GAME TRIBE

Final Project Technical Report

SE/COM S 3190 – Construction of User Interfaces

Spring 2025

Team Members:
Member 1 - Fuchinanya Akpuokwe (fuchicay@iastate.edu)
Member 2 - Iteoluwakishi Osomo (iaosomo@iastate.edu)

May 11, 2025

1. Introduction

Overview: Game Tribe is a full-stack web application that serves as an online game store and community platform. The project addresses the need for a unified platform where users can browse, purchase, and manage their favorite games while connecting with other gamers.

Motivation: Many gaming platforms focus solely on sales or community features. GameTribe combines both, offering a comprehensive solution for gamers who want to purchase games and build their gaming library.

Users: Our target users include:

- Casual and hardcore gamers looking to purchase games
- Administrators who manage the platform's content and users

Goals:

- Create an intuitive e-commerce platform for game purchases
- Develop an administrative dashboard for platform management
- Implement secure user authentication and authorization

Project Type: This is an original project inspired by platforms like Steam, but with our unique implementation focusing on cool features and administrative control

2. Project Description

Major Features:

1. **User Authentication System**
    - Registration with profile customization
    - Secure login/logout functionality
    - Profile management with avatar selection
2. **Game Catalog and Search**
    - Browse games by genre
    - Advanced search and filtering
    - Real-time price and genre filtering
3. **Shopping Cart and Checkout**

    - Add/remove items from the cart
    - Update quantities
    - Complete checkout with shipping information
4. **Community Features**
    - Favorite game system
    - Recently viewed tracking
5. **Admin Dashboard**
    - User management
    - Order management
    - Revenue and statistics tracking

User Flow:

1. Guest users can browse games and view details
2. Registration/login required for purchases and the favorite feature
3. Authenticated users can add items to the cart, checkout, and favorite games
4. Admin users access a separate dashboard for platform management

CRUD Operations:

- **Users**: Create (register), Read (profile), Update (edit profile), Delete (admin)
- **Games**: Read (all users)
- **Orders**: Create (checkout), Read (user/admin), Update (status - admin)

3. File and Folder Architecture

The Game Tribe project follows a clean client-server architecture with separate frontend and backend directories:

Frontend Directory:

- public/ contains static assets, including game images and user avatars
- src/ houses the React application with organized subdirectories:
    - components/ for reusable UI elements like sidebars
    - context/ for React Context providers managing authentication and cart state
    - pages/ containing all route components, with a dedicated Admin/ subfolder
    - services/ for API communication logic
    - styles/ with modular CSS files for each component

Backend Directory:

- models/ defines MongoDB schemas for User, Game, and Order entities
- routes/ contains Express route handlers organized by feature (auth, games, orders, users, admin)
- middleware/ includes authentication and admin authorization logic
- scripts/ provides utility scripts like admin user creation
- server.js serves as the main entry point, configuring Express and MongoDB

Documents Directory: Contains sample data and project documentation.

This structure divides responsibilities clearly, making the code easier to maintain and grow. The frontend uses a component-based style, while the backend uses a RESTful API approach with well-defined routes and middleware.

MN_5/

├── Frontend/gametribe/

│ ├── public/

│ │ └── assets/

│ ├── src/

│ │ ├── components/

│ │ ├── context/

│ │ ├── pages/

│ │ ├── services/

│ │ └── styles/

│ └── package.json

├── Backend/

│ ├── models/

│ ├── routes/

```
│   ├── middleware/
│   ├── scripts/
│   ├── server.js
│   └── package.json
└── Documents/
    └── data.json
```

4. Code Explanation and Logic Flow

4.1. Frontend–Backend Communication

The application uses a centralized API service (api.js) with Axios interceptors for handling authentication tokens and error responses.

```
const api = axios.create({
  baseURL: API_URL,
  headers: {
    'Content-Type': 'application/json'
  }
});


// Intercept requests to add auth token
api.interceptors.request.use(
  config => {
    const token = localStorage.getItem('token');
    if (token) {
      config.headers['x-auth-token'] = token;
    }
    return config;
  },
```

```
  error => {
    return Promise.reject(error);
  }
);
```

4.2. React Component Structure

The application uses React Context for state management:

- **AuthContext**: Manages user authentication state
- **CartContext**: Handles shopping cart operations

Component hierarchy:

App.jsx
├── AuthProvider
│ └── CartProvider
│ ├── Routes
│ │ ├── Public Pages (Home, Search, Login)
│ │ ├── Protected Pages (Profile, Cart)
│ │ └── Admin Pages (Dashboard, Users, Orders, Games)
│ └── Sidebar (persistent navigation)

All routes are handled in the AppRoutes.jsx file.

4.3. Database Interaction

MongoDB with Mongoose ODM:

- **User Schema**: Authentication, profile, favorites
- **Game Schema**: Product info
- **Order Schema**: User reference, items, shipping info

4.4. Code Snippets

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

React Component - Favorites Toggle:

```
const handleToggleFavorite = async (game, e) => {
    e.stopPropagation(); // Prevent card click

    if (!isAuthenticated) {
      navigate('/login');
      return;
    }

    // Use the numeric id from the game object, not _id
    const gameId = game.id || game._id;

    try {
      if (favoriteGames.has(gameId)) {
        await userService.removeFromFavorites(gameId);
        const newSet = new Set(favoriteGames);
        newSet.delete(gameId);
        setFavoriteGames(newSet);
      } else {
        await userService.addToFavorites(gameId);
        const newSet = new Set(favoriteGames);
        newSet.add(gameId);
        setFavoriteGames(newSet);
      }
      // Refresh favorites to sync with server
```

```
      await fetchUserFavorites();
    } catch (error) {
      console.error('Error toggling favorite:', error);
      // Show user-friendly error
      if (error.response && error.response.status === 400) {
        await fetchUserFavorites();
      }
    }
  };
```

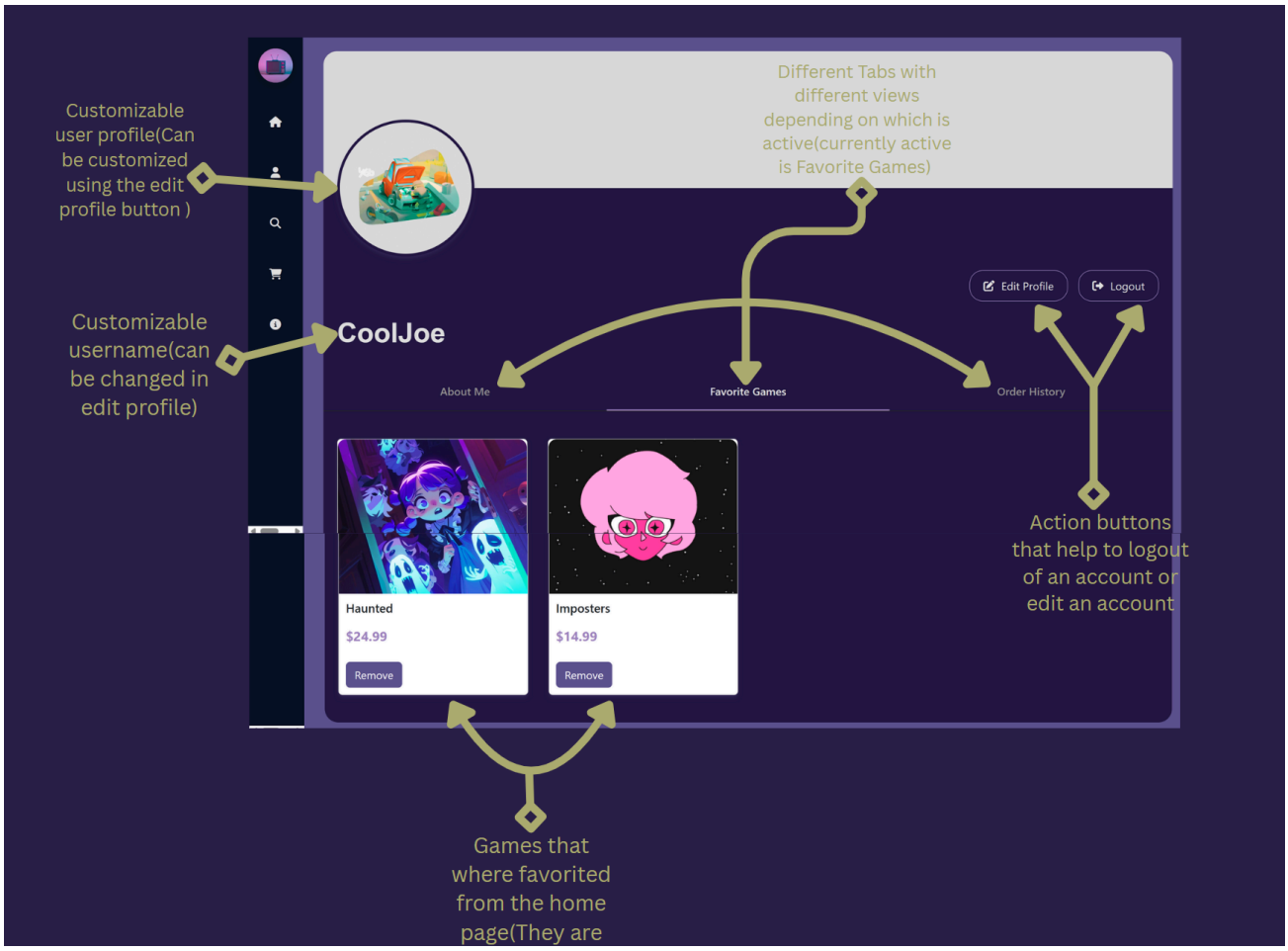
5. Web View Screenshots and Annotations

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

Feature 1: Advanced Search Page with Filtering

Description: The search page provides a comprehensive game discovery experience with multiple filtering options. Users can search games by name, filter by genre tags, adjust price ranges using dual sliders, and sort results by name or price. Each game card displays ratings, price, and genres. The responsive grid layout adjusts to screen size, and the page shows result counts with a reset filters option for user convenience.

Feature 2: User Profile with Favorites Tab Active

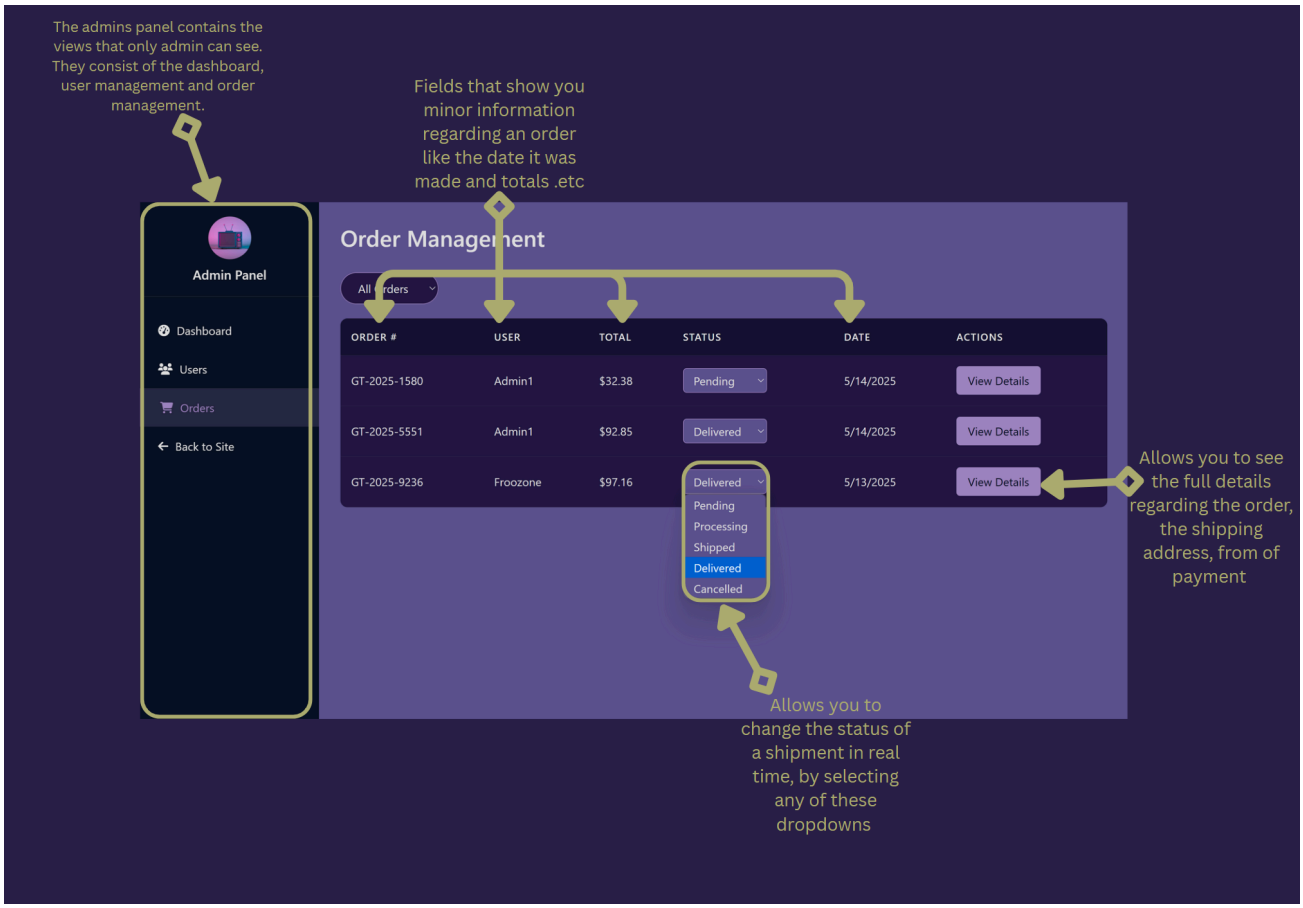


Description: The user profile page serves as a personal dashboard where users can manage their account information and view their gaming activity. The page features a customizable profile header with avatar selection and personal information display. The tabbed interface includes "About Me," "Favorite Games," and "Order History" sections. In the Favorites tab shown, users can view their saved games in a grid layout, with each game displaying its cover image, title, and price. Quick action buttons allow users to view game details or remove items from favorites.

Feature 3: Shopping Cart with Order Summary

Description: The shopping cart page provides a detailed view of selected items with full order management capabilities. The table-based layout shows game thumbnails, names, individual prices, quantity selectors, and line totals. Users can adjust quantities with +/- buttons or direct input, and remove items entirely. The right sidebar displays the order summary with subtotal, tax calculation (8%), and total. Below the cart, a "Recently Viewed" section encourages additional purchases. The checkout button opens a modal for shipping and payment information, but requires user authentication.

Feature 4: Admin Order Management



Description: The admin order management interface provides comprehensive control over all platform orders. The page features a dropdown filter to sort orders by status (All Orders, Pending, Processing, Shipped, Delivered, Cancelled), with a table displaying order numbers, associated users, total amounts, current status, creation dates, and action buttons. Each order status can be updated directly from the table using a styled dropdown selector that changes color based on the selected status. The "View Details" button opens a modal showing complete order information, including items purchased, quantities, shipping address, and payment details.

6. Installation and Setup Instructions

Backend Setup:

cd Backend

npm install

Create .env file:

MONGODB_URI=mongodb://localhost:27017/gameTribeDB

JWT_SECRET=your_secret_key

PORT=5000

Run the backend:

npm start

Frontend Setup:

cd Frontend/gametribe

npm install

Update proxy in package.json:

json"proxy": "http://localhost:5000"

Run the frontend:

npm start

Database Setup:

1. Ensure MongoDB is running
2. Create admin user:

   cd Backend

   node seed/gameData.js

   node scripts/createAdmin.js

Default Admin Credentials:

Email: admin@gametribe.com

Password: admin123

7. Contribution Overview

Feature	**Responsible Member**
Authentication System	Fuchinanya Akpuokwe
Shopping Cart & Checkout	Iteoluwakishi Osomo
User Profile Management	Fuchinanya Akpuokwe
Admin Dashboard	Iteoluwakishi Osomo
Game Management (CRUD)	Iteoluwakishi Osomo
Community Features (Favorites)	Fuchinanya Akpuokwe
Search and Filtering	Fuchinanya Akpuokwe
Order Management	Iteoluwakishi Osomo
Database Schema Design	Both
API Integration	Both

.

8. Challenges Faced

Authentication Token Management

- **Issue:** The API interceptor is causing redirect loops when it receives a 401 error.
- **Solution:** We added checks to the redirects to see which route we are currently on.

MongoDB Population with Virtual Fields

- **Issue:** User orders were not populating correctly in the admin panel
- **Solution:** Added virtual properties and proper population configuration

9. Final Reflections

Learning Outcomes:

- Mastered full-stack development with the MERN stack
- Implemented secure authentication and authorization
- Developed complex state management patterns
- Created responsive, user-friendly interfaces
- Learned MongoDB aggregation for analytics

Improvements for the Future:

1. Implement real-time notifications using WebSockets
2. Add payment gateway integration
3. Implement image upload functionality for game covers
4. Add social features like friend lists and game sharing, and reviews

This project showed the challenges of creating a complete e-commerce platform while keeping the code organized and ensuring a good user experience. The admin features support platform management, and the community features boost user engagement.