

PROJECT REPORT

Project Title: E-Commerce Backend API

Developed By: [Your Name/Team]

Date: December 2025

Table of Contents

- [Abstract](#)
- [Chapter 1: Introduction](#)
- [Chapter 2: System Analysis & Design](#)
- [Chapter 3: Implementation Details](#)
- [Chapter 4: Testing & Security](#)
- [Chapter 5: Conclusion & Future Scope](#)

1. Abstract

The objective of this project is to develop a robust, scalable, and secure Backend API for an E-Commerce platform. The system is built using the **MERN Stack** (specifically Node.js, Express.js, and MongoDB) and TypeScript. It provides essential features such as User Authentication (JWT), Product Management, Media Uploads(Cloudinary), and Role-Based Access Control. The project emphasizes secure coding practices, performance optimization via database indexing, and a clean, maintainable Service-Layer Architecture.

2. Chapter 1: Introduction

1.1 Purpose

The primary purpose is to serve as the server-side logic for an online shopping application. It handles data persistence, business logic, and ensures secure communication between the client and the database.

1.2 Scope

- Admin Module:** Manage products, users, and system settings.
- Customer Module:** Browse products, search/filter, and manage profile/reviews.
- Security:** Protect against common web vulnerabilities.

1.3 Technology Stack

- Language:** TypeScript (v5.x)
- Runtime:** Node.js (v18+)
- Framework:** Express.js (v5.x)
- Database:** MongoDB (v6.x) with Mongoose ODM
- Tools:** Cloudinary (Media), Nodemailer (Email), Zod (Validation), Morgan (Logging)

3. Chapter 2: System Analysis & Design

3.1 Architecture Diagram

The system follows a **Layered Architecture**:

Client Request -> Route -> Middleware (Validation/Auth) -> Controller -> Service (Business Logic) -> Model (Database)

3.2 Database Schema

- User Collection:** Stores name, email, password (**hashed**), role, and timestamps.
- Product Collection:** Stores name, price, description, images (**array**), ratings (**embedded array**), and aggregated stats like averageRating.

3.3 Data Flow

- Authentication:** Users receive an accessToken (15m validity) and a refreshToken (7d validity). Tokens are stored in HttpOnly Cookies for maximum security against XSS attacks.
- Product Search:** Queries use MongoDB's \$text search for keywords and comparison operators (\$gte, \$lte) for price filtering.

4. Chapter 3: Implementation Details

4.1 Service Layer Pattern

Business logic is decoupled from HTTP transport.

Example: AuthService handles the logic of checking passwords, while AuthController only handles req and res.

4.2 Key Modules

- Authentication:** Implements bcrypt for hashing and jsonwebtoken for session management. Includes a robust "Forgot Password" flow using crypto-generated tokens sent via SMTP.
- File Handling:** Uses multer middleware to stream file buffers directly to Cloudinary CDN, returning a secure URL for storage.
- Review System:** A logic wrapper that prevents duplicate reviews and calculates the mathematical average of ratings in real-time.

5. Chapter 4: Testing & Security

5.1 Security Measures

- **Helmet:** Secures HTTP headers.
- **Rate Limiting:** Limits IP to 100 requests per 15 minutes.
- **HPP:** Prevents HTTP Parameter Pollution attacks.
- **Sanitization:** `express-mongo-sanitize` removes malicious \$ signs from inputs.

5.2 Performance

- **Indexing:** Fields `price`, `category`, and `name` are indexed in MongoDB.
- **Compression:** Gzip compression reduces response body size by ~70%.

6. Chapter 5: Conclusion & Future Scope

6.1 Conclusion

The E-Commerce Backend API successfully meets all functional queries for a modern shopping platform. It is "Production Ready", meaning it handles errors gracefully, logs activities, and secures user data.

6.2 Future Scope

- **Payment Gateway:** Integration with Stripe/Razorpay.
- **Order Management:** Full checkout flow and inventory tracking.
- **Real-time Notifications:** Using Socket.io for order updates.