Report DAT076

Elin Hagman, Hampus Rhedin Stam, Amanda Papacosta, Arvid Svedberg

March 2023

1 Introduction

The application is a game where the players get to guess each other's top songs on Spotify. The players join a game by logging in to their Spotify account. When all players have joined the game can be started. A song will appear and the player's have to guess which player has this song as their top song on Spotify. The guessing will be done in real life and not on through the application. To reveal the answer, someone hovers over the "Reveal players" flip card. Then, by clicking on a button the next song is displayed and the game progresses in the same manner.

Link to github repository: `https://github.com/hampusrs/DAT076`

2 Use cases

- **Login with Spotify account -** To actually join the game the user has to log in with their Spotify account.
- **Join a game -** If login was successful the players will automatically be added to the game.
- **See players who join the game -** The user then gets redirected to a page where all players who are waiting to play the game are displayed in a list.
- **Start a game -** One user can press the start game button to redirect all players to the actual game page where the songs are displayed.
- **See current song -** One song is displayed on everyone's screens.
- **Reveal players who has current song -** On the game page there is also a button which when pressed reveals all players who have that song as their top song in the last month.
- **See who are currently in the game during game -** When the game is playing there is a button at the top left corner which users can press to reveal all players that are currently in the game.
- **Choose next song -** During the game play, to move on to the next song after everyone has made their guesses there is a button to reveal the next song.
- **Game Over -** When all songs have been played through the users get redirected to the game is over page.

3 User manual

The following section is a brief introduction to the different pages of the application and how to navigate through the application.

3.1 step 1: Add .env file to server directory

To have the application get access to the Spotify API there is a need of a .env file to be added by the user who runs the application on their device. In the .env file, there are three variables; *CLIENT_ID*, *CLIENT_SECRET* and *REDIRECT_URI*. The *CLIENT_ID* is a unique identifier which is assigned to the application when it is added to the Spotify Developer Dashboard. This identification is needed for when the requests to the API are made in server. The *CLIENT_SECRET* is a key used that Spotify also assigns to the application when it is registered. Is is used when some requests to the Spotify API are made that requires authentication, such as certain user data. In short, both *CLIENT_ID* and *CLIENT_SECRET* are used for secure communication between the application and the Spotify API. The *REDIRECT_URI* is the URL to which the user is redirected after they have logged in with their Spotify account.

Add the .env file named ".env" in the server directory with the following content. The client id and client secret has been sent to the examiner in the Canvas submission.

```
CLIENT_ID=xxxxxxxxxxxxxxxxx
CLIENT_SECRET=xxxxxxxxxxxxxxxx
REDIRECT_URI=http://localhost:8080/callback
```

3.2 step 2: Connection to database

To connect to the database you will need a URI string that has been added as a comment in the canvas assignment submission. Paste the string into the MongoDBCompass URI text field and connect. Now you should be able to see our database and observe how it responds to the different actions made in the game.

3.3 step 3: Log in

When the user first enters the application the following page will be displayed, indicating that the user has to login (see figure 1). Then when the "log in" button to the right is pressed the user is redirected to the Spotify login page where they have to enter their account information (see figure 2)

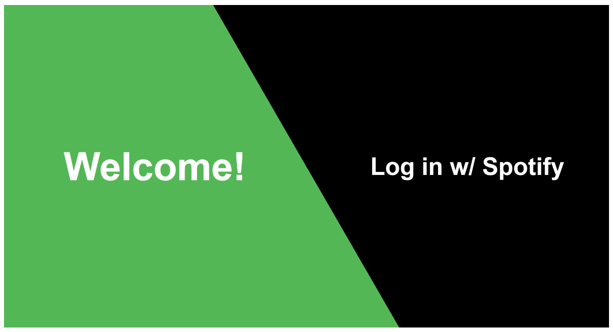
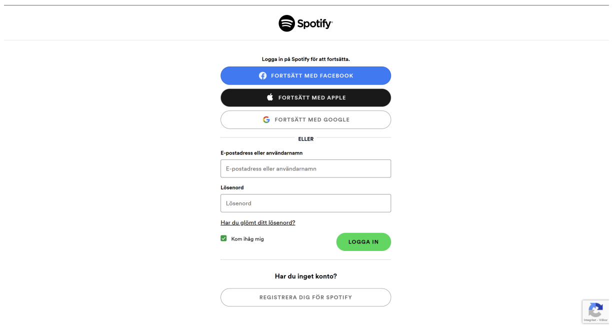


Figure 1: The first page the user sees when deciding to enter the game.

A temporary Spotify account was created for the examinator to use while testing. This is sent to the examinator in the Canvas Submission.

3.4 Wait for players to join

Afterwards, the user is redirected to the pre-game page where all players who have entered their Spotify account information are displayed in a list all players to see. Whenever a new player is joining the game, the list gets updated (see figure 3).


Spotify

Logga in på Spotify för att fortsätta.

FORTSÄTT MED FACEBOOK

FORTSÄTT MED APPLE

FORTSÄTT MED GOOGLE

ELLER

E-postadress eller användarnamn

E-postadress eller användarnamn

Lösenord

Lösenord

Har du glömt ditt lösenord?

✓ Kom ihåg mig

LOGGA IN

Har du inget konto?

REGISTRERA DIG FÖR SPOTIFY

Figure 2: The Spotify login page where the user has to enter their account information to log in.

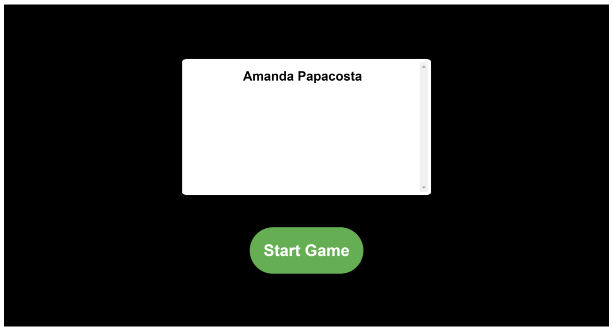


Figure 3: On this page all players who have currently joined the game are displayed.

3.5 Start the game

On the same page there is also a ”Start game” button that when pressed redirects all players to the following page and the game is started (see figure 4).

3.6 Game has started

When the game has started the following page is displayed to all players. In the top left corner there is a button which can be clicked to show all current players in the game. If clicked again it hides the list. In the main area of the page the current song that is playing is displayed with its album cover, title, artist name and album name. Below it is a button that can be clicked to reveal the player(s) who have the current song as their top song. At the bottom of the page there is a next song button that when clicked displays another song for the players to guess (se figure 4).

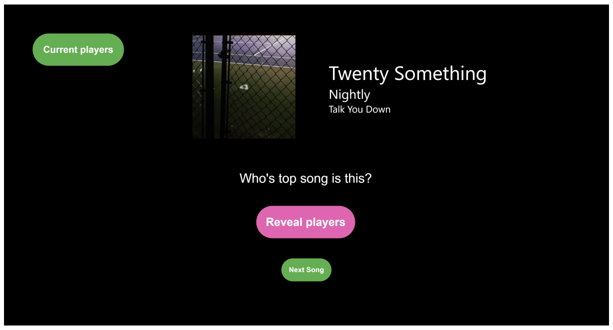


Figure 4: On this page all players who have currently joined the game are displayed.

3.7 Game over

When everyone's top songs have been played through and there are no left all players are redirected to the following game over page (see figure 5). To play the game again, the server and client have to be restarted and all players need to log in with their Spotify accounts again (see figure 1).

Figure 5: The game is over page

4 Design

The project is divided into two main parts, server and client. The server package contains the node.js backend and the MongoDB database. The client package contains the React.js frontend. Figures of the data flow between the frontend, backend and Spotify API can be found in the end of the document.

4.1 Frontend

The frontend part of the application is as aformentioned located in the client directory, where it is divided into the different pages of the application. There is also a components directory which includes the different components used, such as songItem which is the conjunction of album cover and song information. Furthermore, it is in the frontend where the different requests are made to the server to get or post data.

The only framework used in the frontend is Axios, which is used to send requests to the server.

The App component is responsible for rendering the game and handles the redirecting to different pages depending on which state the game is currently in.

4.2 Server

The server part contains a database folder (schema) and a src folder which is divided into three sub folders, i.e model, service and router. The model contains the entities of the project, Player and Song. The service contains the game service, which handles the state of the game, game logic and connection to the database. The router folder contains the endpoints which the client can send requests to. Section 4.4 includes documentation for all endpoints.

The following libraries/frameworks are used in the server:

- `express` - to handle request to the server
- `axios` - to send request to Spotify API
- `dotenv` - to be able to store client id, client secret and redirect uri safely in a hidden .env file.
- `nodemon` - used during development to automatically restart application when saving

- `mongoose` - is used to handle the interaction with the mongoDB database
- `jest` - to test the server

4.3 Database

We are using a mongo database combined with the mongoose framework to more easily work with it in node. The database is used continuously with the game so when a player is added to the game for example, the same player is added to the database. The same goes for popping songs and changing the current song.

The purpose of the database is to be able to recover a game if the server for some reason is disconnected. In the case of a server disconnecting, the server can be restarted. The new database collection that is created when doing so can be restored by posting a get request to "/game/recover".

We have three different schemas, game, song and player. The data is currently stored in a collection called game in the database called test.

The database is also tested and the are in the test folder located in server.

4.4 API

This section contains documentation of the API. It does not include the `POST` request to the `game/players` endpoint, because this is used only for testing. It is used to manually add players to the game, which should be done with the `/login` endpoint in the usual case.

4.4.1 `GET /game`

Returns a list of all players currently in the game.
Request

- **Method:** `GET`
- **Endpoint:** `/game`
- **Headers:** None
- **Body:** None

Response

- **Status Codes:**
  - `200 OK` - The request was successful.
  - `500 Internal Server Error` - There was an error processing the request on the server.
- **Body:** A list of `Player` objects. If there are no players in the game, an empty list is returned.

4.4.2 `POST /game`

Performs an action in the game, i.e starting a game or playing the next song. Cannot start a game if it has already started, or play next song if game has not yet started.
Request

- **Method:** `POST`
- **Endpoint:** `/game`
- **Headers:** None

- **Body:** A JSON object with the following property:
  - `action` - A string describing the action to be performed, that is either "StartGame" or "NextSong".

Response

- **Status Codes:**
  - `200 OK` - The request was successful.
  - `400 Client error` - If the action given is not valid.
  - `500 Internal Server Error` - There was an error processing the request on the server.
- **Body:** A JSON object with the following properties:
  - `currentSong` - A `Song` object of the song that is currently played
  - `players` - An array of `Player` object that has `currentSong` as their top song.

4.4.3 `GET /game/started`

Checks if the game has started.

Request

- **Method:** `GET`
- **Endpoint:** `/game/started`
- **Headers:** None
- **Body:** None

Response

- **Status Codes:**
  - `200 OK` - The request was successful.
  - `500 Internal Server Error` - There was an error processing the request on the server.
- **Body:** A JSON object with the following properties:
  - `gameHasStarted` - A boolean value indicating whether the game has already started.
  - `currentPlayers` - An array of `Player` objects that has the `currentSong` as their top song if the game has started, or all the players in the game if the game has not started.
  - `currentSong` (optional) - A `Song` object representing the current song of the game, if game has started.

4.4.4 `GET /login`

Redirects client to Spotify login URL that promts the player to login and accept the application to access their user information and top items.

4.4.5 Request

- **Method:** `GET`
- **Endpoint:** `/login`
- **Headers:**
- **Body:** None

4.4.6 Response

- **Status Codes:**
  - `302 OK` - The redirect was successful.
  - `500 Internal Server Error` - There was an error processing the request on the server.
- **Body:** None

4.4.7 `GET /callback`

This endpoint is responsible for handling the Spotify OAuth 2.0 callback request after a user has authorized access to their Spotify account. The endpoint receives a ”code” parameter in the query string, which is exchanged for an access token that is used to retrieve the user’s information and top tracks. It then adds the player to the game with the Spotify user name and their top songs and redirect back to the frontend.

4.4.8 Request

- **Method:** `GET`
- **Endpoint:** `/callback`
- **Headers:**
- **Body:** None

4.4.9 Response

- **Status Codes:**
  - `200 OK` - The request was successful
  - `400 Bad Request` - Bad request to Spotify API
  - `401 Unauthorized` - Failed authentication or authorization request to Spotify
  - `403 Forbidden` - Not authorized to access the requested resource (often due to user not being added to dashboard application)
  - `404 Not Found` - The reqeusted resource was not found on the Spotify server
  - `429 Too Many Requests` - The application has exceeded the limit of request to the Spotify server
  - `500 Interal Server Error` - There was an error processing the request on either the application’s server or Spotify’s server.
- **Body:** None

4.4.10 `GET /game/currentSong/isRevealed`

Checks if the players that has `currentSong` as one of their top songs have been revealed or not.

Request

- **Method:** `GET`
- **Endpoint:** `/game/currentSong/isRevealed`
- **Headers:** None
- **Body:** None

Response

- **Status Codes:**
    - `200 OK` - The request was successful.
    - `400 Bad Request` - If game has not started yet
    - `500 Internal Server Error` - There was an error processing the request on the server.
- **Body:** A JSON object with the following properties:
    - `playersAreRevealed` - a boolean value indicating if the players are revealed or not

4.4.11 `POST /game/currentSong/isRevealed`

Changes the reveal status of the players that has `currentSong` as their top song.

Request

- **Method:** `POST`
- **Endpoint:** `/game/currentSong/isRevealed`
- **Headers:** None
- **Body:** A JSON object with the following property:
    - `action` - A string describing if to hide or reveal current song, that is either ”HidePlayers” or ”RevealPlayers”

Response

- **Status Codes:**
    - `200 OK` - The request was successful.
    - `400 Client Error` - If game has not started yet or undefined action
    - `500 Internal Server Error` - There was an error processing the request on the server.
- **Body:** None

4.4.12 `GET /game/recover`

Recovers a game from the database.

Request

- **Method:** `GET`
- **Endpoint:** `/game/recover`
- **Headers:** None
- **Body:** None

Response

- **Status Codes:**
    - `200 OK` - The request was successful.
    - `500 Internal Server Error` - Error occurred when recovering the data from database
- **Body:** A JSON object of the last game:
    - `allPlayers` - A array of players consisting of all players that was participating in the last game.
    - `gameHasStarted` - A boolean, being true if the last game was started, otherwise false.
    - `currentSong` - A `Song` object representing the current song of the last game.
    - `shuffledSongs` - A array of songs, being all the topSongs, shuffled from the last game.

5 Responsibilities

All of us worked on the lab assignments together, creating the foundation of the application. However, we also individually had some areas in the application of further focus:

Amanda: My primary responsibility for this application has been the frontend aspects such as css styling and making sure the sync between the different players and their respective web browsers are working correctly. I was also responsible for the game over page and the redirecting from the actual game page.
Arvid: The last few week I have spent the majority of my time integrating the database into the server. But I have also created tests for the server as well as the database. Additionally I have created quite a lot of functionality in the server and the show all players button and functionallity in the frontend.
Hampus: My primary responsibilities for the project consists of both developing the frontend and backend. This includes the Login-page and it's connection to the Spotify-login and PreGame-page. This led me to developing the router-spotify connection and the spotify-service in client. I was also responsible for the reveal players feature that reveals all of the players that has the current song as top song. I helped develop the synchronization between different clients so that all players view the same song for example. I'm also responsible of the testing of the frontend.
Elin: My main responsibility has been the server, especially the router layer. I have been responsible for the communication with the Spotify API, writing tests for and adding error handling in the router layer.

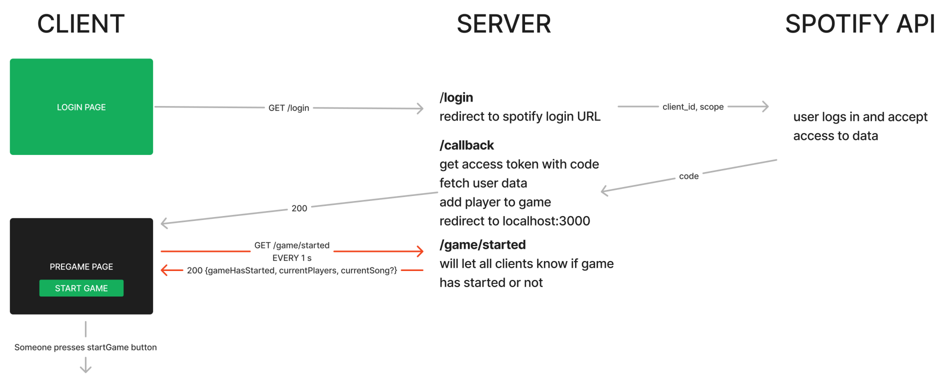

CLIENT
SERVER
SPOTIFY API
LOGIN PAGE
GET /login
/login
redirect to spotify login URL
client_id, scope
user logs in and accept access to data
/callback
get access token with code
fetch user data
add player to game
redirect to localhost:3000
code
200
PREGAME PAGE
START GAME
GET /game/started EVERY 1 s
200 {gameHasStarted, currentPlayers, currentSong?}
/game/started
will let all clients know if game has started or not
Someone presses startGame button

Figure 6: The data flow between client, server and Spotify API before game has started

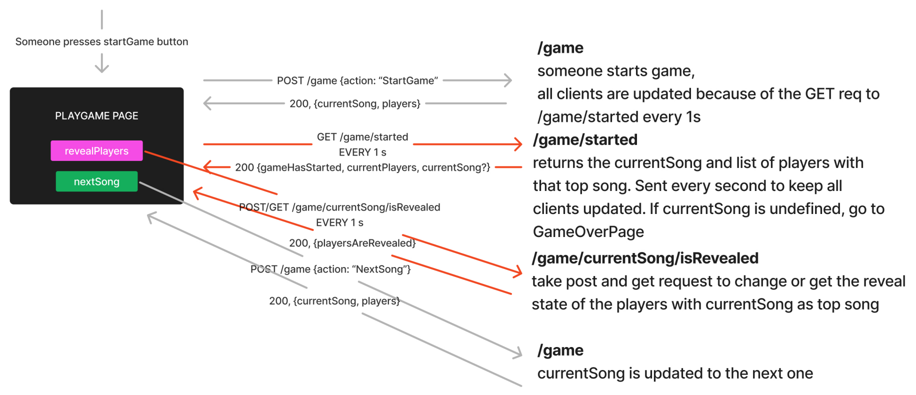

Someone presses startGame button
PLAYGAME PAGE
revealPlayers
nextSong
POST /game {action: "StartGame"
200, {currentSong, players}
/game
someone starts game,
all clients are updated because of the GET req to /game/started every 1s
GET /game/started EVERY 1 s
200 {gameHasStarted, currentPlayers, currentSong?}
/game/started
returns the currentSong and list of players with that top song. Sent every second to keep all clients updated. If currentSong is undefined, go to GameOverPage
POST/GET /game/currentSong/isRevealed EVERY 1 s
200, {playersAreRevealed}
/game/currentSong/isRevealed
take post and get request to change or get the reveal state of the players with currentSong as top song
POST /game {action: "NextSong"}
200, {currentSong, players}
/game
currentSong is updated to the next one

Figure 7: The data flow between client and server after game has started, excluding the GAMEOVER page