For The Record - Music Store

Requirements Analysis

===

Nikolaos Efstathioy
Darrel Nitereka
Sabri Tahir

Table of Contents

==

Introduction

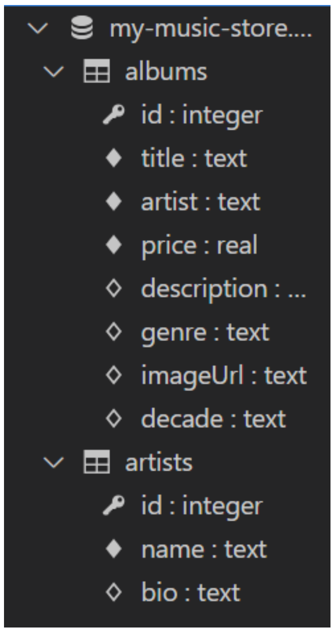
The Requirements Analysis document is a document that provides information regarding our website “For The Record”. More specifically, this document covers use cases and user roles. This document will walk you through the various user roles and the use cases that these roles govern. Each use case provides step-by-step information regarding the case as well as any exceptions and requirements.

Code Overview

Our program has a wide-array of code being utilized. We felt it would be useful for our document to have a walk-through of the program before diving into the use cases.

Before we discuss our models, controllers, and views you should have a more general overview. Our program is made using express and it utilizes a sql database. We are using bootstrap for styling.

SQL Database:

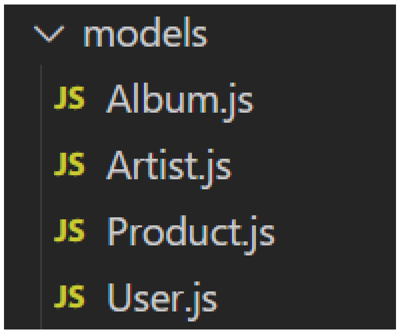


```
my-music-store....
  albums
    id : integer
    title : text
    artist : text
    price : real
    description : ...
    genre : text
    imageUrl : text
    decade : text
  artists
    id : integer
    name : text
    bio : text
```

I. Models

Our models are used to get specific data from our SQLite database as well as to create data for our SQLite database.

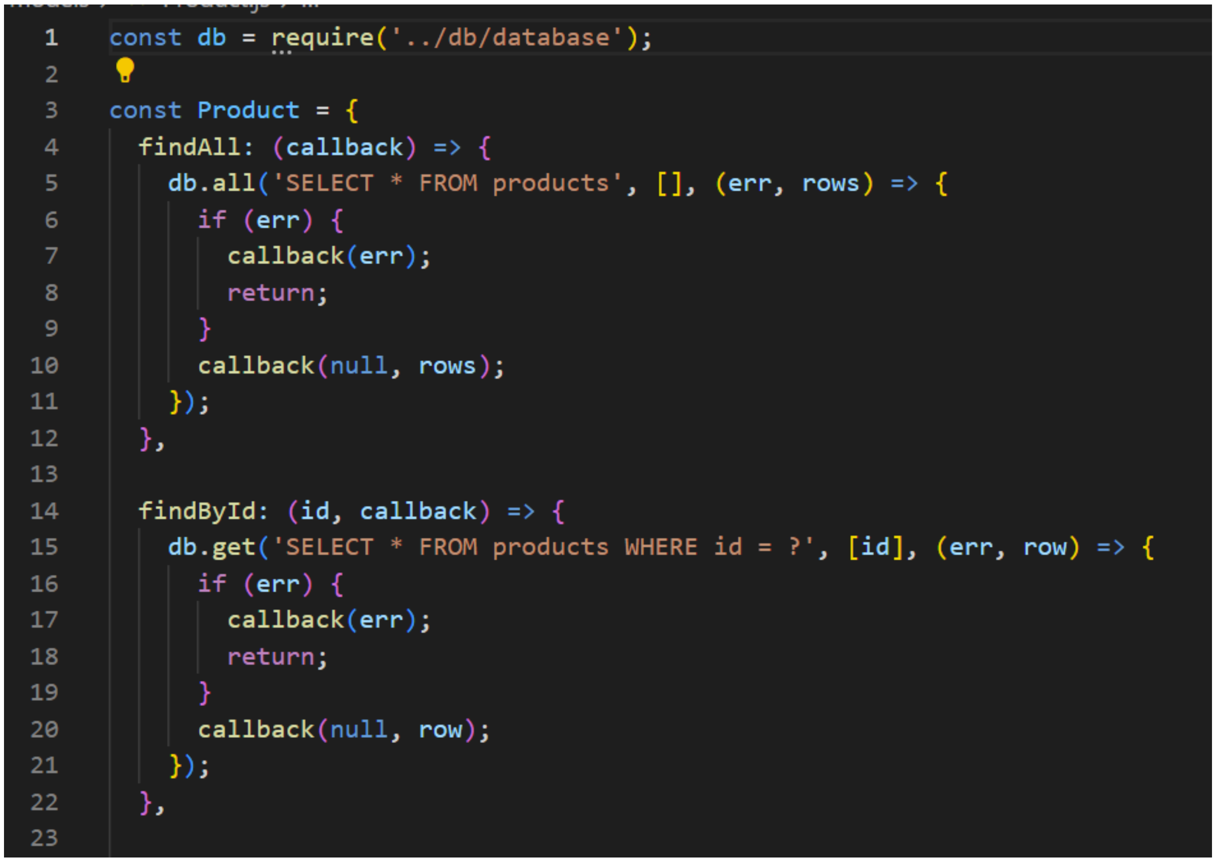
The models:



```
models
  Album.js
  Artist.js
  Product.js
  User.js
```

All four of these models contain various search functions such as findByID, findAll, etc. These models also contain create functions.

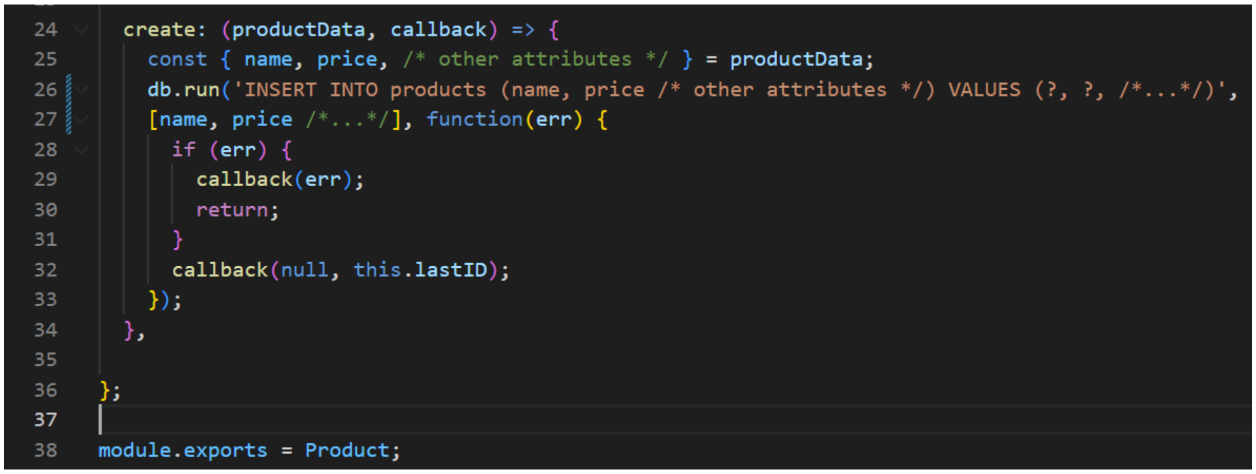
Example model code:



```js
const db = require('../db/database');

const Product = {
  findAll: (callback) => {
    db.all('SELECT * FROM products', [], (err, rows) => {
      if (err) {
        callback(err);
        return;
      }
      callback(null, rows);
    });
  },

  findById: (id, callback) => {
    db.get('SELECT * FROM products WHERE id = ?', [id], (err, row) => {
      if (err) {
        callback(err);
        return;
      }
      callback(null, row);
    });
  },

```

Here you can see two queries from Product.js where we find all products or we find products via id. Both have robust error cases.

Example model code continued:



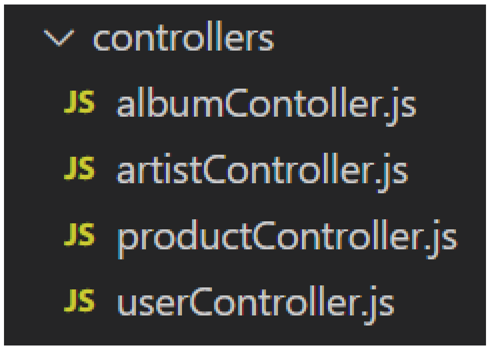
```
  create: (productData, callback) => {
    const { name, price, /* other attributes */ } = productData;
    db.run('INSERT INTO products (name, price /* other attributes */) VALUES (?, ?, /*...*/)',
    [name, price /*...*/], function(err) {
      if (err) {
        callback(err);
        return;
      }
      callback(null, this.lastID);
    });
  },

};

module.exports = Product;
```

Here you can see a create function from Product.js where we insert data into our database.

II. Controllers

Our controllers are used to get data from our SQLite database.

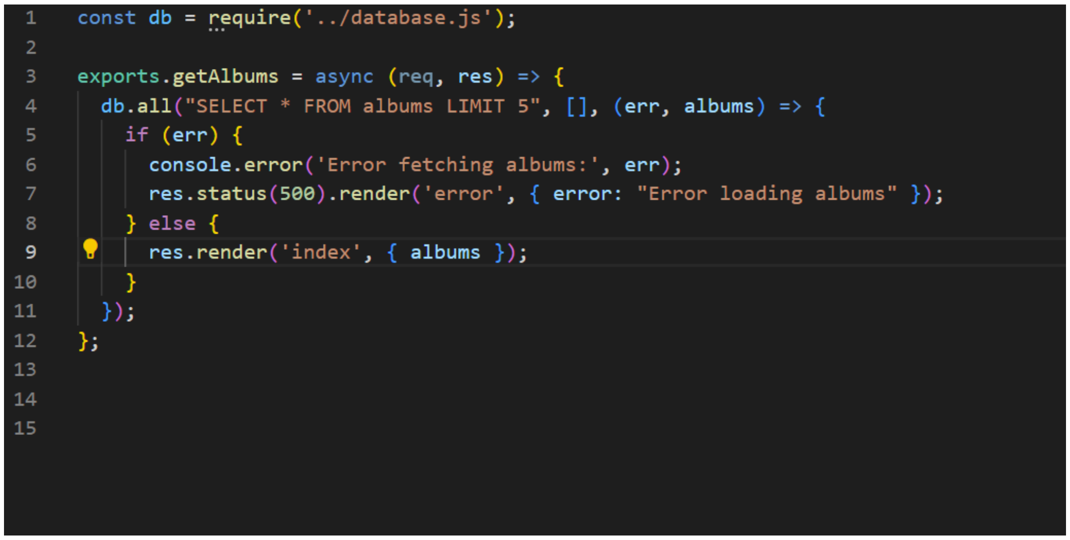
The controllers:



```
controllers
  albumContoller.js
  artistController.js
  productController.js
  userController.js
```

albumController.js and artistController.js both contain a single asynchronous function that queries their respective table. productController.js doesn't have a table but will be used when we add a products table. userController.js ended up not being used in product.

Example controller code:



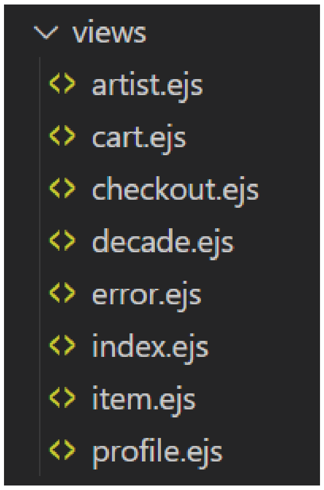
```
const db = require('../database.js');

exports.getAlbums = async (req, res) => {
  db.all("SELECT * FROM albums LIMIT 5", [], (err, albums) => {
    if (err) {
      console.error('Error fetching albums:', err);
      res.status(500).render('error', { error: "Error loading albums" });
    } else {
      res.render('index', { albums });
    }
  });
};
```

As you can see here this is the getAlbums query found in the albumController.js, it uses an asynchronous function to pull the top 5 albums with multiple error handling cases in case it fails.

III. Views

Our views contain the code for displaying their respective page for the website.

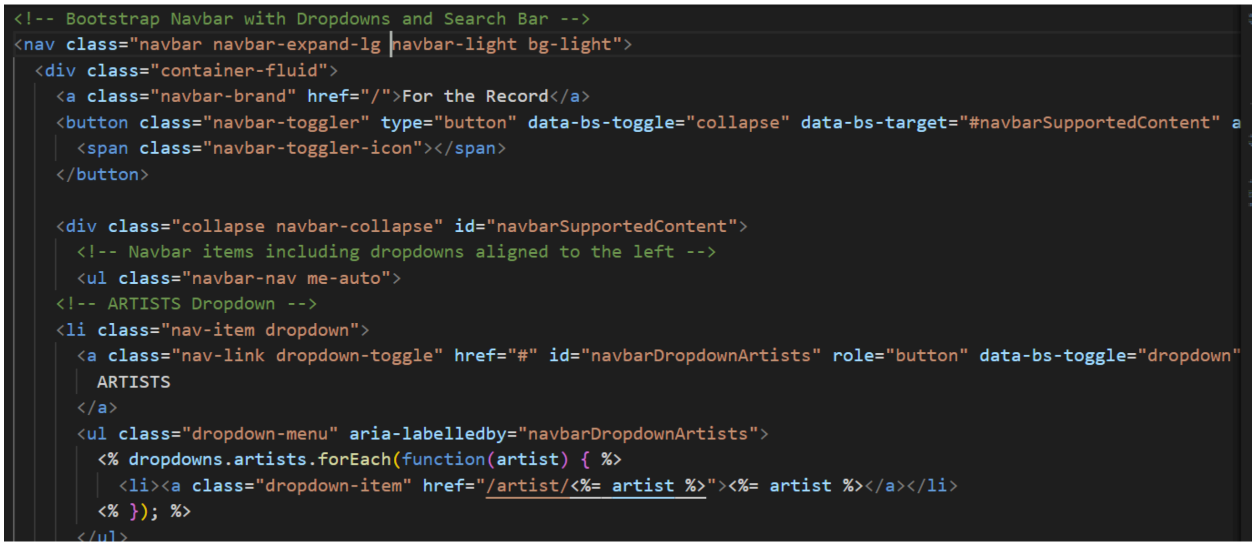
The views:



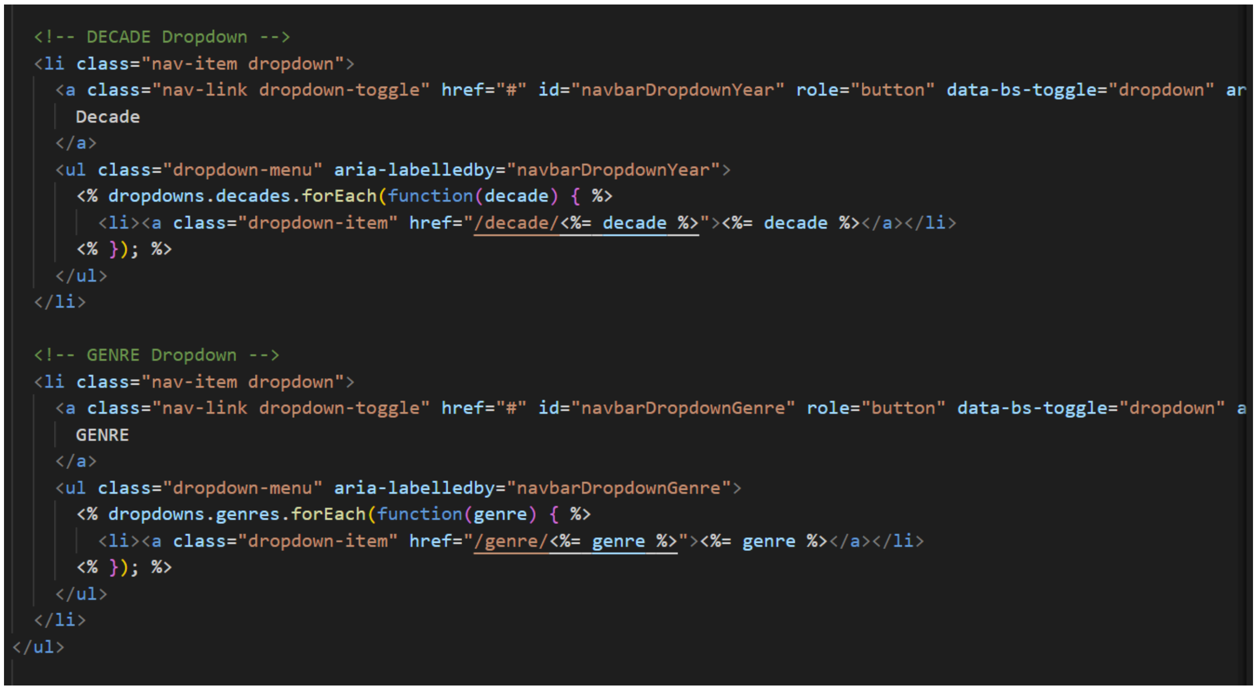
```
views
  artist.ejs
  cart.ejs
  checkout.ejs
  decade.ejs
  error.ejs
  index.ejs
  item.ejs
  profile.ejs
```

Each of these pages contain the same navigation bar code at the top. This is because our website has the same navigation bar on every page so that it's easier to traverse the site.

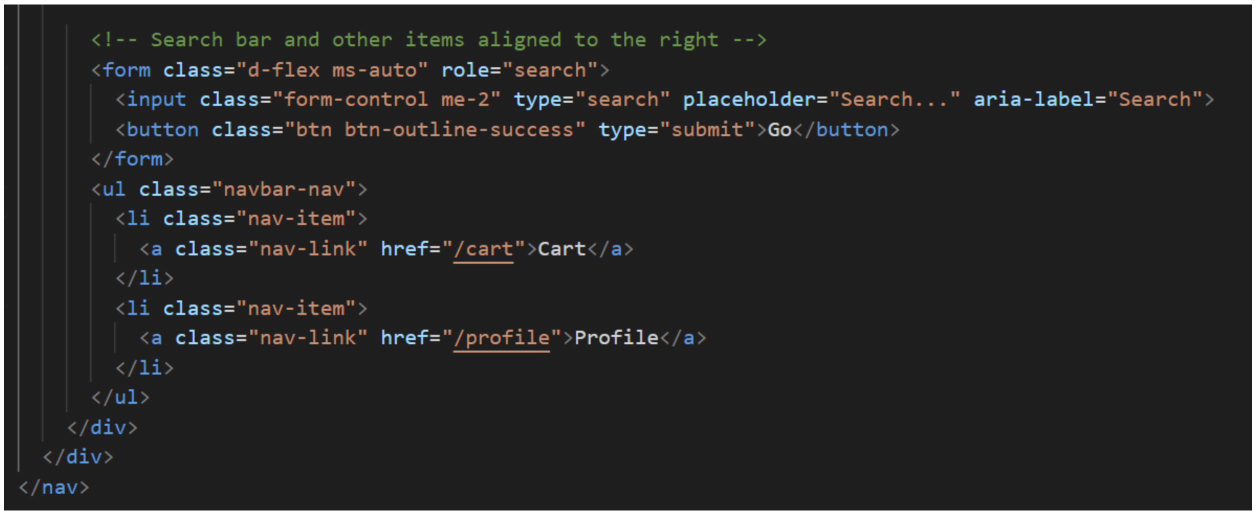
Boilerplate navigation bar code:



```
<!-- Bootstrap Navbar with Dropdowns and Search Bar -->
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="/">For the Record</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" a
      <span class="navbar-toggler-icon"></span>
    </button>

    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <!-- Navbar items including dropdowns aligned to the left -->
      <ul class="navbar-nav me-auto">
    <!-- ARTISTS Dropdown -->
    <li class="nav-item dropdown">
      <a class="nav-link dropdown-toggle" href="#" id="navbarDropdownArtists" role="button" data-bs-toggle="dropdown"
        ARTISTS
      </a>
      <ul class="dropdown-menu" aria-labelledby="navbarDropdownArtists">
        <% dropdowns.artists.forEach(function(artist) { %>
          <li><a class="dropdown-item" href="/artist/<%= artist %>"><%= artist %></a></li>
        <% }); %>
      </ul>
```



```
  <!-- DECADE Dropdown -->
  <li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle" href="#" id="navbarDropdownYear" role="button" data-bs-toggle="dropdown" ar
      Decade
    </a>
    <ul class="dropdown-menu" aria-labelledby="navbarDropdownYear">
      <% dropdowns.decades.forEach(function(decade) { %>
        <li><a class="dropdown-item" href="/decade/<%= decade %>"><%= decade %></a></li>
      <% }); %>
    </ul>
  </li>

  <!-- GENRE Dropdown -->
  <li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle" href="#" id="navbarDropdownGenre" role="button" data-bs-toggle="dropdown" a
      GENRE
    </a>
    <ul class="dropdown-menu" aria-labelledby="navbarDropdownGenre">
      <% dropdowns.genres.forEach(function(genre) { %>
        <li><a class="dropdown-item" href="/genre/<%= genre %>"><%= genre %></a></li>
      <% }); %>
    </ul>
  </li>
</ul>
```



```
      <!-- Search bar and other items aligned to the right -->
      <form class="d-flex ms-auto" role="search">
        <input class="form-control me-2" type="search" placeholder="Search..." aria-label="Search">
        <button class="btn btn-outline-success" type="submit">Go</button>
      </form>
      <ul class="navbar-nav">
        <li class="nav-item">
          <a class="nav-link" href="/cart">Cart</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="/profile">Profile</a>
        </li>
      </ul>
    </div>
  </div>
</nav>
```

As you can see, there is a lot going on here. Simply put, the navigation bar contains a couple drop-down elements for different decades and genres. The navigation bar also contains a search bar and links to the shopping cart, user profile, etc.

Besides the navigation bar, each individual ejs file is displaying information via the SQLite database. We recommend viewing the ejs files yourself to have a full understanding of each page.

Explanation of Use Case Contents

Name of use Case: Short name for the use case that should lend itself to the objective of the use case

Description: Description of both the reason as well as the expected outcome of the use case

Actors: Actors that may be involved in the given use case

Precondition: Any conditions that need to be met for the system to work

Postcondition: State of the system after the conclusion of the use case

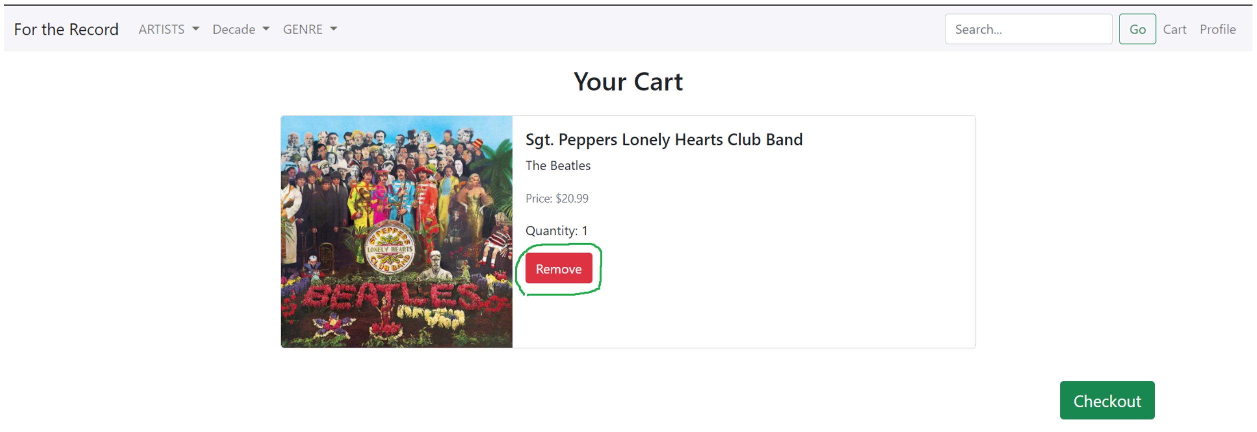
Flow: Describes the steps of the use case

Alternative Flows: Describes any alternate ways in which the use case may be met/used

Exceptions: States any exceptions to the given flow/alternative flows as well as the steps taken to resolve those exceptions

Requirements: Describes anything necessary for the given use case to work

I. Use Cases



Your Cart

Sgt. Peppers Lonely Hearts Club Band

The Beatles

Price: $20.99

Quantity: 1

Name of use Case: Remove item from cart
Last Updated: 4/21/2024
Date Created: 3/7/2024
Description: Buyer clicks the “Remove” button on an item in their shopping cart
Actors: User
Preconditions:

1. Have an item in shopping cart

Postconditions:

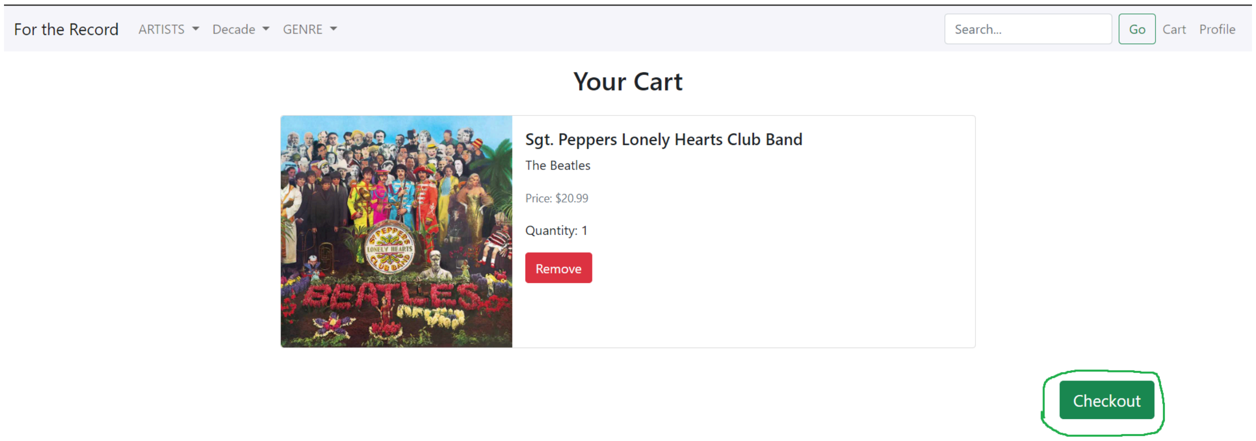
1. Item count is removed from cart list and screen

Flow:

1. Buyer is in the shopping cart screen
2. Buyer identifies an item they are no longer interested in
3. Buyer clicks the correct button

Alternative Flows:
Exceptions:
Requirements:

1. There needs to be an item in the shopping cart


For the Record ARTISTS Decade GENRE Search... Go Cart Profile

Your Cart

Sgt. Peppers Lonely Hearts Club Band

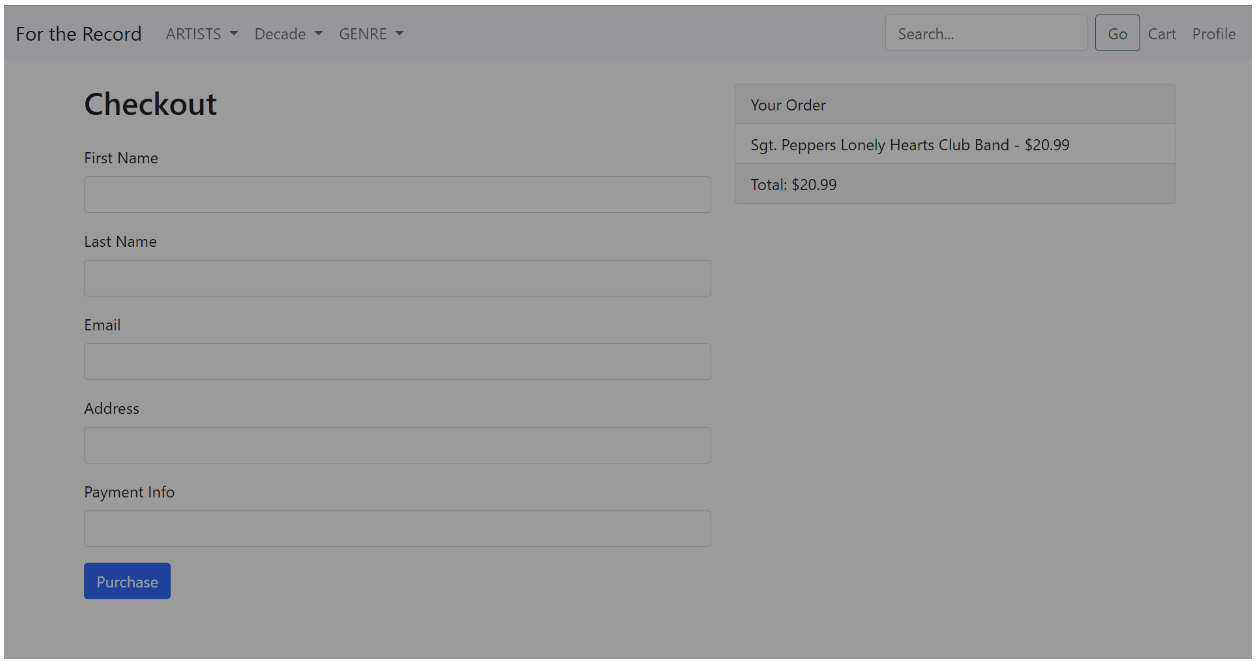
The Beatles

Price: $20.99

Quantity: 1

Remove

Checkout


For the Record ARTISTS Decade GENRE Search... Go Cart Profile

Checkout

First Name

Last Name

Email

Address

Payment Info

Purchase

Your Order

Sgt. Peppers Lonely Hearts Club Band - $20.99

Total: $20.99

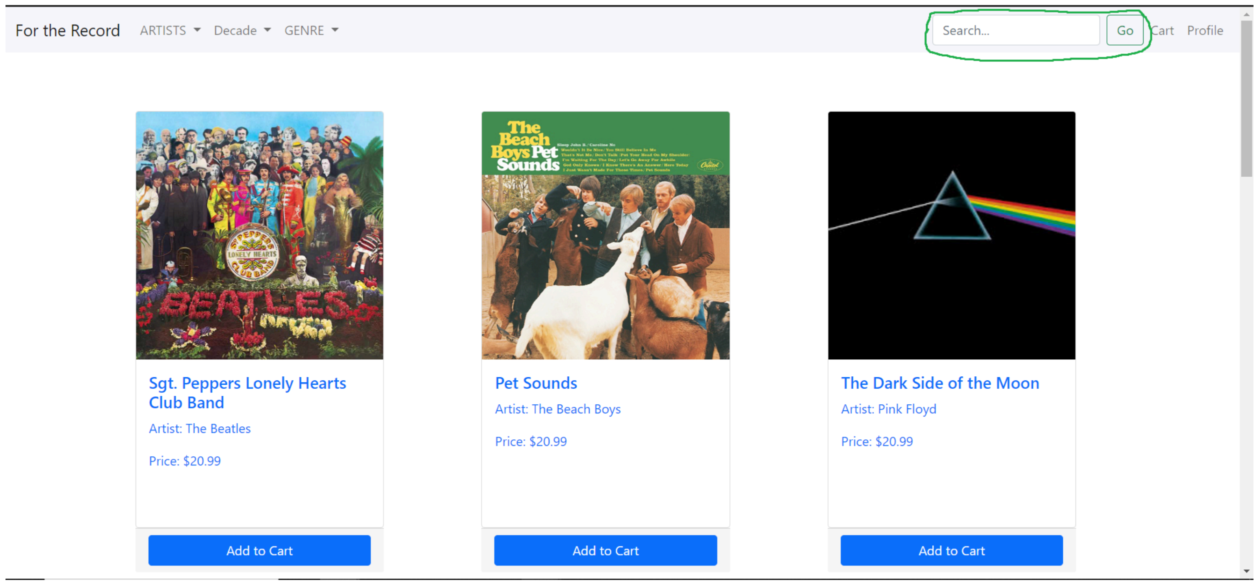
Name of use Case: Checkout items
Last Updated: 4/21/2024
Date Created: 3/7/2024
Description: User clicks on the “Checkout” button to purchase their items, then buys the items after filling out the necessary information. Note that if the cart is empty they can still move to checkout but there will be nothing to purchase
Actors: User
Preconditions:
Postconditions:
Flow:

1. Buyer is in the shopping cart page

2. Buyer clicks the “Checkout” button
3. Buyer inputs a first name, last name, email, address, and payment info
4. Website checks to ensure first and last name aren’t blank, that card info appears to be correct (may implement a banking API later), checks that email contains a ‘@’ and ‘.com’ (may implement a email API later), and lastly checks that address appears correct (may also implement an address API later)

Alternative Flows:
Exceptions:
1. In step 4 of the normal flow, if website identifies any incorrect information
2. Website flags the improper lines (i.e. if a email appears incorrect, that line is flagged)
3. Assuming the buyer fixes their inputs, the purchase succeeds. If a line is still incorrect, we repeat the process of this exception flow

Requirements:

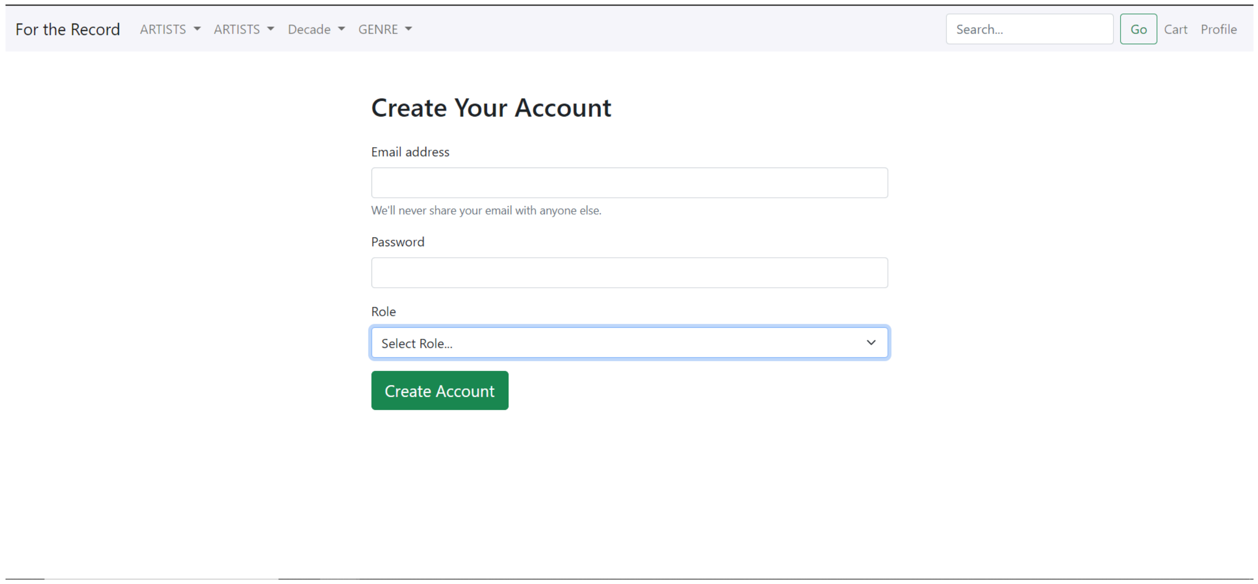


Name of use Case: Search
Last Updated: 4/21/2024
Date Created: 3/7/2024
Description: Buyer clicks on the search bar to look for specific items in the store
Actors: User
Preconditions:
Postconditions:
Flow:
1. User clicks on the search bar and inputs characters
2. User clicks the “Go” button

3. User is sent to a page only containing items that contain the characters inputted in the search bar

Alternative Flows:
1. User inputs nothing into the search bar and presses the “Go” button
2. User is sent to a page containing all items in the store

Exceptions:
Requirements:



Create Your Account

Email address

We'll never share your email with anyone else.

Password

Role

Select Role...

Create Account

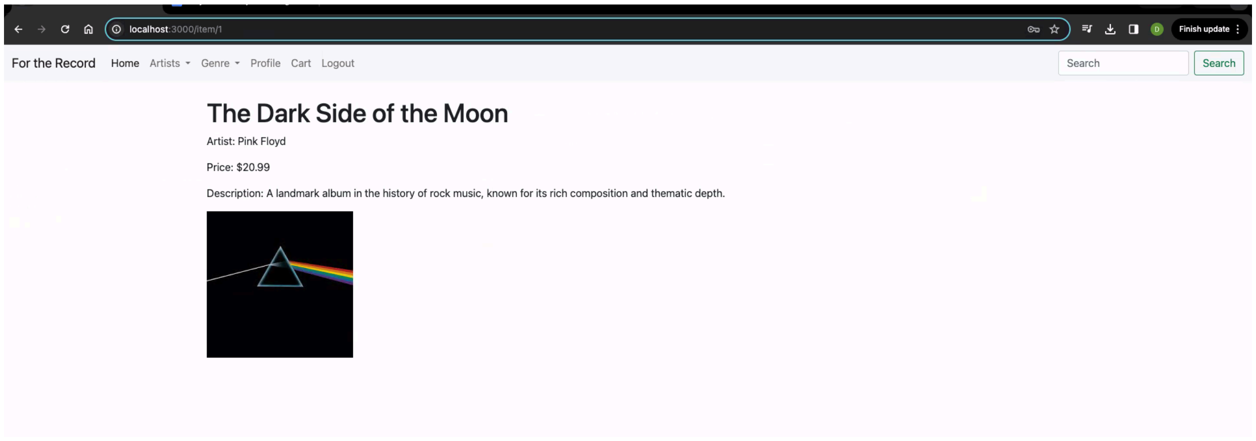
Name of use Case: Create Account
Last Updated: 4/21/2024
Date Created: 3/7/2024
Description: User accesses the create account page
Actors: User
Preconditions:
Postconditions:
1. Account will be remembered by the store

Flow:
1. User is on one of the various pages within the store
2. User clicks on the profile page
3. User is not currently logged in and thus will be prompted to create an account

Alternative Flows:
1. User is about to check out while logged out
2. User is prompted whether they would like to create an account

3. User opts to create an account

Exceptions:
Requirements:
1. User must be logged out (may add functionality for creating an account while logged in later if we deem it to be productive)



The Dark Side of the Moon

Artist: Pink Floyd

Price: $20.99

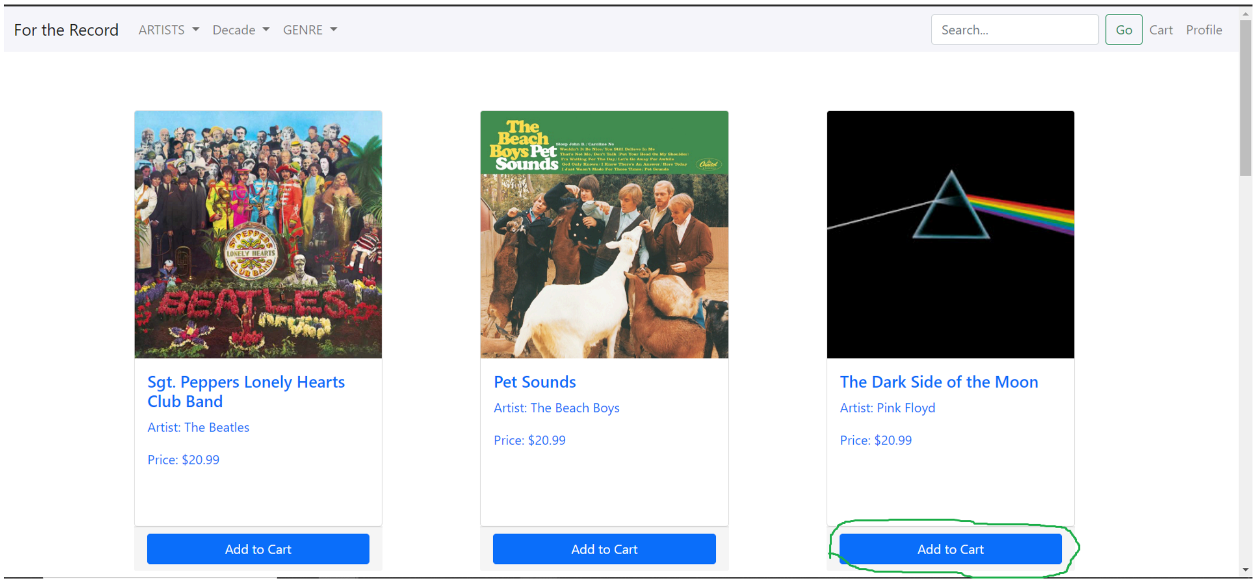
Description: A landmark album in the history of rock music, known for its rich composition and thematic depth.

Name of use Case: View Product Listing
Last Updated: 4/21/2024
Date Created: 3/7/2024
Description: Buyer clicks on some portion of a product listing
Actors: User
Preconditions:
1. The store must have at least one product displayed

Postconditions:
Flow:
1. Buyer identifies a product they want to inquire more about
2. Buyer clicks on some portion of the product while on the shopping page

Alternative Flows:
1. Buyer is viewing their shopping cart
2. Buyer clicks on one of the products in their shopping cart

Exceptions:
Requirements:
1. At least one product must be displayed



Name of use Case: Add item to cart
Last Updated: 4/21/2024
Date Created: 3/6/2024
Description: Buyer clicks the “Add to Cart” button on an item on the shopping screen
Actors: User
Preconditions:

1. Item is in stock
2. Buyer is logged in

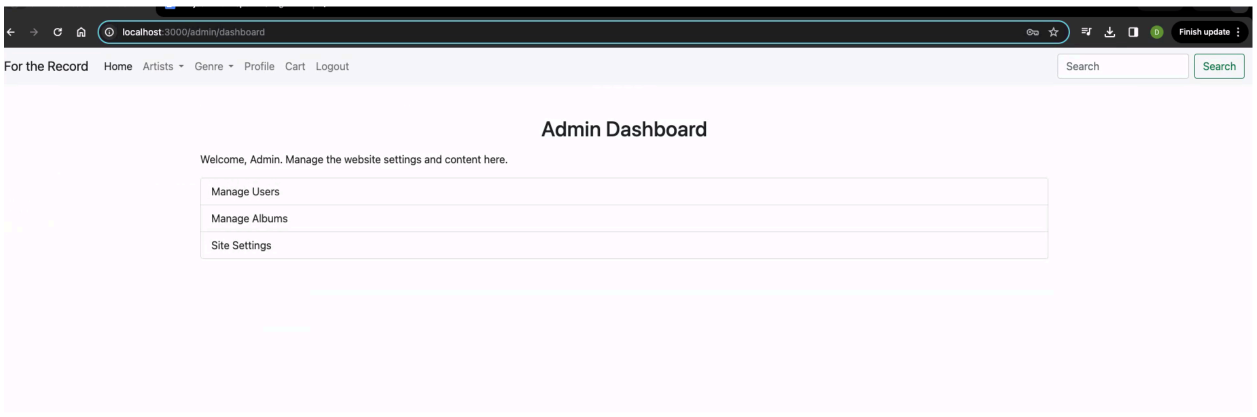
Postconditions:

Flow:

1. Buyer is on a screen that allows them to purchase items (most likely the main page or an individual items description page)
2. Buyer identifies product they are interested in
3. Buyer adds item to cart by clicking the correct button

Alternative Flows:
Exceptions:
Requirements:

1. Item must be in stock



Admin Dashboard

Welcome, Admin. Manage the website settings and content here.

Manage Users

Manage Albums

Site Settings

Name of use Case: Access Admin Page
Last Updated: 4/21/2024
Date Created: 4/21/2024
Description: Admin account holder accesses admin page
Actors: User, Admin
Preconditions:

1. User is logged in as an admin

Postconditions:
Flow:

1. User logs in as admin
2. User is taken to the admin screen

Alternative Flows:

1. User is already logged in as admin
2. User accesses the admin page via the page button

Exceptions:
Requirements:

1. User must be logged in as an admin

II. Actors and User Roles

User - Everyone is initially a user, this may be someone with an account currently logged out, or they're logged in as a customer. In our use cases it will also include admins (admins can do everything a user can do)

Admin - This role was intended to be able to remove products. This role has the highest access to the website (able to edit features, remove products, etc relatively easily).