Capstone Project - Winter 2025

Project Overview
For our Capstone Project, our team has developed BizHorizon – an Operations Management Platform for Small Businesses.

BizHorizon is a fully responsive operations management platform designed for small businesses. It provides a modernized UI, secure authentication, and seamless payment integration via Stripe. With an interactive admin dashboard, real-time data visualization, and automated order processing, BizHorizon helps businesses enhances online presence and optimizes business operations.

Key Features:
- User Authentication (Login, Registration, Profile)
- Product Catalog & Customer Management
- Payment Integration (Stripe)
- Admin Dashboard
- Frontend Implementation (basic frontend implemented)
- RESTful API Development with Express.js and MySQL
- Role-Based Access Control (RBAC)
- Postman API Testing
- Orders Management & Analytics APIs (pending implementation)
- Modern & Responsive UI (Soft Minimalist Palette applied)
- Dashboard with Interactive UI Elements

This README provides an overview of the completed components, project structure, and setup guide.

Project Structure
```
CapstoneProject_T101/
|── config/
|   ├── database.js          # Database connection setup
|   ├── keys.env             # Environment variables (DO NOT COMMIT)
|
|── controllers/
|   ├── catalogController.js  # Handles catalog-related requests
|   ├── customerController.js # Handles customer-related logic
|   ├── paymentController.js  # Handles payment transactions (Stripe)
|   ├── userController.js     # Manages user authentication & profile updates
|   |__ feedbackController.js # Manage feedback-related logic
|
|── middlewares/
|   ├── authMiddleware.js     # Authentication middleware for JWT verification
|
|── models/
|   ├── catalogModel.js      # Catalog schema and queries
|   ├── customerModel.js     # Customer data model
|   ├── paymentModel.js      # Payment transactions schema
|   ├── userModel.js         # User authentication schema
|   |__ feedbackModel.js     # feedback data model
|
|── node_modules/            # Dependencies (Ignored in Git)
|
|── public/
|   ├── css/
|   |   ├── style.css        # Frontend styling
|   ├── js/
```

```
│   │   ├── payment.js       # Payment handling scripts
│   │   ├── script.js        # Main frontend script
│   │   └── adminFeedback.js  # feedback handling script
│   ├── index.html           # Frontend UI
│
├── routes/
│   ├── catalogRoutes.js     # Catalog API routes
│   ├── customerRoutes.js    # Customer-related API routes
│   ├── payment.js           # Payment-related API routes
│   ├── userRoutes.js        # User authentication API routes
│    └── adminRoutes.js       # Feedback API routes
│
├── views/                   # Handlebars/EJS templates (if applicable)
│
├── .env                     # Environment variables (DO NOT COMMIT)
├── .gitignore               # Files/Folders ignored in Git
├── customers.json           # Sample data for customers
├── package-lock.json        # Dependency lock file
├── package.json             # Node.js project dependencies
├── README.md                # Project documentation
├── server.js                # Main server file (Entry Point)

```

Technologies Used
Backend:
- Node.js – JavaScript runtime for server-side execution.
- Express.js – Web framework for building RESTful APIs.
- MySQL – Relational database for storing users, products, payments, and customer data.
- Sequelize ORM – Object-Relational Mapping (ORM) tool for interacting with MySQL.
- JWT (JSON Web Token) – Secure authentication & authorization for API endpoints.
- Bcrypt.js – Password hashing for secure authentication.
- Dotenv – Manages environment variables securely.
- Cors – Enables cross-origin requests for API communication.
- Express Validator – Input validation and sanitization for API endpoints.

Frontend:
- HTML, CSS, JavaScript – Core web technologies for building the user interface.
- Bootstrap – Responsive design framework for styling.
- AJAX (Fetch API / Axios) – Handles API requests asynchronously.
- Stripe.js – Secure online payment integration with Stripe.
- Font Awesome – Icons for UI enhancement.
- Google Fonts – Improved typography and design customization.
- Custom CSS – Modern and responsive UI with Soft Minimalist color palette.

Payment Integration:
- Stripe API – Secure online payment processing.
- Stripe.js & Elements – Client-side payment handling for improved UX.
- Test Cards – Supports Stripe’s test environment for simulating real payments.

API Development & Testing:
- Postman – API testing and debugging tool.
- Express Validator – Used for request validation and sanitization.
- Error Handling & Logging – Implemented structured error responses for debugging.

Version Control & Deployment:
- Git & GitHub – Version control and collaboration.

Completed Features
1. User Authentication & Management
- Secure User Registration, Login, and Profile Management.
- Password hashing using bcrypt.js for enhanced security.
- JWT-based Authentication for session management.
- Ability to view, update, and delete user profiles.
- Token-based authorization for protected routes.
- Improved error handling for authentication failures.

2. Product Catalog Management
- CRUD functionality for product management, including adding, updating, retrieving, and deleting products.
- Product data stored in MySQL with structured queries.
- Validation checks for product creation and updates.

3. Customer Management
- Store and manage customer records in a structured database.
- CRUD operations for customer management.
- Added search functionality for easier customer retrieval.

4. Payment Integration (Stripe)
- Stripe API Integration for secure online transactions.
- Implemented Stripe.js & Elements for a better UI/UX in payment handling.
- Test Card support for payment testing.
- Enhanced error handling for failed or declined transactions.

5. RESTful API Development
- Well-structured API routes, controllers, and models for all key functionalities.
- Implemented role-based access control (RBAC) for better security.
- Tested and Debugged APIs using Postman.
- Optimized database queries for MySQL.
- Improved error handling and validation mechanisms.

6. Admin Dashboard (Backend & Frontend APIs)
- Backend APIs implemented for managing Users, Products, Customers, and Payments
- Frontend UI built with improved styling and Soft Minimalist Palette for modern design.
- Dashboard navigation and sidebar implemented with dynamic sections.
- Button styles and UI elements improved for better user experience.

7. Code & Database Enhancements
- Fixed various issues in Models, Controllers, and Routes.
- Successfully created and validated MySQL tables for users, catalog, and customers.
- Code refactored and cleaned for better maintainability.

Pending Features
- Orders Management – (Planned for future implementation).
- Product Management – (Planned for future implementation).
- Analytics & Reporting (Not implemented yet).
- File Uploads (Multer) – (Feature planned for future versions).
- Deployment & CI/CD Setup – (Final production-ready deployment improvements).

API Endpoints:
- http://localhost:4000/api/users
- http://localhost:4000/api/payment
- http://localhost:4000/api/catalog
- http://localhost:4000/admin
- http://localhost:4000/api/customers
- http://localhost:4000/api/feedbacks

User Authentication & Management
- Register User → POST /api/users/register.
- Login User → POST /api/users/login.
- Get User Profile (Protected) → GET /profile.
- Get All Users → GET /api/users.
- Update User Details → PUT /api/users/:id.
- Delete User → DELETE /api/users/:id.

Product Catalog Management
- Retrieve All Products → GET /api/catalog.
- Add a New Product → POST /api/catalog.
- Update Product Details → PUT /api/catalog/:id.
- Delete a Product → DELETE /api/catalog/:id.

Customer Management
- Retrieve All Customers → GET /api/customers.
- Add a New Customer → POST /api/customers
- Update Customer Details → PUT /api/customers/:id.
- Delete a Customer → DELETE /api/customers/:id.

Payment Processing
- Process Payment via Stripe → POST /api/payment.
- Handle Payment Errors (Automatic error handling for invalid transactions).

Admin Dashboard
- Admin Panel Route → GET /admin
- Backend routes for managing Users, Products, Customers, and Payments.

Defined Frontend Pages
- http://localhost:4000/admin (Dashboard)
- http://localhost:4000/admin/customers (Customers)
- http://localhost:4000/admin/payments (Payments)
- http://localhost:4000/admin/orders (Orders)
- http://localhost:4000/admin/analytics (Analytics)
- http://localhost:4000/admin/feedbacks (Feedbacks)
- http://localhost:4000/api/users/register (User Registration)
- http://localhost:4000/api/users/login (User Login)
- http://localhost:4000/profile (User Profile)
- http://localhost:4000/api/payment (Standalone Payment Page)

Installation & Setup
Prerequisites:
- Node.js installed on your machine.

- MySQL Database setup.
- Stripe Account (required for payments).
- GitHub for version control.

Step 1: Clone the Repository
```sh
git clone https://github.com/Nigar0826/CapstoneProject_Winter2025.git
cd CapstoneProject_T101
```

Step 2: Install Dependencies
```
npm install
```

Step 3: Set Up Environment Variables
1. Create a Stripe Account at [Stripe Dashboard](https://dashboard.stripe.com/register).
2. Get your Secret and Publishable keys from the Stripe Dashboard.
3. Create a `keys.env` file in the `config/` directory and add:
```sh
STRIPE_SECRET_KEY=sk_test_your_secret_key
STRIPE_PUBLISHABLE_KEY=pk_test_your_publishable_key
```
4. Replace `sk_test_your_secret_key` and `pk_test_your_publishable_key` with your actual Stripe API keys.
5. Add `config/keys.env` to `.gitignore` to prevent sensitive key exposure.

Important: Never expose the `STRIPE_SECRET_KEY` in the frontend or GitHub.

Step 4: Start the Server
```sh
node server.js or nodemon server.js
```
The server runs on: `http://localhost:4000`

How to Test the Payment System
1. Open the payment form in a browser: `http://localhost:4000/payment' or 'http://localhost:4000/admin`
2. Enter Stripe test card details:
   - Valid Card: `4242 4242 4242 4242`
   - Expired Card: `4000 0000 0000 0069`
   - Insufficient Funds: `4000 0000 0000 9995`
3. Submit the form and check results:
   - Success: Payment confirmation displayed.
   - Failure: Proper error messages shown.

Next Steps:
- Implement Orders, Products & Analytics APIs.
- Complete final UI design improvements.
- Deploy the project & finalize documentation.
- Finalize documentation.
- Prepare for the final presentation.

```
### Feedback Management    added by Fatima
- Retrieve All Feedbacks → GET /api/feedbacks.
- Add a New Feedback → POST /api/feedbacks.
- Update Feedback Details → PUT /api/feedbacks/:id.
- Delete a Feedback → DELETE /api/feedbacks/:id.

## Contributors
- Nigar - Project Lead, Payment Integration, API Testing, UI/UX Design, Stripe.js Integration,
Documentation, Deployment.
- Elizabeth - Back-End Development, Authentication, Database Optimization.
- Anar - Front-End Development, UI/UX Design, Stripe.js Integration.
- Fatima - Admin Dashboard, API Documentation, Error Handling.
```