TheOnlyGoodCameraStore

Final Project Technical Report

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

Team Members:
Asray Gopa - asray@iastate.edu
Ahmet Okur - emreokur@iastate.edu

May 11, 2025

1. Introduction

This project is an online camera store called **TheOnlyGoodCameraStore**. The goal was to build a full-stack e-commerce site where users can browse and purchase camera products, manage their accounts and orders, and where admins can manage products, track sales, and communicate with customers.

The site includes real-time functionality like order status management, email communication, and live analytics. This was a completely original project, although we drew inspiration from sites like B&H and BestBuy. Our main users are photographers, camera enthusiasts, and business admins.

2. Project Description

The project includes the following major features:

- **Landing Page**
- **User Authentication** (login/register/update/delete)
- **Product Listings** with details and cart functionality
- **Cart Management** (add, update quantity, remove)
- **Checkout System** that stores orders and reduces stock
- **Order Management for Customers**
- **Admin Dashboard** with CRUD operations
- **Order Management for Admins** with status toggles and search
- **Question Submission and Management**
- **Sales Analytics Page** using ChartJS
- **Shipping Estimate Tool** using FedEx API
- **Email Blast Page** using Resend API

CRUD Operations & Entities:

- `users`: create, update, delete, login
- `products`: create, read, update, delete
- `cart`: add, update, remove

- `orders`: create, update (status), delete
- `questions`: create, update (resolve), delete

3. File and Folder Architecture

Our project is organized into three main directories: `frontend/`, `backend/`, and `Documents/`.

`frontend/`

This folder contains all the React code that powers the client side of the application. We used Vite for fast development and hot reloading.

```
frontend/
├── components/           # Shared UI components (Navbar, etc.)
│   └── Navbar.jsx
├── pages/                # All major page components
│   ├── Home.jsx
│   ├── Login.jsx
│   ├── Register.jsx
│   ├── ProductList.jsx
│   ├── ProductDetail.jsx
│   ├── Cart.jsx
│   ├── Orders.jsx
│   ├── AdminDashboard.jsx
│   ├── SalesAnalyticsPage.jsx
│   ├── EmailSenderPage.jsx
│   └── ShippingEstimatePage.jsx
├── App.js                # Main router and layout
├── main.jsx              # ReactDOM render root
└── index.css             # Tailwind/global styles
```

`backend/`

This folder has the Node.js + Express API that handles all data logic. It includes routes for each feature, middleware for authentication, and MongoDB interaction.

```
backend/
├── routes/
│   ├── auth.js             # Login/register logic
│   ├── products.js         # Product CRUD
│   ├── cart.js             # Cart operations
│   ├── orders.js           # Customer and admin order routes
```

```
│   ├── questions.js          # User-submitted Q&A
│   ├── email.js              # Admin email blast with Resend API
│   └── ship.js               # Shipping estimates via FedEx API
├── server.js                # Entry point and route linking
├── db.js                    # MongoDB client/connection logic
└── .env                     # Environment variables (API keys, Mongo
URI)
```

This file organization helped us keep our frontend and backend clearly separated while also making it easy to collaborate. Each major feature had its own route/page, and we reused UI components wherever possible.

4. Code Explanation and Logic Flow

4.1. Frontend–Backend Communication

All frontend API requests are handled using `fetch()` with custom headers for authentication:

- The `UserId` and `UserRole` values are pulled from `localStorage` and attached to each request.
- Most endpoints expect JSON input and return JSON responses.
- Example API calls include:
    - `POST /api/orders` – to place an order
    - `GET /api/products` – to list all available products
    - `PATCH /api/orders/:id/status` – to update order flags (admin only)

4.2. React Component Structure

We built the frontend using React and Vite, structured around individual pages with shared components like `Navbar`.

Component hierarchy (simplified):

```
App.jsx
├── Navbar
├── Routes
│   ├── /products → <ProductList />
```

```
|   ├── /cart → <Cart />
|   ├── /admin → <AdminDashboard />
|   ├── /orders → <OrderManagementPage />
|   ├── /analytics → <SalesAnalyticsPage />
|   └── /ship → <ShippingEstimatePage />
```

Props are used to pass data to components like product details.
useState and **useEffect** manage loading states, data, and user interactions (like toggling checkboxes or submitting forms).

4.3. Database Interaction

We use **MongoDB** for all data storage. Collections include:

- `products`: name, description, price, quantity, image
- `orders`: items, status flags, timestamps, user info
- `users`: email, password (hashed), role (admin or user)
- `questions`: user-submitted inquiries for admin review

The backend uses `mongodb`'s native driver and connects to the database via `MongoClient`.

Each API route either:

- queries the DB (`.find`, `.findOne`)
- inserts new entries (`.insertOne`)
- updates fields (`.updateOne`)
- deletes items (`.deleteOne`)

4.4. Code Snippets

1. React: Add/Edit Product Form (AdminDashboard.jsx)

```
<form onSubmit={formMode === 'add' ? handleAddProduct :
handleEditProduct}>
```

```
  <input
    name="name"
    value={currentProduct.name}
    onChange={handleInputChange}
  />
  <input
    name="price"
    type="number"
    value={currentProduct.price}
    onChange={handleInputChange}
  />
  <button type="submit">
    {formMode === 'add' ? 'Add' : 'Update'} Product
  </button>
</form>
```

2. Backend: Create Order and Reduce Inventory (orders.js)

```
const result = await db.collection('orders').insertOne(order);
await Promise.all(items.map(it =>
  db.collection('products').updateOne(
    { _id: new ObjectId(it.productId) },
    { $inc: { quantity: -it.quantity } }
  )
));
```

3. Analytics Logic (SalesAnalyticsPage.jsx)

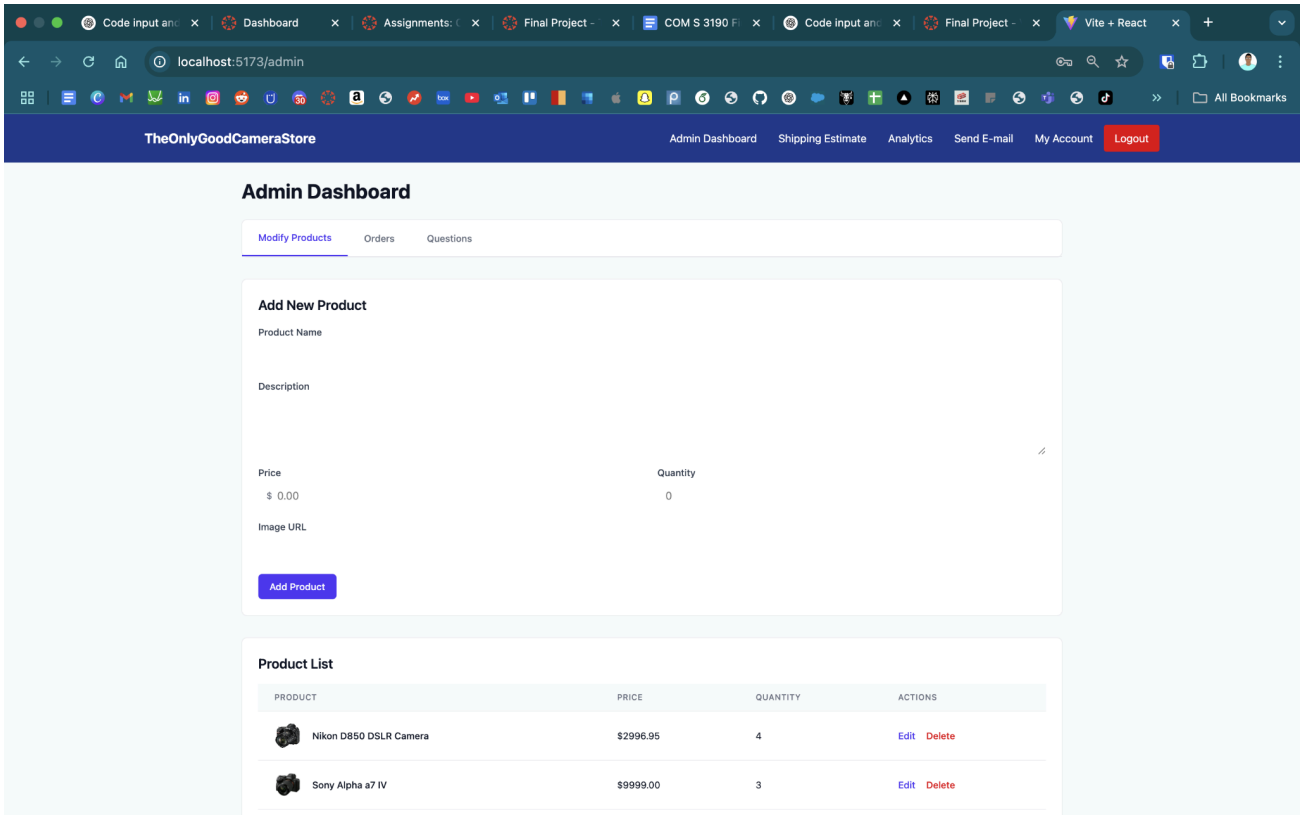
```
const revenueByDate = {};
orders.forEach(o => {
```

```
  const date = new Date(o.createdAt).toISOString().slice(0,10);
  revenueByDate[date] = (revenueByDate[date] || 0) + o.total;
});
```

This section shows how the different parts of our codebase talk to each other: the React frontend sends API calls to our Express backend, which then queries MongoDB and sends back the results for dynamic rendering.

5. Web View Screenshots and Annotations

Ahmet Page 1: Admin Dashboard Modify Products



Description:

This is the main admin panel for managing store inventory. Admin users are automatically redirected here upon login if their role is `"admin"`.

What the user can do:

- **Add new products** by filling in name, description, price, quantity, and an image URL.

- **Edit existing products** using the "Edit" button next to each item. The form pre-fills with the product data for editing.

- **Delete products** by clicking "Delete" next to any listed product.

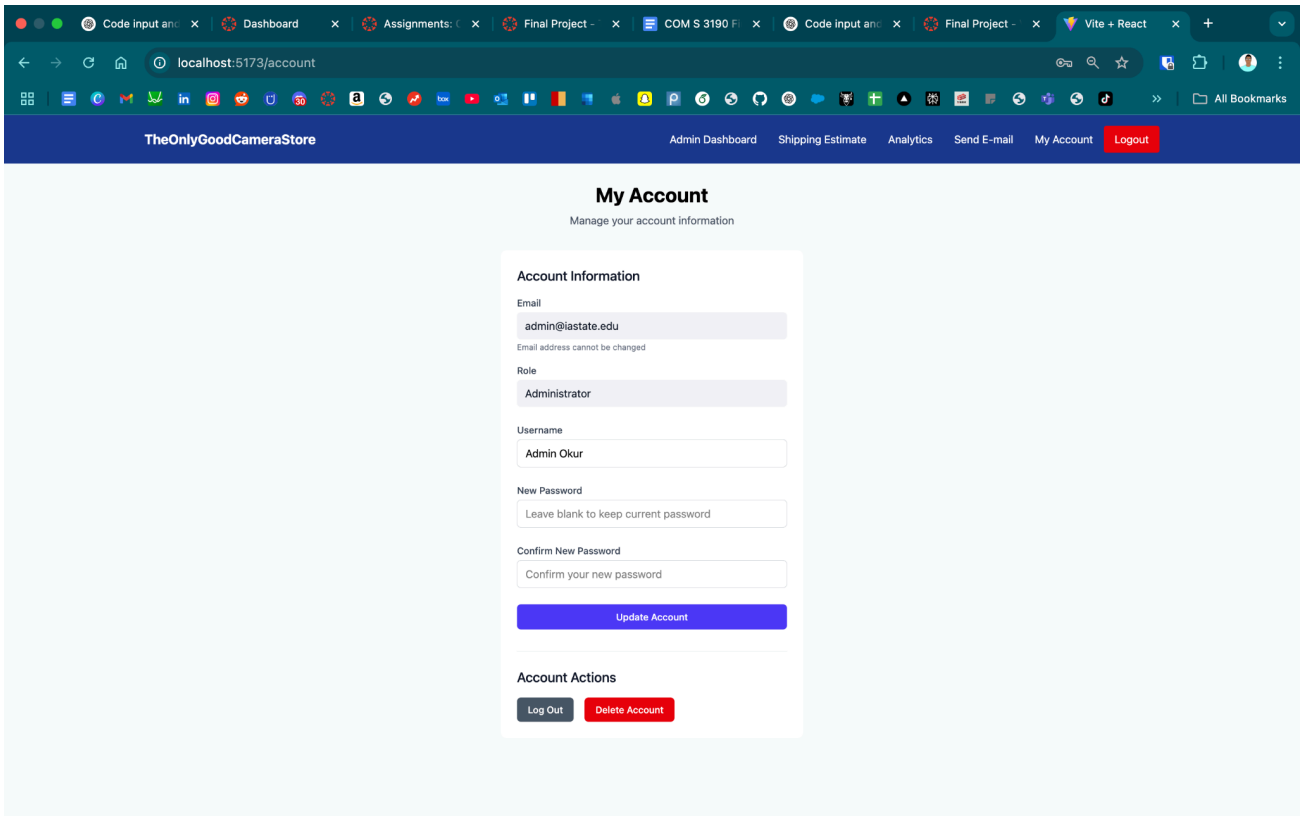
Highlighted Functional Areas:

- *Tabs*: Switch between managing Products, Orders, and Customer Questions.

- *Add Product Form*: Form data is handled with controlled React components and sent via `POST /api/products`.

- *Product Table*: Data is dynamically fetched from the backend (`GET /api/products`) and rendered in a responsive table layout.

Ahmet Page 2: My Account Page



Description:

This is the **My Account** page where a logged-in user can manage their profile settings. This page is

accessible to both admins and customers.

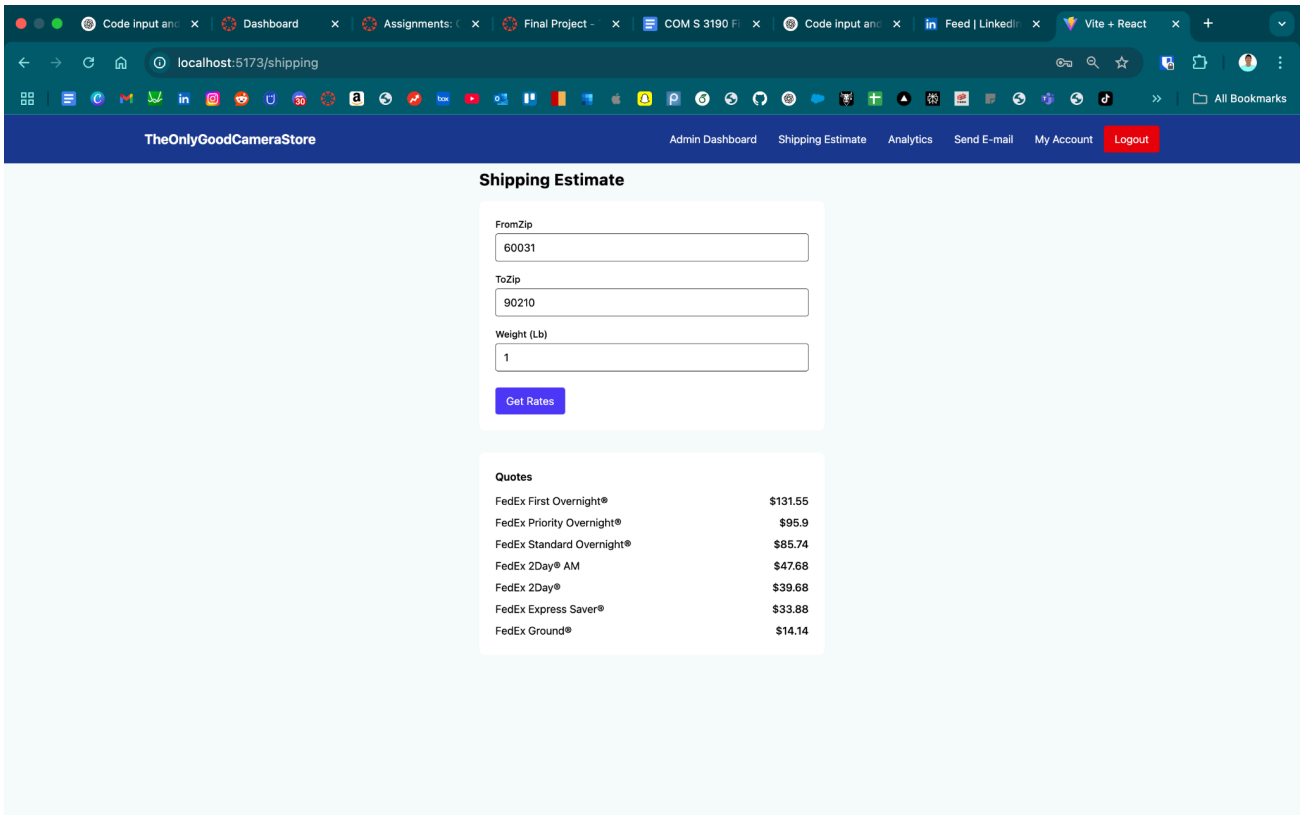
What the user can do:

- View their email and role (both are non-editable for security).
- Update their **username**.
- Change their **password** by filling out the "New Password" and "Confirm New Password" fields.
- Log out of their account.
- Permanently **delete their account** from the system.

Highlighted Functional Areas:

- **Form Handling:** Uses controlled React inputs. On form submit, a `PATCH` request is sent to `/api/users/:id` to update the user's profile.
- **Account Deletion:** Sends a `DELETE` request to the backend to remove the user entirely from the database.
- **Session Management:** The logout button clears localStorage and redirects the user back to the login screen.

Asray Page 1: Shipping Estimate Page



Description:

This is the **Shipping Estimate** tool available to admin users. It uses FedEx's sandbox API to fetch shipping rate quotes for a given ZIP-to-ZIP pair and package weight.

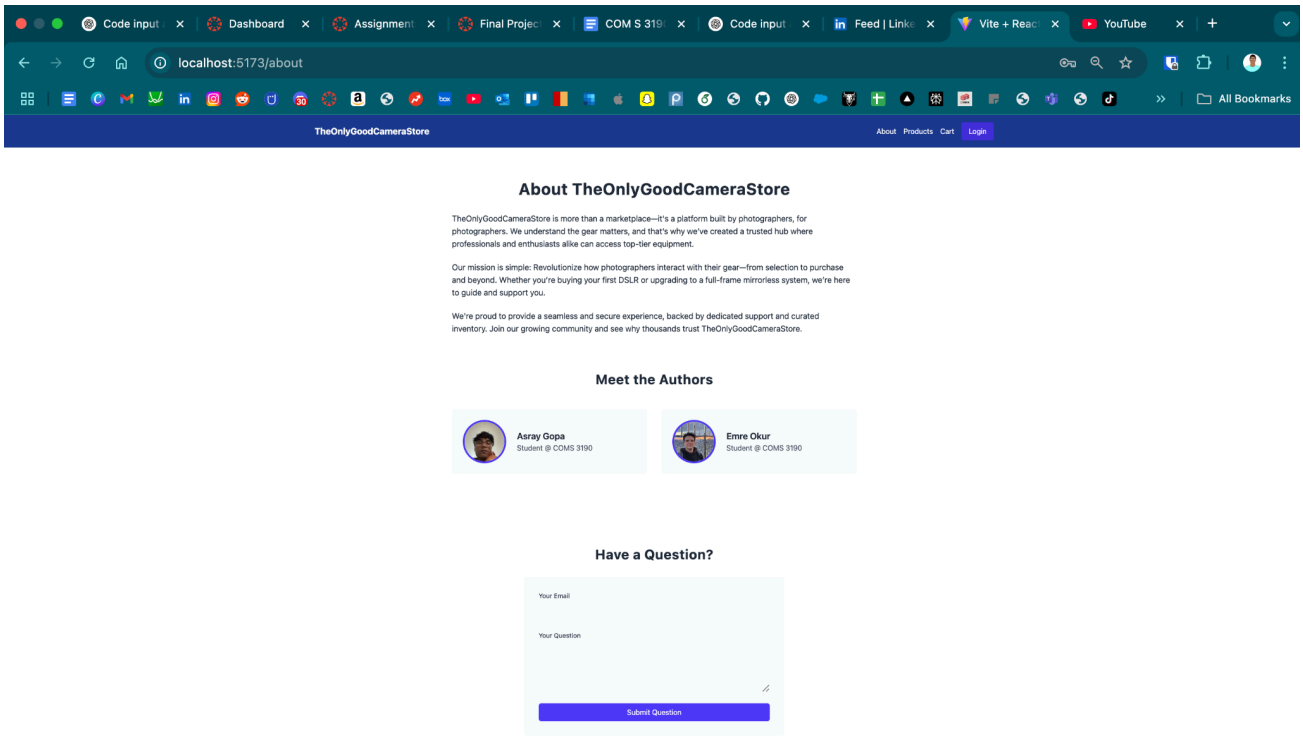
What the user can do:

- Input:
    - Origin ZIP code (`FromZip`)
    - Destination ZIP code (`ToZip`)
    - Package weight in pounds (`Weight (Lb)`)
- On clicking **Get Rates**, a `POST` request is made to `/api/ship/estimate` with the form data.
- The backend calls FedEx's API, processes the response, and returns a list of shipping services with their rates.
- Quotes are displayed clearly in a box below the form.

Key Features:

- Shows real-time rate estimates using live API data.
- Simplifies logistics planning for admins before fulfillment.
- Fully protected: Only admins can access this page based on role headers.

Asray Page 2: About Page with Question Submission



Description:

This is the **About** page that introduces the project and team members. It includes a brief explanation of the platform's mission and vision, along with a form for visitors to submit questions directly to the admin dashboard.

What the user can do:

- Read about the store's purpose and background.

- View the team members who built the project.
- Fill out a **question submission form** with their email and query.
    - The form sends a `POST` request to `/api/questions`.
    - Submitted questions appear in the admin dashboard for review.

Key Features:

- Clean and professional branding.
- Real-time interaction via the question form.
- Backend-linked functionality that supports user engagement with admins.

6. Installation and Setup Instructions

Backend

```
cd backend/
npm install
node server.js
```

Make sure MongoDB is running locally or update `MONGO_URI`.

Frontend

```
cd frontend/
npm install
npm run dev
```

Environment Variables

- Resend API key (Passed inline for now.)

- FedEx client ID and secret (Passed inline for now.)

7. Contribution Overview

Feature	Contributor
Home Page	Emre
Auth & My Account	Emre
Product Listings	Emre
Admin Product CRUD	Emre
Cart System	Emre
Not-Found Page	Emre
Orders (Customer)	Asray
Checkout System	Asray
Admin Orders Manager	Asray
Admin Analytics	Asray
Question Manager	Asray
Shipping Estimate (FedEx)	Asray
Email Sender (Resend)	Asray

8. Challenges Faced

- FedEx OAuth & API Rate Limits: Took a while to understand the token expiration and caching strategy.
- React Re-rendering on State Changes: Fixing stale state and scroll resets during order updates.
- Keeping Permissions Clean: Making sure only admins can access certain views took careful header checking.

9. Final Reflections

This project was a great end-to-end full-stack experience. We got to design, build, and test a real e-commerce system with both customer-facing and admin features.

We learned a lot about RESTful architecture, state management in React, MongoDB schema design, and third-party API integration (like FedEx and Resend).

In the future, we'd like to clean up security, deploy to production (e.g., Vercel + Render), and improve performance with caching and pagination.