**Technical Documentation**

Introduction

This technical documentation outlines the architecture, database schema, and implementation details of the E-Commerce API backend.

Architecture Overview

The E-Commerce API is built using Node.js and Express, utilizing MongoDB as the database for storing user, product, and order information. The application follows a RESTful architecture, providing endpoints for managing users, products, and orders.

Database Schema

* Users Table:

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | ObjectId | Unique identifier for each user |
| username | String | User's username |
| email | String | User's email (unique) |
| password | String | Hashed password |
| address | String | User's shipping address |

* Products Table:

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | ObjectId | Unique identifier for each product |
| name | String | Name of the product |
| description | String | Description of the product |
| price | Number | Price of the product |
| stock | Number | Number of items available |

* Orders Table:

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | ObjectId | Unique identifier for each order |
| user\_id | ObjectId | Reference to the user placing order |
| total\_amount | Number | Total amount for the order |
| status | String | Status of the order (e.g., pending, completed) |

* Order\_Items Table:

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | ObjectId | Unique identifier for each item in an order |
| order\_id | ObjectId | Reference to the associated order |
| product\_id | ObjectId | Reference to the purchased product |
| quantity | Number | Quantity of the product ordered |

API Endpoints Overview

Refer to the User Guide section for detailed descriptions of all available endpoints, including request methods, request bodies, and expected responses.

Error Handling Strategy

The application uses centralized error handling middleware to catch errors and respond appropriately:

javascript

app.use((err, req, res, next) => {

console.error(err.stack);

res.status(500).json({ message: 'Something went wrong!' });

});

Testing Strategy

* Manual testing was conducted using Postman to ensure all endpoints function correctly.
* Automated tests can be implemented using frameworks like Jest or Mocha for unit testing individual components.

Deployment Instructions

To deploy your application on Heroku:

1. Install Heroku CLI and log in:

bash

heroku login

1. Create a new Heroku app and push your code:

bash

heroku create <your-app-name>

git push heroku main

1. Set environment variables on Heroku:

bash

heroku config:set MONGODB\_URI=your\_mongodb\_uri\_here

heroku config:set JWT\_SECRET=your\_jwt\_secret\_here

1. Verify that your application is running correctly by visiting the provided Heroku URL.