IMY 220 Project 2024

Playlist sharing website

The project must be completed by Thursday **3 November**.

We recommend you constantly refer to **this spec as well as your individual deliverable specs** to make sure your project meets all of the requirements by the end.

Make sure you also **continuously test** your project as you add new functionality to make sure everything keeps working as expected.

Contents

Objective

Create a website that allows users to create and share **playlists**. Playlists are collections of songs from music streaming services (in this case, Spotify). Users should be able to create, share, and manage their own playlists.

You must use the technologies and techniques specified in the project spec section to write your own code.

Your website must have a **unique visual design** that matches its theme (i.e., a playlist-sharing website). This could be based on a specific genre of music, an artist/s, or another appropriate theme. You are **not allowed** to

simply use another website's design or create a "knockoff" of another website when designing the visual theme of your site.

Note that you also don't have to use the terminology used in this document to describe functionality, such as "playlist", "friend", etc. You may name these however you want based on the theme of your website.

The purpose of the project is for you to demonstrate the skills that you have learned in IMY 220 - **not anything else**. Therefore, you will **NOT** get marks for proving your proficiency with any other skills OR for using code that is not your own (e.g., from other libraries or templates, or generated by AI).

Submission Instructions

- ***Late projects will not be accepted.***
- ***Email project submissions / Google Drive links will not be accepted.***
- You are required to create a **GitHub repository** to store all of your project files and have some form of source control. The repository must be set to **public** so your code can be marked. The link for this repository should be submitted on **ClickUP** along with your Docker image (when relevant) for each of the deliverables. You will still be required to submit all of your project files for the final deliverable as well as the GitHub link.
- You have to upload your files to **GitHub** as well as **ClickUP**.
- ***IMPORTANT****: Please do not move your files around to different directories (in order to fix the directory structure) after you have finished writing your code. This can result in your code breaking.*

ClickUP

The submission link on ClickUP will close at **exactly 23:59 on Thursday 3 November**.

- **Do not wait** until just before 23:59 to upload because your submission might take longer than usual to upload.
- Instructions:
    - Place: **all your project files**, along with your Docker files and your exported MongoDB database in a folder named with your Position_Surname. Your Position can be found on ClickUP.
    - Compress the **FOLDER** using a zip archive format.
    - Ensure that any **node_modules** folders are **NOT** included in your ZIP folder.
    - Upload the compressed folder (**Position_Surname.zip**) to ClickUP to the relevant project link.

IMPORTANT: Do not expect the lecturer to immediately respond to communications around the submission time. It is your responsibility to make sure you submit all the relevant files well before the submission time.

A Note on AI Use

AI-generated content is **NOT** allowed as a substitute for your own code. **Do not use AI to generate code for your project**. The scope of the project is too large to generate code that is reliable / functional. You will also find it difficult to understand how your code works, which will be detrimental as you **will be required to**

explain parts of your project during your deliverable demo sessions. If you are unable to explain how your code works you will get zero for that section.

Project Demo

You will have to demo your project during the demo sessions to get marks for it.

- You will have to demo using either the latest version of either Google Chrome or Mozilla Firefox.
- When you demo your project, you will be required to show how you have implemented all the instructions by demonstrating the functionality **on the website itself** (not just in the database).
    - You will receive marks for achieving working functionality according to the instructions. For example, if you are required to implement functionality that modifies and displays database data on the website itself, you will receive no marks for simply modifying the database.
- Demo sessions will be made available on ClickUP closer to the deadline.
- Demo sessions will take place on **Wednesday 6 November** and **Thursday 7 November**.
- Project demos will take place in the **Immersive Technology Lab on the 4th floor of the IT building (IT 4-62)**.
- ***You will receive no marks if you do not demo your project.***

*You will **lose marks** in the following conditions*

- If your system does not have all the required functionality.
- If you use any other technology or technique that is not part of the course.
- If you use code that is not your own, i.e., if you use code generators or templates, you will not get marks for it.

*You will get **no marks** in the following conditions*

- If you submit work that is not your own.
- If you do not upload a Dockerfile / Docker Compose file.
- If the Dockerfile / Docker Compose file does not run the application.
- If you do not upload to ClickUP.

Main Pages

All pages must **clearly display the name** of the website creatively and have a **unique visual design**.

Splash Page

- When an unregistered user or a user who is **not logged in** visits the website, they must only be allowed to see a "splash page".
    - *A splash page is the webpage that the user sees first before being given the option to continue to the main content of the site.*
- Splash pages are used to inform new users about the services provided by the website and to promote it. You must thus include a **tagline or other information** that explains the contents and purpose of the website.
- The splash page is the only page that **does not have to have the same consistent layout** as the rest of the web pages.
- The splash page must contain at **least the name of the website as well as a login and registration form**.
    - Users should be able to register using their email address
- ***BONUS****: catch new users' attention and show off your design skills by creating a well-designed landing page that explains the goal of your website. A landing page is generally more detailed compared to a simple splash page and is meant to sell the service you are offering.*

  Divide this page into well-designed sections and use the page scroll to trigger visual effects when the user scrolls down. How you make use of page scroll is up to you.

  (You can find examples of this here: https://www.awwwards.com/inspiration_search/?text=Scrolling)

Home Page

- When a user is **logged in**, the home page must contain the **song feed** for that user in reverse chronological order.
- A user should then be able to switch between their **song feed** and **playlist feed** (see details below). Also display this in reverse chronological order (i.e., newest activity first).

Profile Page

- A user should be able to go to their profile page **when they are logged in**.
- The profile must have a profile **image**. If a user has not uploaded a profile image, there should be a **placeholder** image instead.
    - ***BONUS****: allow users to add an image using drag-and-drop.*
    - ***BONUS****: come up with a way to randomise the placeholder image so that each one is different in some way. This should go beyond simply having a few images to select from.*
- A profile must display **appropriate personal information** about a user, e.g., name, username (if appropriate), pronouns, short bio, links to other social media, etc.. Look at other social media platforms for ideas / guidance on what to include.

  Note*: Since all this information will be publicly visible, no private information must be shown.*

- A user should be able to **edit all their own appropriate information in a logical manner**.
- When viewing the profile page of another user, there must be some logical way to **connect** with a user, i.e., send them a friend request.
- When you visit the profile page of another user, it must **clearly indicate if you are friends on the site**.
- Each user's **list of friends** should appear on their profile as well **but should only be visible if the user who is logged in is friends with the user whose profile they are viewing**. (These details should also be visible if a user is viewing their own profile.)
    - ***BONUS****: in this friend list, show which friend(s) are currently online.*
    - ***BONUS****: when viewing the profile page of another user, show which of their friend(s) are also friends with the current user viewing the profile, i.e., that are "friends with you". This would not make sense for a user who is viewing their own profile, and thus should not show on a user's own profile.*
- A list of all the **playlists created by the user** must appear on their profile page. You must come up with a sensible and visually appealing way to display this. Keep in mind that there might be many created playlists which means you must come up with an intuitive and visually appealing way to provide access to each.
    - A user should also be able to **save a playlist from another user**, which should be shown in a different section on their profile.

Activity Feeds and Playlists

The website should show two activity feeds: a **song** feed and a **playlist** feed.

The song feed should show all songs added by **all users** in reverse chronological order. Songs should be displayed in a similar manner as on the playlist page (see details below).

The playlist feed should show all activity pertaining to playlists for the **friends** of the user currently logged in, also in reverse chronological order. Activity in this case refers to **creating new playlists or saving playlists created by someone else**. Activity should be displayed in an **informative and intuitive way**, e.g., "[user] created a new playlist called [playlist name]" with the relevant information / cover image.

Activities must be displayed in an **aesthetically appealing way**. Simply listing the pieces of activity in Bootstrap cards underneath each other is **not** sufficient; find inspiration from other websites/applications that use news feeds, activity feeds, etc.

Songs

A song in this context refers to an entry with a **name**, **artist**, and song **link**. The following applies with regards to adding songs:

- Any user that has an account can add a song, which in this case involves **adding the required information** using a form.
- A song must also contain the **date that the song was added**, which must be **automatically** added from a timestamp, i.e., the user should **not** be able to add this field manually.

- A song must contain a **link** to the song on a streaming service. This should **only be the URL, not the full embed code**, e.g., https://open.spotify.com/track/4PTG3Z6ehGkBFwjybzWkR8?si=92bc7187e4b44f52
    - The only streaming service you should cater for is **Spotify**. When a song is displayed on the site, the streaming service should be automatically detected from the URL and the song should be shown using the Spotify embed code. This means you may **not display the song links as just links**.
- The user who added a song should also be able to **delete** it.
    - When deleting a song, you should **not delete the entry in the database**. You should just have the ability to **identify** that the song has been deleted. This will be explained in the playlists section of the specification **.

Playlists

A playlist in this context refers to a page with a **name**, **category**, **short description**, **cover image**, **hashtags**, and a **list of songs**. The following applies with regards to creating and managing playlists:

- Any user should be able to create a playlist, which should at least have a **name**.
    - The default text when creating a playlist should be something that indicates the number of playlists a user has already created. E.g., if a user already has 3 playlists, the text might be "New Playlist #4".
- A playlist should also have a **genre**, which has to be selected from a defined list of **genres**, e.g., "pop", "metal". Only admins should be able to manage genres.
- A playlist should also have a **cover image**.
    - ***BONUS**: allow users to add images using drag-and-drop.*
- A user should be able to add **hashtags** to a playlist. You can decide whether users should be able to add them to the text description or in a separate input.
  ***Note**: A hashtag is an **interactive piece of text** that is created by prefixing a predefined symbol to it, e.g., #roadtripmusic. When a user **clicks on a hashtag, it must be the same as performing a search** for that hashtag, i.e., it must display all other playlists that have the same hashtag as part of their summary.*
- A user should be able to **add any song(s)** to a playlist. This should be done via a **context menu** that is accessible per song, i.e., a user should be able to click on the song's menu and add it to any of their created playlists. You need to come up with a usable and intuitive way to integrate this into the website's functionality.
    - A user should also be able to **create a new playlist via this context menu**, which is functionally similar to creating a new playlist "from scratch", except that it already has the selected song included.
- When viewing a playlist by clicking on its page - for example, from the playlist feed or search results - it must have its **own playlist page** which is functionally similar to a profile page but shows the playlist's **details** (name, category, description, cover image, hashtags, etc.). The playlist page should also display the **basic information for each song** (name, artist) and the **embed** code.
    - Songs should be displayed **in the order in which they were added** to the playlist.
    - Users should also be able to **add songs to other playlists** from this page as well, thus it should also include the previously-mentioned context menu with the relevant functionality.

    - ** If a song has since been **deleted**, the name and artist should still appear in the playlist but it should **not** have the embed code and should be **visually indicated as being deleted**. Users should not be able to add deleted songs to playlists.
- Users should be able to **manage (edit) the information for their own playlists** as well as **remove** songs from playlists.
    - ***BONUS**: allow users to reorder songs in a playlist in an intuitive way.*
- Users should be able to **delete** playlists they have created, which should remove it from the database completely and it should not appear on feeds or search results. However, deleting playlists should **not** have any effect on the individual songs themselves.
- Users should also be able to post **comments** on a playlist page and should be able to **include an image along with their comments**. This should be displayed in a **logical** way, for example, if there are 50 comments, only a few of them should be displayed on the playlist's page and the rest should be accessible through some other mechanism.
    - ***BONUS**: allow users to **like** comments which visually indicates the number of likes per comment.*
    - ***BONUS**: allow the user who created the playlist to **pin** a comment, in other words, this comment would always display first and should be **visually indicated** as being pinned.*
- How a user manages their own playlists is up to you, it just needs to be user-friendly and well-integrated into the rest of the website.
    - ***BONUS**: create functionality that allows users to add **a whole playlist to another** existing / new one, i.e., all the songs from one playlist should then be added to the other. Duplicate songs should be ignored.*
    - ***BONUS**: when adding a song to a playlist, create functionality that detects if the song is **already present** in a playlist, and **notifies** the user. The user can then **confirm** if they would still like to add the song as a duplicate.*
    - ***BONUS**: create functionality that allows for **co-authors to playlists**, i.e., more than one user can add songs to the same playlist. Only the user that created the playlist should be able to have this playlist display on their profile, change the playlist details (name, description, image, etc.) or add users as co-authors. Other users can simply add / remove songs from the playlist. Collaborative playlists cannot be private.*

Search

Add **search** functionality that allows a user to search for the following:

- **Other users** by name or username
- **Songs** by name or artist
- **Playlists** by name, genre or hashtags.
- ***BONUS**: The search functionality must be able to find something **even if the search term is a little incomplete or spelled incorrectly**, i.e., fuzzy string searching: the technique of finding strings that match a pattern approximately (rather than exactly).*

  ***Note**: this does **not** refer to simply adding wildcards to your searches. A fuzzy search should be able to find words that have letters missing, swapped around, etc.*

- ***BONUS**: Add **autocomplete** functionality to suggest results as the user is typing. You would need to come up with an intuitive design for this.*

Search results must be **interactive**, i.e., where appropriate they must return links, for example, to user profile pages, playlist pages, etc.

User Roles

The Unregistered User

An unregistered user is a user who does **not currently have a profile or is not logged in**. An unregistered user can do the following:

- **Register** as a new user on the splash page. The splash page does not offer any functionality other than allowing the user to **register** or **login**. The user needs to provide at least their name, email and a password when registering.
  ***Note**: An unregistered user **cannot interact with the website in any other way**.*

The Registered User

A registered user can do the following:

- **Log in** and **log out**.
- See all **activity** from their song and playlist feed.
- **Search** for playlists or users and view other users' profile pages.
- **Edit** all appropriate fields on their own profile.
- **Add** new songs and playlists.
- **Delete** songs they have added (this is discussed further under Songs).
- **Manage** (edit/delete) their own playlists (this is discussed further under Playlists).
- **Delete** their own profile. Deleting their profile removes their user account from the database.
  Note: This should automatically delete all playlists that were added by that user. The same applies as when deleting an individual playlist.

User Interaction

A registered (logged-in) user can do the following:

- **Connect** with other users by following them or sending a friend request to connect.
- **View** the complete user profiles of users that they are friends with.
  ***Note**: When a user is **not friends** with another user, they can only see that user's name, and profile picture.*
- **Unfriend / unfollow** other users.

The Admin (User Account)

Some users should be able to log in as **admin**. The administrator has **all the powers of a registered user**.

Additionally, they can do the following:

- Manage (edit/delete) all **activity**.

- Manage (edit/delete) all **playlists**.
- Manage (edit/delete) all **user** accounts.
- Delete all **songs**.
- Add new **genres**.

- ***BONUS****: add functionality for admins to* ***verify accounts****. A (non-admin) user should only be able to request that their account be verified if their account is more than one week old, if they have added at least one song, and created at least one playlist. Admins should have access to a list of verification requests that they can approve/deny on a case-by-case basis. Once an account is verified, it should be visually indicated on the user's profile somehow.*

Usability

- Take care to design a website that is **intuitive and easy to use**. It is your own responsibility to come up with usable and intuitive interactions for the website's functionality.
- For example:
    - A log out button must not appear if a user is not logged in, or a login button must not be visible if a user is already logged in.
    - When a user **creates** an account, they must **automatically be logged in** with that account.
    - **All form fields must have working labels**, i.e., when you click on the label, focus is given to its accompanying input.
    - **Well-designed error messages should appear when appropriate**, e.g., when a user enters incorrect login details.
    - **Navigation must stay consistent** throughout pages and must provide a **logical way to access the core functionality**.
    - Text that is interactive must **look interactive** (buttons, links, hashtags, etc.).
    - **Clicking on the website logo/name must take the user to the home page**.
- ***Tip****: Test the usability of your site by asking other people (especially non-IT students) to interact with it to find out how easy it is for them to navigate.*
- **You will be penalised for sloppy design that is detrimental to the user experience, such as debugging alerts (or any unnecessary alerts)**.

- ***BONUS****: allow the user to toggle between a standard (light mode) and a dark theme (night mode).*

Visual Design

- Your website **must have a design that is up to professional standards**. Refer to well-designed websites for inspiration, e.g., https://www.awwwards.com, https://dribbble.com/shots/popular/web-design, etc.
- **Use colour schemes that match**, e.g. browse the colour lovers website (https://www.colourlovers.com/) or use Adobe Color (https://color.adobe.com/) or Coolors (https://coolors.co/) to find colour palettes and patterns or search online for colour scheme creators.

- You **must** redesign all form elements to fit with your website's visual theme, in other words, you are **not allowed to have any default HTML/Bootstrap** input boxes, buttons, etc.
- You must include **at least two and maximum three fonts** as part of your website's design. These may **not** be the default fonts for HTML, Tailwind, or Bootstrap. Use fonts that look good and fit the overall design of the site. **You must embed fonts for your website. Consider using something like Google Fonts for this.**
  ***Note**: Fonts must be legible. Do not use more than three fonts for the entire website. Do not use stylised fonts for body text. Do not use standard Windows fonts. Embed fonts with CSS.*

Technologies and Techniques

You may only use the following technologies and techniques:

Technologies:

- **Valid Javascript code everywhere** (no exceptions). This includes any JSX (React-based HTML) and additional styling / class names that you may include in your application. Ensure that valid HTML is returned when the application is viewed in the browser (view source), i.e., ONE doctype, ONE head, ONE body.
- **CSS or TailwindCSS** for styling, embedding of fonts, etc. Do not use inline or embedded style sheets. Use CSS variables where appropriate. No component libraries (like Material UI / shadcn / ChakraUI) are allowed.
- **CSS cascading variables** for aspects relating to the style identity of your website, e.g., your theme colours and fonts.
- You should use **GitHub** throughout the semester and submit a link to your repository for each deliverable.
- You are permitted to use Bootstrap but you will need to customise the styling of the components. You will have to use either CSS or TailwindCSS to style these elements according to your website theme. **You will lose marks if any default Bootstrap is styling visible on your website**.
- **JSON** to transfer your data back and forth between client and server (for example, during fetch calls).
- At least one instance of a Promise for asynchronous behaviour (this can include the built-in async behaviour of using fetch).
- ***BONUS**: demonstrate branching in Github by having different branches for logically separated website content, e.g., having a branch with a README file explaining your project, having a branch dedicated to adding a certain feature (e.g., editing playlists, adding songs, friend/unfriend, etc.) You must have **at least one branch that has been merged back into the main branch** to get the bonus.*

Techniques:

- **Responsive web design with at least two breakpoints.**
  ***Note**: Your website must display correctly on any resolution.*
- **Error-free** React and NodeJS code. Sensible coding, you must only connect to your database in one place. Keep your code DRY (**D**on't **R**epeat **Y**ourself).

    - Separate your code into reusable components. For example, one component for an add / edit form, one component for viewing a playlist summary, one component for the context menu etc.
- **Session / cookie management**.

Favicon

- **Embed** a favicon that fits the theme of your website in all your pages.

Database

- There must be **LOGICAL** test data in the database for demo purposes.
- There must be *at least* the following:
    - 10 existing songs in the database.
    - 5 playlists with at least 2 songs in each playlist.
    - 10 registered users.

  NB: for general testing purposes, you must have one regular user account with the following login details:
  Email address: *test@test.com*
  Password: *test1234*
  This user must have at least 1 added song, 1 created playlist, and 2 friends.
- For the purposes of testing and marking, do **not** hash your passwords.
- **Remember to export your database and include it in your final submission.**

Directory Structure

- Structure all your files into a logical directory structure.
- Make use of correct file naming conventions.
- Use **relative file paths** for references.
- Put images, fonts, CSS files in their respective folders.

Completing the Project

You have until 3 November to complete the project.

You have to submit on ClickUP. **Remember to download and double-check your submissions. Late submissions / submissions in the incorrect place will not be accepted.**

To avoid running into problems on 3 November, it is crucial that you test your project by running the Docker image locally throughout the semester. Ideally, you should test that everything still works each time you add new functionality. **Avoid blind-coding at all costs.** Do not wait until just before the submission deadlines to start testing your project.

Make sure to retest the entire site after submitting to ensure that what you will be assessed on is fully functional.

Docker - Things to Look out for

- Your **entire application** should be Dockerized, this includes the frontend and backend code.
- On marking day, the markers will download your submission and run the relevant commands to run your Docker image.
- Ensure that you test your application thoroughly using Docker. Some things can differ between running your application locally (outside of Docker) and running it using Docker.

Mark breakdown

Deliverable 0 - Wireframe: 10%

Deliverable 1 - Front-end: 15%

Deliverable 2 - Back-end 20%

Final Project (Deliverable 3): 55%

Start Today!