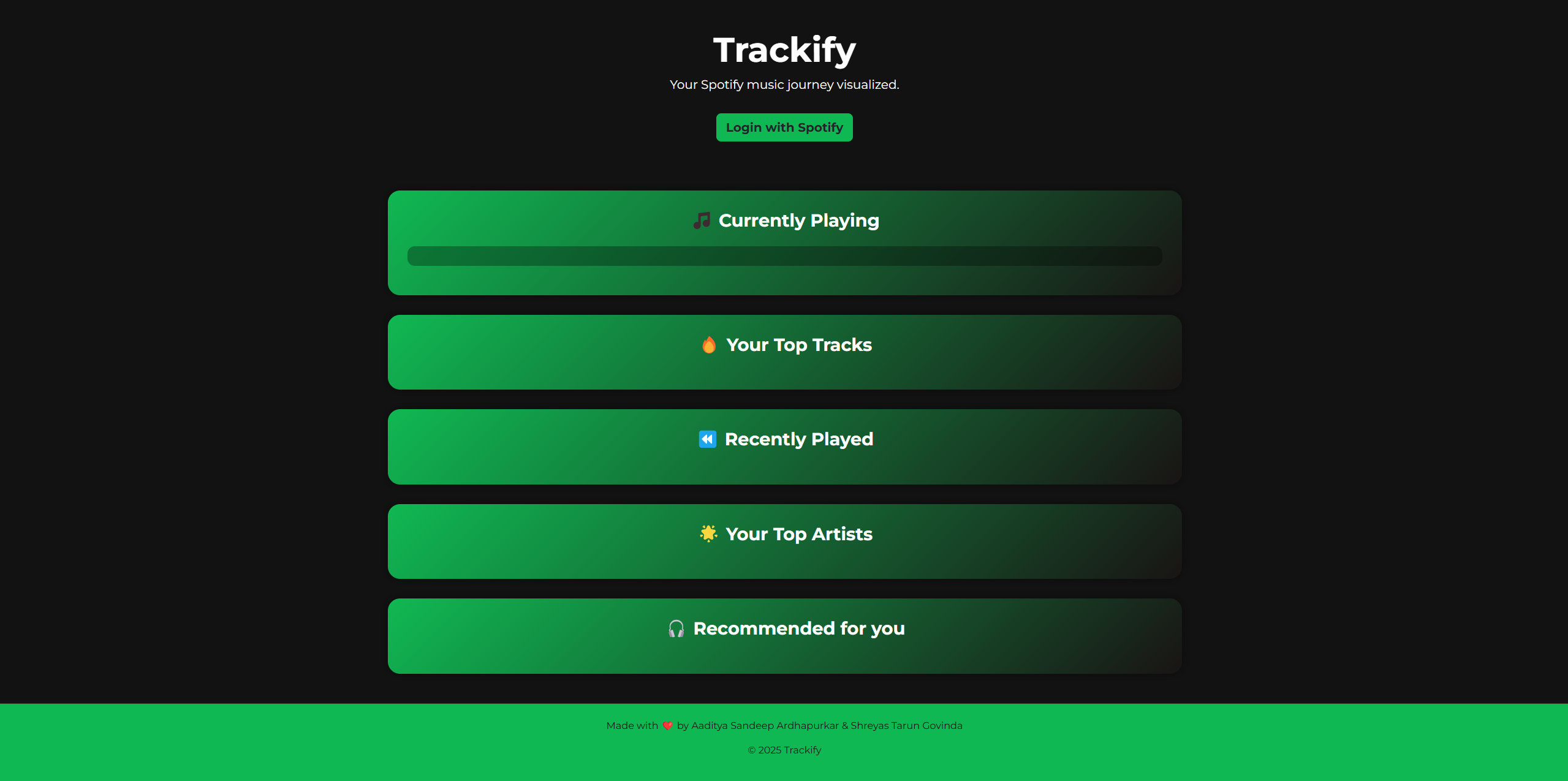
1. User logs in via Spotify authorization.
2. Backend exchanges authorization code for access token.
3. Spotify Web API fetches top tracks, recently played, and top artists.
4. Data is stored in PostgreSQL.
5. Frontend fetches and displays the data.

**Modules Description**

* **AuthRoutes:** Manages Spotify login and callback handling.
* **SpotifyRoutes:** Handles API calls to Spotify.
* **DB Module (db.js):** PostgreSQL Pool setup with environment configs.
* **Spotify Storage Module:** Upserts users, stores track and artist data.
* **Client.js (Frontend):** Handles DOM rendering using API responses.

RESULTS & DISCUSSION

**Visual Output (Screenshots)**

****

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a video chat

AI-generated content may be incorrect.

A screenshot of a video game

AI-generated content may be incorrect.

**Outcome**

* Seamless Spotify integration
* Dynamic and real-time data visualization
* Secure and scalable database storage

CONCLUSION

**Summary**

Trackify successfully provides users with a simple way to explore and understand their listening behavior through clean visuals and solid backend storage. Built using modern web technologies and Spotify APIs, the project delivers both function and aesthetic, making the user experience an easy and delightful one.

REFERENCES

* Spotify Developer API Docs
* PostgreSQL Documentation
* Bootstrap 5 Documentation
* Railway.app Docs