Technical Documentation & Project Report

Project Name: E-Commerce Backend API
Version: 1.0.0
Date: December 7, 2025

---

1. Executive Summary

This document provides a comprehensive technical overview of the E-Commerce Backend API. This system is a production-ready, RESTful API built to handle secure authentication, product management, and media uploads. It is designed with scalability, security, and maintainability as core tenets, following the **Service-Repository Pattern** (adapted as Service-Controller for Mongoose).

2. Technology Stack

Component	Technology	Purpose
Runtime	Node.js	Asynchronous, event-driven JavaScript runtime.
Framework	Express.js	Minimalist web framework for routing and middleware.
Language	TypeScript	Static typing for reliability and developer experience.
Database	MongoDB	NoSQL database for flexible data modeling.
ODM	Mongoose	Object Data Modeling for schema validation.
Auth	JWT (JSON Web Token)	Stateless authentication mechanism.
Validation	Zod	Runtime schema validation for requests.
Logging	Morgan	HTTP request logger.
Uploads	Cloudinary	Cloud-based image management.

---

3. System Architecture

The project follows a **Layered Architecture** to separate concerns and improve testability.

3.1. Layer Breakdown

1. **Presentation Layer (Routes & Controllers)**:
    - Handles incoming HTTP requests.
    - Validates input using **Zod**.
    - Sends HTTP responses to the client.
    - **No business logic** resides here.
2. **Service Layer (Services)**:
    - Contains all **Business Logic**.
    - Interacts with the Database Layer.
    - Handles complex operations like password hashing, token generation, and data aggregation.
3. **Data Access Layer (Models)**:
    - Mongoose Schemas definition.
    - Database indexes for performance.

3.2. Directory Structure

```
src/
├── config/         # Configuration (DB, Cloudinary)
├── constants/      # Static values (Messages, Cookie Options)
├── controllers/    # Request Handlers (Auth, Product)
├── middlewares/    # Interceptors (Auth, Upload, Security)
├── models/         # Mongoose Schemas
├── routes/         # API Endpoint Definitions
├── services/       # Business Logic
├── types/          # TypeScript Interfaces
├── utils/          # Helpers (AppError, catchAsync)
├── app.ts          # App configuration & Middleware
└── server.ts       # Server entry point
```

---

4. Key Features & Implementation

4.1. Authentication & Security

- **Dual-Token System**: Uses short-lived **Access Tokens** (15m) and long-lived **Refresh Tokens** (7d) for secure and persistent sessions.
- **Transport**: Tokens are sent via `HttpOnly Cookies` (to prevent XSS) and `Authorization Headers`.
- **Password Security**: Passwords are hashed using `bcryptjs` with salt.
- **Password Reset**: Secure flow generating a sha256 hashed token, sent via **Email (Nodemailer)**. Link expires in 10 minutes.

4.2. Advanced Product Management

- **Filtering**: Supports MongoDB operators (`gte`, `gt`, `lte`, `lt`) directly in query params (e.g., `price[gte]=100`).
- **Search**: Full-text search on `name` and `description` fields using MongoDB Text Indexes.
- **Pagination**: Efficient data retrieval using `limit` and `skip`.

4.3. Reviews & Ratings

- **Aggregation**: When a review is added, the system **automatically recalculates**:
    - `numOfReviews`: Total count.
    - `averageRating`: Mathematical average.
    - This is optimized to prevent heavy calculation on read operations.

4.4. Image Uploads

- **Pipeline**: `Multer` intercepts the file -> Streams to **Cloudinary** -> Returns URL.
- **Validation**: Restricts uploads to image MIME types only.

---

5. Security Report

The application implements a "Defense in Depth" strategy:

1. **Helmet**: Sets secure HTTP Headers (X-DNS-Prefetch-Control, X-Frame-Options, etc.).
2. **Rate Limiting**: Limits IP requests (100 req / 15 min) to prevent Brute Force and DDoS.
3. **HPP (HTTP Parameter Pollution)**: Prevents pollution attacks where attackers send multiple params with the same name.
4. **MongoSanitize**: Strips `$` and `.` from user input to prevent NoSQL Injection.
5. **CORS**: Configured to allow specific origins.
6. **Zod Validation**: Strict schema validation rejects any malformed or unexpected data payload before it reaches logic.

---

6. Performance Optimizations

1. **Database Indexing**:
    - `price` (Ascending): For range filtering.
    - `category` (Ascending): For category filtering.
    - `name` + `description` (Text): For search queries.
2. **Compression**: Gzip compression enabled for all HTTP responses to reduce payload size.
3. **Lean Queries**: usage of `.select()` and `.lean()` (where applicable) to reduce object overhead.
4. **Graceful Shutdown**: The server listens for `SIGTERM` signals to close the database connection and strictly finish pending requests before shutting down.

---

7. Operational Workflow

7.1. Setup

1. `npm install`
2. Configure `.env` (Database, Cloudinary, SMTP).

7.2. Development

- `npm run dev`: Starts the server with `nodemon` for hot-reloading.

7.3. Production Build

- `npm run build`: Compiles TypeScript to JavaScript in `dist/`.
- `npm start`: Runs the compiled code.

7.4. Testing

- `npm test`: (Placeholder) Ready for test integration.

---

8. Conclusion

The **E-Commerce Backend API** is a robust, secure, and scalable foundation for any e-commerce client. It meets modern web standards and is built to be easily extensible for future requirements like Payment Integration or WebSocket notifications.