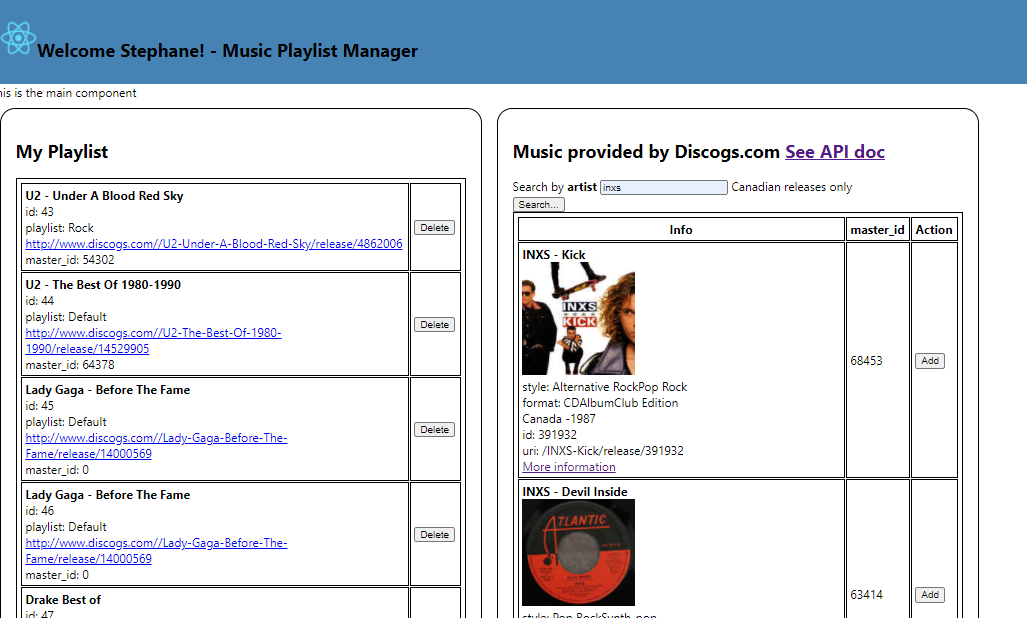
W13 Final Project

## General instructions

1. To be done by teams of 3 students
2. Presented to the group on the last day of class
3. Code shared by two teams is worth 0%

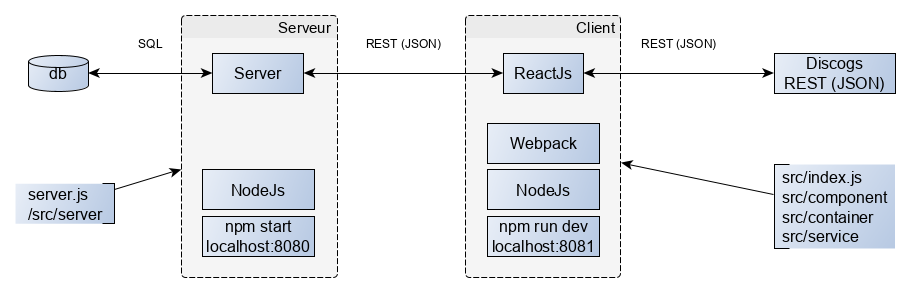


**Example User Interface design**

The objective of the project is to design and code a system to manage a list of favorite music albums. The project consists of 3 parts:

* a Node server with a PostgreSQL database to store the favorite list. The server provides a REST API to access the database
* a React client for user interface. It consists of 2 parts:
  + A component to display the favorite album list obtained from the Node server.
  + A component to search music albums from the music distributing service called Discogs.com. The search results are displayed. The user can add selected albums from Discogs to its favorite list. So this component must also interface with the Node server API to add an album to the favorite list.
  + **Example above is just an example, you are free to do your own layout and visual**

You will create 2 projects in Visual Studio Code. One called **final-project-server** for the Node server and **final-project-client** for the React application



## Creating the database on PostgreSQL server

1. Open pgAdmin4
2. Create a new database **final-project-music-albums**
3. In Moodle you will find an SQL script file **final-project-pg-script.sql**
4. Run the script to create 2 tables:
   1. **track** table contains the list of your favorite music albums (not tracks, really albums)
   2. **playlist** table contains a list of different music genres

DROP TABLE IF EXISTS track;

DROP TABLE IF EXISTS playlist;

CREATE TABLE playlist (

id SERIAL PRIMARY KEY,

title VARCHAR(100) NOT NULL,

creation\_date TIMESTAMPTZ DEFAULT Now()

);

CREATE TABLE track (

id SERIAL PRIMARY KEY,

playlist\_id INTEGER REFERENCES playlist (id),

title VARCHAR(200) NOT NULL,

uri VARCHAR(100) NOT NULL,

master\_id INTEGER NOT NULL

);

## Creating the Node server

1. Create a new folder call **final-project-server**
2. In Moodle your will find **“Creating NodeJS Web Server – Quick Reference”** follow the steps to install required packages
3. Create a file **server.js**
   1. To save time you can copy/modify a **server.js** file from a previous project where we designed a similar API.
   2. Also you can copy file and folder **/src/dao.js** from a previous project. Don’t forget to change the database name in **dao.js**
   3. You only need to handle 3 type of requests in your server API
      1. **GET /tracks** returns the list of all albums in the list. Agree with your partners on what the JSON object returns !
      2. **DELETE /tracks/123** deletes a specific track like id=123
      3. **POST /tracks** adds a new album to the list
   4. Make sure the server runs on port 8000 (not on 3000)
4. Fill the **tracks** table in the database with dummy data using **pgAdmin4**
5. Start the server, make sure it is running when testing the rest of the system,
6. Create an **API\_test.rest** file to test your API. For example <http://localhost:8000/tracks> should return the list of tracks from the database.
7. When the 3 API paths above are working when testing with your file **API\_test.rest** you can tell your partner the server is working.

## Creating the React application

1. In Visual Studio Code select the father directory (do not create a folder for the new application, it will be created automatically
2. **npx create-react-app final-project-client**
3. Check out the PowerPoints **“5-1 React Introduction”** for the other steps (installing eslint etc…)
4. to save time you can copy/paste/modify source files from other project (index.js, Header.js, Footer.js and associated CSS files index.css etc…)

## Create the Playlist component

1. At first create a new React component called **Playlist**
2. While waiting for the server to be up and running, you can test your display and logic code using an hardcoded list of albums in an array, mimicking the expected JSON format returned by the server (show it to your partner, tell him this is what you expect, agree with him !)
3. Create code to display the list of albums returned by the server based on the response JSON object as agreed with the server programmer.
4. Create code to send a **GET http://localhost:8000/tracks** request to your Node server.
5. When the server is ready, test with the server, checking that it displays the list of albums returned by the server.
6. Add a DELETE button next to each item on the list
7. When clicking on DELETE button send a **DELETE /tracks/123** request to the Node server where 123 is the **id** of the track to be deleted. You need to create an event handler for that.
8. The last step is to update the playlist display after deleting an album from the list.

## Create the Discogs component

1. create a new React component called **Discogs**
2. **Create a user account on Discogs.com**
3. **Visit page** <https://www.discogs.com/settings/developers> and click button **create a new application** to obtain a **key** and **secret .** These 2 authentication values are required when querying the Discogs database
4. To search Discogs database you can use this AJAX request

**GET https://api.discogs.com/database/search?key=your\_key&secret=your\_secret&artist="+search\_str+"&country=canada"**

1. Replace **your\_key** and **your\_secret** by your own. The request above is to search by artist name and returns albums released in Canada only. The **search\_str** value can be for example Nirvana.
2. The request returns a JSON response object with an array of 50 results maximum. To know the exact format of the response visit page the Discogs API Documentation page, section about the **SEARCH** request <https://www.discogs.com/developers/#page:database,header:database-search>
3. **Look for response 200 and click on the toggle link**
4. Write code to send a search request to Discogs and display the results. You will need an input box, a search button and an event handler.
5. The array containing the 50 (max) albums is the **.results** property of the response object returned by Discogs. For example the first album info is **results[0].title, results[0].style, results[0].master\_id, results[0].year, results[0].uri, results[0].thumb, etc…**
6. Use the **uri** value in the result to link to the album details on Discogs. You must prepend the string [**http://www.discogs.com/**](http://www.discogs.com/)to the URI value returned **<a href='http://www.discogs.com/some\_uri'>More information</a>**
7. Use the **thumb** value in the result to display the album cover picture. No need to append anything, just use as is  
   **<img src=some\_tumb alt="cover"/>**
8. Next to each result display an **ADD** button.
9. When clicking on the ADD button, send a **POST http://localhost:8000/tracks** request to your Node server to add the album to your favorite list. You need to create an event handler for that.
10. BONUS: have the playlist display automatically update after an album is added to the list.

## Project Grading

* %20 Node Server, testing with API\_test.rest
* %10 React Playlist component displays list from server
* %10 React Playlist component DELETE an album from list
* %10 React Discogs component displays search result
* %5 React Discogs component can link to album details on Discogs and show album cover
* %10 React Discogs component can ADD to album list
* %15 Code design, structure and data, comments,
* %10 CSS visual quality
* %5 Presentation to the group
* %5 Extra Added functionality:
  + playlist auto updates when deleting an album from the list
  + more info saved in database and displayed: album cover image etc…
  + playlist auto updates when adding a new album