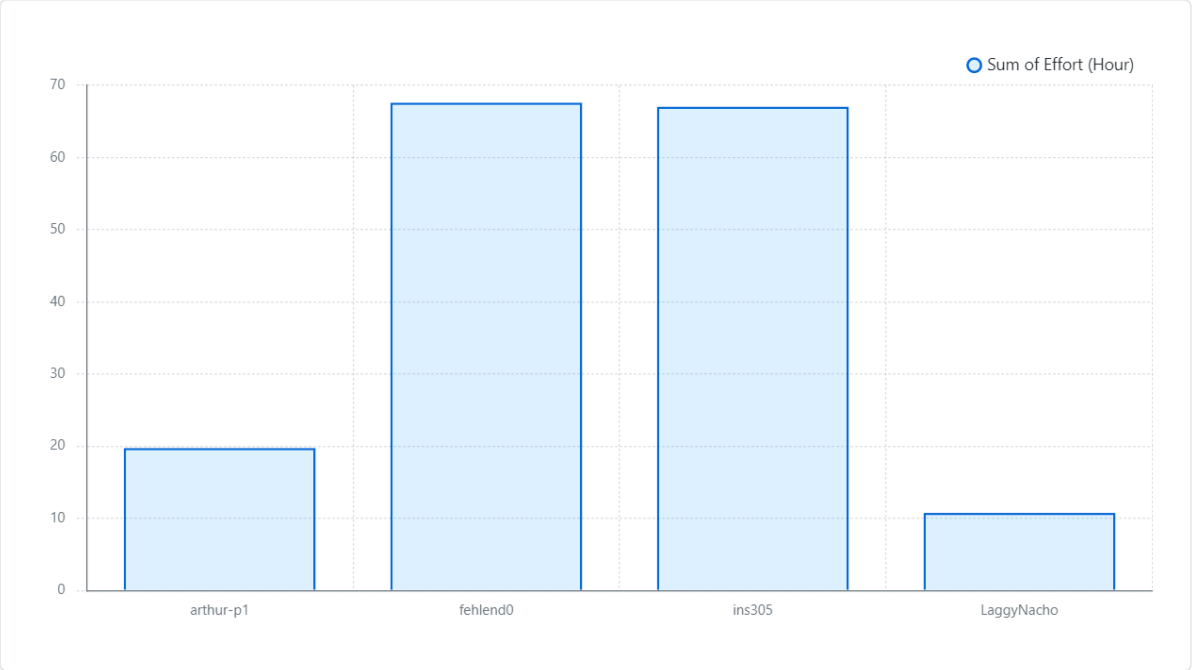
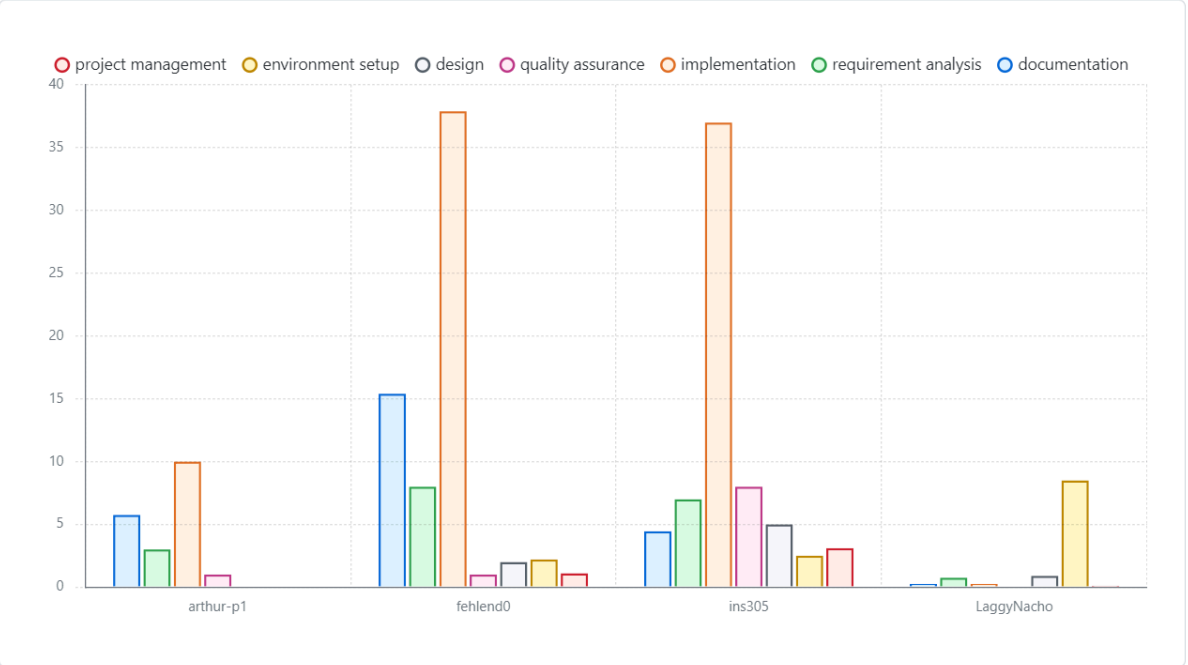
Statify

Hours per Person



Arthur	Viktoriia	Ines
19.75	67.6	67.05

Hours per Workflow

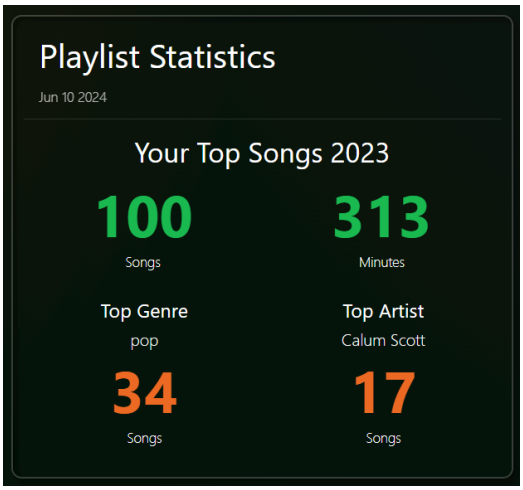


	Arthur	**Viktoriia**	**Ines**
Project Management	0	1.1	3.1
Environment setup	0	2.2	2.5
Design	0	2	5
Implementation	10	37.9	37
Quality Assurance	1	1	8
Requirement Analysis	3	8	7
Documentation	5.75	15.4	4.45

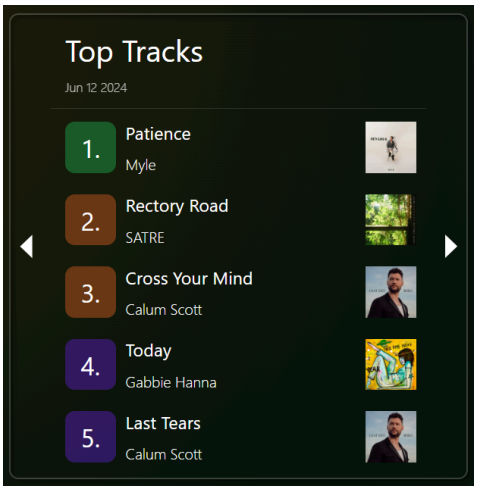
Major contributions per person:

Arthur	Weekly Reports
Ines	Frontend
Viktoriia	Database

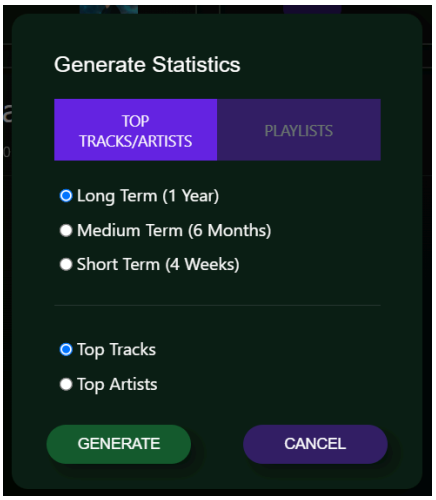
Highlights of our demo

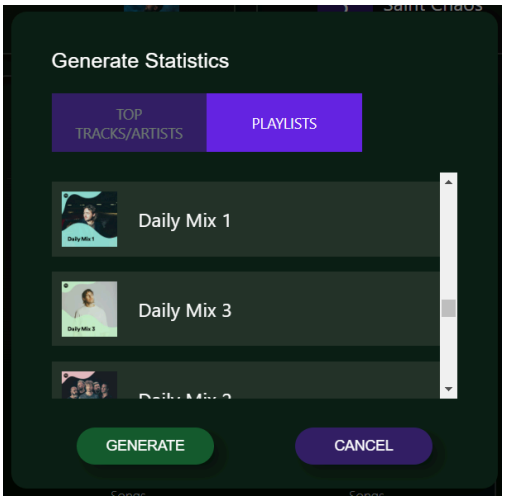


Statistics about a playlist from your library

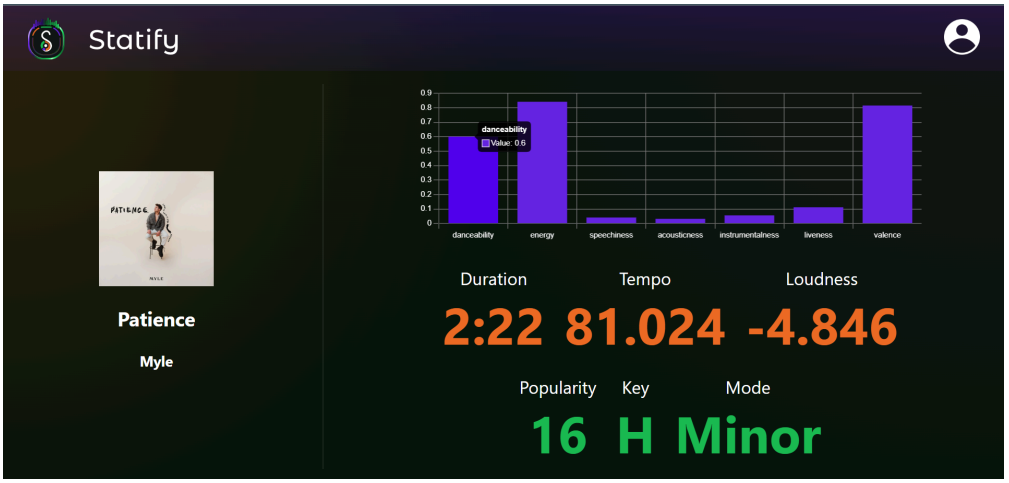


Statistics about your top 5 tracks





Dialog for generating new statistics

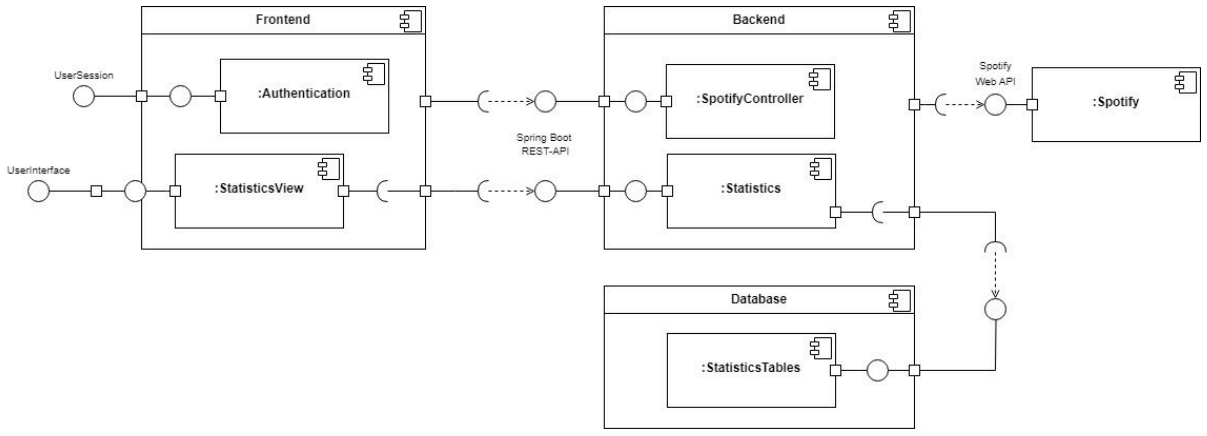


Page with information about a specific track

Highlights of our project

Tech Stack

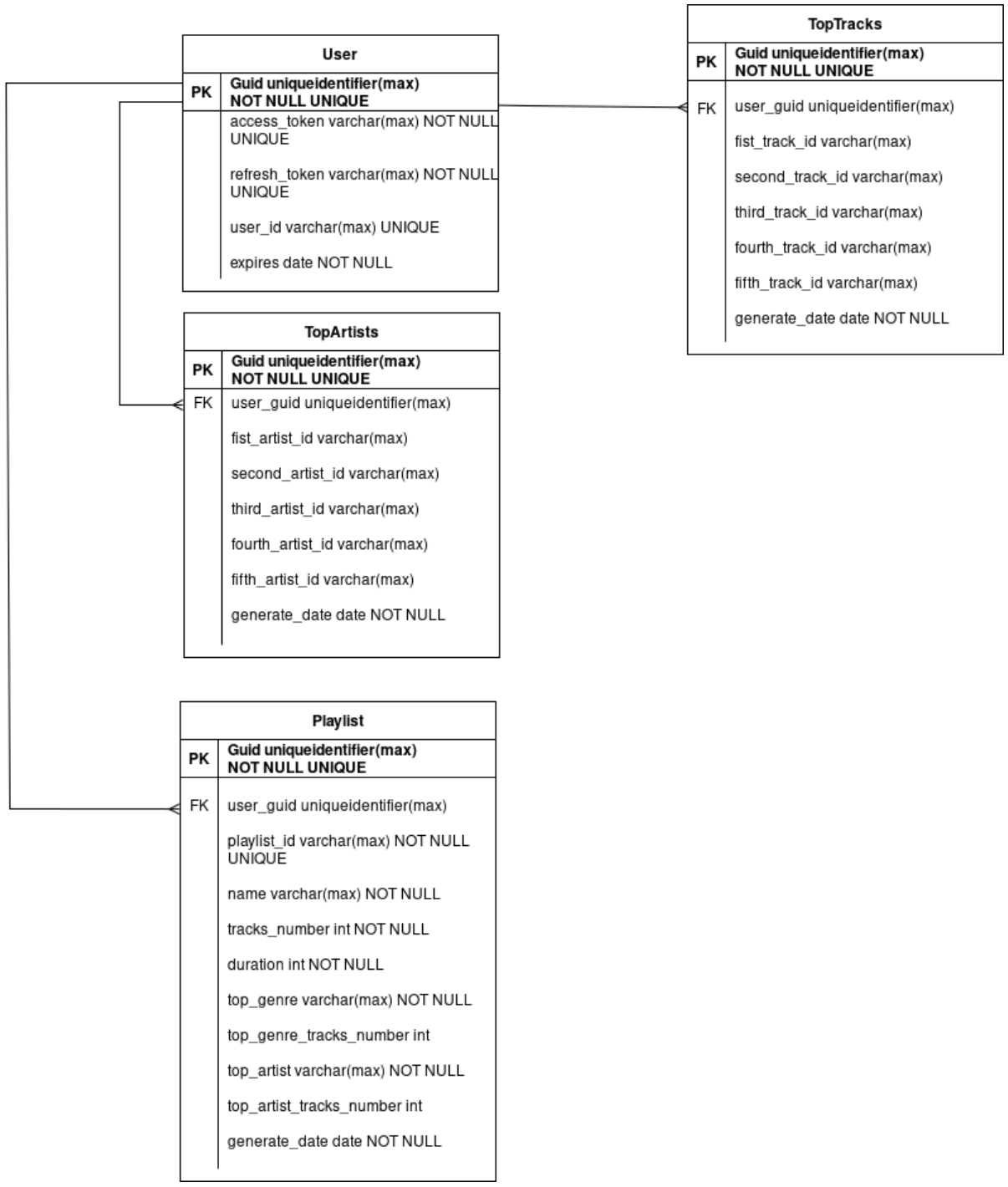
Architecture:



Our architecture contains 3 main components: Frontend, Backend, Database.

The Frontend is connected to the backend via the provided Spring Boot REST API defined in the SpotifyController. The Frontend displays a user's Statistics via the user interface. For that purpose, it requests the Backend's Statistics component, which provides either already generated Statistics from the Database or generates new statistics by sending requests to the Spotify API.

Database Design:



Our database includes four tables designed to manage user information, playlists, and top artists and tracks:

→ The **user** table stores user information, including a unique identifier (GUID), access and refresh tokens, user ID, and token expiry date.

→ The **playlist** table stores details about user playlists, including a unique playlist identifier (GUID), user GUID, playlist ID, name, number of tracks, duration, top genre, and top artist.

→ The **top_artists** table records the top 5 artists for each user, identified by a unique GUID, user GUID, artist IDs, and the date this data was generated.

→ The **top_tracks** table records the top 5 tracks for each user, with a unique GUID, user GUID, track IDs, and the date this data was generated.

Each user can have multiple playlists, top artists, and top tracks, all linked through the user's GUID.

CI/CD

Node.js CI Workflow

Purpose: to install Node.js dependencies, cache them, build the source code, and run tests.
Triggered: on pushes to the "main" and "develop" branches, and on pull requests to the "main" branch.

Steps:
→ retrieves the repository code
→ installs Node.js
→ configures npm caching
→ executes “npm ci” to cleanly install dependencies
→ runs “npm run build” if a build script is present
→ executes `npm test` to run the test suite

Java CI with Maven Workflow

Purpose: to build the Java project with Maven and ensure dependencies are up to date, as well as to generate a dependency graph.

Triggered: on pushes to the "main" and "develop" branches, and on pull requests to these branches.

Steps:
→ retrieves the repository code
→ installs JDK and Maven
→ configures the environment variables for Java
→ executes “mvn package” to build the project
→ uses the Depgraph Maven plugin to update the project's dependency graph