

# **ELECTRONICS INVENTORY APP USING MONGODB, EXPRESS, AND NODE.JS**

---

BUILT WITH MEN STACK | ES MODULES | MONGODB ATLAS

## Tools and Technologies Used

Node.js – JavaScript runtime for backend

Express – Web framework for building REST APIs

MongoDB Atlas – Cloud NoSQL database

Mongoose – ODM for MongoDB

Dotenv – Secure environment variable management

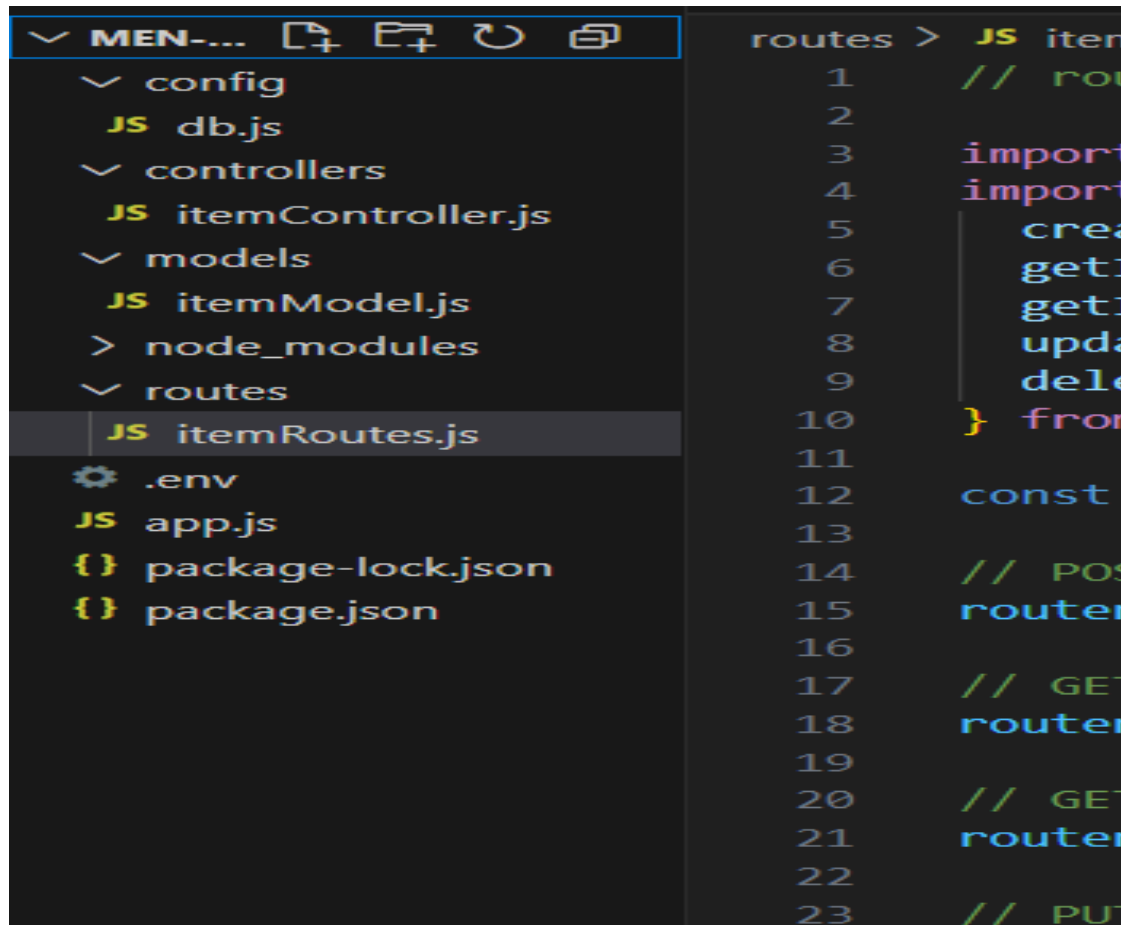
Nodemon – Dev tool for auto-reloading server

## Dependencies

Installing the required packages:

- `npm init -y`
- `npm install express dotenv mongoose`
- `npm install nodemon --save-dev`

# Project Structure



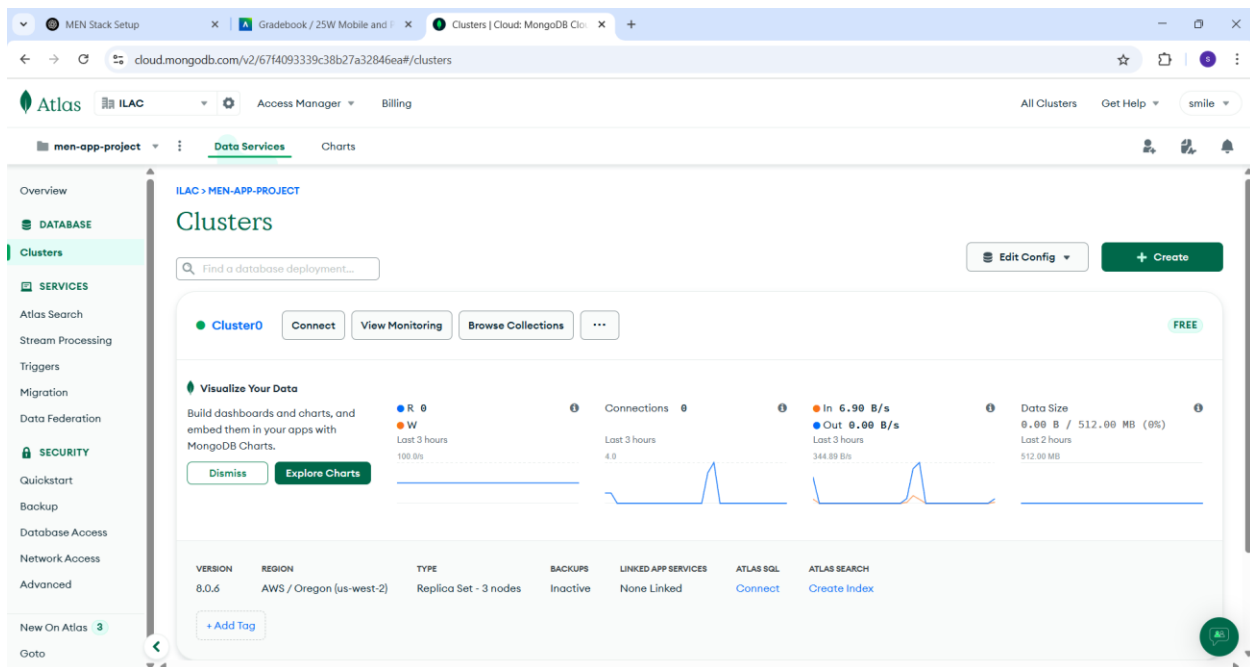
Men-app/

- |—— config/      # DB config
- |—— controllers/    # Route logic
- |—— models/        # Mongoose schema
- |—— routes/        #Contains API route definitions
- |—— app.js         # Main entry
- |—— .env            # Mongo URI stored here

# Creating MongoDB Atlas setup

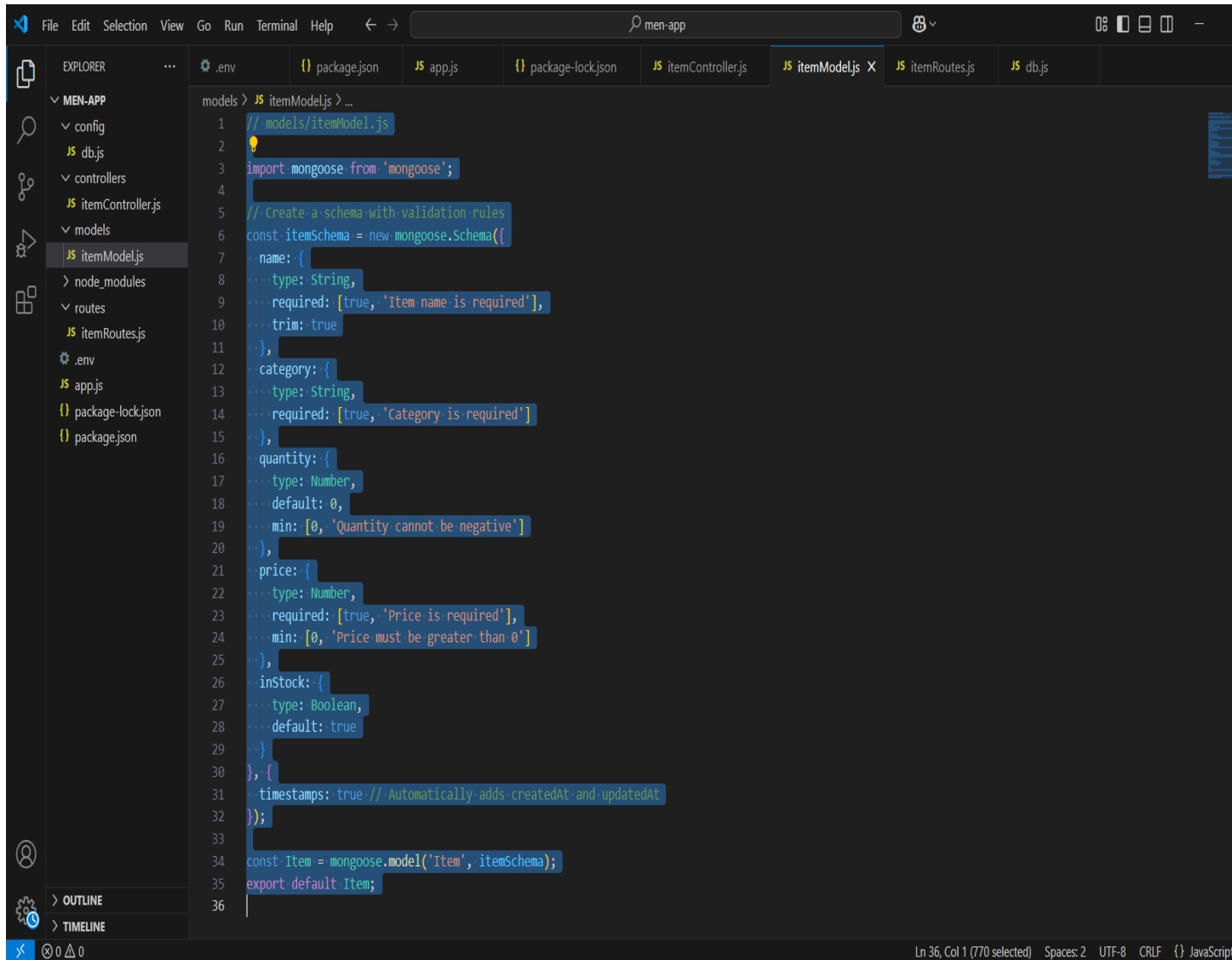
- Go to <https://www.mongodb.com/cloud/atlas> and log in.
- Create a free cluster (select latest version ).
- Create a new database and a collection (e.g., items).
- Get connection string and place it in the .env file:

`MONGO_URI=mongodb+srv://myuser27:mypassword27@cluster0.yz3rag3.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0`



# Creating Mongoose Schema

Inside models/itemModel.js:



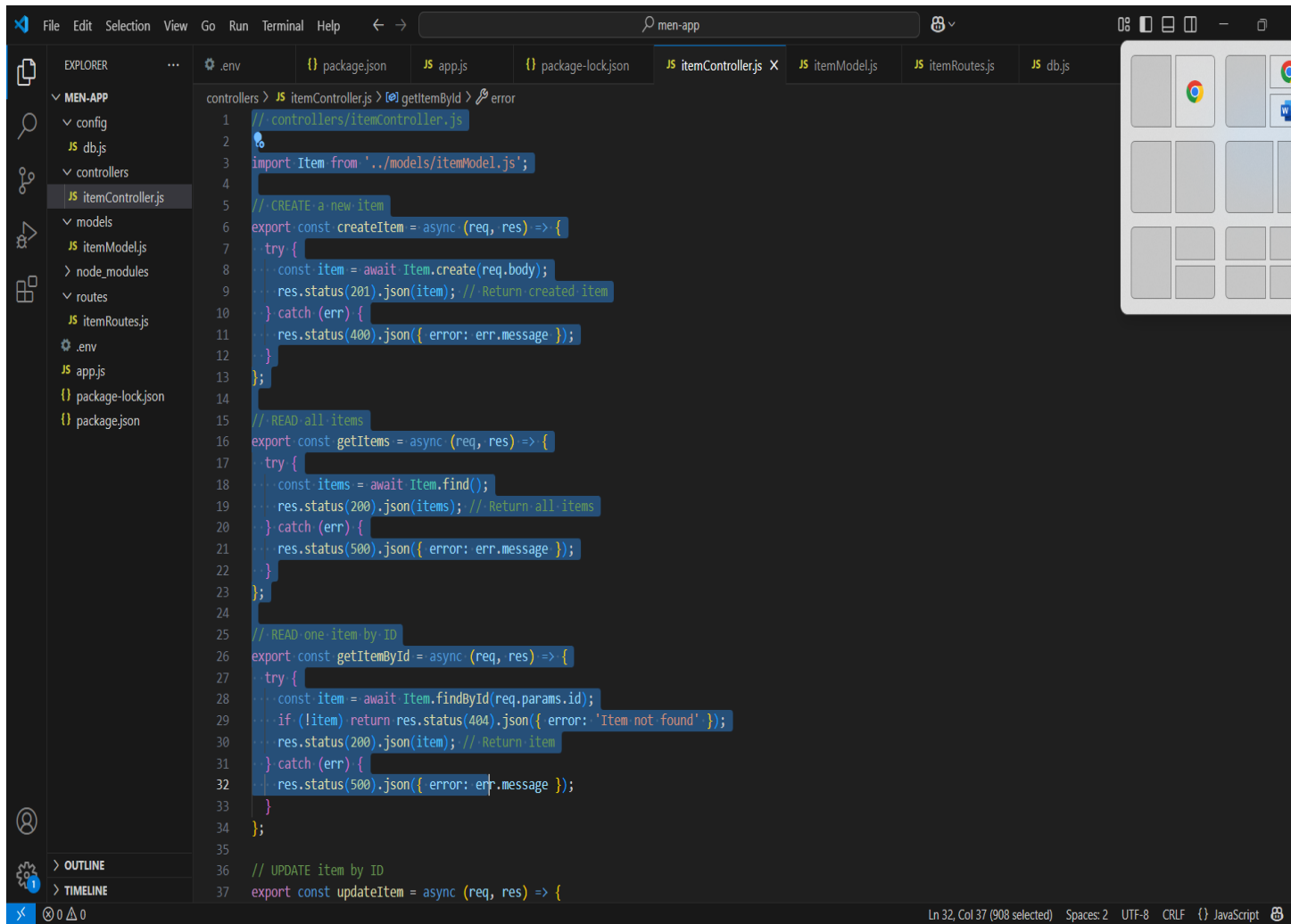
The screenshot shows a Visual Studio Code editor window with the file explorer on the left and the code editor in the center. The file explorer shows a project structure with folders like config, controllers, models, routes, and node\_modules. The file 'itemModel.js' is selected in the models folder. The code editor shows the following JavaScript code:

```
1 // models/itemModel.js
2
3 import mongoose from 'mongoose';
4
5 // Create a schema with validation rules
6 const itemSchema = new mongoose.Schema({
7   name: {
8     type: String,
9     required: [true, 'Item name is required'],
10    trim: true
11  },
12  category: {
13    type: String,
14    required: [true, 'Category is required']
15  },
16  quantity: {
17    type: Number,
18    default: 0,
19    min: [0, 'Quantity cannot be negative']
20  },
21  price: {
22    type: Number,
23    required: [true, 'Price is required'],
24    min: [0, 'Price must be greater than 0']
25  },
26  inStock: {
27    type: Boolean,
28    default: true
29  }
30 }, {
31   timestamps: true // Automatically adds createdAt and updatedAt
32 });
33
34 const Item = mongoose.model('Item', itemSchema);
35 export default Item;
36
```

The status bar at the bottom indicates the cursor is at line 36, column 1, with 770 characters selected. The encoding is UTF-8 and the line endings are CRLF.

# Creating Controller Logic

Inside controllers/itemController.js:

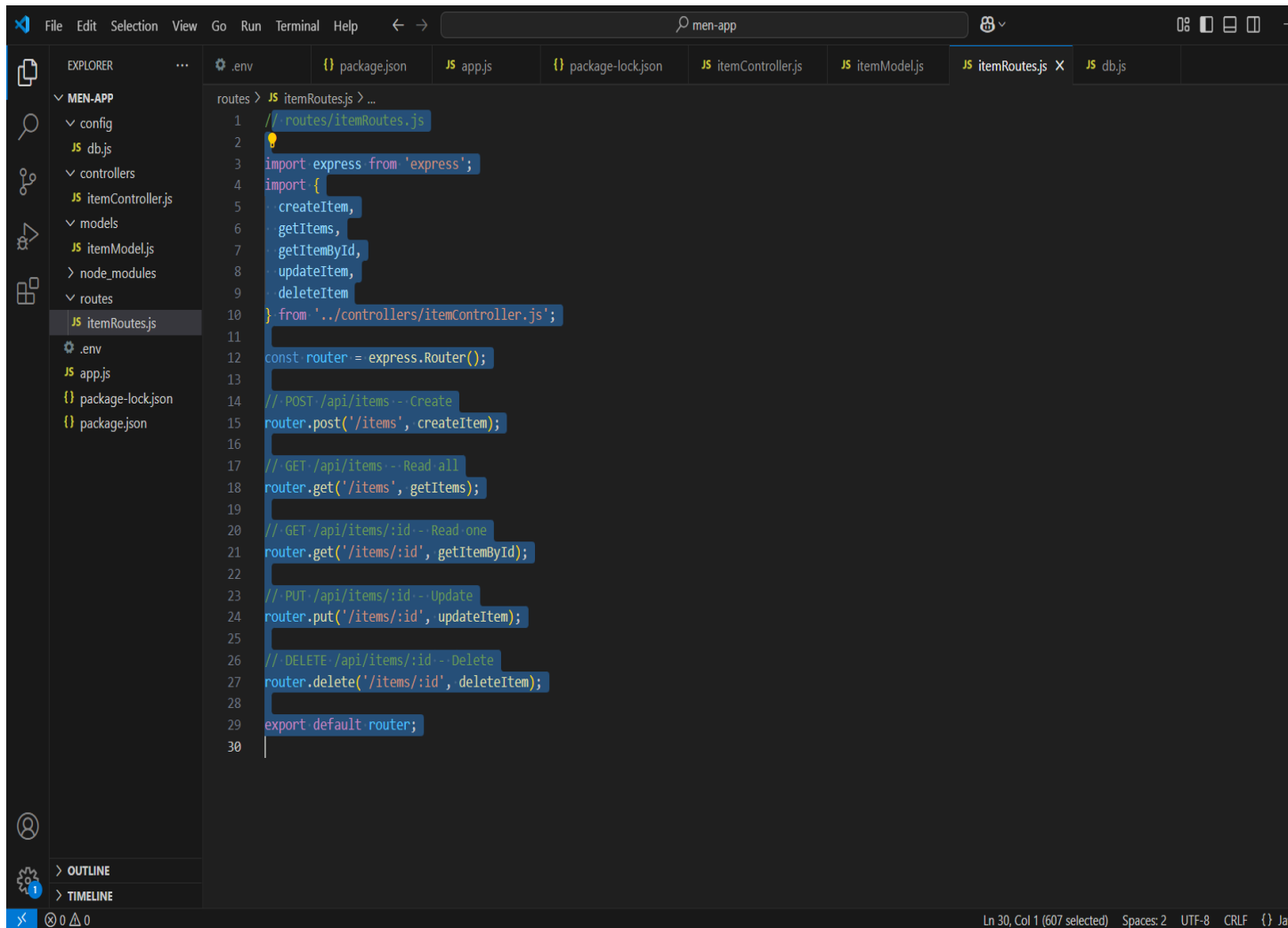


```
1 // controllers/itemController.js
2
3 import Item from '../models/itemModel.js';
4
5 // CREATE a new item
6 export const createItem = async (req, res) => {
7   try {
8     const item = await Item.create(req.body);
9     res.status(201).json(item); // Return created item
10  } catch (err) {
11    res.status(400).json({ error: err.message });
12  }
13 };
14
15 // READ all items
16 export const getItems = async (req, res) => {
17   try {
18     const items = await Item.find();
19     res.status(200).json(items); // Return all items
20   } catch (err) {
21     res.status(500).json({ error: err.message });
22   }
23 };
24
25 // READ one item by ID
26 export const getItemById = async (req, res) => {
27   try {
28     const item = await Item.findById(req.params.id);
29     if (!item) return res.status(404).json({ error: 'Item not found' });
30     res.status(200).json(item); // Return item
31   } catch (err) {
32     res.status(500).json({ error: err.message });
33   }
34 };
35
36 // UPDATE item by ID
37 export const updateItem = async (req, res) => {
```

Ln 32, Col 37 (908 selected) Spaces: 2 UTF-8 CRLF {} JavaScript

# Define API Routes

Inside routes/itemRoutes.js:

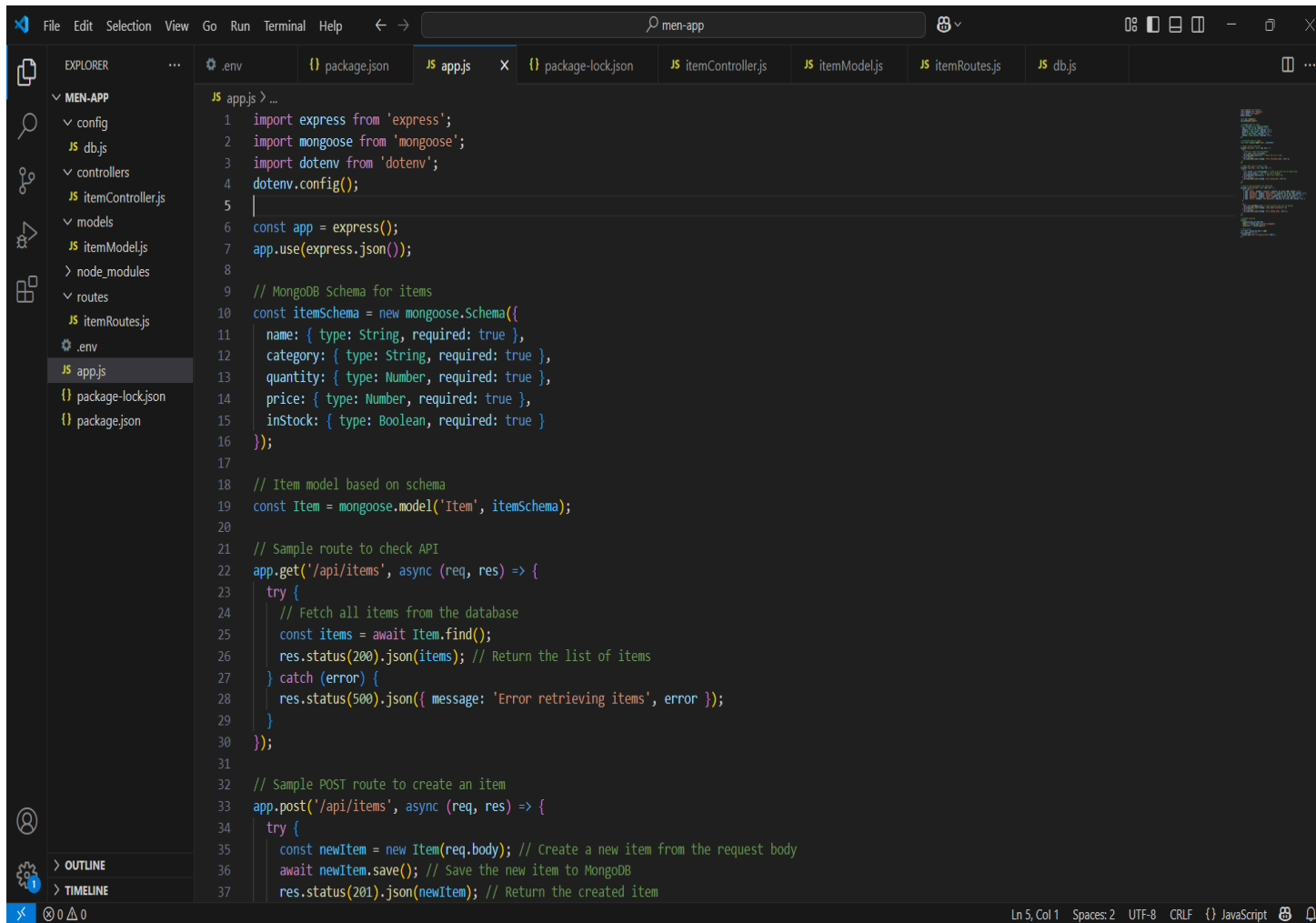


```
1 // routes/itemRoutes.js
2
3 import express from 'express';
4 import {
5   createItem,
6   getItems,
7   getItemById,
8   updateItem,
9   deleteItem
10 } from '../controllers/itemController.js';
11
12 const router = express.Router();
13
14 // POST /api/items -- Create
15 router.post('/items', createItem);
16
17 // GET /api/items -- Read all
18 router.get('/items', getItems);
19
20 // GET /api/items/:id -- Read one
21 router.get('/items/:id', getItemById);
22
23 // PUT /api/items/:id -- Update
24 router.put('/items/:id', updateItem);
25
26 // DELETE /api/items/:id -- Delete
27 router.delete('/items/:id', deleteItem);
28
29 export default router;
30
```

Ln 30, Col 1 (607 selected) Spaces: 2 UTF-8 CRLF {} Ja

# Set Up Main App File

In app.js:



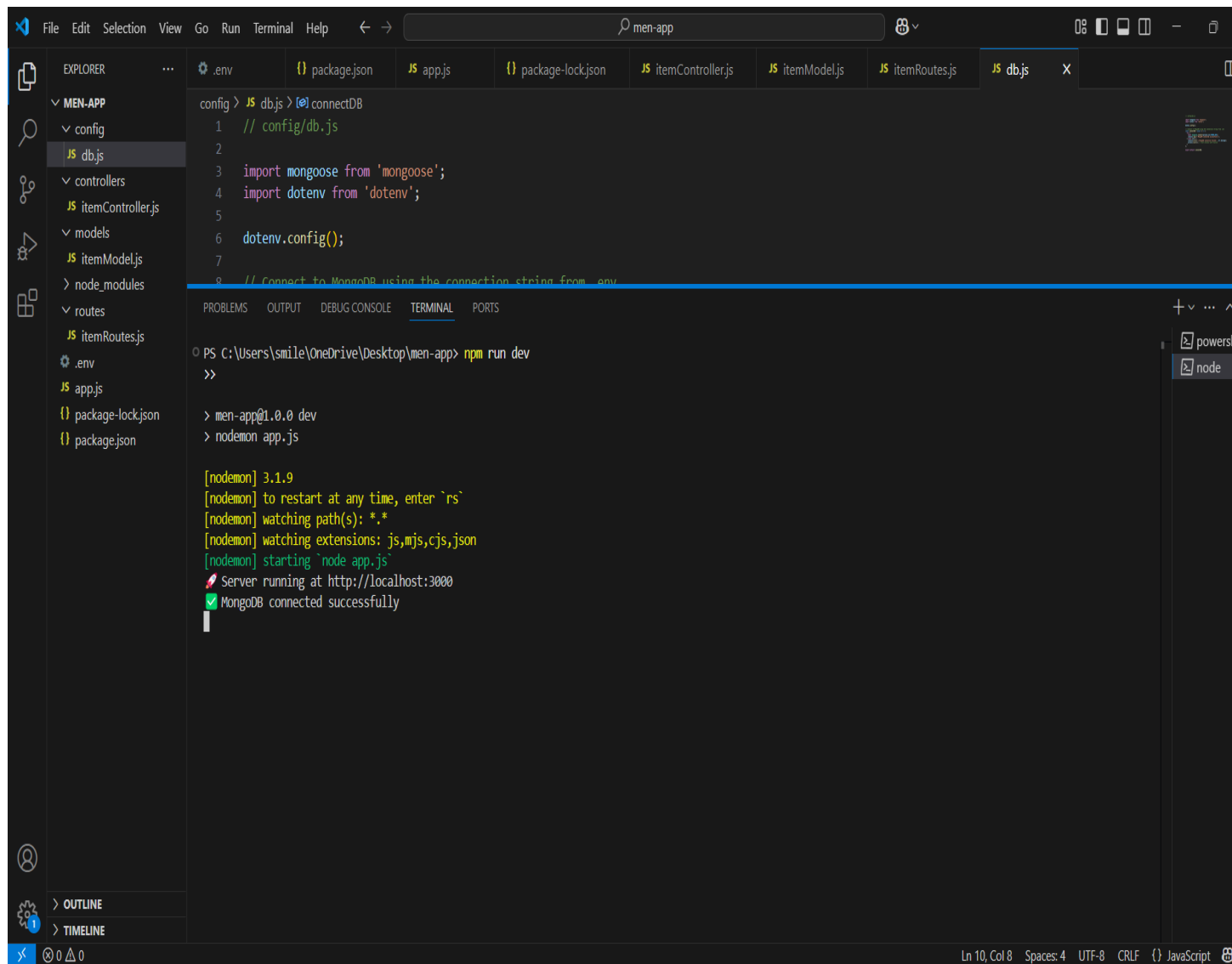
```
JS app.js > ...
1  import express from 'express';
2  import mongoose from 'mongoose';
3  import dotenv from 'dotenv';
4  dotenv.config();
5  |
6  const app = express();
7  app.use(express.json());
8  |
9  // MongoDB Schema for items
10 const itemSchema = new mongoose.Schema({
11   name: { type: String, required: true },
12   category: { type: String, required: true },
13   quantity: { type: Number, required: true },
14   price: { type: Number, required: true },
15   inStock: { type: Boolean, required: true }
16 });
17 |
18 // Item model based on schema
19 const Item = mongoose.model('Item', itemSchema);
20 |
21 // Sample route to check API
22 app.get('/api/items', async (req, res) => {
23   try {
24     // Fetch all items from the database
25     const items = await Item.find();
26     res.status(200).json(items); // Return the list of items
27   } catch (error) {
28     res.status(500).json({ message: 'Error retrieving items', error });
29   }
30 });
31 |
32 // Sample POST route to create an item
33 app.post('/api/items', async (req, res) => {
34   try {
35     const newItem = new Item(req.body); // Create a new item from the request body
36     await newItem.save(); // Save the new item to MongoDB
37     res.status(201).json(newItem); // Return the created item
```



## Test with Postman

- Open Postman.
- Send a POST request to:

<http://localhost:3000/api/items>



The screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left displays the project structure for 'MEN-APP', including files like .env, package.json, app.js, package-lock.json, itemController.js, itemModel.js, itemRoutes.js, and db.js. The main editor area shows the content of 'db.js', which includes database connection logic using mongoose and dotenv. Below the editor, the TERMINAL panel is active, showing the command 'npm run dev' being executed. The terminal output indicates that the application is running successfully on http://localhost:3000 and that MongoDB is connected.

```
config > JS db.js > connectDB
1 // config/db.js
2
3 import mongoose from 'mongoose';
4 import dotenv from 'dotenv';
5
6 dotenv.config();
7
8 // Connect to MongoDB using the connection string from .env

PS C:\Users\smile\OneDrive\Desktop\men-app> npm run dev
>>
> men-app@1.0.0 dev
> nodemon app.js

[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
🔥 Server running at http://localhost:3000
✅ MongoDB connected successfully
```

GET

▼

http://localhost:3000/api/items

ParamsAuthorizationHeaders (6)BodyScriptsSettings

BodyCookiesHeaders (7)Test Results↺

{ } JSON ▼

▶ Preview

🔗 Visualize ▼

```
21      "_id": "67f417f955f571e4d08a12d1",
22      "name": "Smartwatch",
23      "category": "Wearables",
24      "quantity": 15,
25      "price": 250,
26      "inStock": true,
27      "__v": 0
28    },
29    {
30      "_id": "67f417f955f571e4d08a12d2",
31      "name": "Headphones",
32      "category": "Audio",
33      "quantity": 100,
34      "price": 140,
35      "inStock": true,
36      "__v": 0
37    },
38    {
39      "_id": "67f417f955f571e4d08a12d3",
40      "name": "LED TV",
41      "category": "Home Appliances",
42      "quantity": 10,
43      "price": 650,
44      "inStock": false,
45      "__v": 0
46    }
47  ]
```

Find and replaceConsole

```
1    {
2      "_id": "67f417f955f571e4d08a12d0",
3      "name": "Smartphone",
4      "category": "Mobile Phones",
5      "quantity": 50,
6      "price": 1050,
7      "inStock": true,
8      "__v": 0
9    },
10   {
11     "_id": "67f417f955f571e4d08a12d1",
12     "name": "Smartwatch",
13     "category": "Wearables",
14     "quantity": 15,
15     "price": 250,
```

HomeWorkspacesAPI Network

Search Postman

Invite

Upgrade

OverviewGetting startedGET http://localhost:3000/api/items

No environment

http://localhost:3000/api/items

SaveShare

</>

GET

http://localhost:3000/api/items

Send

ParamsAuthorizationHeaders (6)BodyScriptsSettings

Cookies

Query Params

Key	Value	Description		Bulk Edit
Key	Value	Description		

BodyCookiesHeaders (7)Test Results

200 OK · 110 ms · 875 B

JSONPreviewVisualize

```
2  {
3    "_id": "67f417f955f571e4d08a12cf",
4    "name": "Laptop",
5    "category": "Computers",
6    "quantity": 25,
7    "price": 1600,
8    "inStock": true,
9    "__v": 0
10  },
11  {
12    "_id": "67f417f955f571e4d08a12d0",
13    "name": "Smartphone",
14    "category": "Mobile Phones",
15    "quantity": 50,
16    "price": 1050,
17    "inStock": true,
18    "__v": 0
19  }
20 }
```

OnlineFind and replaceConsole

PostbotRunnerStart ProxyCookiesVaultTrash

Share

Send

## Cookies

	Key	Value	Description	⋮ Bulk Edit
	Key	Value	Description	

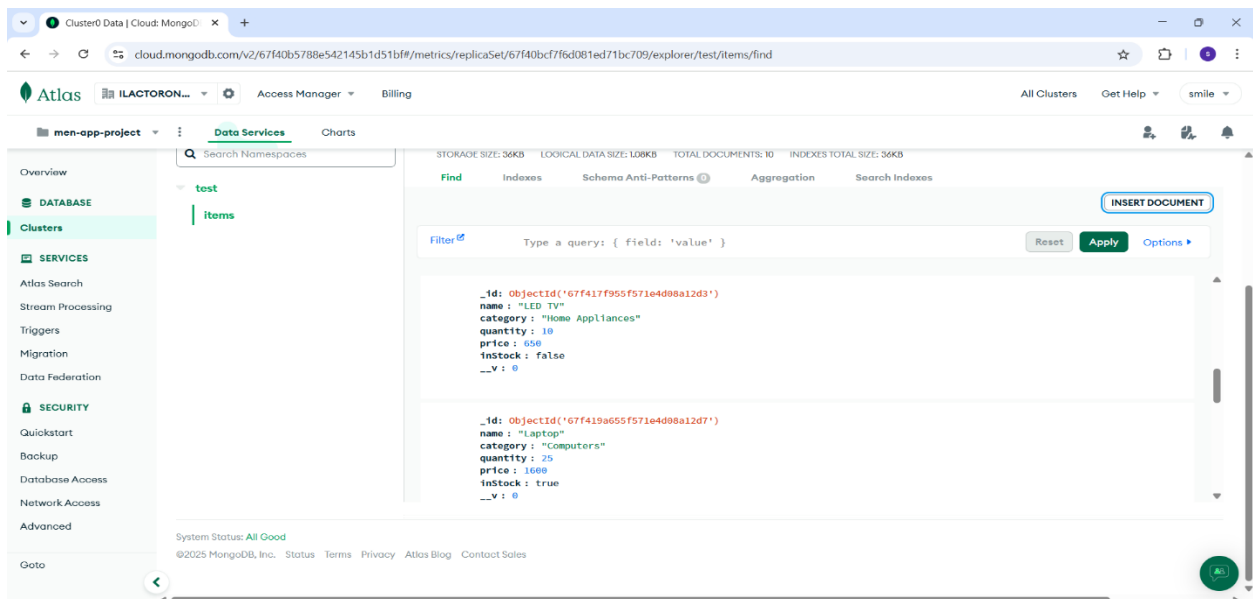
 | 

```
1
2      "message": "Items added successfully!"
3
```

# MongoDB Atlas Collection with One Document

The items collection is visible in my MongoDB Atlas database.

Electronic things are represented by the documents in this collection. The five essential fields in every document are name, category, quantity, price, and inStock. To guarantee data quality, validation procedures are used and the data is kept in a professional schema.



Cluster0 Data | Cloud: Mongo0

cloud.mongodb.com/v2/67f40b5788e542145b1d51bf#/metrics/replicaSet/67f40bcf7f6d081ed71bc709/explorer/test/items/find

Atlas

ILACTORON...

Access Manager

Billing

All Clusters

Get Help

smile

men-app-project

Data Services

Charts

Overview

test

Items

STORAGE SIZE: 36KB

LOGICAL DATA SIZE: 1.08KB

TOTAL DOCUMENTS: 10

INDEXES TOTAL SIZE: 36KB

Find

Indexes

Schema Anti-Patterns

Aggregation

Search Indexes

INSERT DOCUMENT

Filter

Type a query: { field: 'value' }

Reset

Apply

Options

QUERY RESULTS: 1-10 OF 10

\_id: ObjectId('67f417f955f571e4d88a12cf')

name: "Laptop"

category: "Computers"

quantity: 25

price: 1600

inStock: true

\_v: 0

\_id: ObjectId('67f417f955f571e4d88a12d0')

name: "Smartphone"

category: "Mobile Phones"

quantity: 50

price: 1000

System Status: All Good

©2025 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales