

The Album Archive

Final Project Technical Report

SE/COM S 3190 – Construction of User Interfaces Spring 2025

Team Members:

Member 1 - will2027@iastate.edu

Member 2 - owentg3@iastate.edu

May 11, 2025

1. Introduction

Overview of the problem, motivation, users, and goals. State whether the project is original or inspired.

A lot of times people want to keep track of some of their favorite music/albums, but can never remember what their names are. This site functionality allows you do add any album you want to your collection, and even add albums that you haven't heard of to the home page! The idea overall is original.

2. Project Description

Explain major features, user flow, CRUD operations and entities affected.

We used **MongoDB** to store album data, enabling full **CRUD operations**—Create, Read, Update, and Delete. This allowed users to add, view, edit, and delete albums from the database. MongoDB's flexibility made it easy to store and manage the different fields associated with each album.

3. File and Folder Architecture

Describe structure of frontend/, backend/, Documents/. Include a folder tree diagram.

```
/backend  
  
  Final_project.js  
  
/frontend  
  
  /src  
  
    /assets  
  
    /pages (Where all jsx code is)  
  
    App.jsx  
  
    main.jsx  
  
    tailwind.config.js  
  
/documents
```

This is the general folder structure taken from my ReadMe file, obviously there are other key files in there not mentioned but implied like nodemodules and others.

4. Code Explanation and Logic Flow

4.1. Frontend–Backend Communication

Describe how API requests are made and handled.

API requests are made from the React frontend using the `fetch` API. When a user submits a form, such as adding a new album, a POST request is sent to the backend server at the appropriate endpoint (e.g., `/addAlbum`). The backend, built with Express.js, receives the request, processes the data, and interacts with MongoDB to perform the desired operation. After completing the request, the server responds with a status or data, which the frontend uses to update the UI or navigate the user accordingly.

4.2. React Component Structure

Overview of component hierarchy, use of props/state.

The app uses a modular React component structure, with key components like Home, AddAlbum, AlbumCard, MyCollection, and ConfirmationPage. State is managed with `useState` in higher-level components and passed down via props for rendering and interaction. Components like AlbumCard receive album data and handlers as props. Navigation is handled with `react-router-dom`, and updates occur through local state or callbacks to parent components.

4.3. Database Interaction

Overview of how the database is used.

The MongoDB database is used to store album data through standard CRUD operations—Create, Read, Update, and Delete. When users add, remove, or update albums, those changes are reflected in the database. The app then fetches and displays this data dynamically to keep the interface in sync with the backend.

4.4. Code Snippets

Include 2–3 meaningful snippets (React component, backend route, DB logic).

```
app.post("/addAlbum", async (req, res) => {  
  
  try {  
  
    await client.connect();  
  
    console.log("MongoDB connected");  
  
    const newDocument = {  
  
      name: req.body.name,  
  
      artist: req.body.artist,  
  
      rating: req.body.rating,  
  
      image: req.body.image,  
  
      link: req.body.link,  
  
    };  
  
    const result = await client.db('albums').collection('albums').insertOne(newDocument);  
  
    res.status(201).json(result);  
  
  } catch (error) {  
  
    console.error(error);  
  
    res.status(500).json({ message: 'Server error' });  
  
  }  
})
```

```
    genre: req.body.genre,

  };

  const result = await db.collection("albums").insertOne(newDocument);

  res.status(200);

  res.send(result);

} catch (error) {

  console.error("Could not add the new album" + error);

  res.status(500);

  res.send("Error adding new album");

} finally {

  await client.close();

}

});
```

```
const handleAddAlbum = async () => {

  try {

    const response = await fetch('http://localhost:8080/addAlbum', {

      method: 'POST',

      headers: {

        'Content-Type': 'application/json',

      },

      body: JSON.stringify(newAlbum),

    });

    if (response.ok) {

      console.log('Album added successfully');

      setAlbums((prev) => [...prev, newAlbum]); // Update local state if
successful
```

```
        navigate('/home'); // Navigate to the home page after adding the
album

    } else {

        console.error('Failed to add album:', response.statusText);

    }

} catch (error) {

    console.error('Error adding album:', error);

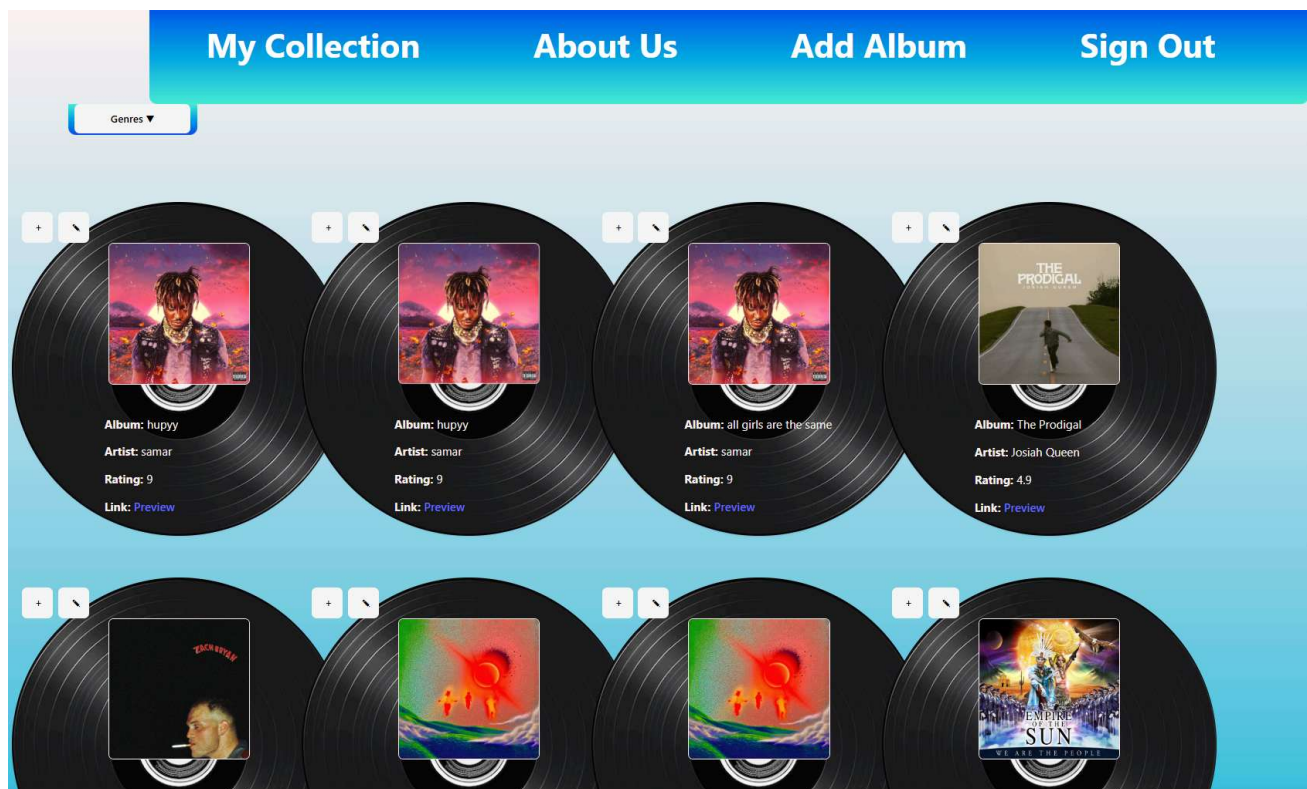
}

};
```

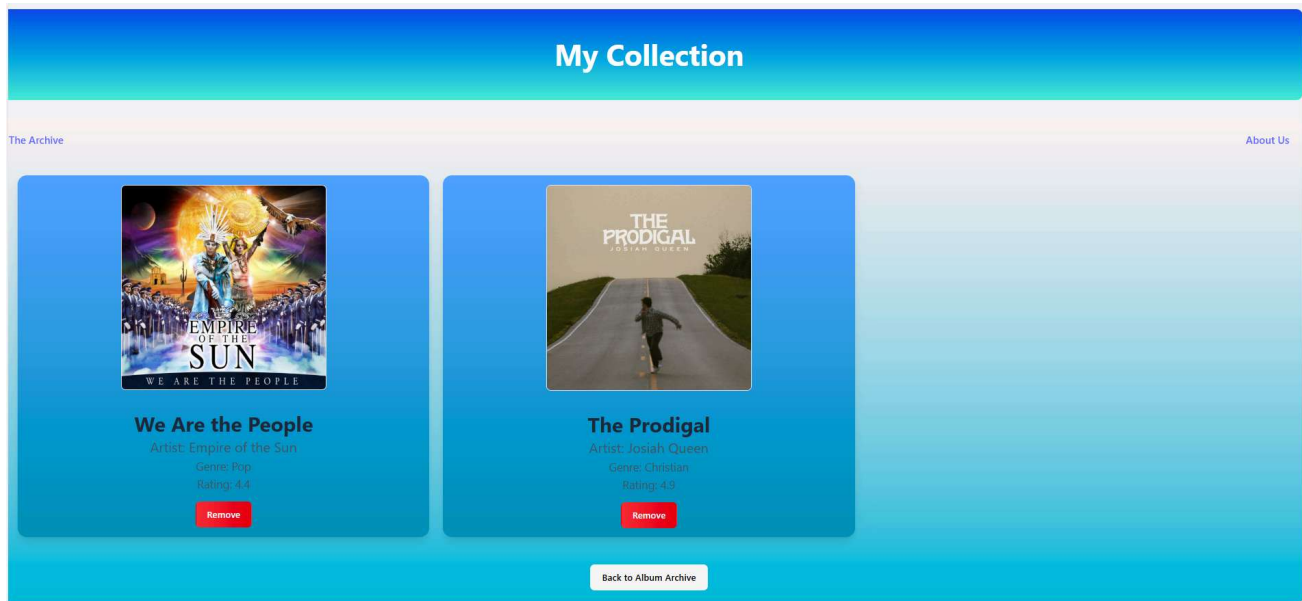
5. Web View Screenshots and Annotations

Insert and annotate full-page screenshots for 4 features (2 for each team member)

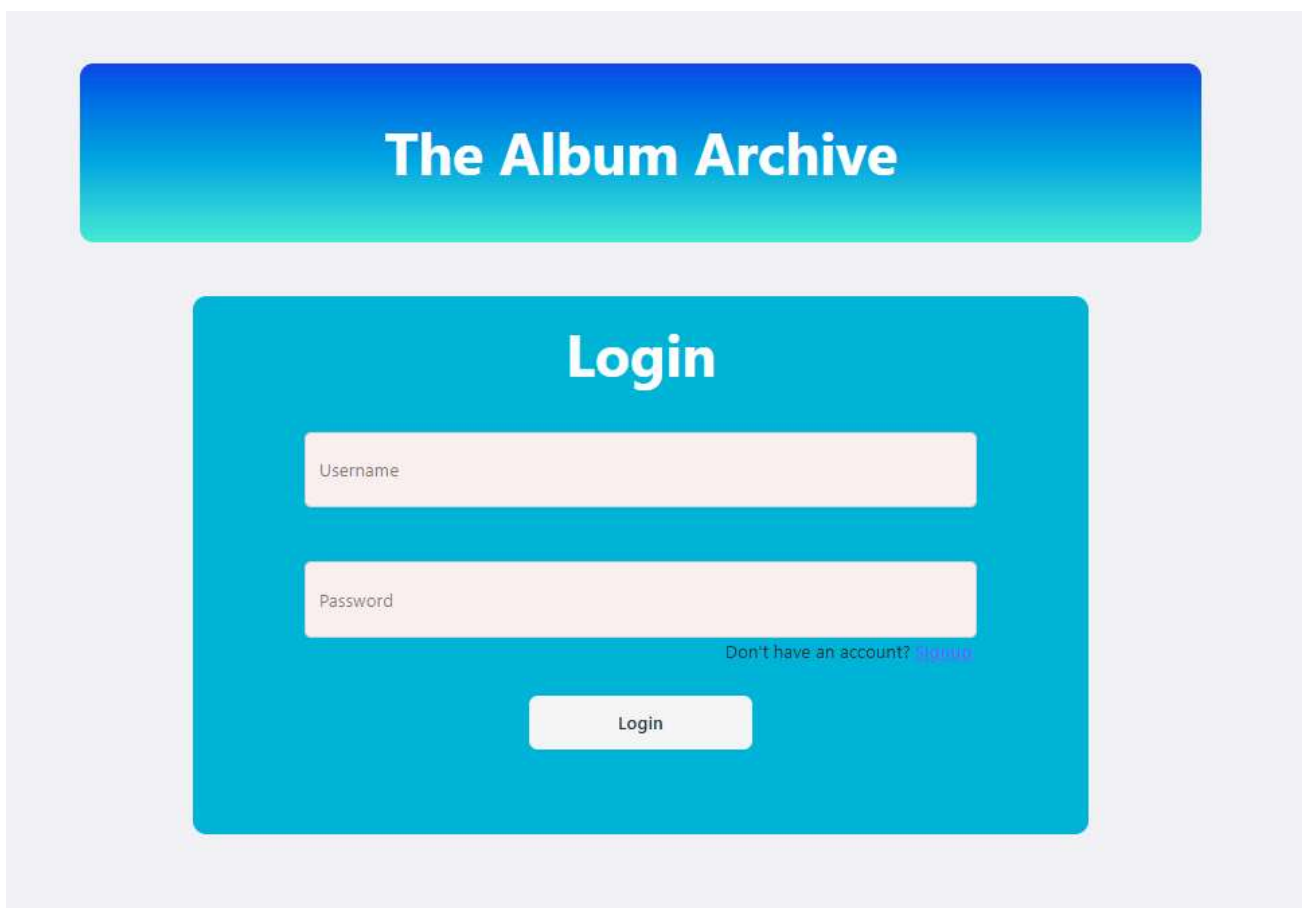
Description: What this page does and what the user can do.



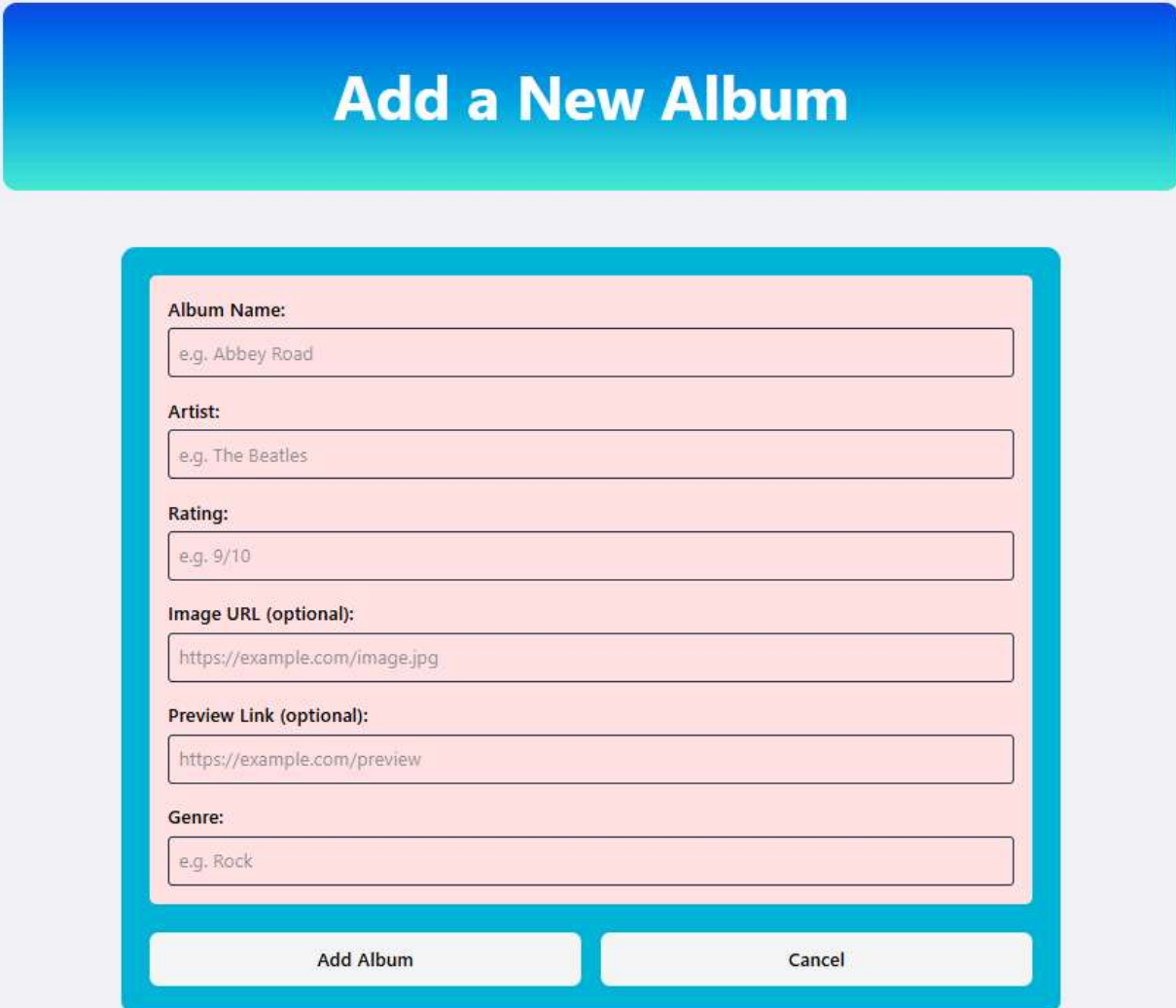
this is our home page with random sample data, you can navigate to any site you want using the buttons (+ button adds to collection, pencil button edits the album)



This is our collection page consisting of the data from MongoDB, you can remove from your collection here, and then navigate back to the home page or to the about us page. (We didn't allow you to navigate to the addAlbum function because it wouldn't be adding to collection, you must first add the album to the home page, then add to the collection)



Simple login page here: must log in or sign up to access the webpage: they are stored within MongoDB as well



The image shows a web form titled "Add a New Album" in a blue gradient header. The form itself is a light pink box with a blue border. It contains several input fields with placeholder text: "Album Name:" (e.g., Abbey Road), "Artist:" (e.g., The Beatles), "Rating:" (e.g., 9/10), "Image URL (optional):" (https://example.com/image.jpg), "Preview Link (optional):" (https://example.com/preview), and "Genre:" (e.g., Rock). At the bottom are two buttons: "Add Album" and "Cancel".

Add Album page is pretty simple as well, wanted to keep it simple to focus on how the main pages look. Takes in data, and then submits it to store in MongoDB

6. Installation and Setup Instructions

Step-by-step instructions to run frontend and backend. Mention environment variables and DB setup.

cd backend

node Final_project.js

cd frontend

npm run dev

7. Contribution Overview

A table or list mapping each feature to the responsible team member.

Login + SignUp - Will Frommelt

AddAlbum - Owen Gilbertson

EditAlbum - Owen Gilbertson

Home - William Frommelt

About - William Frommelt

Confirmation - Owen Gilbertson

MyCollection - Owen Gilbertson

App.jsx - Owen Gilbertson/William Frommelt

Final_project.js - Owen Gilbertson/William Frommelt

MongoDB - Owen Gilbertson/William Frommelt

8. Challenges Faced

List 2–3 significant development/debugging issues and how you addressed them.

1. Getting the main page's formatting to look how we wanted it to. The album pictures in the background wouldn't line up correctly, and this took us longer than expected to fix. Along with that was figuring out how to use special formatting to create a space to hold the picture the way it does.
2. Getting the login to check the credentials the user is inputting to ensure they're valid. It was hard to debug because our error messages at first weren't good enough to debug, but over time, we finally organized them more and made it easier to find the issue.

9. Final Reflections

Summary of learning outcomes and things you would improve.

Throughout this project we learned a lot about utilizing backend development through mongo DB to go along with react, and using tailwind in-line styling. Although there were some learning curves when it felt like we weren't sure how to fix things, we feel like we ended with a successful project after the time we put into it. Some things we could improve are making it harder to login and sign up (needing an access code for company purposes, so not everyone can sign up).