E-Commerce Web App — AI-Assisted Development (Report-5)

Tools Used

ChatGPT – Guided the creation of new admin and order APIs, product review endpoints, and stock update logic. Assisted with JWT-based authentication, middleware design, error handling, and controller optimizations.

Claude AI – Recommended consistent use of async error wrappers (`catchAsyncErrors`), centralized error handling with `ErrorHandler`, and modular helper functions (e.g., stock update). Suggested schema validation improvements and route protection.

GitHub Copilot – Auto-completed controller boilerplate, Mongoose schema updates, route scaffolding, and review/order logic.

Project Overview

This phase focused on expanding the backend with:

1. **Admin User Management**
    - Get all users
    - Get single user details
    - Update user role
    - Delete user
2. **Product Reviews**
    - Create or update a review
    - Get all reviews for a product
    - Delete review (by owner or admin)

3. **Order Management**
    - Create new orders
    - Get a single order (with user details)
    - Get logged-in user's orders
    - Get all orders (admin)
    - Update order status (with stock updates)
    - Delete orders
4. **Auth & Middleware Enhancements**
    - JWT-based `isAuthenticatedUser` middleware
    - `authorizeRoles` middleware for admin-only routes
    - `catchAsyncErrors` wrapper for async route handling
    - `ErrorHandler` class for centralized error responses

Admin User Management

Controllers

```
// Get all users
export const getAllUsers = catchAsyncErrors(async (req, res, next) =>
{
  const users = await User.find().select('-password');
  res.status(200).json(users);
});

// Get single user
export const getUserById = catchAsyncErrors(async (req, res) => {
  const user = await User.findById(req.params.id).select('-password');
  if (!user) return next(new ErrorHandler('User not found', 404));
```

```
  res.status(200).json(user);
});

// Update user role
export const updateUserRole = catchAsyncErrors(async (req, res) => {
  const user = await User.findById(req.params.id);
  user.role = req.body.role;
  await user.save();
  res.status(200).json({ message: 'User role updated', user });
});

// Delete user
export const deleteUser = catchAsyncErrors(async (req, res) => {
  const user = await User.findById(req.params.id);
  await user.deleteOne();
  res.status(200).json({ message: 'User deleted successfully' });
});
```

Middleware

```
export const isAuthenticatedUser = async (req, res, next) => {
  const token = req.headers.authorization?.split(' ')[1] ||
req.cookies.token;
  if (!token) return res.status(401).json({ message: 'Not authorized'
});
  const decoded = jwt.verify(token, process.env.JWT_SECRET);
  req.user = await User.findById(decoded.id).select('-password');
  next();
};

export const authorizeRoles = (...roles) => (req, res, next) => {
  if (!roles.includes(req.user.role)) return res.status(403).json({
message: 'Admin only' });
  next();
};
```

Product Reviews

Controllers

```
// Create or update a review
export const createOrUpdateReview = catchAsyncErrors(async (req, res)
=> {
  const { rating, comment } = req.body;
  const product = await Product.findById(req.params.id);
  const existingReview = product.reviews.find(r => r.user.toString()
=== req.user._id.toString());

  if (existingReview) {
    existingReview.rating = rating;
    existingReview.comment = comment;
  } else {
    product.reviews.push({ user: req.user._id, name: req.user.name,
rating, comment });
  }

  product.numOfReviews = product.reviews.length;
  product.ratings = product.reviews.reduce((acc, r) => acc + r.rating,
0) / product.reviews.length;
  await product.save({ validateBeforeSave: false });

  res.status(200).json({ message: 'Review submitted successfully' });
});

// Get reviews
export const getProductReviews = catchAsyncErrors(async (req, res) =>
{
  const product = await Product.findById(req.params.id);
  res.status(200).json({ reviews: product.reviews, numOfReviews:
product.numOfReviews, averageRating: product.ratings });
});

// Delete review
export const deleteReview = catchAsyncErrors(async (req, res) => {
  const { productId, reviewId } = req.params;
```

```
  const product = await Product.findById(productId);
  product.reviews = product.reviews.filter(r => r._id.toString() !==
reviewId);
  product.numOfReviews = product.reviews.length;
  product.ratings = product.reviews.reduce((acc, r) => acc + r.rating,
0) / (product.reviews.length || 1);
  await product.save({ validateBeforeSave: false });
  res.status(200).json({ message: 'Review deleted successfully' });
});
```

Order Management

Mongoose Schema

```
const orderSchema = new mongoose.Schema({
  user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required:
true },
  orderItems: [{ name: String, price: Number, quantity: Number, image:
String, product: { type: mongoose.Schema.Types.ObjectId, ref:
'Product' } }],
  shippingInfo: { address: String, city: String, state: String,
postalCode: String, country: String, phoneNo: String },
  paymentInfo: { id: String, status: String, method: String },
  itemsPrice: { type: Number, required: true },
  taxPrice: { type: Number, required: true },
  shippingPrice: { type: Number, required: true },
  totalPrice: { type: Number, required: true },
  orderStatus: { type: String, default: 'Processing', enum:
['Processing', 'Shipped', 'Delivered', 'Cancelled'] },
  deliveredAt: Date,
  paidAt: Date,
}, { timestamps: true });
```

Controllers

```
// Create new order
export const createOrder = catchAsyncErrors(async (req, res) => {
```

```
  const order = await Order.create({ ...req.body, user: req.user._id,
paidAt: Date.now() });
  res.status(201).json({ message: 'Order created', order });
});

// Update order status with stock adjustment
const updateStock = async (productId, quantity) => {
  await Product.findByIdAndUpdate(productId, { $inc: { stock:
-quantity } }, { new: true });
};

export const updateOrderStatus = catchAsyncErrors(async (req, res) =>
{
  const order = await Order.findById(req.params.id);
  if (req.body.status === 'Delivered') {
    for (const item of order.orderItems) await
updateStock(item.product, item.quantity);
    order.deliveredAt = Date.now();
  }
  order.orderStatus = req.body.status;
  await order.save({ validateBeforeSave: false });
  res.status(200).json({ message: 'Order status updated', order });
});
```

AI Interaction & Prompts

Admin Controllers

ChatGPT suggested centralized `ErrorHandler` and `catchAsyncErrors` to avoid repetitive try/catch blocks and simplify error handling. Also provided role-based access middleware (`authorizeRoles`).

Product Reviews

ChatGPT recommended merging create/update endpoints, recalculating average ratings, and validating review ownership for delete operations.

Orders

AI guided the design of a comprehensive order schema, proper population of user info, automatic stock adjustment upon delivery, and safe async handling to prevent `req.user` undefined errors.

Impact & Code Highlights

Security Improvements

- JWT middleware ensures authenticated access (`req.user` attached reliably)
- Admin-only routes protected with `authorizeRoles`

Functionality Enhancements

- Product reviews fully CRUD-enabled
- Order management includes stock updates automatically

Code Reusability

- `catchAsyncErrors` and `ErrorHandler` used consistently
- `updateStock` helper ensures modular stock updates

Debugging & Testing Logs

- Logged `req.user` in middleware to verify JWT decoding
- Confirmed correct order totalPrice and stock decrement logic

Advantages

- Full backend coverage for users, products, reviews, and orders
- Centralized async error handling

- Modular, maintainable code architecture
- Real-world features: stock updates, order workflow, admin management

Limitations

- Email notifications for order/payment not implemented yet
- No rate-limiting for review or order endpoints
- Frontend integration of admin dashboards ongoing

Reflection

ChatGPT: Scaffolded controllers, middleware, and helper functions for all new backend features.
Claude AI: Suggested async wrappers, centralized error handling, and modular helpers.
Copilot: Auto-completed boilerplate, Mongoose schemas, and route scaffolding.

Hybrid AI-assisted development accelerated feature implementation while ensuring security, scalability, and maintainability.