Backend Web Development 101 Project Documentation

Project Title: Inventory Management System (IMS)

Author:	Aldrin John Tamayo
Date Created:	November 28, 2024
Version:	0.1

Table of Contents

Project Description

The **Inventory Management System (IMS)** is a backend web application designed to manage an inventory of items. Users can perform basic **CRUD** operations—Create, Read, Update, and Delete—on the inventory. The application features a simple and user-friendly interface, leveraging modern backend technologies for seamless performance and scalability.

Objective

The objective of this project is to:

1. Demonstrate the use of Node.js, Express.js, EJS, and MongoDB/Mongoose to build a full-stack CRUD application.
2. Implement a structured and scalable backend architecture.
3. Provide a real-world example of managing data and server-client communication in web development.

Technology Stack

Backend

- **Node.js**: Runtime environment for executing JavaScript on the server.
- **Express.js**: Framework to handle routing and middleware.
- **MongoDB**: NoSQL database for storing inventory data.
- **Mongoose**: ODM library for MongoDB to manage database schemas and queries.

Frontend

- **EJS**: Templating engine for rendering dynamic HTML pages.

Core Features

1. **Item Management**:
    - **Create**: Add new items to the inventory.
    - **Read**: View a list of all items in the inventory.
    - **Update**: Edit details of existing items.
    - **Delete**: Remove items from the inventory.
2. **Search and Filter**: (Optional)
    - Search items by name or category.
    - Filter items based on specific criteria (e.g., category or stock status).

System Architecture

The IMS is designed using the **Model-View-Controller (MVC)** architecture for maintainability and scalability:

- **Model**: Defines the data structure and interacts with the database using Mongoose.
- **View**: Renders dynamic content using EJS templates.
- **Controller**: Handles application logic and routes requests between models and views.

Database Schema

The MongoDB database will store inventory data using the following schema:

```
const mongoose = require('mongoose');

const itemSchema = new mongoose.Schema({
 name: { type: String, required: true },
 category: { type: String, required: true },
 quantity: { type: Number, default: 0 },
 price: { type: Number, required: true },
 description: { type: String, default: '' },
 createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Item', itemSchema);
```

API Endpoints

The system exposes RESTful API endpoints for CRUD operations:

1. Create Item

- **Endpoint**: POST /items
- **Request Body**:

```
{
"name": "Sample Item",
"category": "Electronics",
"quantity": 10,
"price": 150.00,
"description": "A sample electronic item."
}
```

- **Response:**

```
{
"message": "Item created successfully.",
"item": { ... }
}
```

2. Read Items

- **Endpoint: GET /items**
- **Response:**

```
[
{
  "_id": "item_id",
  "name": "Sample Item",
  "category": "Electronics",
  "quantity": 10,
  "price": 150.00,
  "description": "A sample electronic item."
},
...
]
```

3. Update Item

- **Endpoint: PUT /items/:id**
- **Request Body:**

```
{

"name": "Updated Item",

"quantity": 15

}
```

- **Response:**

```
{

"message": "Item updated successfully.",

"item": { ... }

}
```

4. Delete Item

- **Endpoint: DELETE /items/:id**
- **Response**

```
{

"message": "Item deleted successfully."

}
```

Frontend Integration

EJS templates will render dynamic pages for the CRUD operations:

1. Index Page: Displays a table listing all items.
2. Add Item Page: Contains a form to create a new item.
3. Edit Item Page: Pre-fills item data for editing.
4. Delete Confirmation Page: Confirms item deletion.

Future Enhancements

- Implement user authentication for secure access.
- Add support for image uploads for inventory items.
- Include advanced filtering and sorting options.

Conclusion

The Inventory Management System showcases how to build a robust backend application with Node.js, Express.js, EJS, and MongoDB. This project serves as a foundational example for CRUD-based web applications and can be expanded with additional features based on real-world requirements.