ShopTune

Final Project Technical Report

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

Team Members:
Member 1-tmagikar@iastate.edu
Member 2-shubamc@iastate.edu

May 11, 2025

1. Introduction

Overview: ShopTune is an e-commerce web application designed to simulate a complete shopping experience with frontend-backend integration and a connected MySQL database. The motivation behind this project was to understand full-stack development, including user authentication, data storage, and CRUD operations.

Users: Customers looking to browse, add to cart, and purchase fashion items categorized as Women, Men, and Accessories.

Goals:

- Browse and categorize products
- Manage cart and checkout process
- Signup and log in securely
- Connect frontend to backend and database using REST APIs

Originality: Inspired by commercial e-commerce platforms, but completely original in implementation.

2. Project Description

Major Features:

- Category-based product browsing (Women, Men, Accessories)
- Cart and checkout system
- User login and signup
- Music player integration

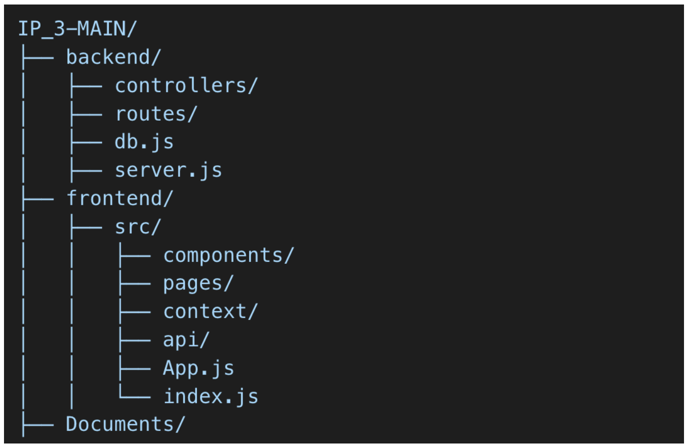
User Flow:

- User lands on HomePage with category selection
- Navigates to specific category page to view products
- Adds items to cart
- Proceeds to checkout or login/signup if not authenticated

CRUD Operations:

- **Create**: New user signup, cart creation
- **Read**: Product list from database
- **Update**: Cart updates and item removal
- **Delete**: Remove items from cart

3. File and Folder Architecture

- 

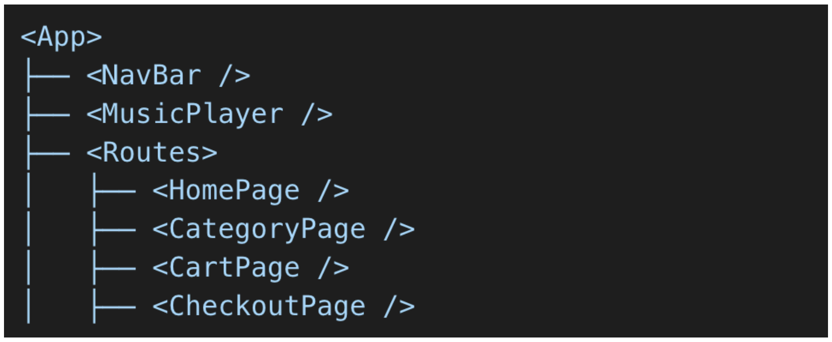
```
IP_3—MAIN/
├── backend/
│   ├── controllers/
│   ├── routes/
│   ├── db.js
│   ├── server.js
├── frontend/
│   ├── src/
│   │   ├── components/
│   │   ├── pages/
│   │   ├── context/
│   │   ├── api/
│   │   ├── App.js
│   │   └── index.js
├── Documents/
```

4. Code Explanation and Logic Flow

4.1. Frontend–Backend Communication

- fetch(/api/products) on frontend calls Express backend
- Express routes use MySQL to query product data
- JSON results sent to frontend and rendered via React

4.2. React Component Structure



```
<App>
├── <NavBar />
├── <MusicPlayer />
├── <Routes>
│   ├── <HomePage />
│   ├── <CategoryPage />
│   ├── <CartPage />
│   ├── <CheckoutPage />
```

CartContext used for global cart state

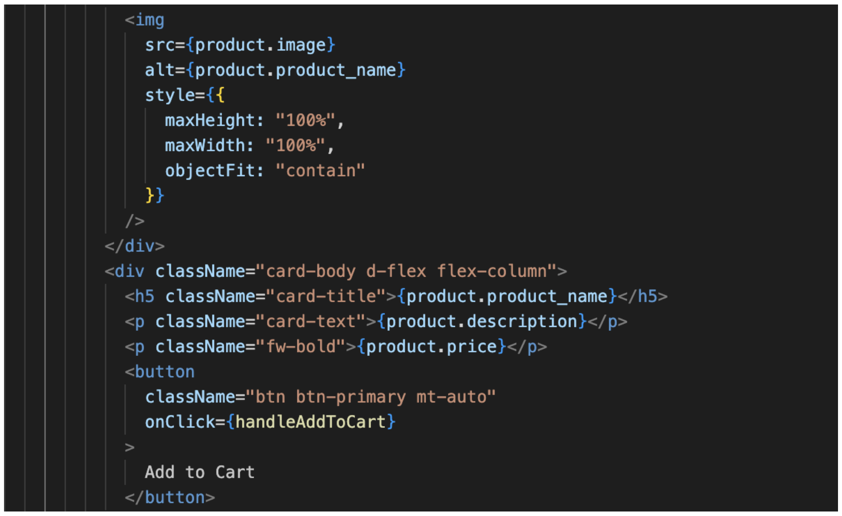
Props passed to ProductCard to display product info

4.3. Database Interaction

MySQL database with tables: users, products

Node.js uses mysql2 to execute queries in server.js

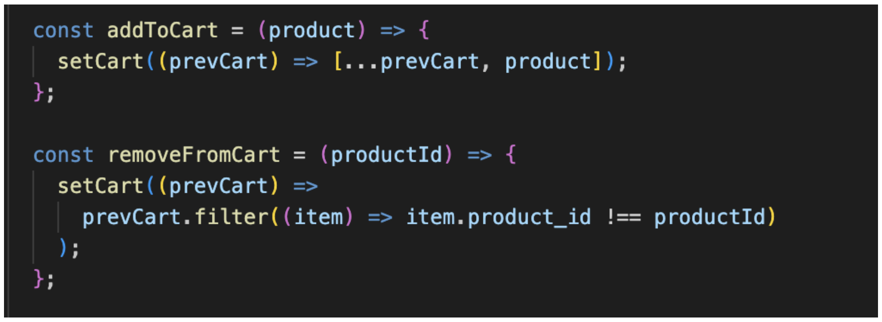
4.4. Code Snippets



```
          <img
            src={product.image}
            alt={product.product_name}
            style={{
              maxHeight: "100%",
              maxWidth: "100%",
              objectFit: "contain"
            }}
          />
        </div>
        <div className="card-body d-flex flex-column">
          <h5 className="card-title">{product.product_name}</h5>
          <p className="card-text">{product.description}</p>
          <p className="fw-bold">{product.price}</p>
          <button
            className="btn btn-primary mt-auto"
            onClick={handleAddToCart}
          >
            Add to Cart
          </button>
```

Purpose: Ensures images are consistently contained within the card box.

The button calls the global addToCart() method from CartContext.



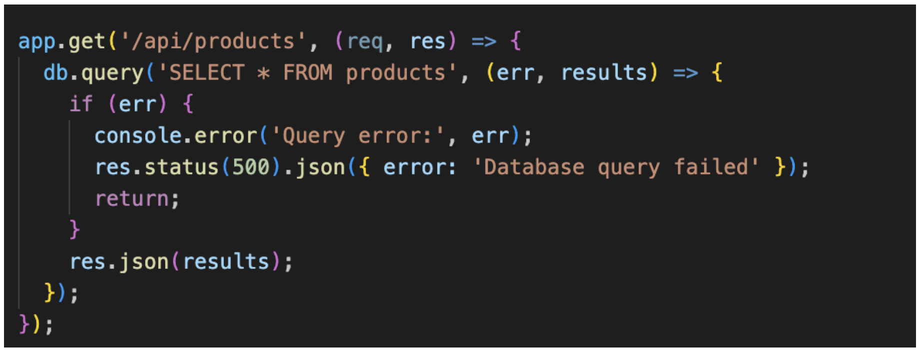
```
const addToCart = (product) => {
  setCart((prevCart) => [...prevCart, product]);
};

const removeFromCart = (productId) => {
  setCart((prevCart) =>
    prevCart.filter((item) => item.product_id !== productId)
  );
};
```

Purpose: addToCart adds selected products to cart state.

removeFromCart filters out a product by its ID.

These are provided via Context to any component that needs them

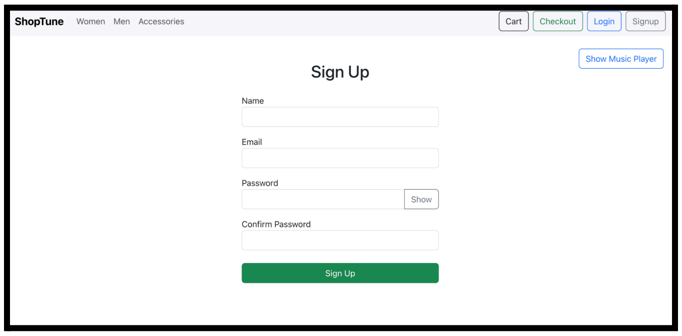


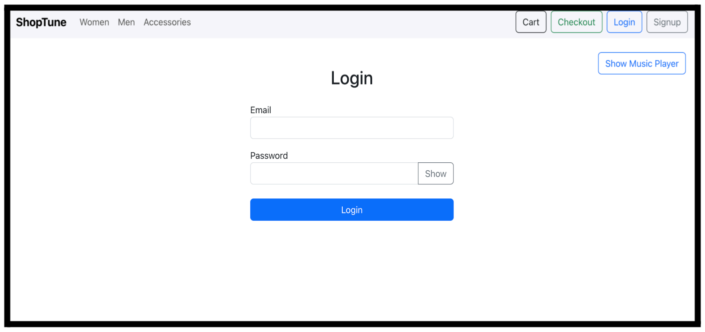
```
app.get('/api/products', (req, res) => {
  db.query('SELECT * FROM products', (err, results) => {
    if (err) {
      console.error('Query error:', err);
      res.status(500).json({ error: 'Database query failed' });
      return;
    }
    res.json(results);
  });
});
```

Purpose: This route connects our React frontend to the MySQL products table.

If successful, returns a JSON array of all available products.

5. Web View Screenshots and Annotations





Description:

LoginPage:

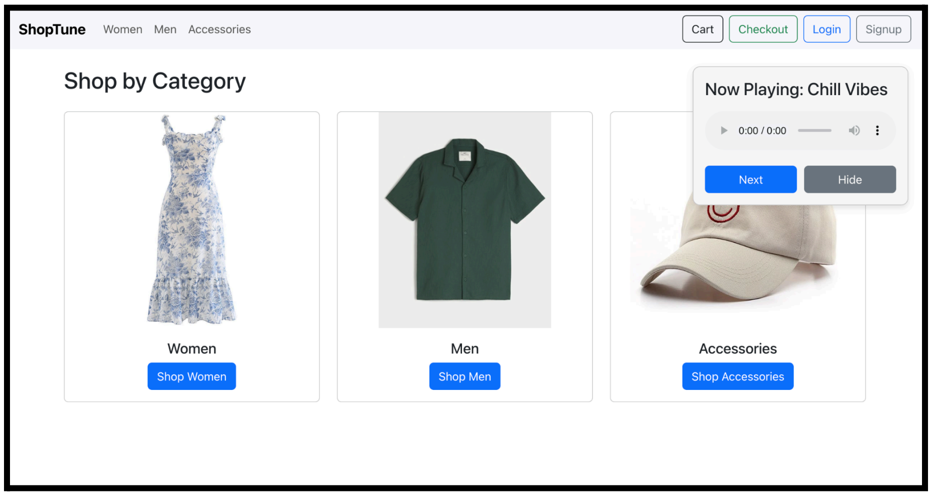
- Presents a form with email and password fields.
- Includes basic validation and displays feedback for incorrect login (e.g., “Invalid credentials”).
- On successful login, redirects user to the homepage or a protected route.

SignupPage:

- Allows new users to register by providing a username, email, and password.
- Checks for existing users and confirms password validity.

Backend-Ready:

- The UI is structured to easily connect to backend APIs like /api/login and /api/signup.



Description: This is the landing page that introduces users to the three major shopping categories:

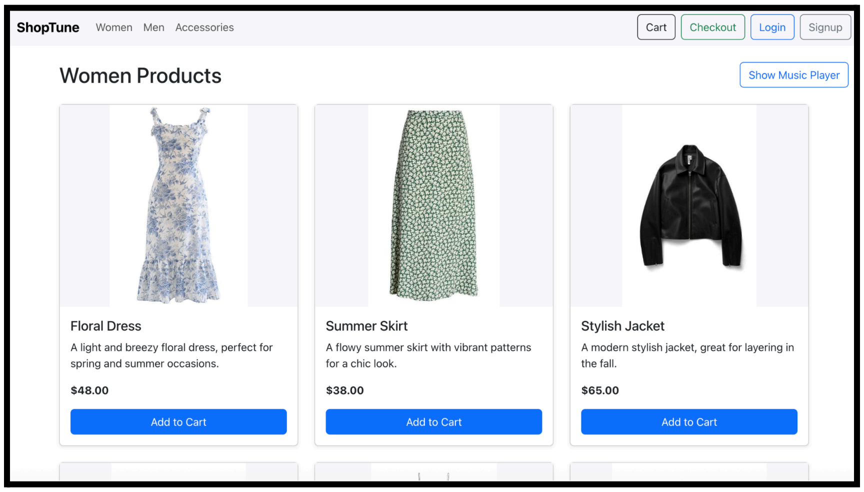
Women, Men, and Accessories.

Design:

- Each category is represented as a card with a representative image and a title.
- Cards are neatly laid out using Bootstrap's responsive grid system for consistent alignment across devices.

Functionality:

- Each card includes a "Shop [Category]" button that navigates users to the respective /women, /men, or /accessories page..
- This creates a smooth, single-page application transition experience.



Women Products

Floral Dress

A light and breezy floral dress, perfect for spring and summer occasions.

$48.00

Summer Skirt

A flowy summer skirt with vibrant patterns for a chic look.

$38.00

Stylish Jacket

A modern stylish jacket, great for layering in the fall.

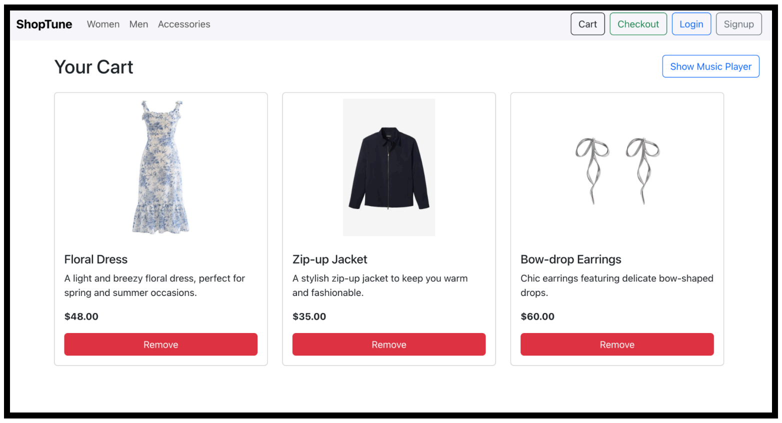
$65.00

Description: To display all products under a selected category in a clean, card-based layout.

Data Source: Products are dynamically filtered using the category route parameter (from useParams()) and fetched from backend.

Each Product Card Includes:

- A high-quality image, properly contained using CSS (object-fit: contain) within a fixed height area.
- Product name as the title.
- Short description for context.
- Price displayed in bold.
- A "Add to Cart" button that calls addToCart() from the global Cart Context.
- Shows a confirmation dialog on successful add.

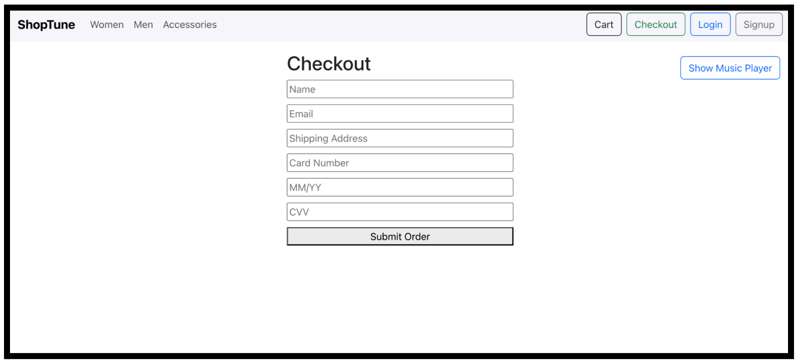


Your Cart

Floral Dress
A light and breezy floral dress, perfect for spring and summer occasions.
$48.00

Zip-up Jacket
A stylish zip-up jacket to keep you warm and fashionable.
$35.00

Bow-drop Earrings
Chic earrings featuring delicate bow-shaped drops.
$60.00



Checkout

Name
Email
Shipping Address
Card Number
MM/YY
CVV

Description:

CartPage:

- Lists all added products in card format.
- Each entry shows product name, image, price, and a Remove button (calls removeFromCart()).
- At the bottom, displays total cost calculated dynamically.
- Empty cart condition is also handled with a message.

CheckoutPage:

- Acts as a summary screen before payment.
- Confirms all items in the cart and could simulate order submission
- Clears cart on checkout.

6. Installation and Setup Instructions

Backend:

cd backend

npm install

node server.js

Frontend:

cd frontend

npm install

npm start

Database:

CREATE DATABASE shopTuneDB;

USE shopTuneDB;

-- Run table creation scripts/ whatever necessary

7. Contribution Overview

Feature	Team Member
Frontend UI	Tanisha
Cart/Checkout Logic	Tanisha
Backend API & DB	Shubam
User Auth System	Shubam

8. Challenges Faced

1. CORS issues when connecting frontend to backend - fixed using app.use(cors())
2. MySQL password policy errors - fixed by adjusting validation rules
3. JSON fetch error due to missing route closure - fixed syntax in server.js

9. Final Reflections

This project gave us hands-on experience with full-stack development. We learned how to connect React with a Node.js/Express backend, and how to structure clean, modular code. We would improve user auth and add payment integration next