

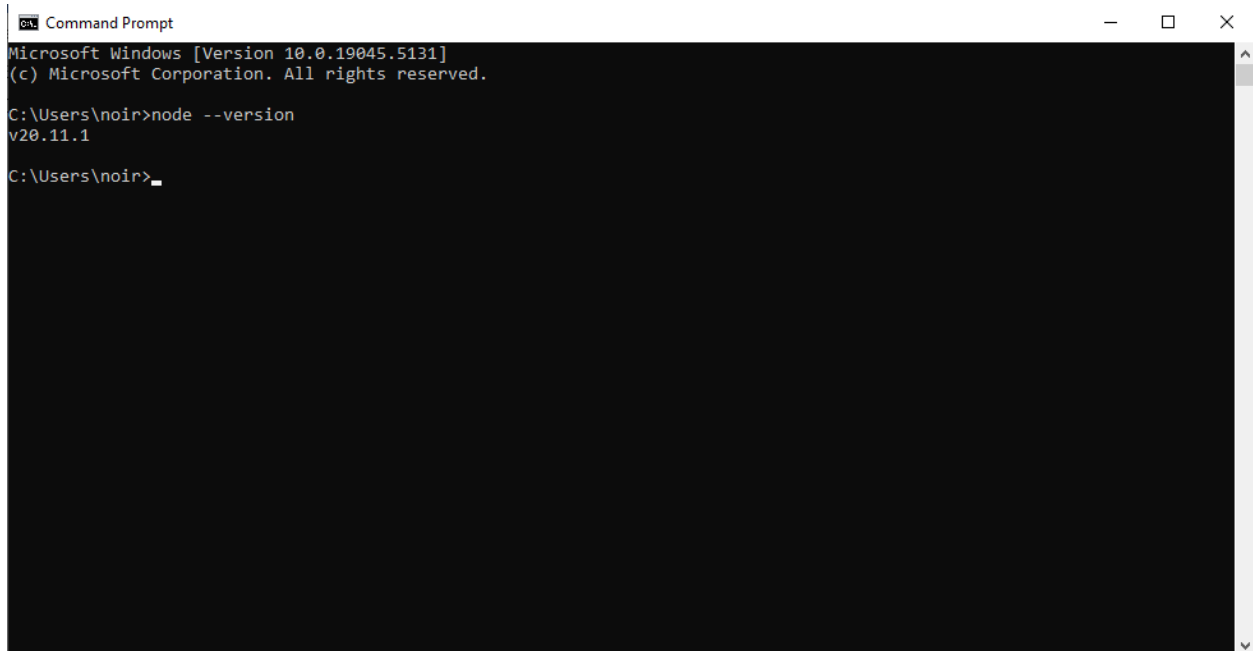
## BACKEND

### STEP 1.

Download and Install Node.js, prebuilt installer: <https://nodejs.org/en/download/prebuilt-installer>

### STEP 2.

Check if Node.js is installed, by typing “node –version” in cmd.



```
Command Prompt
Microsoft Windows [Version 10.0.19045.5131]
(c) Microsoft Corporation. All rights reserved.

C:\Users\noir>node --version
v20.11.1

C:\Users\noir>
```

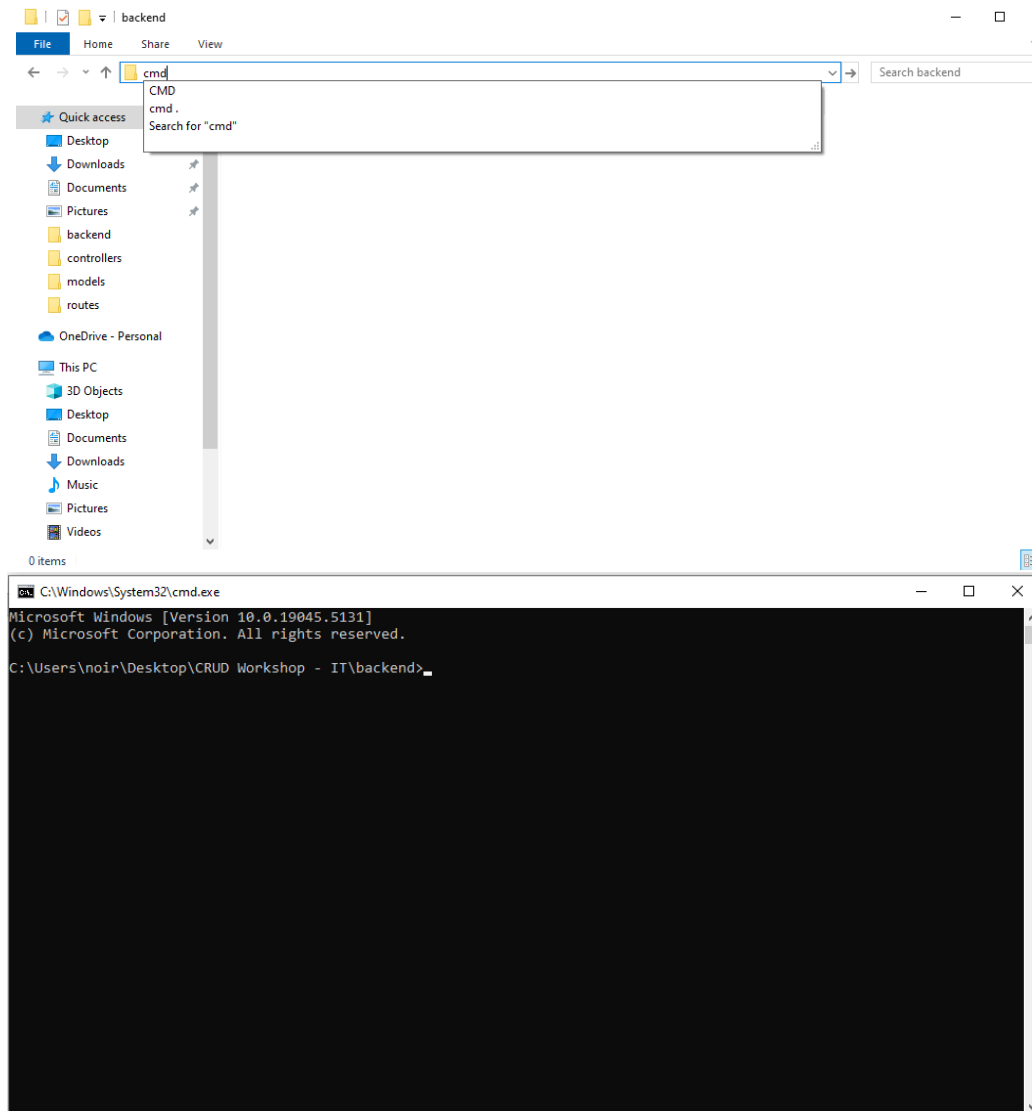
### STEP 3.

Create a folder name “BASIC CRUD APP”. Inside the root folder, create another folder named “backend”.

- BASIC CRUD APP
  - o backend

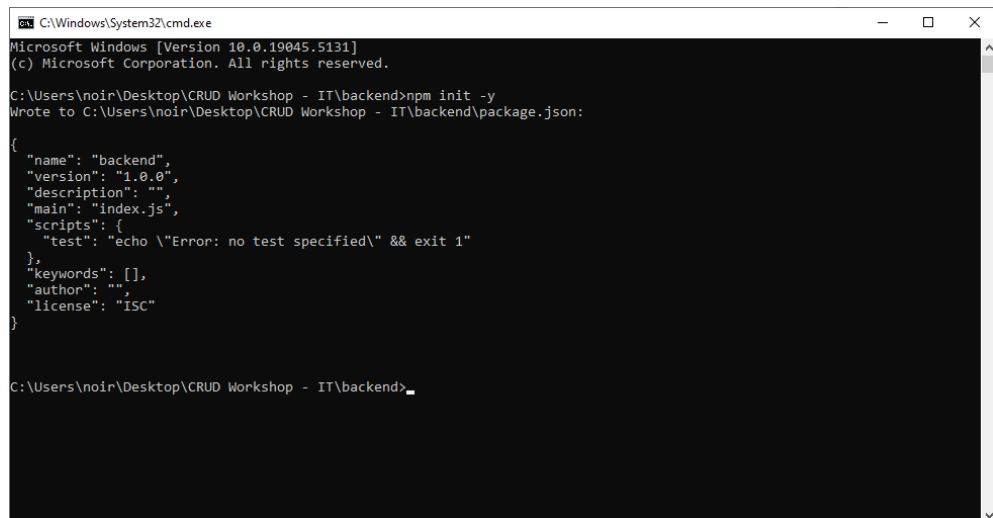
### STEP 4.

Access the backend folder, in the path field type “cmd”, then hit enter



### STEP 5.

In the cmd, type “npm init -y”, then hit enter.



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.5131]
(c) Microsoft Corporation. All rights reserved.

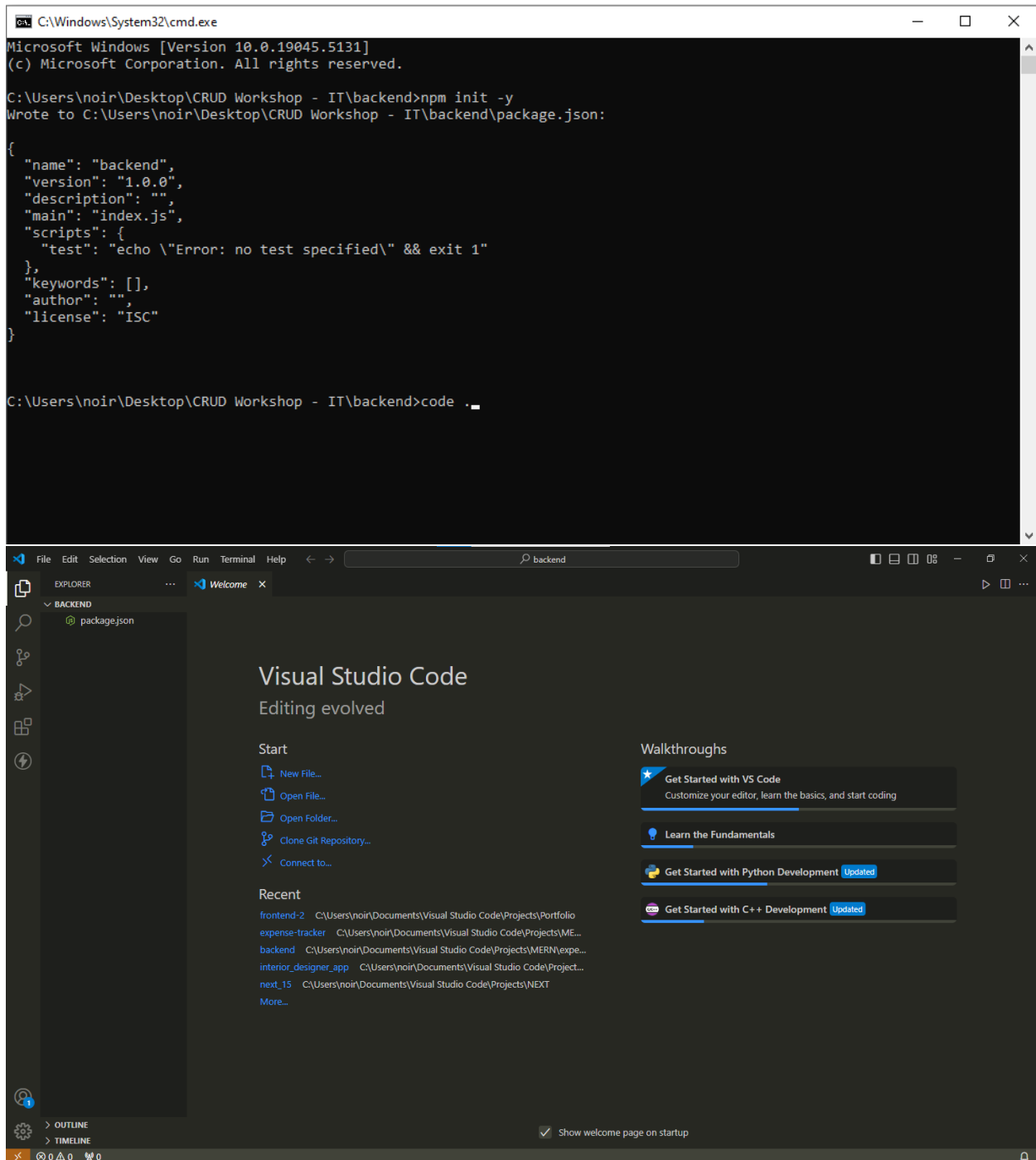
C:\Users\noir\Desktop\CRUD Workshop - IT\backend>npm init -y
Wrote to C:\Users\noir\Desktop\CRUD Workshop - IT\backend\package.json:

{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

C:\Users\noir\Desktop\CRUD Workshop - IT\backend>
```

## STEP 6

In the cmd, type "code .", then hit enter, it should the VS Code Editor.



STEP 7.

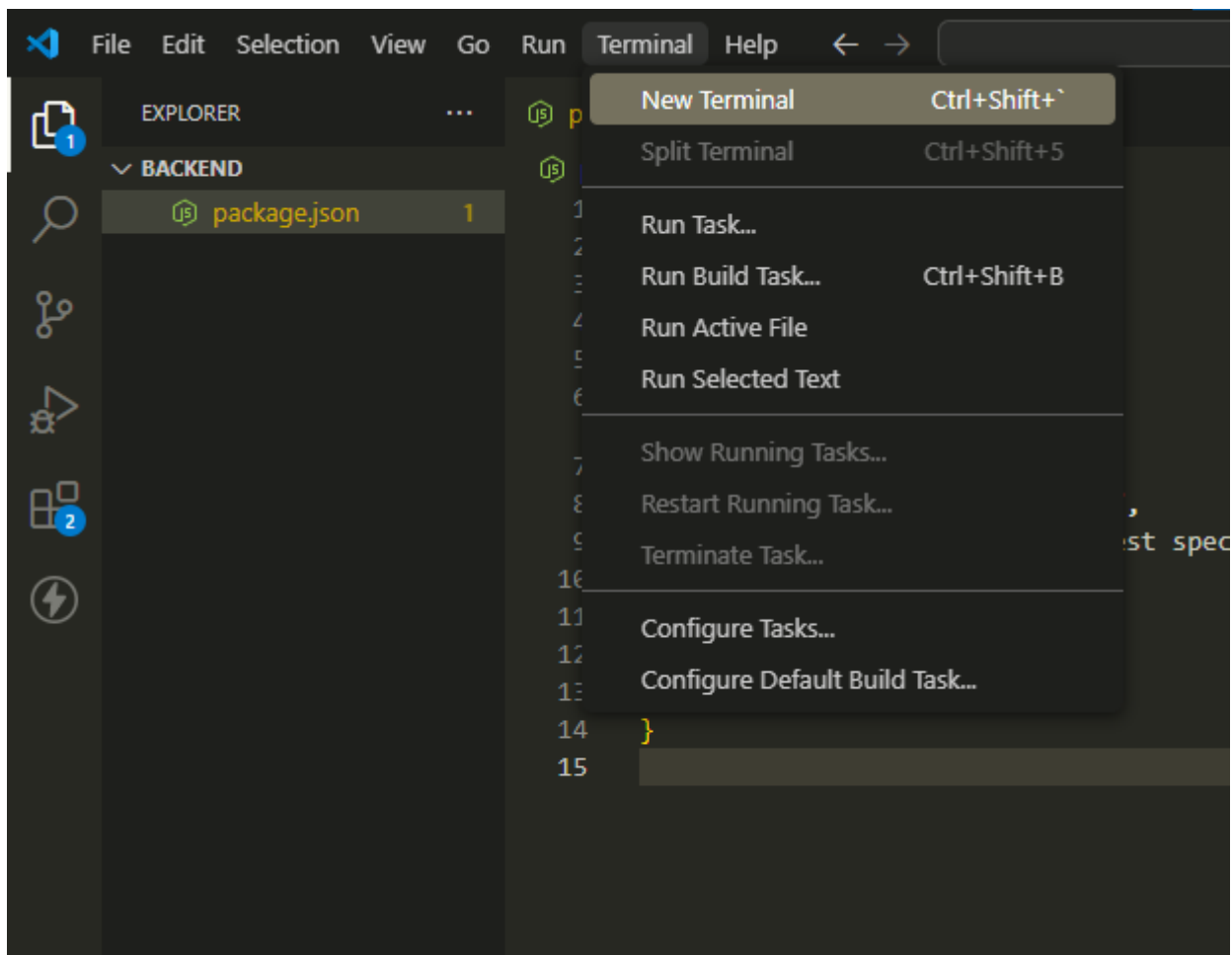
In the explorer section, click the "package.json", then add this code block:

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
```

```
"type": "module",
"main": "server.js",
"scripts": {
  "start": "nodemon server.js",
  "test": "echo \\\"Error: no test specified\\\" && exit 1"
},
"keywords": [],
"author": "",
"license": "ISC"
}
```

STEP 8.

Go to terminal, then click new terminal.

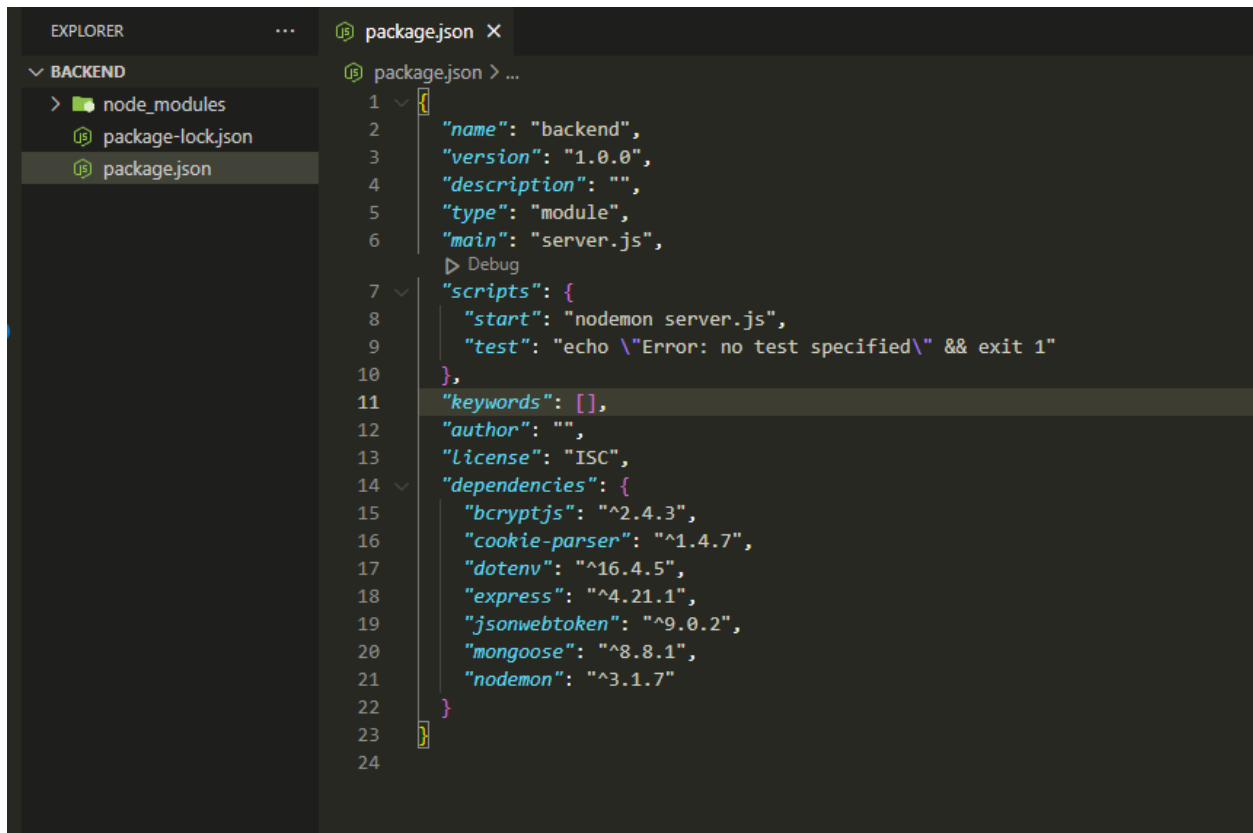


STEP 9.

Install the required dependencies by hitting enter.

```
npm install nodemon express mongoose dotenv multer
```

Your “package.json” should have the dependencies installed.

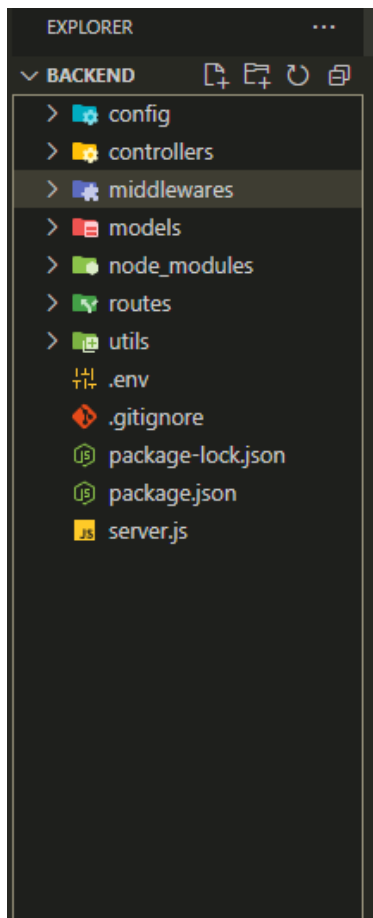


The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar is open, showing a project structure with a folder named 'BACKEND'. Inside 'BACKEND', there are three files: 'node\_modules', 'package-lock.json', and 'package.json'. The 'package.json' file is selected and its content is displayed in the main editor. The content of 'package.json' is as follows:

```
1 {
2   "name": "backend",
3   "version": "1.0.0",
4   "description": "",
5   "type": "module",
6   "main": "server.js",
7   "scripts": {
8     "start": "nodemon server.js",
9     "test": "echo \"Error: no test specified\" && exit 1"
10  },
11  "keywords": [],
12  "author": "",
13  "license": "ISC",
14  "dependencies": {
15    "bcryptjs": "^2.4.3",
16    "cookie-parser": "^1.4.7",
17    "dotenv": "^16.4.5",
18    "express": "^4.21.1",
19    "jsonwebtoken": "^9.0.2",
20    "mongoose": "^8.8.1",
21    "nodemon": "^3.1.7"
22  }
23 }
```

#### STEP 10.

In the explorer section, create a new file name “server.js”, “.env”, and “.gitignore”. Also create folders, “config”, “controllers”, “routes”, “models”, and “middlewares”



STEP 11.

Click the “.env” and add this port:

```
PORT=5000
```

Crtl + s to save

STEP 12.

Click the “.gitignore” and add this statement:

```
.env
```

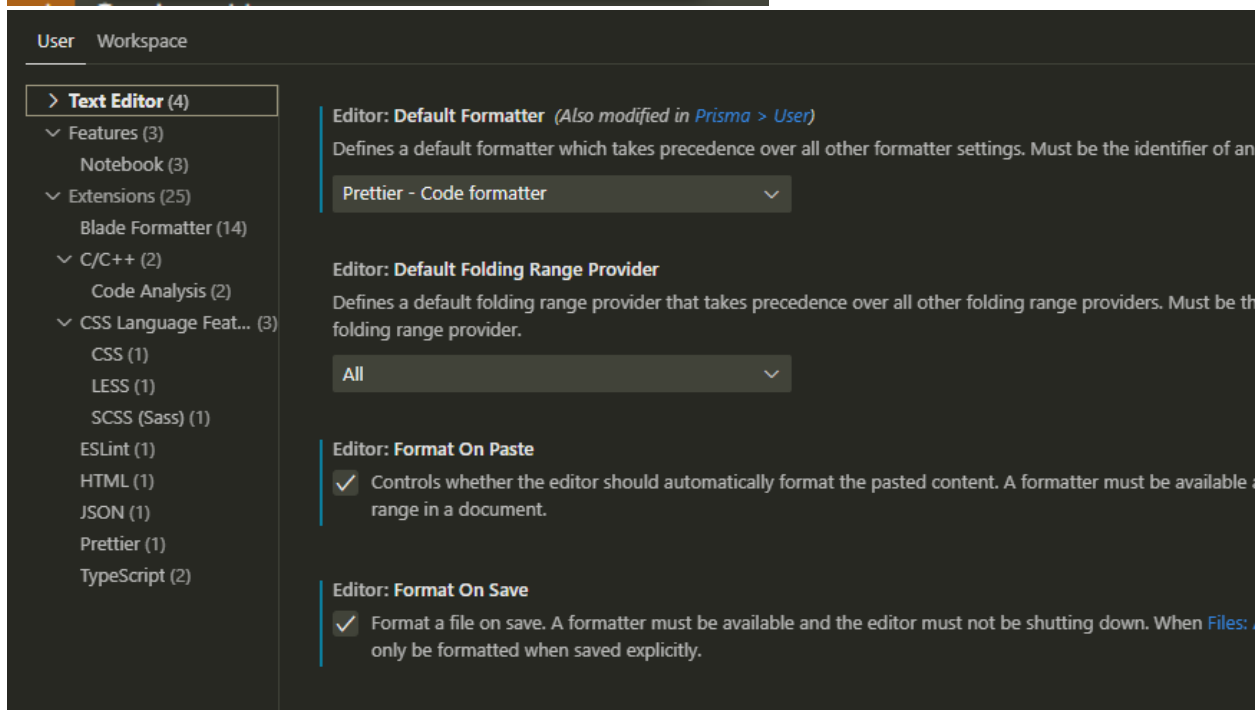
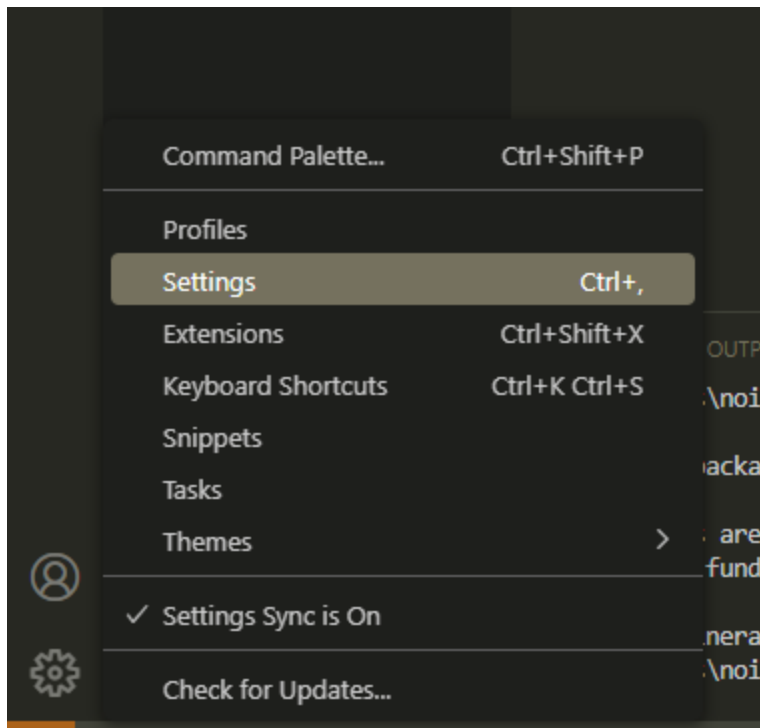
Crtl + s to save

STEP 13. (Optional)

Click the “Extensions tab”, then install “material icon” and “prettier code formatter”.

STEP 13.1. (Optional)

In the settings, type “formatter”. Click the Text Editor. Set the default formatter to “Prettier – Code Formatter”, and check “Format on Paste” and Format on Save”



STEP 14.

Click the "server.js", then add this code block:

```
import express from "express";
import dotenv from "dotenv";
```



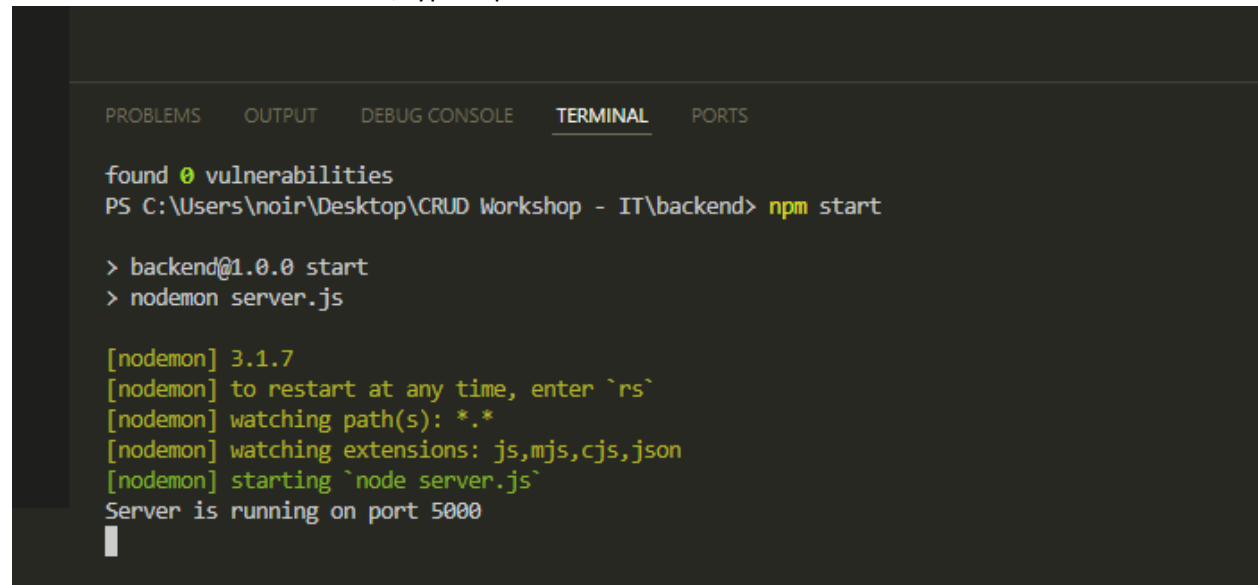
```
dotenv.config();

const app = express();

// Middlewares
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Listen
const port = process.env.PORT || 4000;
app.listen(port, () => console.log(`Server is running on port ${port}`));
```

CTRL + s to save. In the terminal, type “npm start” then hit enter.



```
found 0 vulnerabilities
PS C:\Users\noir\Desktop\CRUD Workshop - IT\backend> npm start

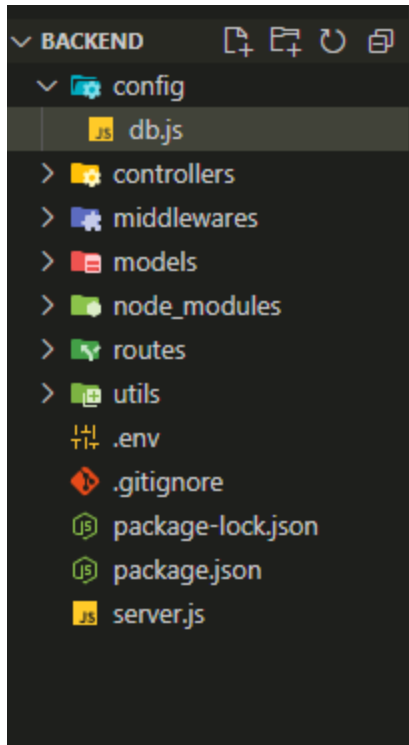
> backend@1.0.0 start
> nodemon server.js

[nodemon] 3.1.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
Server is running on port 5000
```

Good job you have created a server.

STEP 15. Create a mongo db connection.

In the explorer, create a file named “db.js” under the “config” folder.



Click the “db.js”, then add this code block:

```
import mongoose from "mongoose";

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI);
    console.log(`MongoDB is Connected: ${mongoose.connection.host}`);
  } catch (error) {
    console.log(`Error: ${error.message}`);
    process.exit(1);
  }
};

export default connectDB;
```

CRTL + S to save.

STEP 16. Create a mongo db account

Sign in to MongoDB Atlas:

<https://account.mongodb.com/account/login?n=https%3A%2F%2Fcloud.mongodb.com%2Fv2%2F65f6b51c49a07b25297e2fd4&nextHash=%23clusters&signedOut=true>

Use google account to sign in.

After Signing in. Create a new project.

black's Org

Access Manager

Billing

All ClustersGet Helpblack

ORGANIZATION

Projects

Alerts

Activity Feed

Settings

Integrations

Access Manager

Billing

Support

Live Migration

BLACK'S ORG - 2024-11-14

New Project

Find a project...

Project Name	Clusters	Tags	Users	Teams	Alerts	Actions
CRUD	1 Cluster	+ Add Tags	1 User	0 Teams	0 Alerts	...
CRUD-APP	1 Cluster	+ Add Tags	1 User	0 Teams	0 Alerts	...
Project 0	0 Clusters	+ Add Tags	1 User	0 Teams	0 Alerts	...

System Status: All Good Last Login: 158.62.64.110  
©2024 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales

Name the project. Then next and create project.

black's Org

Access Manager

Billing

ORGANIZATION

Projects

Alerts

Activity Feed

Settings

Integrations

Access Manager

Billing

Support

Live Migration

Name Your Project

Add Members

Name Your Project

Project names have to be unique within the organization (and other restrictions).

BASIC-CRUD-APP

Add Tags (Optional)

Use tags to efficiently label and categorize your projects. A project can have a maximum of 50 tags. You can modify tags for the project later. [Learn more](#)

Key	Value	Actions
Select a key or enter your own	: Select a value or enter your own	
+ Add tag		
		0 TAGS

CancelNext

System Status: All Good Last Login: 158.62.64.110  
©2024 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales

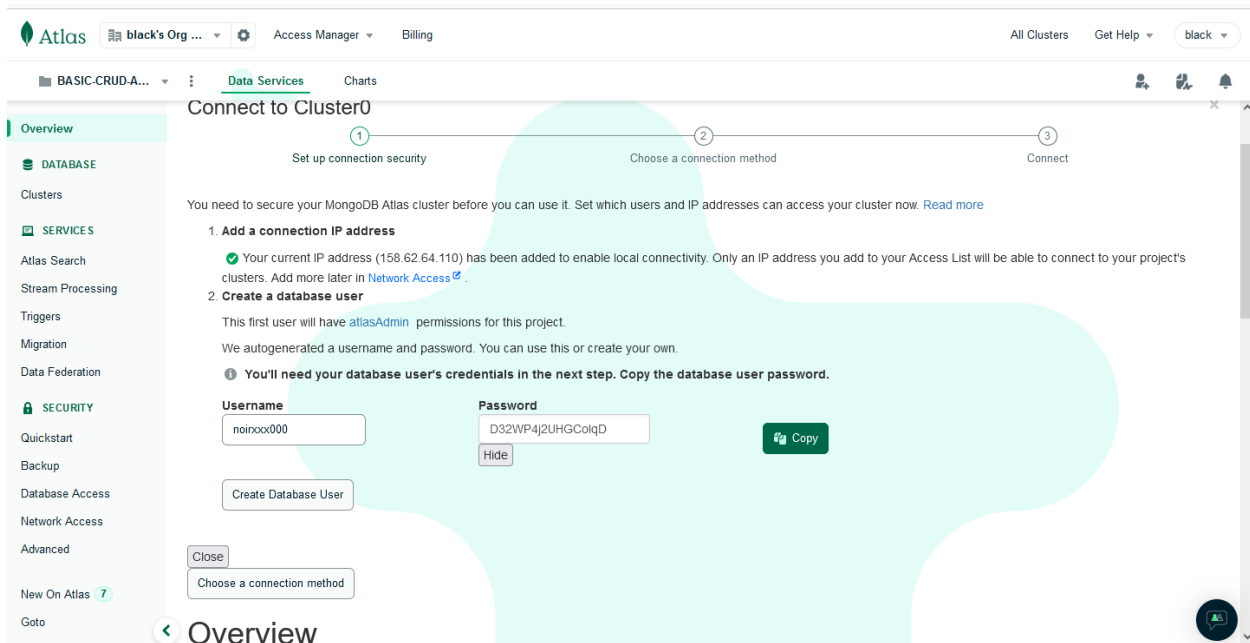
After creating, create a cluster.

The screenshot shows the MongoDB Atlas Overview page. The top navigation bar includes the Atlas logo, the organization name 'black's Org', and links to 'Access Manager' and 'Billing'. The breadcrumb trail indicates the current location: 'black's Org - 2024-11-14 > BASIC-CRUD-APP'. The left sidebar contains a navigation menu with sections for 'Overview', 'DATABASE' (Clusters), 'SERVICES' (Atlas Search, Stream Processing, Triggers, Migration, Data Federation), 'SECURITY' (Backup, Database Access, Network Access, Advanced), and 'New On Atlas'. The main content area is titled 'Overview' and features a large 'Create a cluster' button with a '+ Create' label. Below the button, it says 'Choose your cloud provider, region, and specs.' To the right, there is a 'Toolbar' section with 'Featured Resources' (Get Started with Atlas, Reference MongoDB Documentation, Develop Applications with the Developer Center, Ask the MongoDB Community) and a 'Support Plan' section (You are on the Basic Plan. You can view the Support page to learn more. Got it).

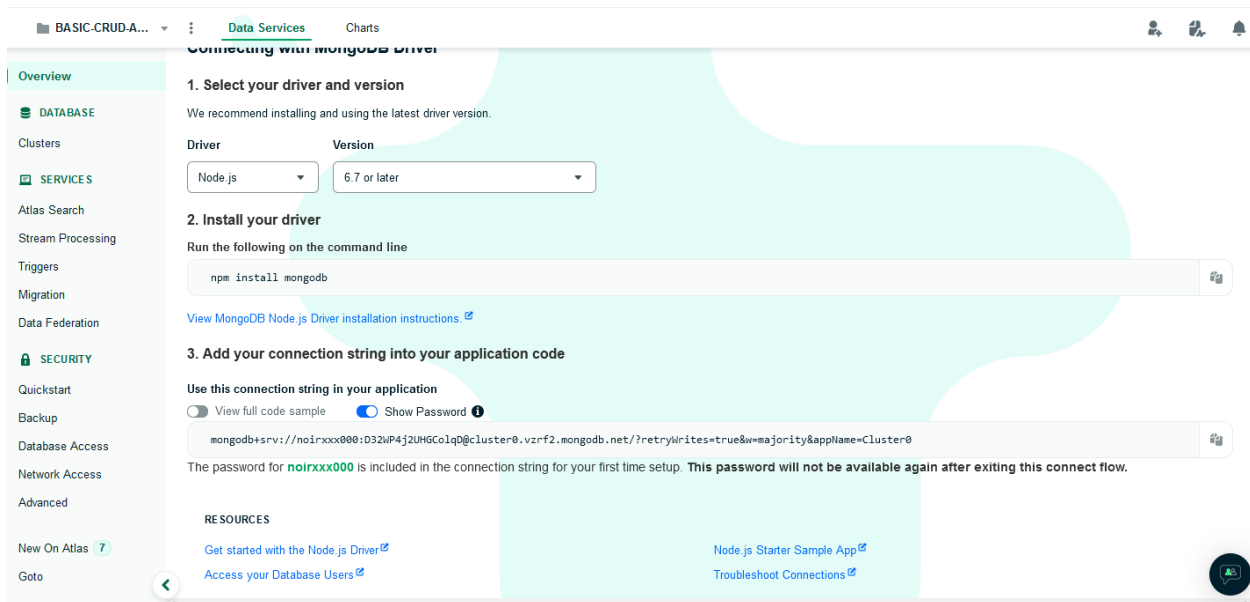
Choose the free tier, then create deployment

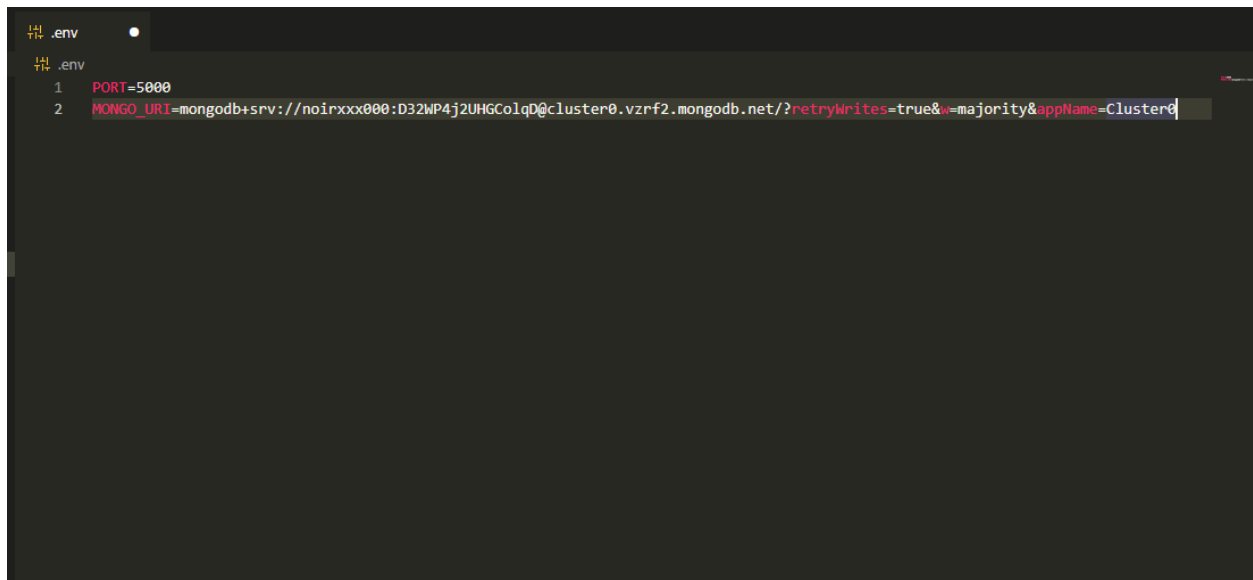
The screenshot shows the 'Deploy your cluster' page. The title is 'Deploy your cluster' and the subtitle is 'Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.' There are three deployment options: 'M10' (\$0.10/hour), 'Flex' (From \$0.01/hour), and 'Free' (Selected). The 'Free' option is described as 'For learning and exploring MongoDB in a cloud environment.' Below the options, there is a green banner that says 'Free forever! Your free cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.' The 'Configurations' section includes a 'Name' field (Cluster0) and a 'Quick setup' section with checkboxes for 'Automate security setup' (checked) and 'Preload sample dataset' (unchecked). At the bottom, there is a 'Provider' section with a button 'I'll do this later' and a 'Create Deployment' button.

Copy the password and paste on a notepad. After that “create database user”



After creating, choose connection, then select “driver”. Copy the connection string then paste it on “.env” file.



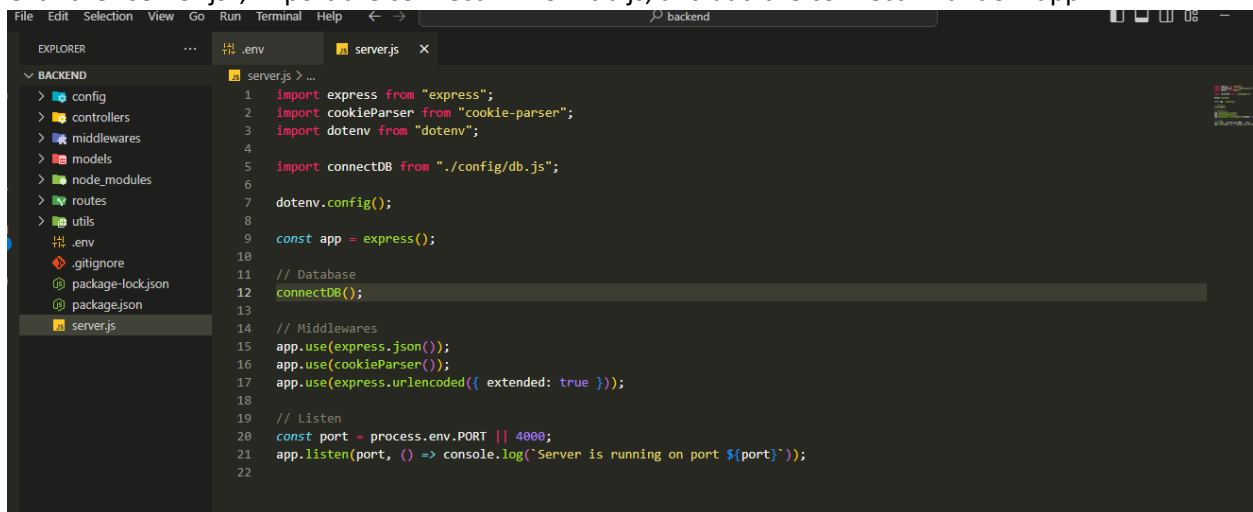


```
.env
1 PORT=5000
2 MONGO_URI=mongodb+srv://noirxxx000:D32WP4j2UHGColqD@cluster0.vzrf2.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0
```

CRTL + S to save

STEP 17.

Click the “server.js”, import the connectDB from db.js, and add the connectDB under “app”



```
server.js
1 import express from "express";
2 import cookieParser from "cookie-parser";
3 import dotenv from "dotenv";
4
5 import connectDB from "./config/db.js";
6
7 dotenv.config();
8
9 const app = express();
10
11 // Database
12 connectDB();
13
14 // Middlewares
15 app.use(express.json());
16 app.use(cookieParser());
17 app.use(express.urlencoded({ extended: true }));
18
19 // Listen
20 const port = process.env.PORT || 4000;
21 app.listen(port, () => console.log(`Server is running on port ${port}`));
22
```

CRTL + S to save. Then restart the server. It should display like this

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  node

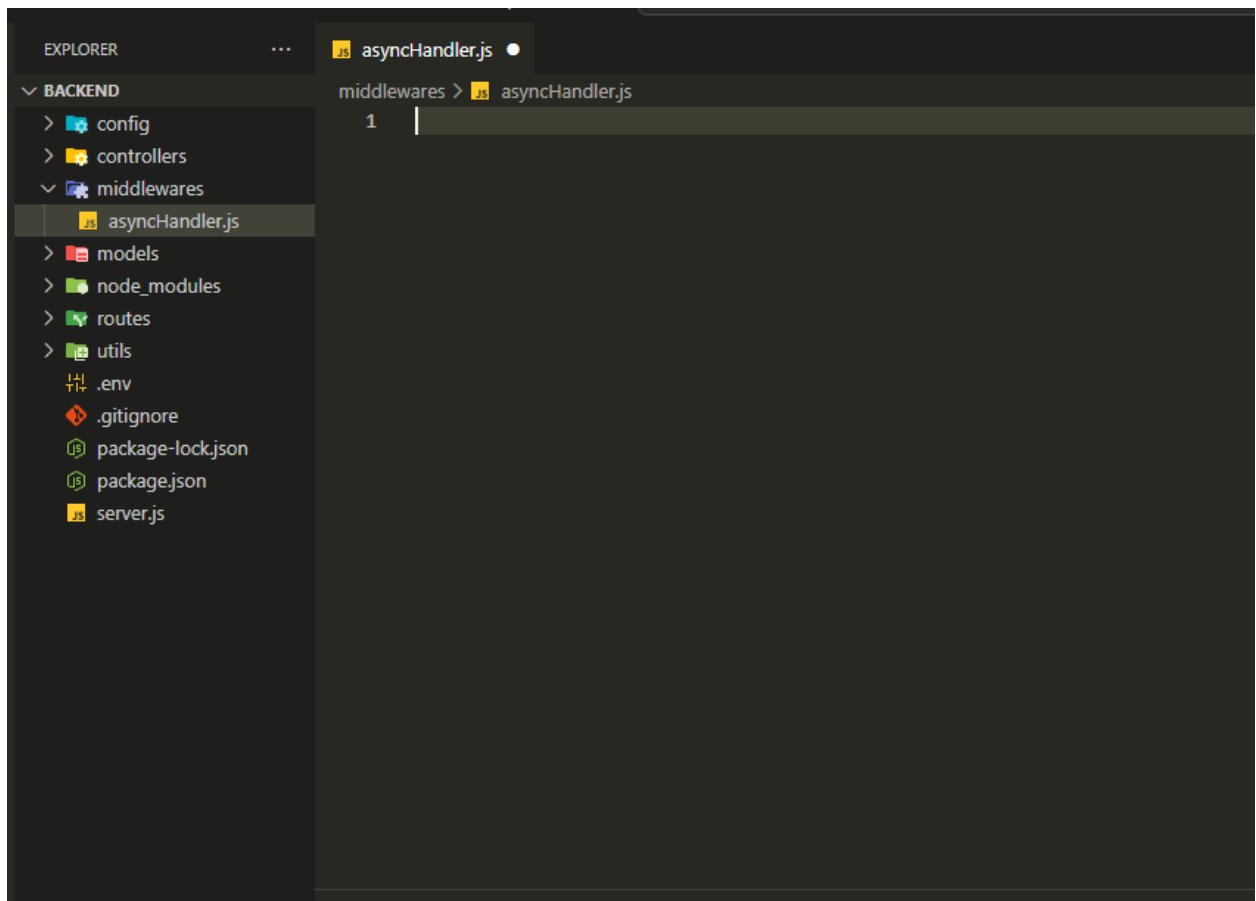
Terminate batch job (Y/N)? y
PS C:\Users\noir\Desktop\CRUD Workshop - IT\backend> npm start

> backend@1.0.0 start
> nodemon server.js

[nodemon] 3.1.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
Server is running on port 5000
MongoDB is Connected: cluster0-shard-00-00.vzrf2.mongodb.net
```

STEP 19.

Create a file named “asyncHandler.js” under the “middlewares” folder



```
EXPLORER  ...  asyncHandler.js

▼ BACKEND
  > config
  > controllers
  ▼ middlewares
    asyncHandler.js
  > models
  > node_modules
  > routes
  > utils
  .env
  .gitignore
  package-lock.json
  package.json
  server.js

middlewares > asyncHandler.js
1 |
```

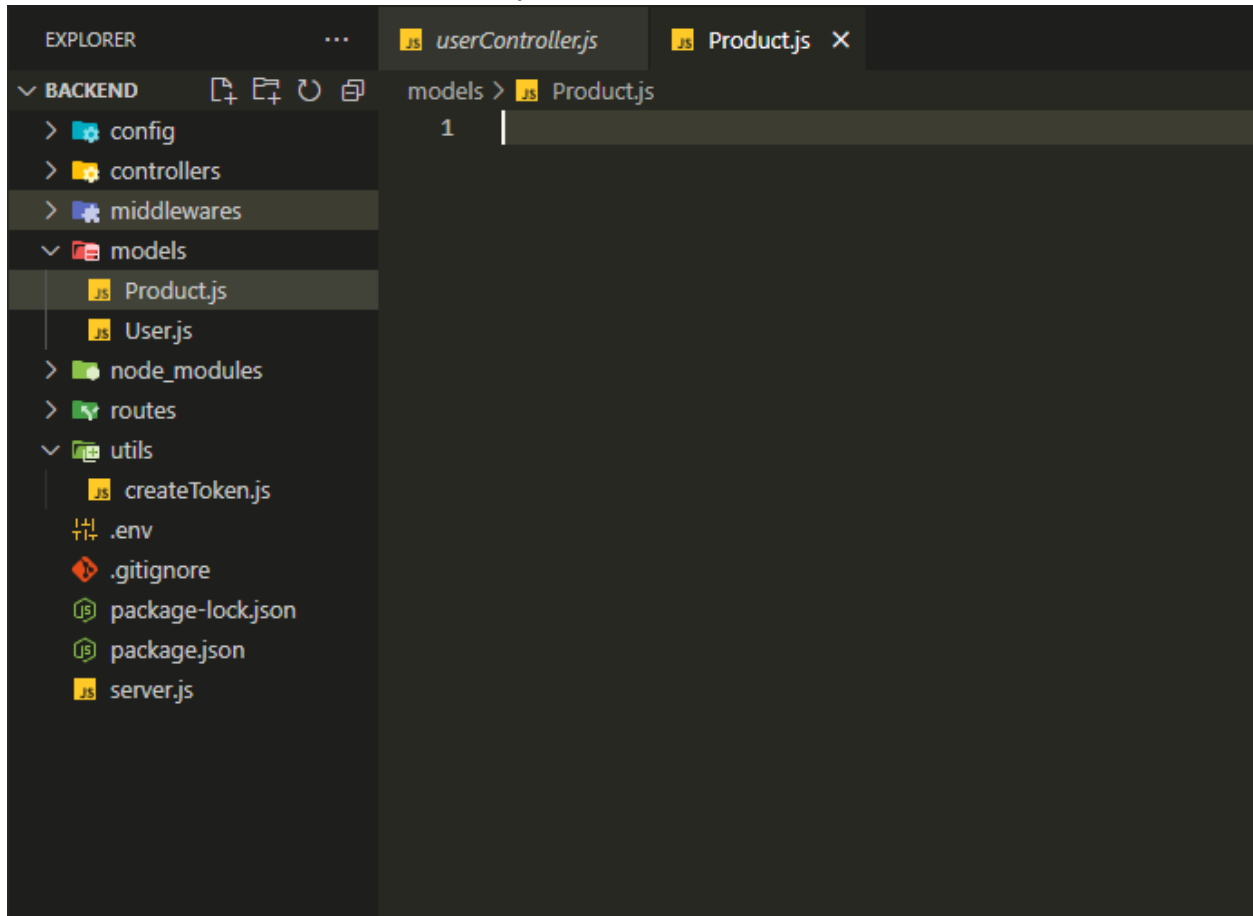
Add this code block:

```
const asyncHandler = (fn) => (req, res, next) => {
  Promise.resolve(fn(req, res, next)).catch((err) => {
    res.status(500).json({ message: err.message });
  });
};
```

```
export default asyncHandler;
```

STEP 26:

In the editor, create a file named “Product.js” under the models folder.



Add this code block:

```
import mongoose from "mongoose";

const productSchema = new mongoose.Schema(
  {
    productName: {
      type: String,
      required: true,
    },
    price: {
      type: Number,
      required: true,
    },
  }
);
```

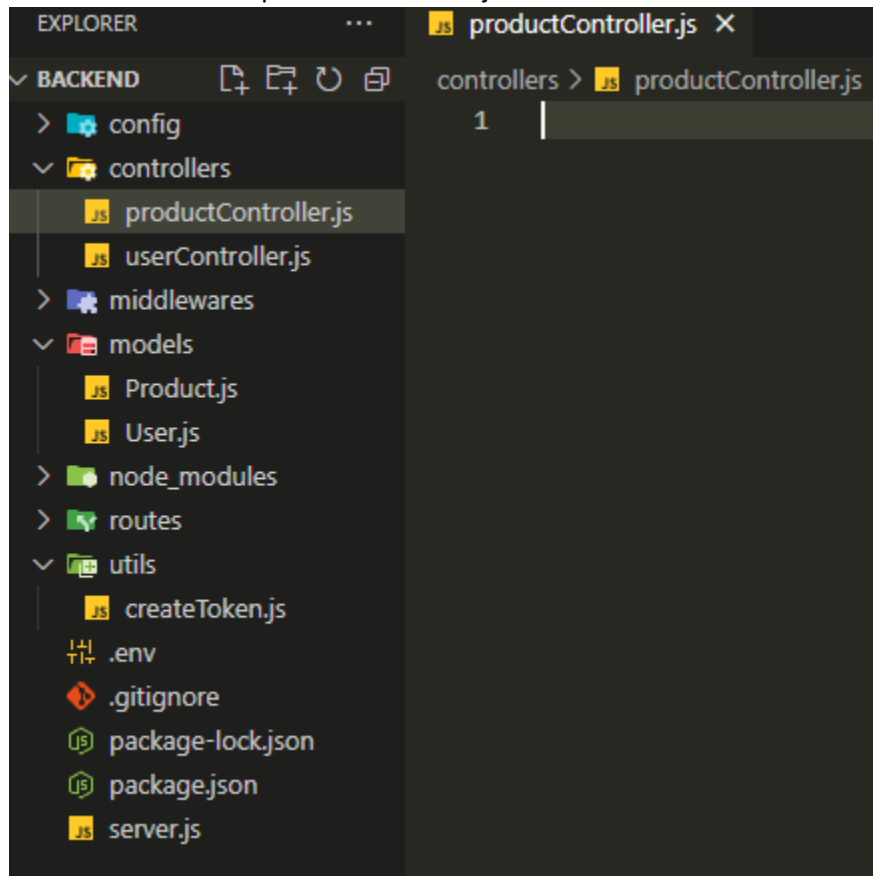


```
    min: 0,
  },
  description: {
    type: String,
    required: true,
    trim: true,
  },
  image: {
    type: String,
    required: true,
  },
},
{
  timestamps: true,
}
);

const Product = mongoose.model("Product", productSchema);
export default Product;
```

STEP 27:

Create a file named “productController.js” under the “controllers” folder.



Add this code block:

```
import Product from "../models/Product.js";
import asyncHandler from "../middlewares/asyncHandler.js";

export const createProduct = asyncHandler(async (req, res) => {
  try {
    const { productName, price, description } = req.body;
    const image = req.file ? req.file.path : null; // Get the image path from
    multer

    const newProduct = new Product({
      productName,
      price,
      description,
      image, // Save the image path to the product model
    });

    const savedProduct = await newProduct.save();
    res.status(201).json({
      message: "Product created successfully",
```

```

        savedProduct,
    });
} catch (error) {
    console.log(error);
    res.status(400).json({ message: "Error creating a product", error });
}
});

export const updateProduct = asyncHandler(async (req, res) => {
    try {
        const { productId } = req.params;
        const updates = req.body;

        if (req.file) {
            updates.image = req.file.path; // Add the new image path if an image is
            uploaded
        }

        const product = await Product.findByIdAndUpdate(productId, updates, {
            new: true,
        });

        if (!product) {
            return res.status(400).json({ message: "Product not found" });
        }

        res.status(200).json(product);
    } catch (error) {
        res.status(400).json({ message: "Error updating product", error });
    }
});

export const getAllProducts = asyncHandler(async (req, res) => {
    try {
        const products = await Product.find();
        res.status(200).json(products);
    } catch (error) {
        res.status(400).json({ message: "Error fetching products", error });
    }
});

export const getProductById = asyncHandler(async (req, res) => {
    try {
        const { productId } = req.params;
        const product = await Product.findById(productId);

```

```

    if (!product) {
      return res.status(400).json({ message: "Product not found" });
    }

    res.status(200).json(product);
  } catch (error) {
    res.status(400).json({ message: "Error fetching product", error });
  }
});

export const deleteProduct = asyncHandler(async (req, res) => {
  try {
    const { productId } = req.params;
    const result = await Product.deleteOne({ _id: productId });

    if (result.deletedCount == 0) {
      return res.status(400).json({ message: "Product not found" });
    }

    res.status(200).json({ message: "Product deleted successfully" });
  } catch (error) {
    res.status(400).json({ message: "Error deleting product", error });
  }
});

```

STEP 28: create a file name "upload.js" under the middlewares folder

Add this code block:

```

import multer from "multer";
import path from "path";

// Define where the images will be stored and their file naming convention
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, "uploads/"); // Specify the 'uploads' folder for storing images
  },
  filename: (req, file, cb) => {
    cb(null, `${Date.now()}_${file.originalname}`); // Use timestamp to prevent
    filename collisions
  },
});

```

```
// File filter for image files only
const fileFilter = (req, file, cb) => {
  const allowedTypes = /jpeg|jpg|png|gif/;
  const extname = allowedTypes.test(
    path.extname(file.originalname).toLowerCase()
  );
  const mimetype = allowedTypes.test(file.mimetype);

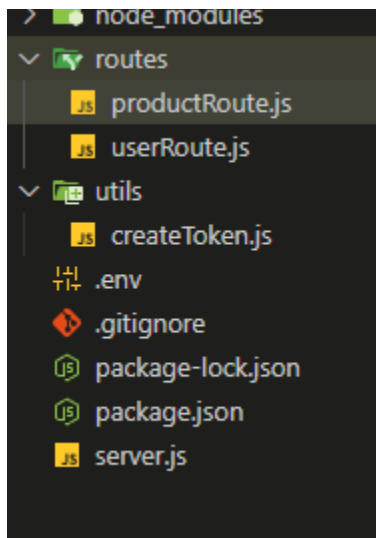
  if (extname && mimetype) {
    return cb(null, true);
  } else {
    cb(new Error("Only image files are allowed"), false);
  }
};

const upload = multer({
  storage: storage,
  fileFilter: fileFilter,
});

export default upload;
```

STEP 29:

Create a file named “productRoute.js” under the “routes” folder.



Add this code block:

```
import express from "express";
```

```

import {
  createProduct,
  getAllProducts,
  getProductById,
  updateProduct,
  deleteProduct,
} from "../controllers/productController.js";
import upload from "../middlewares/upload.js"; // Import multer configuration

const router = express.Router();

// Route to create a new product with image upload
router.route("/").post(upload.single("image"), createProduct);

// Route to fetch all products
router.route("/").get(getAllProducts);

// Route to fetch, update or delete a single product by ID
router
  .route("/:productId")
  .get(getProductById)
  .put(upload.single("image"), updateProduct) // Handle image upload during
update
  .delete(deleteProduct);

export default router;

```

STEP 30:

Click the "server.js", and then add this statement

```

import path from "path";
import { fileURLToPath } from "url";

import productRoutes from "../routes/productRoute.js"

```

```

app.use("/api/products", productRoutes);

// Serve images from the "uploads" folder
const __filename = fileURLToPath(import.meta.url); // Get the current file URL
const __dirname = path.dirname(__filename); // Get the directory name
app.use("/uploads", express.static(path.join(__dirname, "uploads")));

```

```

server.js > ...
1  import express from "express";
2  import dotenv from "dotenv";
3  import path from "path";
4  import { fileURLToPath } from "url"; // Import fileURLToPath for handling URL paths
5
6  import connectDB from "../config/db.js";
7  import productRoutes from "../routes/productRoute.js";
8
9  dotenv.config();
10
11 const app = express();
12
13 // Database
14 connectDB();
15
16 // Middlewares
17 app.use(express.json());
18 app.use(express.urlencoded({ extended: true }));
19
20 // APIs
21 app.use("/api/products", productRoutes);
22
23 // Serve images from the "uploads" folder
24 const __filename = fileURLToPath(import.meta.url); // Get the current file URL
25 const __dirname = path.dirname(__filename); // Get the directory name
26 app.use("/uploads", express.static(path.join(__dirname, "uploads")));
27
28 // Listen
29 const port = process.env.PORT || 5000;
30 app.listen(port, () => console.log(`Server is running on port ${port}`));
31

```

STEP 31:

Open Postman, and TEST the APIs

“/create”

POST localhost:5000/api/prod

POST localhost:4000/api/user/

+

...

HTTP

localhost:5000/api/products/create

POST

localhost:5000/api/products/create

Params

Authorization

Headers (9)

Body ●

Pre-request Script

Tests

Settings

● none

● form-data

● x-www-form-urlencoded

● raw

● binary

JSON ▼

```
1 {
2   "productName": "Ipad Pro Air",
3   "price": "80000",
4   "description": "Best Ipad of the year"
5 }
```

Body

Cookies (1)

Headers (7)

Test Results

St

Pretty

Raw

Preview

Visualize

JSON ▼

≡

```
1 {
2   "message": "Product created successfully",
3   "savedProduct": {
4     "productName": "Ipad Pro Air",
5     "price": 80000,
6     "description": "Best Ipad of the year",
7     "user": "6738c7243b4106c362068b00",
8     "_id": "673939c0bdf9962f3f6a3f2b",
9     "createdAt": "2024-11-17T00:33:04.224Z",
10    "updatedAt": "2024-11-17T00:33:04.224Z",
11    "__v": 0
12  }
13 }
```

Get all products - "/"



localhost:5000/api/products/

GET localhost:5000/api/products/

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

```
1 {
2   "productName": "Ipad Pro Air",
3   "price": "80000",
4   "description": "Best Ipad of the year"
5 }
```

Body Cookies (1) Headers (7) Test Results

Status: 200 OK Time: 174 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "_id": "6739393bbdf9962f3f6a3f27",
4     "productName": "Ipad Pro Max",
5     "price": 90000,
6     "description": "Best Ipad of the year",
7     "user": "6738c7243b4106c362068b00",
8     "createdAt": "2024-11-17T00:30:51.694Z",
9     "updatedAt": "2024-11-17T00:30:51.694Z",
10    "__v": 0
11  },
12  {
13    "_id": "673939c0bdf9962f3f6a3f2b",
```

Get product by ID – “/:productId”

GET localhost:5000/api/products/6739393bbdf9962f3f6a3f27

GET localhost:5000/api/products/6739393bbdf9962f3f6a3f27

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● raw ● binary JSON ▾

```
1 {
2   "productName": "Ipad Pro Air",
3   "price": "80000",
4   "description": "Best Ipad of the year"
5 }
```

Body Cookies (1) Headers (7) Test Results

Pretty Raw Preview Visualize JSON ▾

```
1 {
2   "_id": "6739393bbdf9962f3f6a3f27",
3   "productName": "Ipad Pro Max",
4   "price": 90000,
5   "description": "Best Ipad of the year",
6   "user": "6738c7243b4106c362068b00",
7   "createdAt": "2024-11-17T00:30:51.694Z",
8   "updatedAt": "2024-11-17T00:30:51.694Z",
9   "__v": 0
10 }
```

Update product by Id – “/productId”

localhost:5000/api/products/6739393bbdf9962f3f6a3f27

PUT localhost:5000/api/products/6739393bbdf9962f3f6a3f27

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

```
1 {
2   "productName": "Samsung S2",
3   "price": "80000",
4   "description": "Best phone of the year"
5 }
```

Body Cookies (1) Headers (7) Test Results

Status: 200 OK Time: 184 m

Pretty Raw Preview Visualize JSON

```
1 {
2   "_id": "6739393bbdf9962f3f6a3f27",
3   "productName": "Samsung S2",
4   "price": 80000,
5   "description": "Best phone of the year",
6   "user": "6738c7243b4106c362068b00",
7   "createdAt": "2024-11-17T00:30:51.694Z",
8   "updatedAt": "2024-11-17T00:36:12.246Z",
9   "__v": 0
10 }
```

Delete product by Id – “/productId”

localhost:5000/api/products/6739393bbdf9962f3f6a3f27

DELETE localhost:5000/api/products/6739393bbdf9962f3f6a3f27

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

```
1 {
2   "productName": "Samsung S2",
3   "price": "80000",
4   "description": "Best phone of the year"
5 }
```

Body Cookies (1) Headers (7) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Product deleted successfully"
3 }
```

CONGRATULATIONS YOU HAVE FINISHED THE BACKEND  
PART. !!! JAORY PA FRONTEND