**Lesson 06: Analyzing and Migrating an Existing Project for Improving Efficiency Using Generative AI**

**Overview:**

This project aims to analyze, improve, and migrate an existing retail management system's codebase to enhance its efficiency using generative AI. The task involves migrating from CherryPy to Node JS Express Framework and leveraging GitHub Copilot to streamline the transition. Furthermore, a comprehensive analysis of time and space complexity will identify opportunities for optimization. Ultimately, the project aims to optimize functionality, security, and compatibility within the retail management system, ensuring an enhanced user experience.

**Instructions:**

1. Analyze the existing codebase to identify areas for improvement, focusing on code coverage and performance
2. Migrate the current framework from CherryPy to Express Framework, utilizing GitHub Copilot for guidance and code generation
3. Test the migrated applications to ensure they function correctly on the new framework
4. Optimize the code for better performance and readability by using suggestions from GitHub Copilot

**Tasks:**

1. Mongo DB set up
2. Analyzing and executing the existing codebase
   1. Analyze and set up the existing codebase to test its functionality
3. Evaluate the best project packages using Snyk Advisor
   1. Evaluate and identify the best packages in the code using Snyk Advisor
4. Migrate from CherryPy to Express Node JS Framework using GitHub Copilot
   1. Migrate the CherryPy application to Express Node JS Framework using GitHub Copilot
5. Analyze the time and space complexity of the migrated code using GitHub Copilot
6. Analyze the time and space complexity of the migrated application using GitHub Copilot to enhance efficiency

**Tools required:**

1. Visual Studio Code
2. GitHub Copilot
3. Python
4. Node JS
5. Mongo DB

**Guided Practice Solution**

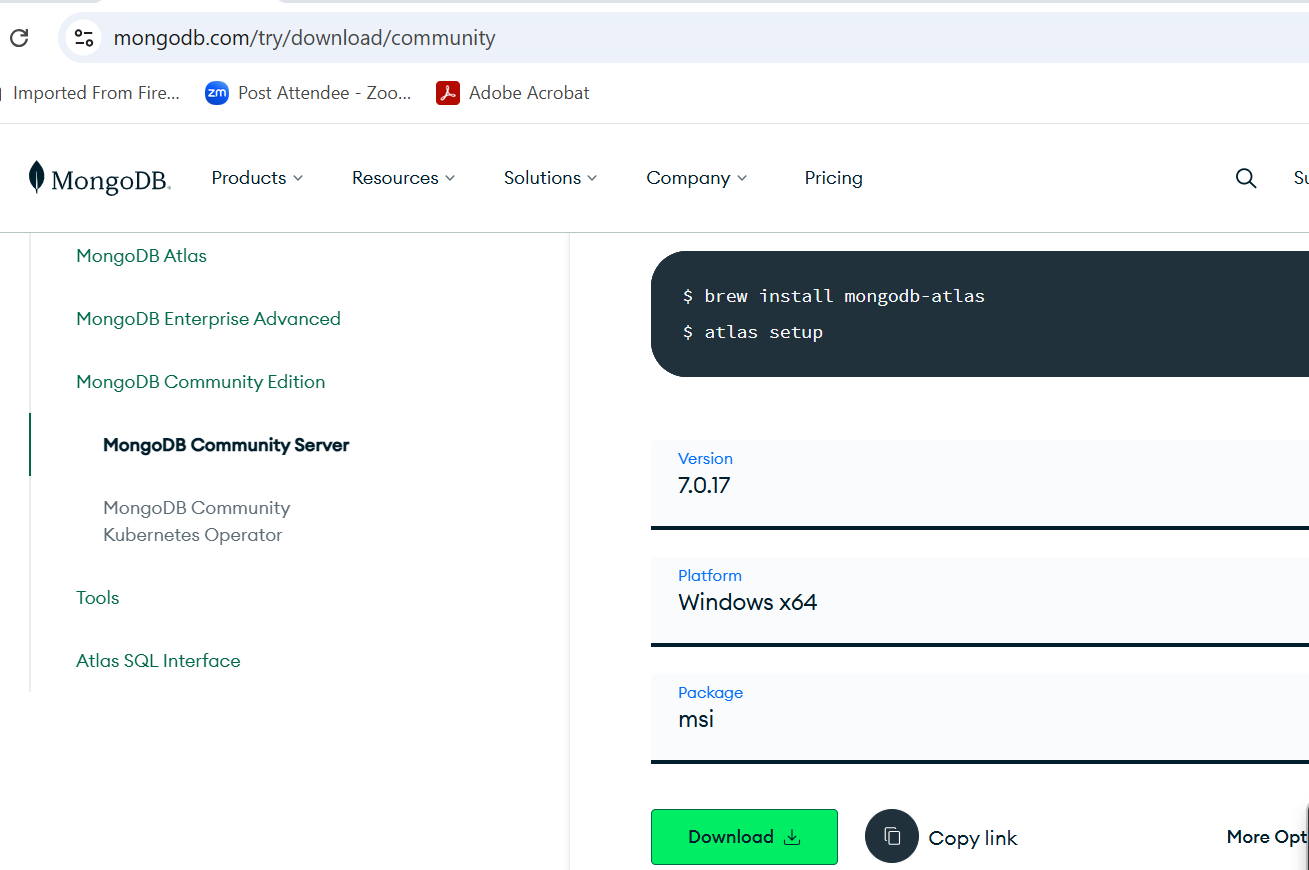
**Disclaimer:** Please note that all the generative AI tools used in this exercise can produce varied outputs even when presented with similar prompts. Thus, you may get different output for the same prompt.

**Task 1: Mongo DB set up**

1.1 **MongoDB 7.x Setup**

1.2 Download MongoDB – Get the MSI installer from

[**https://www.mongodb.com/try/download/community**](https://www.mongodb.com/try/download/community)



1.3 Install MongoDB – Run the MSI installer and select the Complete installation other OS user please download the mongo DB with respective OS.

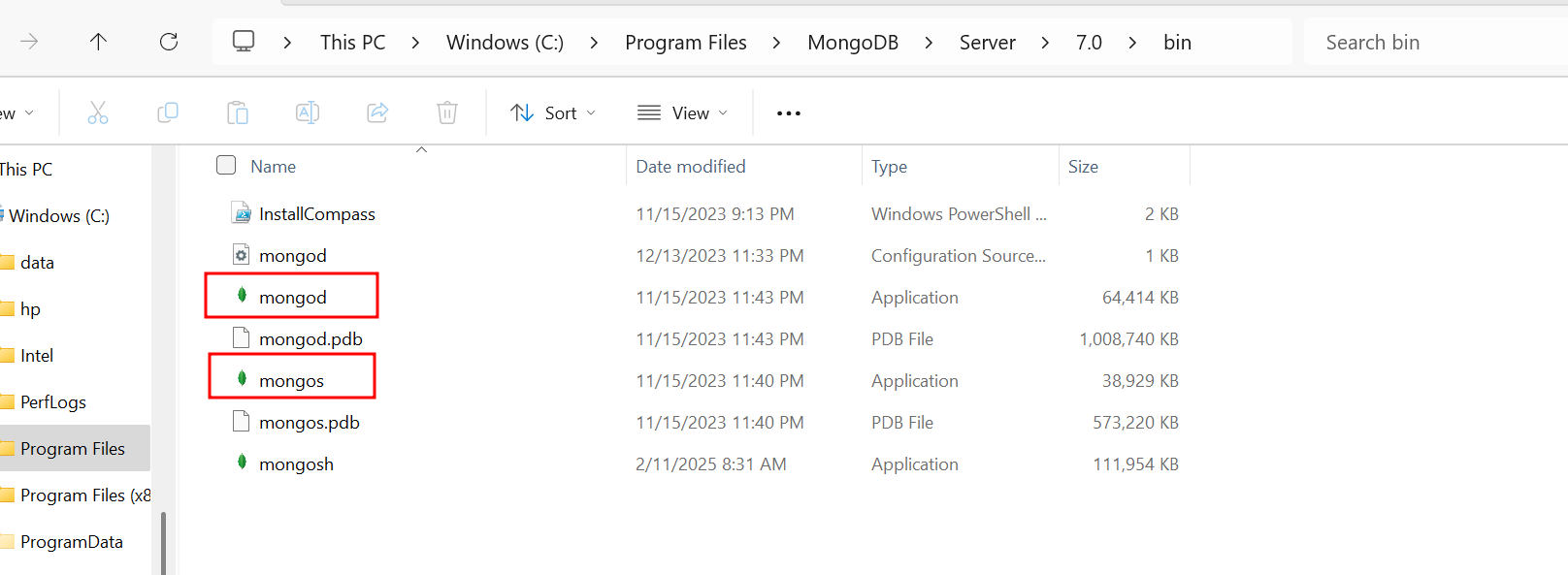
1.4 Configure MongoDB – Ensure "Run MongoDB as a Service" is checked.

1.5 Start MongoDB – Open Command Prompt, navigate to MongoDB's bin folder, and start the service.

1.6 download mongo shell using below command

<https://www.mongodb.com/try/download/shell>

and extract and copy pasts mongos inside bin folder



1.7 Before run the client terminal you need to create the folder as inside C drive

**mkdir C:\data\db**

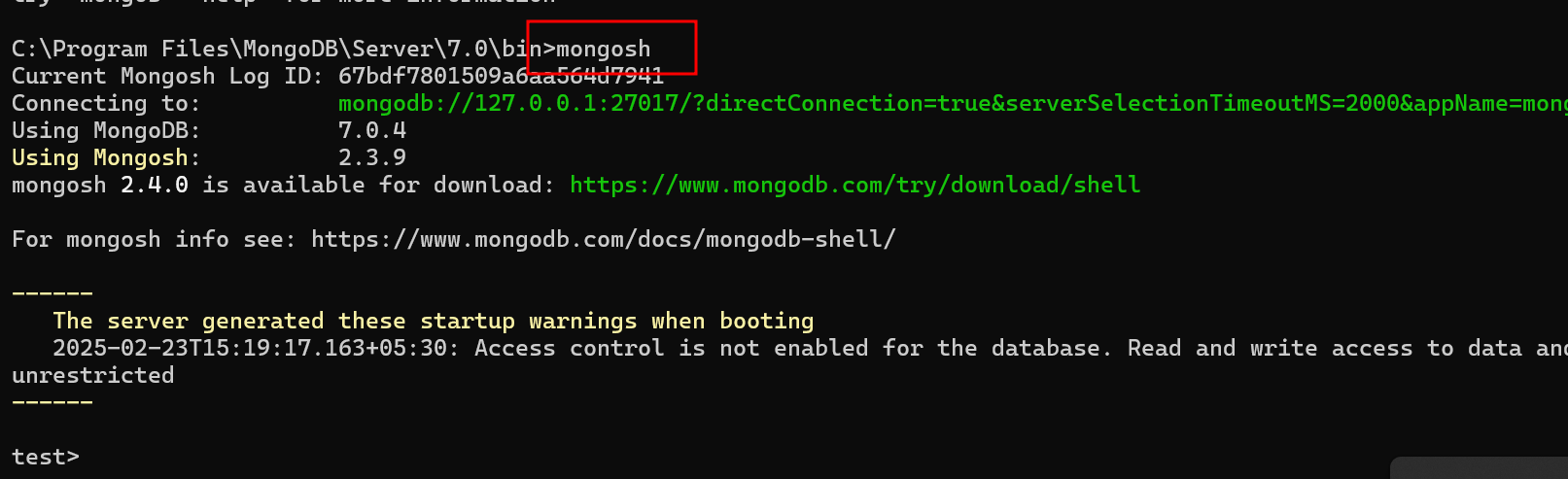
**means create a folder as data and inside data create the folder as db**

1.8 macOS & Linux:

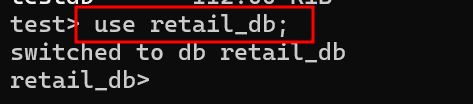
If starting MongoDB manually, you may need to create the folders and set permissions

sudo mkdir -p /data/db sudo chown -R $(whoami) /data/db

To run the mongo client terminal run the command as



**Below command is use to create the database.**



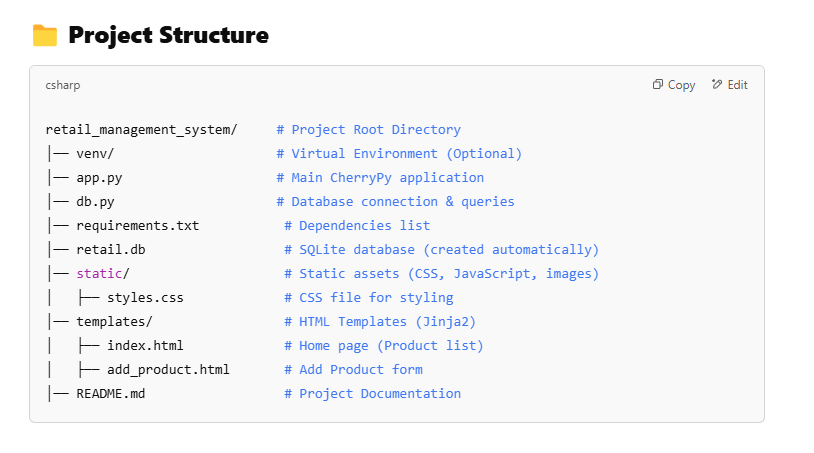
**Insert sample data in mongo db**

**db.products.insertMany([ { "name": "Laptop", "price": 1000, "stock": 10 }, { "name": "Smartphone", "price": 500, "stock": 20 } ])**

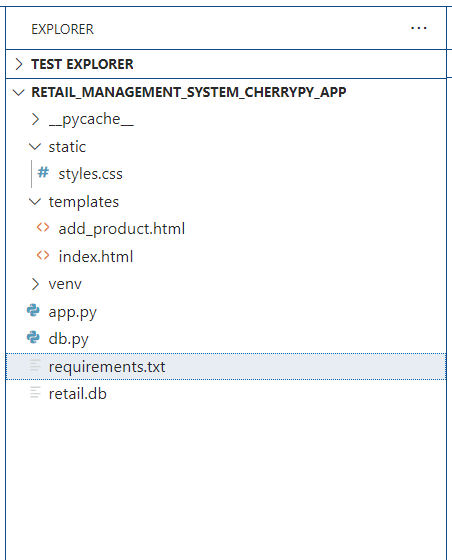
**Task 1: Analyzing and executing the existing codebase**

**Step 1: Analyze and set up the existing codebase to test its functionality**

* 1. retail\_management\_system\_cherrypy\_app



* 1. Download, extract, and save the project in the local system. Next, open this project in VSCode as shown in the screenshot below:



The code for all these modules is as follows:

**db.py**

from pymongo import MongoClient

class Database:

    def \_\_init\_\_(self):

        self.client = MongoClient("mongodb://localhost:27017/")

        self.db = self.client["retail\_db"]

        self.products = self.db["products"]

    def get\_products(self):

        return list(self.products.find({}, {"\_id": 0}))

    def add\_product(self, name, price, stock):

        self.products.insert\_one({"name": name, "price": float(price), "stock": int(stock)})

    def delete\_product(self, name):

        self.products.delete\_one({"name": name})

**app.py**

import cherrypy

from jinja2 import Environment, FileSystemLoader

from db import Database

db = Database()

env = Environment(loader=FileSystemLoader('templates'))

class RetailApp:

    @cherrypy.expose

    def index(self):

        products = db.get\_products()

        template = env.get\_template("index.html")

        return template.render(products=products)

    @cherrypy.expose

    def add\_product(self, name=None, price=None, stock=None):

        if name and price and stock:

            db.add\_product(name, float(price), int(stock))

            raise cherrypy.HTTPRedirect("/")

        template = env.get\_template("add\_product.html")

        return template.render()

    @cherrypy.expose

    def delete\_product(self, name):

        db.delete\_product(name)

        raise cherrypy.HTTPRedirect("/")

if \_\_name\_\_ == "\_\_main\_\_":

    config = {

        '/static': {

            'tools.staticdir.on': True,

            'tools.staticdir.dir': "static"

        }

    }

    cherrypy.quickstart(RetailApp(), "/", config)

**index.html** file code part of templates folder

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Retail Management System</title>

    <link rel="stylesheet" href="/static/styles.css">

</head>

<body>

    <h1>Retail Management System</h1>

    <a href="/add\_product">Add Product</a>

    <table>

        <tr>

            <th>Name</th>

            <th>Price</th>

            <th>Stock</th>

            <th>Action</th>

        </tr>

        {% for product in products %}

        <tr>

            <td>{{ product.name }}</td>

            <td>${{ product.price }}</td>

            <td>{{ product.stock }}</td>

            <td><a href="/delete\_product?name={{ product.name }}">Delete</a></td>

        </tr>

        {% endfor %}

    </table>

</body>

</html>

**add\_product.html** file code part of templates folder

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Add Product</title>

    <link rel="stylesheet" href="/static/styles.css">

</head>

<body>

    <h1>Add Product</h1>

    <form method="post">

        <label>Name:</label>

        <input type="text" name="name" required><br>

        <label>Price:</label>

        <input type="number" step="0.01" name="price" required><br>

        <label>Stock:</label>

        <input type="number" name="stock" required><br>

        <button type="submit">Add</button>

    </form>

    <a href="/">Back to Home</a>

</body>

</html>

**styles.css** file coder part of static folder

body {

    font-family: Arial, sans-serif;

    text-align: center;

}

table {

    width: 80%;

    margin: 20px auto;

    border-collapse: collapse;

}

th, td {

    border: 1px solid black;

    padding: 10px;

}

a {

    display: inline-block;

    margin: 10px;

    padding: 10px;

    background: blue;

    color: white;

    text-decoration: none;

}

button {

    padding: 10px;

    background: green;

    color: white;

    border: none;

    cursor: pointer;

}

**requirement.txt** file which contains all dependencies

cherrypy

pymongo

jinja2

* 1. Go to VSCode, open the CherryPy project, and execute the main.py file using the following command in the terminal:

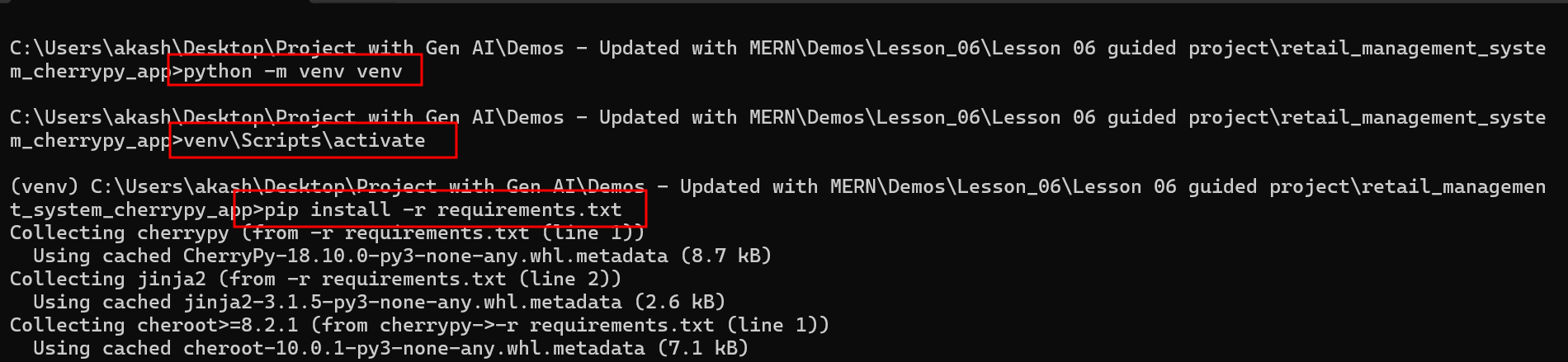
**Set up a virtual environment**

python -m venv venv

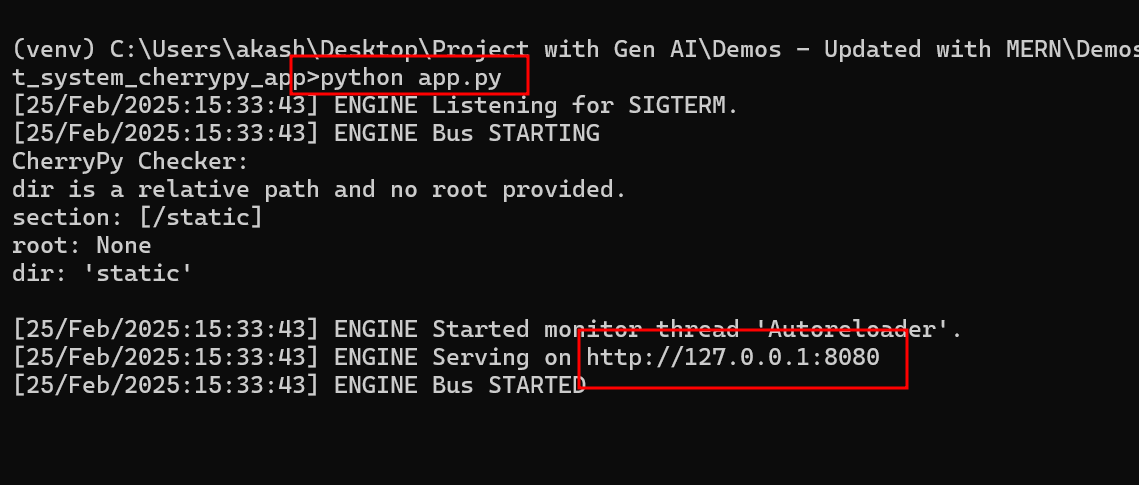
source venv/bin/activate # For macOS/Linux

venv\Scripts\activate # For Windows

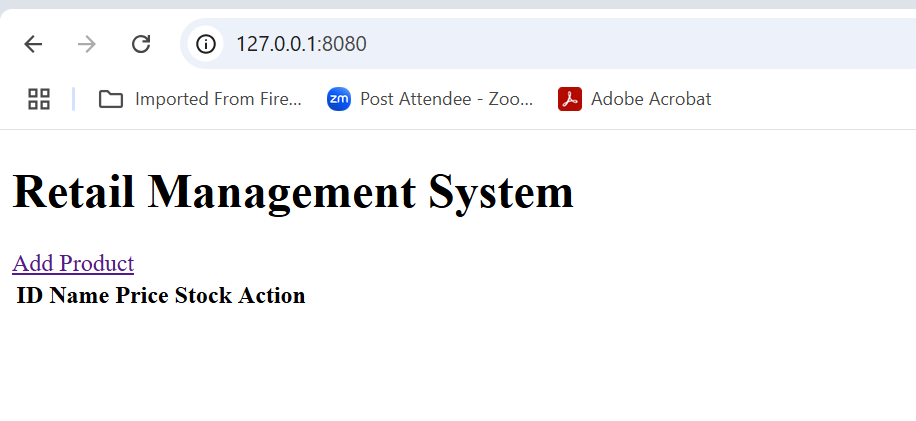
**pip install -r requirements.txt**



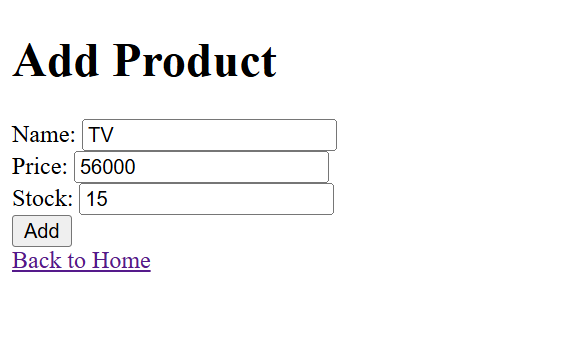
**Run the app**

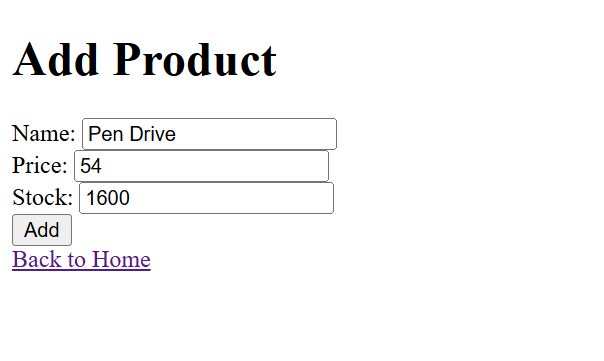


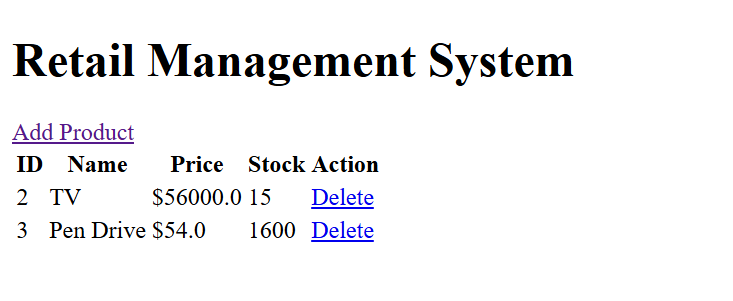
**Now open the application on browser**



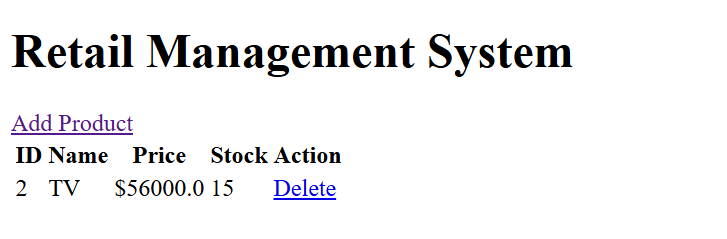
**Add the product**







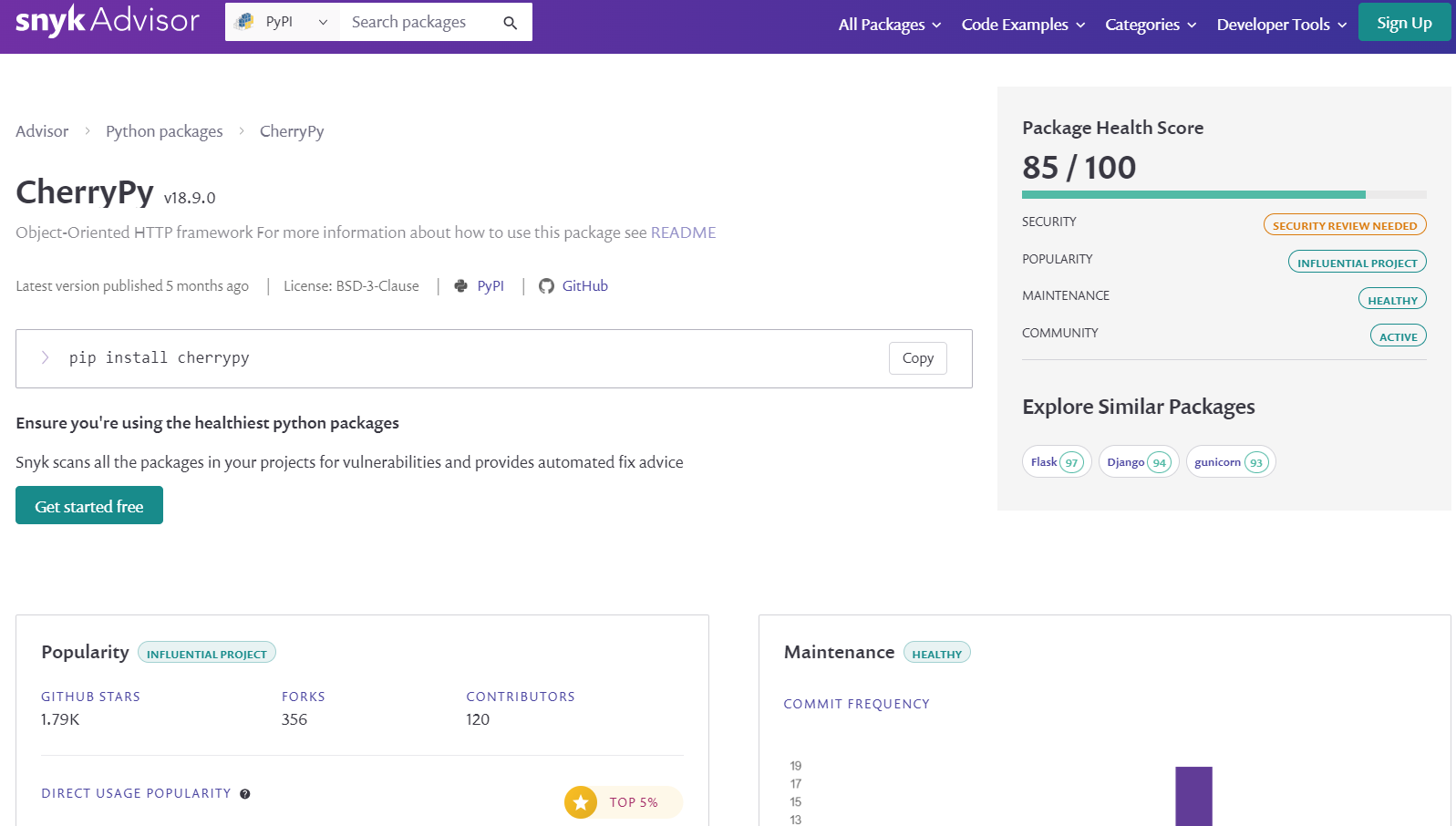
**Delete the product details**



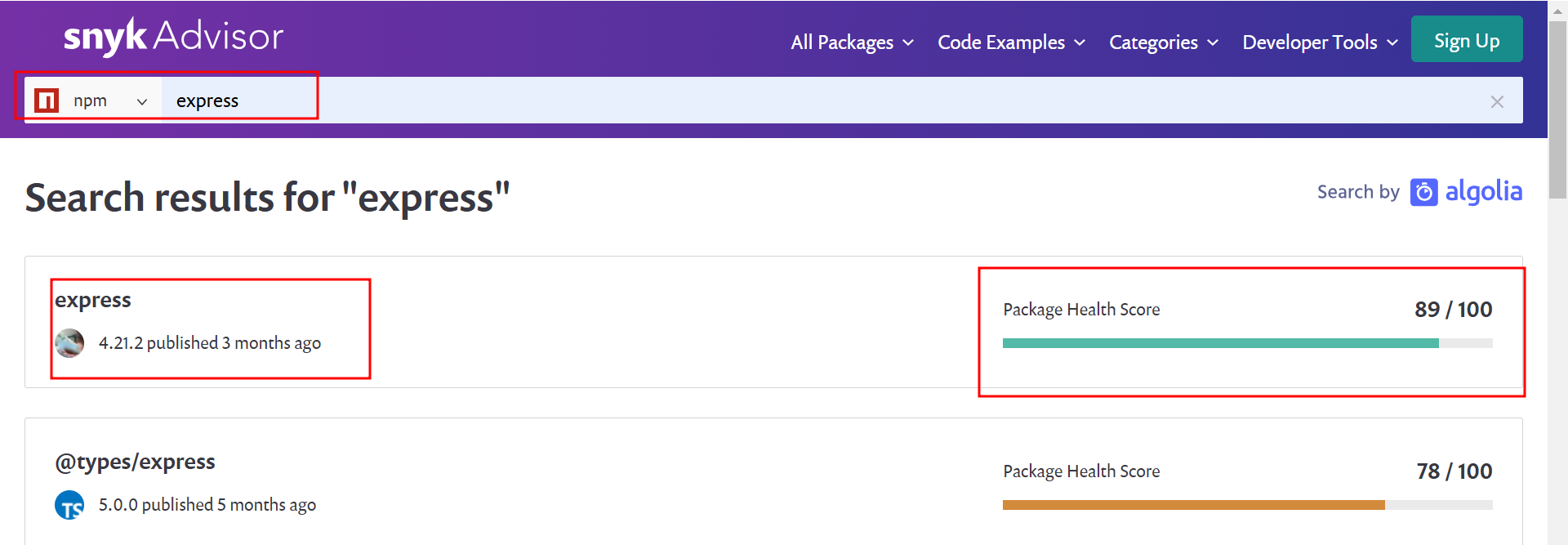
**Task 2: Evaluate the best project packages using Snyk Advisor**

**Step 1: Evaluate and identify the best packages in the code using Snyk Advisor**

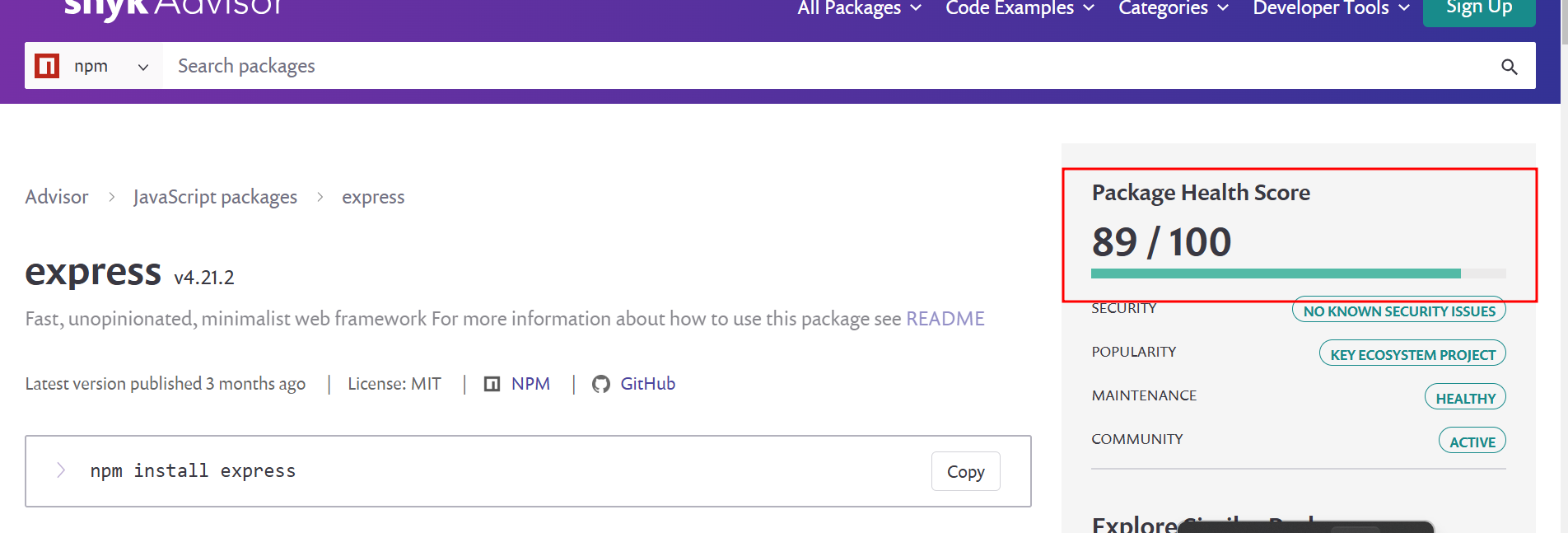
* 1. Snyk Advisor is a tool that helps developers assess the security, popularity, and maintenance of open-source packages before using them. It provides insights into potential vulnerabilities, outdated dependencies, licensing issues, and general package health. The tool is particularly useful for developers who want to ensure the safety and reliability of their projects when integrating third-party libraries from ecosystems like **npm, PyPI, Maven, Go, and more**.
  2. Navigate to the **Snyk Advisor** page for **CherryPy** using the following URL and evaluate the package for vulnerabilities and security issues:  
     **https://snyk.io/advisor/python/cherrypy**

  
  
The information on **CherryPy** shows that there are security issues, and its security health score is 85.

* 1. Search for other packages with better security health scores. Add **npm with express** in the search bar, as shown in the screenshot below:



* 1. Click on **Express**, as shown in the screenshot below:



The package health score for Express is high and secure.

**Note:** You may explore other packages by adding the desired package name.

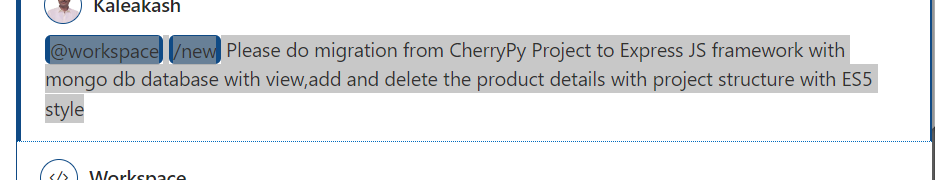
It shows that all the scores are green and secure and therefore, Express for the migration from CherryPy.

**Task 3: Migrate from CherryPy to Express framework using GitHub Copilot**

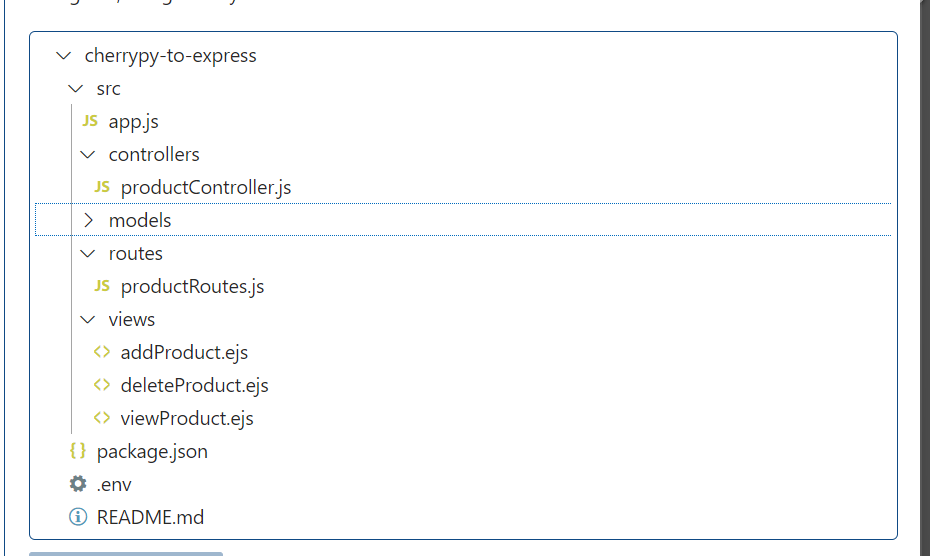
**Step 1: Migrate the CherryPy to Express using GitHub Copilot**

* 1. Open the **CherryPy** project in Visual Studio Code, navigate to the GitHub Copilot chat, and enter the following prompt to generate a Flask workspace similar to the CherryPy project:

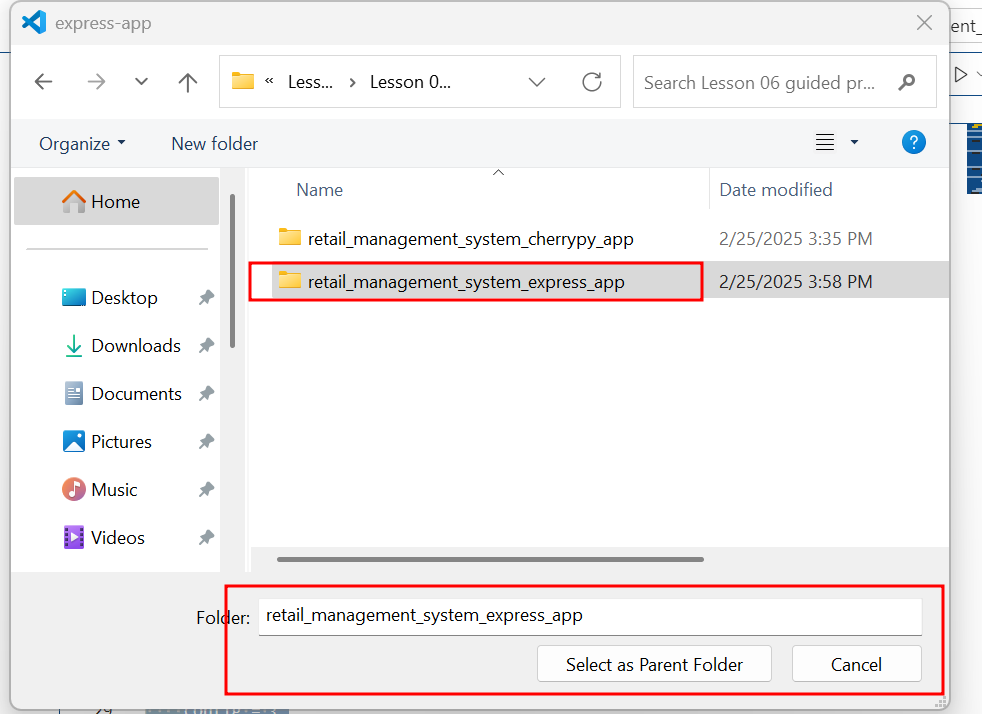
**@workspace /new Please do migration from CherryPy Project to Express JS framework with mongo db database with view,add and delete the product details with project structure with ES5 style**



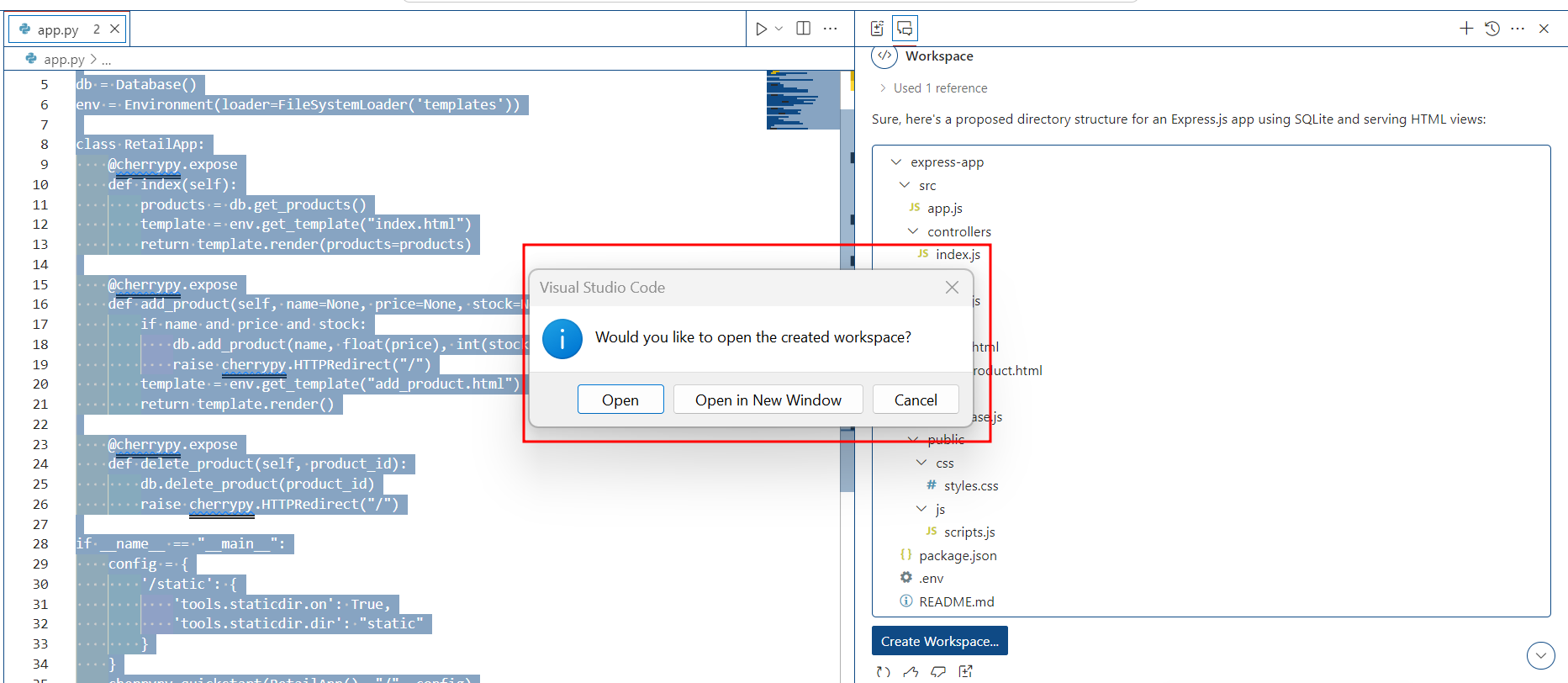
The new workspace for Express JS code is as follows:



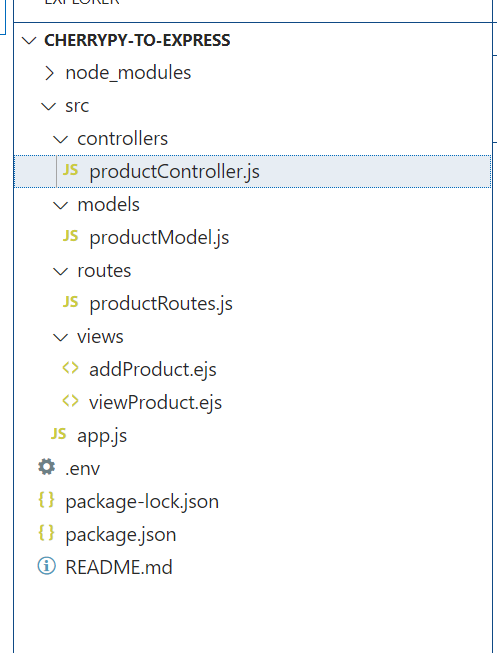
Once you click on create workspace. It will ask you workspace folder name



It create all files and folder and then display pop up message as

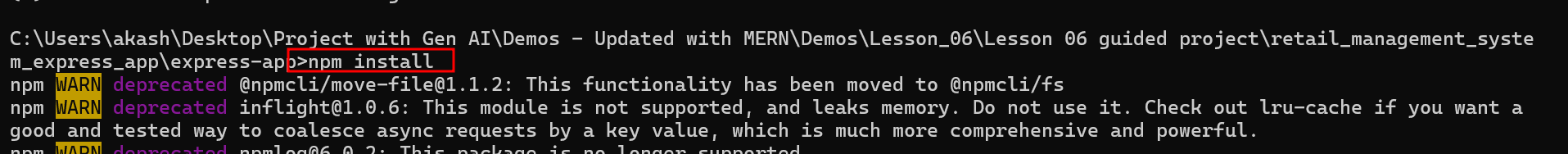


It create new folder structure for express js project as below



Now the command prompt or terminal inside a root directory of project.

Run **npm install** command to installed all required dependencies.



Then run main class part of src folder with name as app.js

**node start**

**Note**: some time you will get the error. Because of version, or missing code based upon the prompt message. So, you need to debug the code properly. If you get any error. Please refer the all-file code.

**package.json file code**

{

  "name": "cherrypy-to-express",

  "version": "1.0.0",

  "description": "A project to migrate from CherryPy to Express.js with MongoDB for product management.",

  "main": "src/app.js",

  "scripts": {

    "start": "node src/app.js",

    "dev": "nodemon src/app.js"

  },

  "dependencies": {

    "express": "^4.17.1",

    "mongoose": "^5.10.9",

    "ejs": "^3.1.6",

    "dotenv": "^8.2.0"

  },

  "devDependencies": {

    "nodemon": "^2.0.4"

  },

  "author": "",

  "license": "ISC"

}

**app.js file code**

var express = require('express');

var bodyParser = require('body-parser');

var path = require('path');

var mongoose = require('mongoose');

var productRoutes = require('./routes/productRoutes');

let url = 'mongodb://localhost:27017/retail\_db'

var app = express();

// Middleware for parsing request bodies

app.use(bodyParser.urlencoded({ extended: true }));

app.use(bodyParser.json());

// Set the view engine to EJS

app.set('view engine', 'ejs');

app.set('views', path.join(\_\_dirname, 'views'));

// Connect to MongoDB

mongoose.connect(url, { useNewUrlParser: true, useUnifiedTopology: true })

    .then(() => console.log('MongoDB connected'))

    .catch(err => console.error('MongoDB connection error:', err));

// Set up routes

app.use('/', productRoutes);

// Start the server

var PORT = process.env.PORT || 3000;

app.listen(PORT, function() {

    console.log('Server is running on port ' + PORT);

});

**viewProduct.ejs file code part of views folder**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>View Products</title>

</head>

<body>

    <h1>Product List</h1>

    <table>

        <thead>

            <tr>

                <th>Name</th>

                <th>Price</th>

                <th>Description</th>

                <th>Actions</th>

            </tr>

        </thead>

        <tbody>

            <% products.forEach(function(product) { %>

                <tr>

                    <td><%= product.name %></td>

                    <td><%= product.price %></td>

                    <td><%= product.description %></td>

                    <td>

                        <form action="/products/delete/<%= product.\_id %>" method="POST">

                            <button type="submit">Delete</button>

                        </form>

                    </td>

                </tr>

            <% }); %>

        </tbody>

    </table>

    <a href="/add">Add New Product</a>

</body>

</html>

**addProduct.ejs part of views folder**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Add Product</title>

</head>

<body>

    <h1>Add New Product</h1>

    <form action="/products/add" method="POST">

        <label for="name">Product Name:</label>

        <input type="text" id="name" name="name" required>

        <br>

        <label for="price">Price:</label>

        <input type="number" id="price" name="price" required>

        <br>

        <label for="description">Description:</label>

        <textarea id="description" name="description" required></textarea>

        <br>

        <button type="submit">Add Product</button>

    </form>

</body>

</html>

**productRoutes.js file part of routes folder**

const express = require('express');

const   routes = express.Router();

var productController = require('../controllers/productController');

routes.get('/', productController.viewProduct);

routes.get('/add', productController.addProductPage);

routes.post('/products/add', productController.addProduct);

routes.post('/products/delete/:id', productController.deleteProduct);

module.exports = routes;

**productModel.js file part of models folder**

var mongoose = require('mongoose');

var productSchema = new mongoose.Schema({

    name: {

        type: String,

        required: true

    },

    price: {

        type: Number,

        required: true

    },

    description: {

        type: String,

        required: true

    }

});

var Product = mongoose.model('Product', productSchema);

module.exports = Product;

**productController.js file part of controllers folder**

var ProductModel = require('../models/productModel');

function ProductController() {}

ProductController.prototype.addProductPage = function(req, res) {

    res.render('addProduct');

}

ProductController.prototype.addProduct = function(req, res) {

    var newProduct = new ProductModel({

        name: req.body.name,

        price: req.body.price,

        description: req.body.description

    });

    newProduct.save(function(err) {

        if (err) {

            return res.status(500).send(err);

        }

        ProductModel.find({}, function(err, products) {

            if (err) {

                return res.status(500).send(err);

            }

            res.render('viewProduct', { products: products });

        });

    });

};

ProductController.prototype.deleteProduct = function(req, res) {

    ProductModel.findByIdAndRemove(req.params.id, function(err) {

        if (err) {

            return res.status(500).send(err);

        }

        ProductModel.find({}, function(err, products) {

        if (err) {

            return res.status(500).send(err);

        }

        res.render('viewProduct', { products: products });

    });

    });

};

ProductController.prototype.viewProduct = function(req, res) {

    ProductModel.find({}, function(err, products) {

        if (err) {

            return res.status(500).send(err);

        }

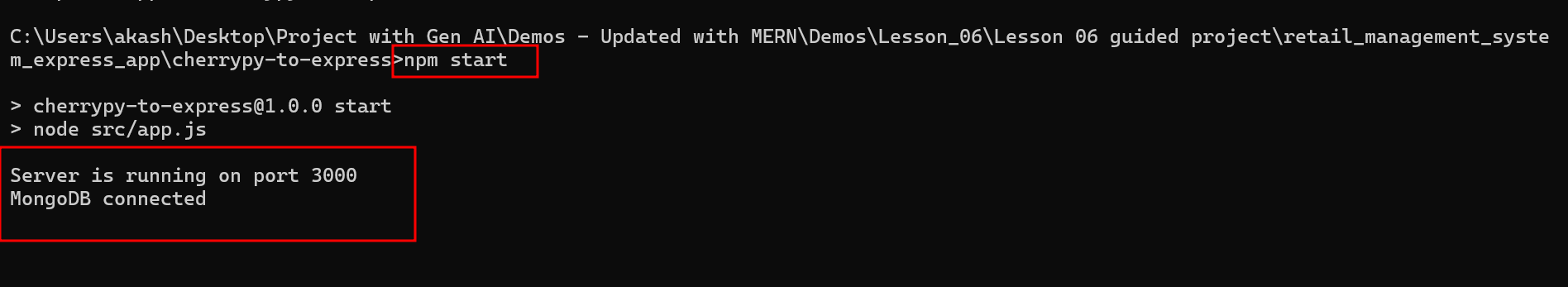
        res.render('viewProduct', { products: products });

    });

};

module.exports = new ProductController();

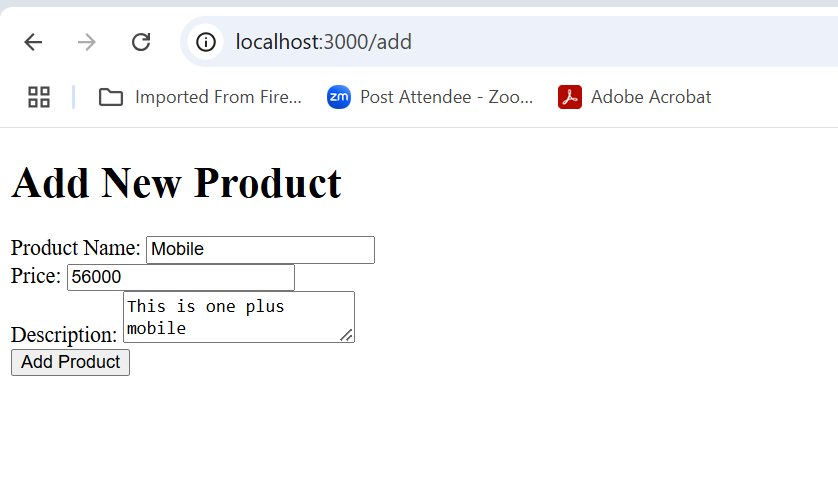
After debug and remove all error.

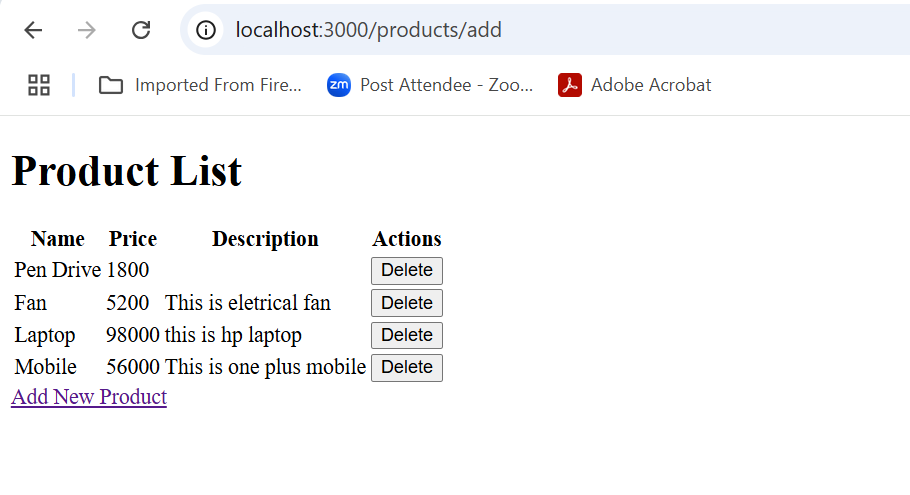


Open the browser

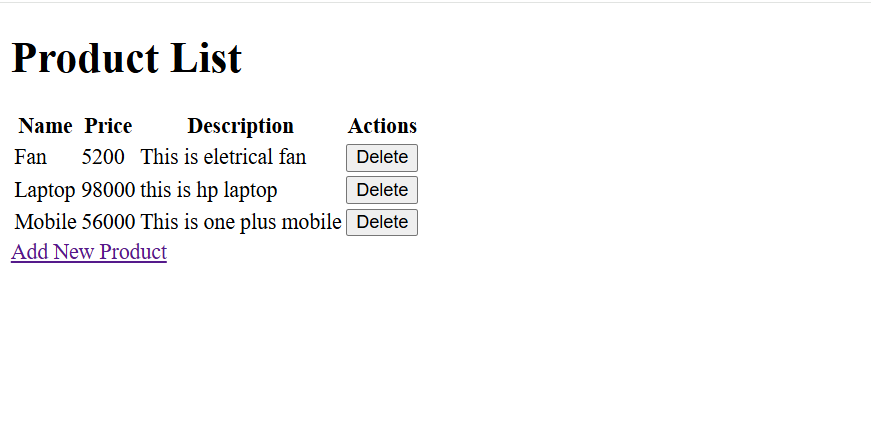
<http://localhsot:3000>







We can delete the particular product

**Task 4: Analyze the time and space complexity of the migrated code**

**using GitHub Copilot**

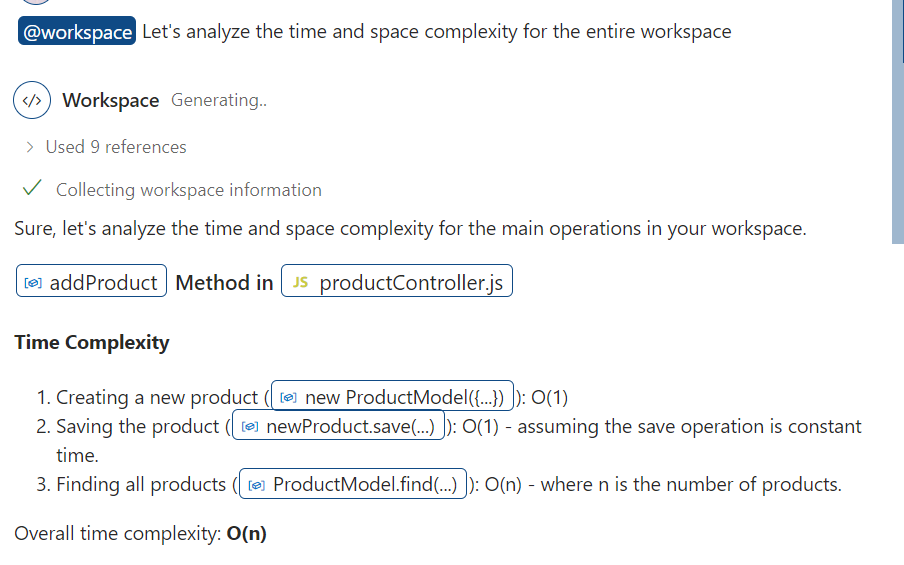
**Step 1: Analyze the time and space complexity of the migrated APIs**

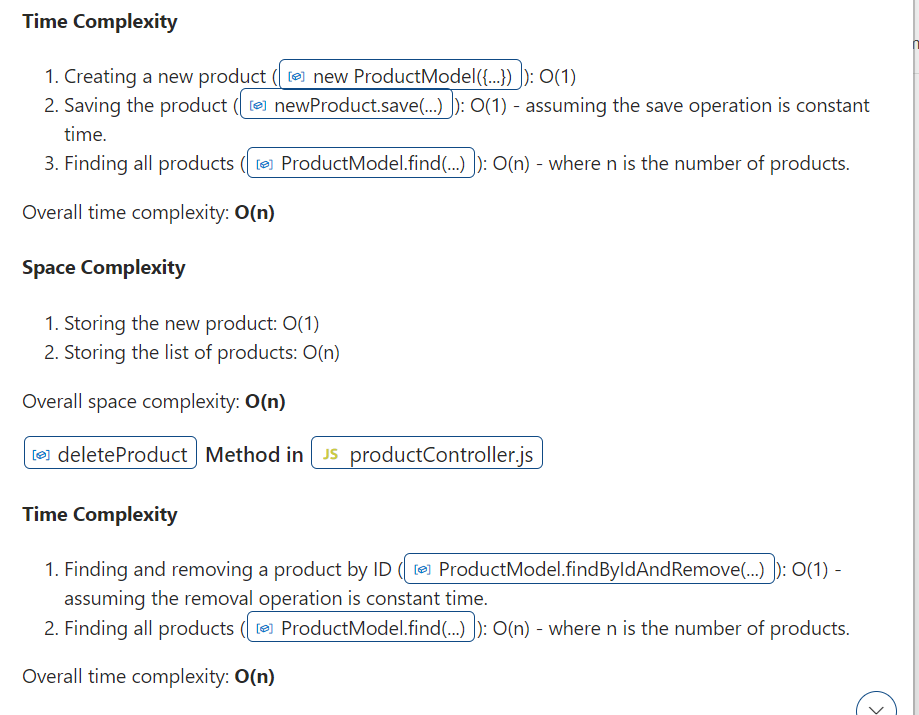
**using GitHub Copilot to enhance efficiency**

* 1. Analyze the migrated time and space complexity by using the following command:

**@workspace Let's analyze the time and space complexity for the entire workspace**

It provides the following response:



The existing code is optimized in terms of time and space complexity.